# ユーザーガイド AWS CodePipeline



### API バージョン 2015-07-09

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

### AWS CodePipeline: ユーザーガイド

Copyright © 2025 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
トリガー	9
[変数]	10
条件	10
ルール	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
パイプライン実行の停止方法	15
SUPERSEDED モードでの実行の処理方法	19
QUEUED モードでの実行の処理方法	20
PARALLEL モードでの実行の処理方法	22
パイプラインのフローを管理する	22
入力および出力アーティファクト	25
ステージ条件はどのように機能しますか?	28
ステージ条件のルール	31
パイプラインのタイプ	32
適切なパイプラインのタイプの選択	32
入門	38
ステップ 1: AWS アカウント および管理ユーザーを作成する	38

にサインアップする AWS アカウント	38
管理アクセスを持つユーザーを作成する	39
ステップ 2: CodePipeline への管理アクセスのためのマネージドポリシーを適用する	40
ステップ 3: をインストールする AWS CLI	42
ステップ 4: CodePipeline 用のコンソールを開く	43
次のステップ	43
製品およびサービスの統合	44
CodePipeline アクションタイプとの統合	44
ソースアクションの統合	44
ビルドアクションの統合	52
テストアクションの統合	54
デプロイアクションの統合	56
Amazon Simple Notification Service との承認アクションの統合	63
呼び出しアクションの統合	64
CodePipeline との一般的統合	65
コミュニティからの例	68
ブログ記事	68
チュートリアル	73
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	74
前提条件	74
ステップ 1: Amazon EC2 Linux インスタンスを作成する	75
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	F
る	77
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	78
ステップ 4: パイプラインを作成する	79
ステップ 5: パイプラインをテストする	85
チュートリアル: CodePipeline を使用して Docker イメージを構築して Amazon ECR にプッ	
シュする (V2 タイプ)	86
前提条件	87
ステップ 1: ソースリポジトリに Dockerfile を追加する	87
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	89
ステップ 3: パイプラインを作成する	90
ステップ 4: パイプラインのテスト	95
チュートリアル: CodePipeline を使用して Amazon EKS にデプロイする	96
前提条件	96
ステップ 1: (オプション) Amazon EKS でクラスターを作成する	97

ステップ 2: Amazon EKS でプライベートクラスターを設定する	100
ステップ 3: IAM で CodePipeline サービスロールポリシーを更新する	100
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	103
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	104
ステップ 6: パイプラインを作成する	105
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 (V2 タイ	
プ)	107
	108
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	108
ステップ 2: パイプラインを作成する	109
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	112
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	113
前提条件	114
ステップ 1: CloudShell を開いてリポジトリを複製する	115
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	115
ステップ 3: リリースに対するコミットにタグを付ける	119
ステップ 4: 変更をリリースしてログを表示する	120
チュートリアル: パイプラインを開始するためのプルリクエストのブランチ名をフィルタリン	
グする (V2 タイプ)	121
	122
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	122
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	ţ
る	125
チュートリアル: パイプラインレベルの変数を使用する	126
前提条件	127
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	127
ステップ 2: 変更をリリースしてログを表示する	130
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)	131
S3 バケットを作成する	132
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。	134
CodeDeploy でアプリケーションを作成する	137
最初のパイプラインを作成する	139
別のステージを追加する	142
	· · —

リソースをクリーンアップする	150
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)	151
「CodeCommit リポジトリを作成」	152
コードのダウンロード、コミット、プッシュする	153
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	156
CodeDeploy でアプリケーションを作成する	158
最初のパイプラインを作成する	159
CodeCommit リポジトリ内のコードを更新する	162
リソースをクリーンアップする	164
詳細情報	164
チュートリアル: 4 ステージのパイプラインを作成する	165
前提条件を満たす	166
パイプラインを作成する	171
ステージを追加する	173
リソースをクリーンアップする	176
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。	177
Amazon SNS を使用してEメール通知を設定します。	178
CodePipeline の CloudWatch Events 通知ルールを作成します。	179
リソースをクリーンアップする	181
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	181
Device Farm テストを使用するように CodePipeline を設定します。	182
チュートリアル: を使用して iOS アプリをテストする AWS Device Farm	187
Device Farm テスト(例 Amazon S3) を使用するように CodePipeline を設定する	189
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	194
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	195
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	200
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	205
例 1: を使用して AWS CodeCommit パイプラインを作成する AWS CloudFormation	205
例 2 : AWS CloudFormationを使用して Amazon S3 パイプラインを作成します。	208
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	211
前提条件: AWS CloudFormation サービスロールと CodeCommit リポジトリを作成しま	
す。	212
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	213

ステップ 2: パイプラインを作成する	. 214
ステップ 3: AWS 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	. 217
ステップ 4: 手動承認アクションを追加する	. 218
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	. 218
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	219
チュートリアル: CodePipeline を使用した Amazon ECS 標準デプロイ	. 220
. 前提条件	. 221
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	. 224
ステップ 2: 継続的デプロイパイプラインを作成する	. 226
ステップ 3: CodeBuild ロールに Amazon FCR 権限を追加する	228
ステップ 4・パイプラインのテスト	228
チュートリアル: Amazon FCR ソース FCS - CodeDeploy 間のデプロイでパイプラインを作	Ξ
	229
前提条件	231
ステップ 1 イメージを作成して Amazon FCR リポジトリにプッシュする	231
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	. 201
て CodeCommit Uポジトリにプッシュする	232
て、ObdeCommit リホントリピノンコリビーディーデットグループを作成する	236
ステップ 4: Amazon ECS クラスターとサービスを作成する	. 200
ステップ 5: CodeDeploy アプリケーションとデプロイグリープ (ECS コンピューティング	. 209
ステンテリ、CodeDeploy テラテーションとテラロキラルーラ (ECS コンピューテキンラ プラットフォーム) を作成する	242
ノノンドノオーム) を作成する	. 242
ステップ 0. ハイノノインを作成 9 る	. 243
ステッノ 7: ハイノノイノに変更を加えてテノロイを確認 9 る	. 240
テュートリアル: Amazon Alexa Skill をナノロイ 9 るハイノノイノを作成 9 る	. 248
	. 248
ステップ 1: Alexa テベロッハーサービス LWA セキュリティブロノアイルを作成する	. 249
ステップ 2 : Alexa スキルのソースファイルを作成して CodeCommit リホントリにノッシ	<u>т</u>
	. 249
ステッフ 3: ASK CLI コマンドを使用して更新トークンを作成する	. 251
ステップ 4: パイプラインを作成する	. 252
ムテップ 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	. 254
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	. 254
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	. 256
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 にテ	2
プロイする	. 261

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	266
[開始する前に]	267
ステップ 1: buildspec.yml ファイルを作成する	268
ステップ 2: パイプラインを作成して設定する	268
ステップ 3: 発行アプリケーションをデプロイする	270
ステップ 4: 発行アクションを作成する	271
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	272
前提条件	273
ステップ 1: Lambda 関数を作成する	273
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 そ	070
る チュートリアル・ANA/C Oten Functions IIIバルトアクションを使用する	270
テュートリアル: AWS Step Functions 呼び出しアクションを使用する	270
前提条件: シノノルなハイノノイノを作成または選択 9 る	270
ステップ 1. リノノルスナートマンノを作成する	279
ステッフ Z. ハイノフインにステップ 関数時の山しアウションを追加する	219
ブニードウアル. AppColling をアプロイプロバイヌとして使用するバイアフィンを作成しよ	ຉ៰៱
9。	200
前旋米什 フテップ 1: AM/S AppConfig リソーフを作成すろ	201
ステップ 1. AVVS AppCointing ワノースをIF成する	201
ステップ 2・ファイルを 35 ノースパテットにテップロートしょう。	202
ステップ /・任音のソースファイルに 恋面を加えてデプロイを確認します	203
イノンシューロミック ハノンイルに支史を加えてリンロイを確認したり。 チュートリアル・CodoCommit パイプラインソーフで空全たクローンを使用する	204
デュードラアル CodeCommit バイアライフラーバビル主なフローラ ど使用する	204
前旋来日	200
ステップ 0· パイプラインを作成してプロジェクトをビルドする	286
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	200
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	200
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	291
が「「「」」、「」、「」、「」、「」、「」、「」、「」、「」、「」、「」、「」、「	292
ステップ 1・RFADMF ファイルを作成する	292
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	293
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	296
ステップ 4: 構築出力でリポジトリコマンドを表示する	296
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	296

前提条件	297
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	298
ステップ 2: パイプラインを作成する	300
ステップ 3: 初期デプロイを表示する	303
ステップ 4: CloudFormationsStackInstances アクションを追加する	303
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	304
ステップ 6: スタックセットを更新する	305
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	305
前提条件	306
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	307
ステップ 2: パイプラインを作成する	307
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	310
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	312
ベストプラクティスとユースケース	315
CodePipeline の使用方法例	315
Amazon S3 で CodePipeline を使用する AWS CodeCommit、および Amazon S3 AWS	
CodeDeploy	315
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	316
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	317
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	317
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	317
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	317
AWS 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	318
Amazon VPC で CodePipeline を使用する	319
可用性	319
CodePipeline 用の VPC エンドポイントポリシーを作成する	320
VPC 設定のトラブルシューティング	321
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	322
パイプライン、ステージ、アクションを作成する	323
カスタムパイプラインを作成する (コンソール)	324

パイプラインを作成する (CLI)	337
静的テンプレートからパイプラインを作成する	343
パイプラインを編集する	350
パイプラインを編集する (コンソール)	351
パイプラインを編集する (AWS CLI)	354
パイプラインと詳細を表示する	359
パイプラインを表示する (コンソール)	359
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)	364
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。	367
パイプラインの詳細と履歴を表示する (CLI)	368
実行履歴でステージ条件のルール結果を表示する	369
パイプラインを削除します。	372
パイプラインを削除する (コンソール)	372
パイプラインを削除する (CLI)	372
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	374
前提条件: AWS KMS 暗号化キーを作成する	376
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	377
ステップ 2: パイプラインを編集する	385
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	388
ポーリングパイプラインを移行する方法	388
アカウント内のポーリングパイプラインの表示	390
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	395
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	417
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	445
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	481
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ウェブフックに	
移行する	484
CodePipeline サービスロールを作成する	502
CodePipeline サービスロールを作成する (コンソール)	502
CodePipeline サービスロールを作成する (CLI)	503
リソースのタグ付け	505
パイプラインにタグ付けする	506
パイプラインにタグ付けする (コンソール)	507
パイプラインにタグ付けする (CLI)	508
通知ルールの作成	511

	515
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	518
Amazon ECR ソースアクションと EventBridge	518
Amazon ECR ソースに対する EventBridge ルールを作成する (コンソール)	519
Amazon ECR ソースに対する EventBridge ルールを作成する (CLI)	521
Amazon ECR ソースの EventBridge ルールを作成する (AWS CloudFormation テンプレー	-
ト)	525
EventBridge イベントを使用する Amazon S3 ソースアクションへの接続	530
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 (CLI)	531
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 (AWS CloudFormatio	n
テンプレート)	535
EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail	558
Amazon S3 ソースに対する EventBridge ルールを作成する (コンソール)	559
Amazon S3 ソースに対する EventBridge ルールを作成する (CLI)	563
Amazon S3 ソースの EventBridge ルールを作成する (AWS CloudFormation テンプレート	<b>`</b> )
Amazon S3	569
CodeCommit ソースアクションと EventBridge	581
CodeCommit ソースに対する EventBridge ルールを作成する (コンソール)	582
CodeCommit ソースに対する EventBridge ルールを作成する (CLI)	585
CodeCommit ソースの EventBridge ルールを作成する (AWS CloudFormation テンプレー	
ト)	590
CodeConnections 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	598
Bitbucket Cloud への接続	598
Bitbucket Cloud への接続を作成する (コンソール)	600
Bitbucket Cloud への接続を作成する (CLI)	604
GitHub コネクション	606
GitHub (コンソール) への接続を作成する	608
GitHub (CLI) への接続を作成する	611
GitHub Enterprise Server 接続	613
GitHub Enterprise Server への接続を作成する (コンソール)	614
GitHub Enterprise Server (CLI) への接続を作成します。	618
GitLab.com への接続	621
GitLab.com への接続を作成する (コンソール)	623
GitLab.com への接続を作成する (CLI)	627
GitLab セルフマネージドの接続	629

GitLab セルフマネージドへの接続を作成する (コンソール)	. 631
GitLab セルフマネージドへのホストと接続を作成する (CLI)	. 634
別のアカウントと共有されている接続を使用する	637
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	. 638
トリガーフィルターに関する考慮事項	640
プロバイダー別のトリガーのプルリクエストイベント	. 641
トリガーフィルターの例	. 642
フィルターなしでコードプッシュにトリガーを追加する	651
コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する	. 651
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 (コンソール)	653
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 (CLI)	655
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 (AWS	
CloudFormation テンプレート)	658
トリガーを追加して変更検出をオフにする	. 659
パイプラインを手動で開始および停止する	. 661
CodePipeline でパイプラインを編集する	661
パイプラインを手動で開始する	663
スケジュールに基づいたパイプラインの開始	664
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	668
パイプライン実行を停止する	671
パイプライン実行を停止する (コンソール)	672
インバウンド実行を停止します (コンソール)。	. 675
パイプライン実行を停止する (CLI)	675
インバウンド実行 (CLI) を停止します。	. 677
パイプライン実行の履歴を表示し、モードを設定する	679
実行を表示する	679
パイプライン実行の履歴を表示する (コンソール)	679
実行のステータスを表示する (コンソール)	. 681
インバウンドの実行(コンソール)を表示します。	683
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)	684
アクション実行を表示する (コンソール)	686
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)	. 687
パイプラインの詳細と履歴を表示する (CLI)	. 687
パイプライン実行モードを設定または変更する	. 699
実行モードの表示に関する考慮事項	. 700
実行モード間を切り替える場合の考慮事項	703

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)	704
パイプライン実行モードを設定する (CLI)	705
ステージをロールバックまたは再試行する	709
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	709
ステージ再試行に関する考慮事項	710
失敗したステージを手動で再試行する	710
ステージ障害時の自動再試行を設定する	714
ステージロールバックの設定	720
ロールバックに関する考慮事項	720
ステージを手動でロールバックする	720
ステージの自動ロールバックを設定する	726
実行リストでロールバックステータスを表示する	730
ロールバックステータスの詳細を表示する	733
ステージの条件を設定する	738
ステージ条件のユースケース	739
ステージ条件に設定する結果に関する考慮事項	739
ステージ条件に設定するルールに関する考慮事項	740
入力条件の作成	740
入力条件の作成 - CloudWatchAlarm ルールの例 (コンソール)	741
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)	742
入力条件の作成 (CLI)	744
入力条件の作成 (CFN)	746
失敗時の条件の作成	747
失敗時の条件の作成 (コンソール)	747
再試行結果の例を使用した OnFailure 条件の作成 (コンソール)	748
失敗時の条件の作成 (CLI)	749
失敗時の条件の作成 (CFN)	751
成功時の条件の作成	752
成功時の条件の作成 (コンソール)	752
成功時の条件の作成 (CLI)	754
成功時の条件を作成する (CFN)	756
ステージ条件の削除	757
ステージ条件の上書き	758
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	760
アクションタイプの使用	760
アクションタイプをリクエストする	762

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)	768
アクションタイプを表示する	770
アクションタイプを更新する	771
パイプラインのカスタムアクションを作成する	773
カスタムアクションを作成する	775
カスタムアクションのジョブワーカーを作成する	779
パイプラインにカスタムアクションを追加する	786
CodePipeline でカスタムアクションにタグ付けする	789
カスタムアクションにタグを追加する	789
カスタムアクションのタグを表示する	790
カスタムアクションのタグを編集する	791
カスタムアクションからタグを削除する	791
パイプラインで Lambda 関数を呼び出す	791
ステップ 1: パイプラインを作成する	794
ステップ 2:Lambda 関数を作成する	795
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	799
ステップ 4:Lambda 関数でパイプラインをテストする	800
ステップ 5: 次のステップ	801
JSON イベントの例	802
追加のサンプル関数	803
手動の承認アクションをステージに追加する	816
手動の承認アクションに関する設定オプション	817
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	818
CodePipeline で IAM ユーザーに承認アクセス許可を付与する	819
サービスロールへの Amazon SNS アクセス許可の付与	821
手動の承認アクションを追加する	823
承認アクションを承認または拒否する	827
手動の承認通知の JSON データ形式	831
パイプラインにクロスリージョンアクションを追加する	832
パイプラインのクロスリージョンアクションを管理する (コンソール)	834
パイプラインにクロスリージョンアクションを追加する (CLI)	837
パイプラインにクロスリージョンアクションを追加する (AWS CloudFormation)	842
変数の操作	845
変数のアクションを設定する	846
出力変数を表示する	850
例: 手動承認で変数を使用する	853

例:CodeBuild 環境変数で BranchName 変数を使用する	853
ステージ移行の操作	856
移行を無効化または有効化する (コンソール)	856
移行を無効化または有効化する (CLI)	858
パイプラインのモニタリング	860
CodePipeline イベントのモニタリング	861
詳細タイプ	862
パイプラインレベルのイベント	865
ステージレベルのイベント	873
アクションレベルのイベント	877
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	885
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	889
イベントのプレースホルダーバケット名 (リージョン別)	890
AWS CloudTrail を使用した API コールのログ記録	893
CloudTrail での CodePipeline 情報	894
CodePipeline ログファイルエントリについて	895
CodePipeline CloudWatch メトリクス	897
PipelineDuration	898
FailedPipelineExecutions	898
トラブルシューティング	899
パイプラインのエラー: AWS Elastic Beanstalk で設定されたパイプラインは次のようなコ	ラー
メッセージを返します。「デプロイに失敗しました。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」	900
デプロイエラー: 「DescribeEvents」アクセス許可がない場合、 AWS Elastic Beanstalk 🗄	デプロ
イアクションで設定したパイプラインは、失敗ではなくハングアップ状態になります。 .	901
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。」	901
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。	902
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 1 つの	
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。	902
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したが AWS	;

	パノプニノン ㅠㅠ ノファムレフェル ダクゼロレきゅう キマいえ ヒミに 日ミキキ	000
	ハイノフインのアーティノアクトノオルダ名が切り詰められているように見えます	903
	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。	904
	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。	905
	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< td=""><td></td></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。	907
	GitHub (OAuth アプリ経由) ソースアクション: リポジトリリストには異なるリポジトリが表示	Ŧ
	されます	907
	GitHub (GitHub App 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	907
	Amazon S3 エラー: CodePipeline サービスロール <arn> により、S3 バケット</arn>	
	<bucketname> に対する S3 アクセスが拒否されました。</bucketname>	908
	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。	910
	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」	912
	実行モードを QUEUED または PARALLEL モードに変更したパイプラインは、実行制限に達	
	すると失敗します	912
	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。	913
	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。	913
	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	<u>.</u>
	の作成時に開始しない可能性があります	914
	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	914
	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-	_
	致しない可能性があります	915
	SC2 デプロイアクションがTラーメッセージで失敗する No such file	915
	FKS デプロイアクションがcluster unreachableTラーメッセージで失敗する	916
	別の問題があるため問い合わせ先を教えてください。	917
╆	キュリティ	918
	イーク保護	919
	インターネットトラフィックのプライバシー	920
	保管中の暗号化	920
	転送中の暗号化	020 021
		<u>5</u> 21

暗号化キーの管理	921
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	921
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	. 924
アイデンティティ/アクセス管理	925
対象者	926
アイデンティティを使用した認証	. 926
ポリシーを使用したアクセスの管理	929
が IAM と AWS CodePipeline 連携する方法	. 932
アイデンティティベースのポリシーの例	938
リソースベースのポリシーの例	. 975
トラブルシューティング	. 976
CodePipeline 許可リファレンス	. 979
CodePipeline サービスロールを管理する	. 989
インシデントへの対応	. 996
コンプライアンス検証	. 997
耐障害性	. 998
インフラストラクチャセキュリティ	998
セキュリティに関するベストプラクティス	999
パイプライン構造リファレンス	1001
パイプライン宣言	1004
name	1006
roleArn	1007
artifactStore または artifactStores	1007
stages	1008
version	1009
executionMode	1009
pipelineType	1009
variables	1010
triggers	1010
metadata	1014
ステージ宣言	1015
name	1018
actions	1018
conditions	1018

r	ules	1019
アク	?ションの宣言	1019
		1019
r	name	1023
r	egion	1023
r	coleArn	1023
r	namespace	1024
a	actionTypeId	1024
I	InputArtifacts	1025
С	outputArtifacts	1026
c	configuration (アクションプロバイダー別)	1027
I	unOrder	1029
Cod	lePipeline の有効なアクションプロバイダー	1030
Pol	1ForSourceChanges パラメータの有効な設定	1035
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	1037
プロ	1バイダータイプ別の有効な設定パラメータ	1038
アクシ	ョン構造リファレンス	1043
Ama	azon EC2 アクションリファレンス	1044
7	アクションタイプ	1045
Ē	没定パラメータ	1045
,	入力アーティファクト	1049
L	出力アーティファクト	1049
E	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	1049
7	アクションの宣言	1052
	関連情報	1053
Ama	azon ECR ソースアクションリファレンス	1053
7	アクションタイプ	1054
Ē	没定パラメータ	1055
,	入力アーティファクト	1055
Ļ	出力アーティファクト	1055
L	出力変数	1055
+	ナービスロールのアクセス許可: Amazon ECR アクション	1056
7	アクションの宣言 (Amazon ECR の例)	1056
	関連情報	1058
ECR	BuildAndPublish ビルドアクションリファレンス	1058
7	アクションタイプ	1059

設定パラメータ	1059
入力アーティファクト	1060
出力アーティファクト	1060
出力変数	1060
サービスロールのアクセス許可: ECRBuildAndPublishアクション	1061
アクションの宣言	1063
関連情報	1064
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	1064
アクションタイプ	1065
設定パラメータ	1065
入力アーティファクト	1067
出力アーティファクト	1068
サービスロールのアクセス許可: CodeDeployToECSアクション	1068
アクションの宣言	1070
関連情報	1072
Amazon Elastic Container Service デプロイアクションリファレンス	1072
アクションタイプ	1073
設定パラメータ	1074
入力アーティファクト	1074
出力アーティファクト	1075
サービスロールのアクセス許可: Amazon ECS 標準アクション	1075
アクションの宣言	1077
関連情報	1078
Amazon Elastic Kubernetes Service EKSデプロイアクションリファレンス	1078
アクションタイプ	1079
設定パラメータ	1080
入力アーティファクト	1081
出力アーティファクト	1081
環境変数	1081
出力変数	1082
サービスロールのポリシーのアクセス許可	1082
アクションの宣言	1085
関連情報	1086
Amazon S3 デプロイアクションリファレンス	1086
アクションタイプ	1086
設定パラメータ	1087

入力アーティファクト	1088
出力アーティファクト	1088
サービスロールのアクセス許可: S3 デプロイアクション	1089
アクション設定の例	1089
関連情報	1092
Amazon S3 ソースアクションリファレンス	1093
アクションタイプ	1094
設定パラメータ	1094
入力アーティファクト	1096
出力アーティファクト	1096
出力変数	1097
サービスロールのアクセス許可: S3 ソースアクション	1097
アクションの宣言	1098
関連情報	1099
AWS AppConfig デプロイアクションリファレンス	1100
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	1101
出力アーティファクト	1101
サービスロールのアクセス許可: AppConfigアクション	1101
アクション設定の例	1102
関連情報	1103
AWS CloudFormation デプロイアクションリファレンス	1103
アクションタイプ	1104
設定パラメータ	1104
入力アーティファクト	1109
出力アーティファクト	1110
出力変数	1110
サービスロールのアクセス許可: AWS CloudFormation アクション	1110
アクションの宣言	1112
関連情報	1113
AWS CloudFormation StackSets	1114
Stack AWS CloudFormation StackSets アクションの仕組み	1115
パイプラインで StackSets アクションを構成する方法	1117
CloudFormationStackSet アクション	1118
CloudFormationStackInstances アクション	1132

サービスロールのアクセス許可: CloudFormationStackSetアクション	1142
サービスロールのアクセス許可: CloudFormationStackInstancesアクション	1143
スタックセットオペレーションのアクセス許可モデル	1143
テンプレートパラメータのデータタイプ	1144
関連情報	1113
AWS CodeBuild ビルドおよびテストアクションリファレンス	1146
アクションタイプ	1147
設定パラメータ	1147
入力アーティファクト	1149
出力アーティファクト	1150
出力変数	1151
サービスロールのアクセス許可: CodeBuild アクション	1151
アクション宣言(CodeBuild の例)	1151
関連情報	1153
AWS CodePipeline アクションリファレンスを呼び出す	1153
アクションタイプ	1154
設定パラメータ	1154
入力アーティファクト	1157
出力アーティファクト	1158
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	1158
アクションの宣言	1158
関連情報	1159
AWS CodeCommit ソースアクションリファレンス	1159
アクションタイプ	1160
設定パラメータ	1161
入力アーティファクト	1162
出力アーティファクト	1162
出力変数	1163
サービスロールのアクセス許可: CodeCommit アクション	1164
アクション設定の例	1164
関連情報	1167
AWS CodeDeploy デプロイアクションリファレンス	1167
アクションタイプ	1168
設定パラメータ	1168
入力アーティファクト	1168
出力アーティファクト	1168

サービスロールのアクセス許可: AWS CodeDeploy アクション	1169
アクションの宣言	1170
関連情報	1171
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)	1172
アクションタイプ	1176
設定パラメータ	1176
入力アーティファクト	1177
出力アーティファクト	1177
出力変数	1178
サービスロールのアクセス許可: CodeConnections アクション	1179
アクションの宣言	1179
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	1181
関連情報	1181
コマンドアクションリファレンス	1182
コマンドアクションに関する考慮事項	1183
サービスロールのポリシーのアクセス許可	1184
アクションタイプ	1185
設定パラメータ	1185
入力アーティファクト	1190
出力アーティファクト	1190
環境変数	1190
サービスロールのアクセス許可: コマンドアクション	1192
アクションの宣言 (例)	1192
関連情報	1194
AWS Device Farm テストアクションリファレンス	1194
アクションタイプ	1194
設定パラメータ	1195
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: AWS Device Farm アクション	1199
アクションの宣言	1200
関連情報	1201
Elastic Beanstalk デプロイアクションリファレンス	1202
アクションタイプ	1202
設定パラメータ	1202

入力アーティファクト	1203
出力アーティファクト	1203
サービスロールのアクセス許可: アクションをElasticBeanstalkデプロイする	1203
アクションの宣言	1204
関連情報	1205
Amazon Inspector InspectorScan 呼び出しアクションリファレンス	1205
アクションタイプ ID	1206
設定パラメータ	1207
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
サービスロールのアクセス許可: InspectorScanアクション	1209
アクションの宣言	1210
関連情報	1211
AWS Lambda アクションリファレンスを呼び出す	1211
アクションタイプ	1212
設定パラメータ	1212
入力アーティファクト	1213
出力アーティファクト	1213
出力変数	1213
アクション設定の例	. 1213
JSON イベントの例	1214
関連情報	1217
AWS OpsWorks デプロイアクションリファレンス	1217
アクションタイプ	1217
設定パラメータ	1217
入力アーティファクト	1218
出力アーティファクト	1218
サービスロールのアクセス許可: AWS OpsWorks アクション	1218
アクション設定の例	. 1219
関連情報	1220
AWS Service Catalogデプロイアクションリファレンス	1220
 アクションタイプ	1220
設定パラメータ	1220
入力アーティファクト	1221
出力アーティファクト	1221

サービスロールのアクセス許可: Service Catalog アクション	1221
設定ファイルの種類別のアクション設定の例	1222
関連情報	1223
AWS Step Functions	1223
アクションタイプ	1217
設定パラメータ	1224
入力アーティファクト	1226
出力アーティファクト	1226
出力変数	1226
サービスロールのアクセス許可: StepFunctionsアクション	1226
アクション設定の例	1227
行動	1230
関連情報	1103
ルール構造リファレンス	1233
CloudwatchAlarm	1233
ルールタイプ	1234
設定パラメータ	1234
ルール設定の例	1234
関連情報	1235
CodeBuild ルール	1236
サービスロールのポリシーのアクセス許可	1236
ルールタイプ	1237
設定パラメータ	1237
ルール設定の例	1239
関連情報	1240
コマンド	1240
コマンドルールに関する考慮事項	1240
サービスロールのポリシーのアクセス許可	1236
ルールタイプ	1237
設定パラメータ	1237
ルール設定の例	1239
関連情報	1240
DeploymentWindow	1245
ルールタイプ	1246
設定パラメータ	1246
ルール設定の例	1248

関連情報	1249
LambdaInvoke	1249
ルールタイプ	1234
設定パラメータ	1250
ルール設定の例	1250
関連情報	1251
VariableCheck	1251
ルールタイプ	1252
設定パラメータ	1252
ルール設定の例	1254
関連情報	1255
統合モデルのリファレンス	1256
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	1256
概念	1257
サポートされている統合モデル	1259
Lambda 統合モデル	1260
Lambda 関数を更新して CodePipeline からの入力を処理します。	1260
Lambda 関数の結果を CodePipeline に返す	1265
継続トークンを使用して、非同期プロセスの結果を待つ	1267
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。	1267
ジョブワーカー統合モデル	1267
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	1268
イメージ定義ファイルのリファレンス	1271
Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル	1271
Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル	1274
変数リファレンス	1279
概念	1280
[変数]	1280
名前空間	1281
変数のユースケース	1282
変数の設定	1283
パイプラインレベルの変数を設定する	1283
アクションレベルの変数の設定	1284
変数の解決	1286
変数のルール	1287

パイプラインアクションで使用できる変数	1288
定義された変数キーを持つアクション	1288
ユーザー設定の変数キーを使用したアクション	1292
構文での glob パターンの使用	1295
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	1297
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	2
更新する	1298
ステップ 1: (OAuth アプリを介して) GitHub アクションを置き換える	1300
ステップ2:GitHub への接続を作成する	1300
ステップ 3: GitHub のソースアクションを保存する	1301
クォータ	1303
付録 A: GitHub (OAuth アプリ経由) ソースアクション	1319
GitHub (OAuth アプリ経由) ソースアクションの追加	1320
GitHub (OAuth アプリ経由) ソースアクションリファレンス	1321
アクションタイプ	1322
設定パラメータ	1322
入力アーティファクト	1324
出力アーティファクト	1324
出力変数	1324
アクションの宣言 (GitHub の例)	1325
GitHub (OAuth) への接続	1326
関連情報	1327
ドキュメント履歴	1328
以前の更新	1363
CodePipeline 機能リファレンス	1375
mcccl	xxviii

## 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、<u>コスト</u>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
- <u>パイプライン実行の仕組み</u>
- 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?
- <u>パイプラインのタイプ</u>
- 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。

「<u>継続的な配信</u>」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。

<u>継続的統合</u> 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。現在 CodePipeline で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<u>CodePipeline との製品とサー</u> 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. <u>CodePipeline チュートリアル</u> 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. <u>パイプライン、ステージ、アクションを作成する</u>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、「<u>DevOps パイプライン</u>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
- <u>パイプライン実行</u>
- <u>ステージオペレーション</u>
- アクション実行
- <u>実行タイプ</u>
- アクションタイプ
- アーティファクト
- ソースリビジョン
- トリガー
- [変数]
- <u>条件</u>
- ルール

### 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。 ステージ

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<u>アーティファクト</u>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。

Pip	eline
Stage	
Action	Action
	Transition
St	tage
Action	Action
	Action

#### 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 run0rder 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つ実 行は、インバウンドの実行を参照してください。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、「<u>インバウン</u> 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、「<u>パイプラ</u> 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 FAILED ステータスで、ステージはロック解除されます。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、「<u>SUPERSEDED モードでの実行の処理方</u> 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、「<u>ステージ実行</u>」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、「<u>ステージ</u> <u>ロールバックの設定</u>」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<u>アーティファクト</u>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 の製品や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。

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、<u>統合モデル</u> 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。

#### トリガー

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、「<u>CodePipeline でパイプラインを編集する</u>」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、「<u>チュー</u> 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になっているパイプラインでは、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た変数の値は、そのアクションが正常に完了した後に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。

#### 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。

ルール

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、「<u>ステージ条件はどのように機能しますか?</u>」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。

- 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。
- 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。
- 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。
- 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、「<u>パイプライン実行モードを設定また</u> 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、「<u>PARALLEL モードの CodeCommit または S3 ソースリビジョ</u> 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、「<u>PARALLEL モードの CodeCommit または S3 ソースリビジョンは、EventBridge</u> <u>イベントと一致しない可能性があります</u>」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。

## 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。

 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。

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。
- 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。

Execution history Info			Stop execution View d		
Q					< 1 )
	Execution ID	Status	Source revisions	Duration	Completed
0	bf3dc924-	⊖ Stopped	Source – Added main.css	22 seconds	Jan 16, 2020 8:28 AM (UTC-8:00)

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。 AWS CloudFormation アクションの場 合、パイプライン内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。



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、「<u>パイプライン実行モードを</u> <u>設定または変更する</u>」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、「<u>AWS</u> 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。



## MyPipeline

Note: maximum of 50 concurrent executions per pipeline

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、「<u>パイプライン実行モードを</u> <u>設定または変更する</u>」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、「<u>AWS</u> 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、他の実行が完了するまで 待たずに開始します。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、「<u>パイプライン実行モードを</u> <u>設定または変更する</u>」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、「<u>AWS</u> 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。



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、パイプラインソースアクションにアプリケーションソース コードファイルが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 のように ZIP ファイルとして提供されますSampleApp\_Windows.zip。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、そのアクションの出力 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す。 別の例として、ビルドアクションは、アプリケーションソースコードファイルである入力アーティ ファクトのアプリケーションソースコードをコンパイルするビルドコマンドを実行する場合があり ます。CodeBuild アクションの <u>AWS CodeBuild ビルドおよびテストアクションリファレンス</u> 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。

以下に例を示します。

- 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。
- 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。
- 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 トは codepipeline-*region-12345EXAMPLE* という名前で、*region* はパイプラインを作成した AWS リージョン、12345EXAMPLE はバケット名が一意であることを確認する 12 桁の乱数です。

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-regiondefexample 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は、アクションの作成先のリージョンにデフォルトのアーティ ファクトバケットが 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。条件が失敗すると、結果が適用され ます。ステージは、条件が失敗した場合にのみ、指定された結果を実行します。ルールのー 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。

- 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。

ステージ条件はどのように機能しますか?

- 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。

Stage	configured for entry condition	
Condition: Entry Results: Block or Enter	Override       Rule1 { }       Nule2 { }	CWE
Rule2 status: Failed (Error), Result: Block Rule2 status: Succeeded, Result: Enter		
<ul> <li>Rule1 status: Failed (Alarm), Result: Block</li> <li>Rule2 status: Succeeded, Result: Enter</li> <li>Override results in Stage status: Enter</li> </ul>		

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。

Stage co	onfigured for On Failure condition	-
Condition: OnFailure Result: Rollback	Override       Rule1 { }       Nule2 { }	CWE
	Rule2 status: Failed (Er Rule2 status: Succeeder Rule1 status: Failed (Al Rule2 status: Succeeder	g ror), Result: <b>Rollback</b> ed, Result: <b>Fail stage</b> arm), Result: <b>Rollback</b> ed, Result: <b>Fail stage</b>
Override results in Stage status: Failed		

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 Lambda Invoke 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。

Condition: Entry Results: Rollback or Fail	Override
	<ul> <li>Rule2 status: Failed (Error), Result: Rollback or Fail</li> <li>Rule2 status: Succeeded, Result: Proceed</li> <li>Rule1 status: Failed (Alarm), Result: Rollback or Fail</li> <li>Rule2 status: Succeeded, Result: Proceed</li> <li>Qverride results in Stage status: Proceed</li> </ul>

Stage configured for On Success condition

## 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。条件の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。アラーム状態として OK、ALARM、または INSUFF\_DATA がチェックで返 されると、ルールは満たされます。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、<u>CodePipeline パイプライン構造リファレンス</u>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、「<u>適切な</u> 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	<ul> <li>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</li> <li>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</li> </ul>
<u>アクションレベルの変数</u>	サポート	サポート
PARALLEL 実行モード	サポート外	サポート
<u>パイプラインレベルの変数</u>	サポート外	サポート
<u>QUEUED 実行モード</u>	サポート外	サポート
<u>パイプラインステージのロー</u> <u>ルバック</u>	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
<u>ステージ条件</u>	サポート外	サポート
<u>Git タグ、プルリクエスト、ブ</u> 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
<u>コマンドアクション</u>	サポート外	サポート
<u>スキップ結果を使用した入力</u> 条件の作成	サポート外	サポート
<u>ステージ障害時の自動再試行</u> <u>を設定する</u>	サポート外	サポート

CodePipeline の料金については、<u>料金</u>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。
- 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:</pre>
    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))):</pre>
                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:'))</pre>
   print ("| {:<40} | {:<10}".format('______</pre>
            ___'))
    print ("| {:<40} | {:<10}".format('Total action executions:',</pre>
 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))</pre>
```

- analyze\_cost\_in\_v2(pipeline)
- 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。ここで、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:

Ì	Total action executions:	Ì	9
I	Total billing action executions:	L	9
l	Total billing action execution minutes:	L	5.59
I	Cost of moving to V2 in \$:	I	-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。

0

)。

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

- ステップ 1: AWS アカウント および管理ユーザーを作成する
- ステップ 2: CodePipeline への管理アクセスのためのマネージドポリシーを適用する
- ステップ 3: をインストールする AWS CLI
- ステップ 4: CodePipeline 用のコンソールを開く
- 次のステップ

ステップ 1: AWS アカウント および管理ユーザーを作成する

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

- 1. https://portal.aws.amazon.com/billing/signup を開きます。
- 2. オンラインの手順に従います。

サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力 するように求められます。

にサインアップすると AWS アカウント、 AWS アカウントのルートユーザー が作成されます。 ルートユーザーには、アカウントのすべての AWS のサービス とリソースへのアクセス権があ ります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルー トユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。<u>https://</u> <u>aws.amazon.com/</u> の [マイアカウント] をクリックして、いつでもアカウントの現在のアクティビ ティを表示し、アカウントを管理することができます。

### 管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、日常的なタスクにルートユーザーを使用しないように AWS アカウントのルートユーザー、 のセキュリティを確保し AWS IAM Identity Center、 を有効に して管理ユーザーを作成します。

を保護する AWS アカウントのルートユーザー

 ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有 者<u>AWS Management Console</u>として にサインインします。次のページでパスワードを入力しま す。

ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイ ドのルートユーザーとしてサインインするを参照してください。

2. ルートユーザーの多要素認証 (MFA) を有効にします。

手順については、IAM <u>ユーザーガイドの AWS アカウント 「ルートユーザー (コンソール) の仮</u> 想 MFA デバイスを有効にする」を参照してください。

#### 管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

手順については、「AWS IAM Identity Center ユーザーガイド」の「<u>AWS IAM Identity Centerの</u> 有効化」を参照してください。

2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリア ルについては、 AWS IAM Identity Center ユーザーガイドの<u>「デフォルトを使用してユーザーア</u> クセスを設定する IAM アイデンティティセンターディレクトリ」を参照してください。

### 管理アクセス権を持つユーザーとしてサインインする

 IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティ センターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

IAM Identity Center ユーザーを使用してサインインする方法については、AWS サインイン 「 ユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラク ティスに従ったアクセス許可セットを作成します。

手順については、「AWS IAM Identity Center ユーザーガイド」の「<u>権限設定を作成する</u>」を参 照してください。

グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

手順については、「AWS IAM Identity Center ユーザーガイド」の「<u>グループの結合</u>」を参照し てください。

# ステップ 2: CodePipeline への管理アクセスのためのマネージドポ 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、<u>「 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</u>」 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、<u>CodePipeline サービスロールからアクセス許可を削除する</u>を参照してください。

アクセス権限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。

・ 以下のユーザーとグループ AWS IAM Identity Center:

アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「<u>権限設定を</u> 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「<u>サード</u> <u>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</u>」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「<u>IAM</u> 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。詳細については「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセ ス権限の追加」を参照してください。

# ステップ 3: をインストールする 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

 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、<u>AWS 「 コマンド</u> 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、「 <u>のインストール AWS</u> <u>Command Line Interface</u>」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。

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、パイプラインを作成するリージョン (us-east-2 など) を指定します。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 configure

```
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、<u>Managing</u> <u>Access Keys for IAM Users</u> および <u>How Do I Get Credentials?</u> 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。

# ステップ 4: CodePipeline 用のコンソールを開く

 にサインイン AWS Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」で CodePipeline コンソールを開きます。

# 次のステップ

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
- <u>Amazon S3 ソースアクション</u>

- <u>Bitbucket Cloud、GitHub (GitHub アプリ経由)、GitHub Enterprise Server、GitLab.com,GitLab セ</u> 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

<u>Amazon ECR</u> 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

<u>Amazon S3</u> 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、<u>Amazon S3 ソースアクションリ</u> ファレンス を参照してください
- <u>ステップ 1: アプリケーションの S3 バケットを作成する</u>

### ・ <u>パイプラインを作成する (CLI)</u>

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。「<u>CodePipeline との一般的統合</u>」を参照してください。

Bitbucket Cloud、GitHub (GitHub アプリ経由)、GitHub Enterprise Server、GitLab.com,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、 「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドア</u> クションの場合)」の注意を参照してください。

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。「<u>Bitbucket</u> 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、<u>CodeStarS</u> <u>ourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、Gi</u> <u>tLab.com、および GitLab セルフマネージドアクションの場合)</u>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 のリポジトリをソースとして使用するように CodePipeline を設定 GitHub Enterpris 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。

コンソールの GitHub (GitHub App 経由) プロバイダーオプションまたは CLI の CodestarSourceConnection アクションを使用します。「<u>GitHub コ</u> <u>ネクション</u>」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、<u>CodeStarS</u> <u>ourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、Gi</u> <u>tLab.com、および GitLab セルフマネージドアクションの場合)</u>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 App 経由) アクションは、GitHub のバージョン 2 のソースアクションです。GitHub (OAuth アプリ経由) アクションは、OA uth 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。G itHub (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) アクションを使用する既存のパイプラインは引き続 き機能し、影響はありません。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 (Bitbucke t 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。Gi tHub (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、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 ステップを参照してください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。
- GitHub Enterpris 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 e Server 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、<u>CodeStarS</u> <u>ourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、Gi</u> <u>tLab.com、および GitLab セルフマネージドアクションの場合)</u>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 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 CodestarS ourceConnection アクションとGitLab プロバイダーを使用します。 「<u>GitLab.com への接続</u>」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、<u>CodeStarS</u> ourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、Gi <u>tLab.com、および GitLab セルフマネージドアクションの場合)</u> を参照して ください

GitLab セルフマ ネージド

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。

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

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、<u>CodeStarS</u> <u>ourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、Gi</u> <u>tLab.com、および GitLab セルフマネージドアクションの場合)</u>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、「<u>GitLab セ</u> <u>ルフマネージドの接続</u>」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 Apps 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 <u>CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise</u> <u>Server、GitLab.com、および GitLab セルフマネージドアクションの場合)</u> 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、 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ステップを参照してくださ い<u>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</u> 更新する。

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) アクションを使用する既存のパイプラインは引き続き機能し、影響はありません。

詳細はこちら:

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

## 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

- <u>CodeBuild ビルドアクション</u>
- 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
- <u>TeamCity ビルドアクション</u>

CodeBuild ビルドアクション

<u>CodeBuild</u> 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。詳細情報 は、<u>AWS CodeBuild ビルドおよびテストアクションリファレンス</u>の CodePipeline アクション設定 リファレンスを参照してください。

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、<u>AWS CodeBuild ビルドおよびテス</u> 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、<u>CloudBees</u> 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

<u>Amazon ECR</u> 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、 の CodePipeline Action 設定リファレンスを参照してくださ いECRBuildAndPublish ビルドアクションリファレンス。

Jenkins ビルドアクション

CodePipeline を設定し、<u>Jenkins CI</u>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、<u>TeamCity</u>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

- <u>CodeBuild またはテストアクション</u>
- 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
- <u>テストオートメーションを反映</u>

### CodeBuild またはテストアクション

<u>CodeBuild</u> 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。詳細情報は、<u>AWS CodeBuild</u> <u>ビルドおよびテストアクションリファレンス</u>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アプリ ケーションを含む、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、<u>AWS Device Farm テストアクショ</u> 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、<u>Ghost Inspector</u> 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 <u>OpenText LoadRunner Cloud</u> 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

テストオートメーションを反映

Reflect 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。ノーコードテスト自動化により、Refle 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 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
- AWS CloudFormation アクションをデプロイする
- AWS 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
- AWS 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
- <u>Amazon Alexa デプロイアクション</u>
- 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

<u>Amazon EC2</u> では、クラウドでコンピューティングを作成および管理できます。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:

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

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

<u>Amazon EKS</u> では、kubernetes クラスターを作成および管理できます。クラスターにイメージをデ 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:

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

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
- <u>チュートリアル: デプロイプロバイダーとして AWS AppConfig を使用するパイプラインを作成する</u>

AWS CloudFormation アクションをデプロイする

<u>AWS CloudFormation</u>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 Serverless Application Model (AWS SAM) を拡張 AWS 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 AWS 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 AWS 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 AWS CloudFormation で きます。その結果、 AWS 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:

- <u>AWS CloudFormation デプロイアクションリファレンス</u>の CodePipeline アクション設定リファレンス
- <u>CodePipeline を使用した継続的デリバリー</u> 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 AWS 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 AWS 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。

AWS CloudFormation StackSets デプロイアクション

AWS 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 AWS 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:

- <u>AWS CloudFormation StackSets デプロイアクションリファレンス</u>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で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:

- ・ <u>Amazon ECS とは</u>
- <u>チュートリアル</u>: 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

<u>Elastic Beanstalk</u> 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。

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、「<u>CodePipeline との製品とサービス</u> 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

AWS OpsWorks アクションをデプロイする

AWS 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 AWS OpsWorks。

- カスタム Chef クックブック Chef クックブック AWS 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 AWS 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、 AWS 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 AWS OpsWorks アプリケーションを作成できます。

の CodePipeline サポート AWS OpsWorks は現在、米国東部 (バージニア北部) リージョン (useast-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

<u>Service Catalog</u> 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

<u>Amazon Alexa Skills Kit</u> 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、<u>デプロイアクショ</u> <u>ンの統合</u> 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、 <u>XebiaLabs</u> 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 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 呼び出しアクション

Amazon Inspector は、ワークロードを自動的に検出し、ソフトウェアの脆弱性や意図しないネット ワークへの露出を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、Amazon Inspector は Rust や Go バイナリなどのバイナ リをサポートしています。

CodePipeline 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:

の CodePipeline アクション設定リファレンス <u>Amazon Inspector InspectorScan 呼び出しアク</u>
 ションリファレンス

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:

- の CodePipeline アクション設定リファレンス AWS Lambda アクションリファレンスを呼び出す
- 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

<u>Step Functions</u> 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:

- <u>AWS Step Functions アクションリファレンスを呼び出す</u>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	<u>Amazon CloudWatch</u> は AWS リソースをモニタリングします。
	詳細はこちら:
	・ <u>Amazon CloudWatch とは</u>
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。
	<ul> <li>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         <ul> <li>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             。</li> </ul> </li> </ul>
	詳細はこちら:
	What Is Amazon EventBridge?
	・ <u>CodePipeline でパイプラインを編集する</u> .
	<ul> <li><u>CodeCommit ソースアクションと EventBridge</u></li> </ul>
	<ul> <li>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 - EventBrid ge 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。</li> </ul>

	詳細はこちら:
	<ul> <li>CodePipeline イベントのモニタリング</li> </ul>
	<ul> <li>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。</li> </ul>
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、「 <u>AWS Cloud9 ユーザーガイド</u> 」を参照してください。 詳細はこちら:
	・ <u>AWS Cloud9のセットアップ</u>
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:
	・ <u>を使用した CodePipeline API コールのログ記録 AWS CloudTrail</u>
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、「 <u>通知ルールの作成</u> 」を 参照してください。

AWS Key Managemen t 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、CodePipeli ne 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
	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、「 <u>別の AWS アカウントのリソースを使用するパイプラインを</u> <u>CodePipeline で作成する</u> 」を参照してください。
	<ul> <li>AWS 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。</li> </ul>
	<ul> <li>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、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。</li> </ul>
	AWS KMS key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、 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

 <u>AWS CodeCommit、、およびを使用して CI/CD AWS CodeBuildAWS CodeDeployを完了する</u> 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

### • CodePipeline を使用して GitHub から Amazon EC2 にデプロイする方法

CodePipeline をゼロから設定して、開発、テスト、本番稼働の各ブランチを別々のデプロ イメントグループにデプロイする方法を説明します。IAMロール、CodeDeploy エージェン ト、CodeDeploy、および CodePipelineの使用方法と設定方法について説明します。

2020 年 4 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

 <u>CodePipeline、CodeBuild、Amazon ECR、およびを使用したAmazon ECSへの継続的なデプロ</u> イ AWS 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 AWS CloudFormationを使用して Docker コンテナとして配信されます。

 サンプル AWS CloudFormation テンプレートとそれを使用して、GitHub の <u>ECS リファレンス</u> <u>アーキテクチャ:継続的デプロイ</u>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

 <u>AWS CloudFormation と CodeCommit を使用して CodePipeline AWS Elastic Beanstalk で のパイ</u> 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 AWS 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 AWS CloudFormation

AWS 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

・ <u>でクロスアカウントパイプラインを作成する AWS CodePipeline</u>

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。 AWS 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 <u>チュートリアル: 4 ステージのパイプライ</u> 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<u>CodePipeline のパイプラインで</u> <u>AWS Lambda 関数を呼び出す</u>。

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 AWS 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

の手順を完了したら<u>CodePipeline の使用開始</u>、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 を使用して Docker イメージを構築して Amazon ECR にプッシュ する (V2 タイプ)
- 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で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: パイプラインを開始するためのプルリクエストのブランチ名をフィルタリングする (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: を使用して 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
- ・ <u>チュートリアル</u>: 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

- <u>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</u> Serverless Application Repository
- 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
- <u>チュートリアル: デプロイプロバイダーとして AWS AppConfig を使用するパイプラインを作成する</u>
- ・ <u>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</u>
- ・ <u>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</u>
- チュートリアル: AWS CloudFormation StackSets デプロイアクションで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では、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。

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、「」を参照してください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。

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. https://console.aws.amazon.com/iam/ で IAM コンソール 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]を選択します。
- [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] (次へ: アクセス許可) を選択します。
- AWSSystemsManagerDefaultEC2InstanceManagementRoleeployAction 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。
- 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)。

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]を選択します。

### インスタンスを起動するには

- 1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
- 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 Enable 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。

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、作成したロールは選択されません。

- 9. 「概要」の「インスタンス数」に「」と入力します2。
- 10. Launch instance (インスタンスの起動)を選択します。
- [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ときは、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。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アクセス許可を使用して、デフォルトのロールを作成するか、 既存のロールを更新します。

- 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ストアの名前と場所を表示します。アーティファクトバケット Amazon リソースネーム (ARN) を書き留めてコピーします。
- 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. [Update] (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。
- [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。
- 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。
- サービスロールで、既存のサービスロールを使用を選択し、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、「」を参照してください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。
- 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. ソースプロバイダーで、GitHub (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。

[Next (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。

Х

### Instance type

Choose the instance type that you want to deploy to. Supported types are Amazon EC2 instances or AWS Systems Manager (SSM) managed nodes. You must have already created, tagged, and installed the SSM agent on all instances.

T

EC2

You can add one group of tags for EC2 instances to this deployment group.

One tag group: Any instance identified by the tag group will be deployed to.

Key	
Q Name	X

Q Name

Value

Q my-instances

Matching instances

### 2 unique matched instances.

### Click here for details [2]

### Target directory

Specify the location of the target directory you want to deploy to. Use an absolute path like /home/ec2-user/deploy.

/home/ec2-user/testhelloworld

### PreScript - optional

Path to the executable script file that runs BEFORE the Deploy phase. It should start from the root directory of your uploaded source artifact. Use an absolute path like uploadDir/preScript.sh.

#### PostScript

Path to the executable script file that runs AFTER the Deploy phase. It should start from the root directory of your uploaded source artifact. Use an absolute path like uploadDir/postScript.sh.

test/script.sh

Advanced

#### Max batch - optional

Specify the number or percentage of targets that can deploy in parallel.

targets

O percentage

targets: from 1 to the number of instances you have

2

a. Target ディレクトリに、 など、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/ home/ec2-user/testhelloworld。

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 には、 などのスクリプトのパスとファイル名を入力しますtest/script.sh。
- c. [Next (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。
| Source                                      |              |
|---|--------------|
| GitHub (via GitHub App)                     |              |
| Succeeded - <u>42 minutes ago</u>           |              |
| <u>f8c490e7</u>                             |              |
| View details                                |              |
|   |              |
| f8c490e7 Source: Edited script.sh           |              |
| DeploytoEC2 Succeeded                       | Start rollba |
| Pipeline execution ID: <u>e4e931ec-56cb</u> |              |
|   |              |
| EC2Deploy                                   |              |
| Amazon EC2                                  |              |
|   |              |
| Succeeded - <u>38 minutes ago</u>           |              |
| View details                                |              |
| View details                                |              |

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の出力を表示します。

Logs Summary Input Output
Showing the last 38 lines of the build log. <u>View entire log</u>
A Show previous logs
<pre>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;</pre>
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
7 ava ažani got object – bucket codeninaline us cost 1 2028
rbtest/SourceArti/rwfBtWb /tmp/codepipeline/530
8 unzip -o /tmp/codepipeline/530398 /tmp/codepipeline/5303
9 rm /tmp/codepipeline/5303985c-fc99-4
<pre>10 [2025/02/13 00:15:38.340] Deploy event DOWNLOAD succeeded on instances: 1-0145 11 [2025/02/13 00:15:38.397] Processing deploy event BEFORE_DEPLOY on instances: i- 0145;</pre>
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/testhelloworld15 cp -r /tmp/codepipeline/530:;/* /home/ec2-user/testhelloworld16 rm -rf /tmp/codepipeline/536/*17 [2025/02/13 00:16:13.616] Deploy event DEPLOY succeeded on instances: i-01457.
18 [2025/02/13 00:16:13.673] Processing deploy event AFTER_DEPLOY on instances: i-

Log	s Summary Input Output
29	aws spapi get-objectbucket couepipeline-us-east-1-29900999000key
30	unzip -o /tmp/codepipeline/530
	/tmp/codepipeline/5303
31	rm /tmp/codepipeline/53
32	[2025/02/13 00:17:17.891] Deploy event DOWNLOAD succeeded on instances: i-05866
33	[2025/02/13 00:17:17.942] Processing deploy event BEFORE_DEPLOY on instances: i-
24	05866
54	[2025/02/15 00:1/:1/.955] Skipping deploy event BEFORE_DEPLOY on instances: 1-
35	[2025/02/13 00:17:17.974] Processing deploy event DEPLOY on instances: i-05866
36	[2025/02/13 00:17:18.062] Executing commands on instances i-05866
	d7abd80c- , commands: mkdir -p /home/ec2-user/testhelloworld
37	cp -r /tmp/codepipeline/55055050 7050 7050 7050 7060 7060 7060 7
38	rm -rf /tmp/codepipeline/
39	[2025/02/13 00:17:18.129] Total instances 2, pending instances 0, in progress instances 1.
40	[2025/02/13 00:17:49.738] Deploy event DEPLOY succeeded on instances: i-05866
41	[2025/02/13 00:17:49.787] Processing deploy event AFTER_DEPLOY on instances: 1-
12	[2025/02/13 00:17:49 880] Executing commands on instances i-05866
42	06365
	user/testhelloworld/test/script.sh
43	/home/ec2-user/testhelloworld/test/script.sh
44	[2025/02/13 00:17:49.938] Total instances 2, pending instances 0, in progress instances 1.
45	[2025/02/13 00:18:20.868] Deploy event AFTER_DEPLOY succeeded on instances: i-
	05866
46	[2025/02/13 00:18:20.921] Processing deploy event UNBLOCK_TRAFFIC on instances: i-
47	05866
47	[2023/02/13 00.10.20.935] Skipping depioy event UNBLOCK_TRAFFIC on instances: 1-
48	[2025/02/13 00:18:21.075] Describing tag, tag key = Name, tag value = my-instances.
49	[2025/02/13 00:18:21.322] Found 0 more instances:
50	[2025/02/13 00:18:21.415] Deployment SUCCEEDED
51	

# 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。

- 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、「<u>EC2 デプロイアクションがエラーメッセージで</u> 失敗する No such file」を参照してください。

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

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 でビルドアクションを作成するのに役立ちます。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も示しま す。

A 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安全であり、信頼できる AWS ア カウント によって所有されていることを確認します。

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。CodePipeline で Amazon ECS から CodeDeploy へのブルー/グリーンデプロイアクションのソースとして ECR イメージリ ポジトリを持つパイプラインを使用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: 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。

### ▲ 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「<u>リポジトリの作成</u>」と「<u>イメージを</u> 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に作成された 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 <u>クラスターの作成</u>と <u>サービスの</u> 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、 そのイメージを Amazon ECR にプッシュします。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。CodeBuild にその方法を指示するために、ソースコードリポジトリにbuildspec.ymlファイル を追加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
```

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

ファイルを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theAmazon 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。 CodePipeline 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、「」を参照してください<u>Amazon ECS 標準デプロイアクション用の</u> imagedefinitions.json ファイル。

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。などの Dockerfile の名前を使用しhello-world、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。

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

ファイルを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。
- [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。
- [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。
- 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. ソースプロバイダーで、GitHub (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 ECRBuildAndPublish を選択します。

tep 1 Thoose creation option	Add build stage Info Step 4 of 6
tep 2	•
oose pipeline settings	Build - optional
ep 3 Id source stage	Build provider Choose the tool you want to use to run build commands and specify artifacts for your build action.
p 4 d build stage	O Commands     O Other build providers
p 5 d deploy stage	AWS ECRBuildAndPublish
p 6	ECR Repository Name Enter a name for the ECR repository
view	Q actions/image-repo X
	Image Tag - optional Enter an image tag
	Dockerfile Path - optional Enter the path to the Dockerfile
	Region
	US East (N. Virginia)
	Input artifacts Choose an input artifact for this action. Learn more [2]

- a. ECR リポジトリ名で、イメージリポジトリを選択します。
- b. [次へ]を選択します。
- 10. ステップ 5: テストステージを追加し、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ます。次のステップで 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ーで、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、などのソースステージから入力アーティファクトを選択しま すSourceArtifact。
  - 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表示します。

•		
⊘ Build Succeeded	Start rollback	
Pipeline execution ID: <u>186696f4-0925-</u>		
Build		
AWS ECRBuildAndPublish		
Succeeded - 5 minutes and		
View details		
3524bc39 Source: Added imagedefinitions.json		
Disable transition		
$\checkmark$		
Opploy Succeeded	Start rollback	
Pipeline execution ID: <u>186696f4-0925-4</u>		
DeploytoECS		
Amazon ECS 2		
View details		
3524bc39 Source: Added imagedefinitions.ison		

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の出力を表示します。



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が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。

Developer Tools > CodePipeline > Pipelines > ecrbuild-to-ecs > Debug ecrbuild-to-ecs		
ecrbuild-to-ecs		Back to pipeline Release change
√ ⊡ ~ ⊙ Source	<ul> <li>Pipeline execution details</li> </ul>	
Source GitHub (via GitHub App)     DeploytoECS       ✓ O Build     Action execution ID: 68e3ec5a-5f06-4		
Build     AWS ECRBuildAndPublish     S Deploy	Summary Input	
BeploytoECS     Amazon ECS	Status 😢 Failed	Last updated Just now
	Action execution ID	
	Error code Invalid action configuration	
	Error message Did not find the image definition file imagedefinit file is stored in your pipeline's Amazon S3 artifact key: ecrbuild-to-ecs/SourceArti/	tions.json in the input artifacts ZIP file. Verify the bucket: codepipeline-us-east-1-

# 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。
- 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 よってコード変更から作成された ECR に Docker イメージがプッシュされます。

# 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では、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)。
- 既存の CodePipeline サービスロールを使用する必要があります。このロールは、ステップ 3: IAM <u>で CodePipeline サービスロールポリシーを更新する</u>以下を使用してこのアクションのアクセス許 可で更新します。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 クラスターを作成す ることを選択できます。

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 ターを既に作成している場合は、このステップはオプションです。

Amazon EKS でパブリッククラスターを作成する

このステップでは、EKS で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 コンソールを開き、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。[Next (次へ)] を選択します。
- 3. [Create] (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 を作成する

- 1. VPC コンソールを開き、VPC の作成を選択します。
- 2. [VPC 設定] で、[VPC など] を選択します。
- 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。
- 図を表示し、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。

VPC Show details	Subnets (6)	Route tables (6)	
Your AWS virtual network	Subnets within this VPC	Route network traffic to resources	
project-vpc	us-east-1a	<ul> <li>project-rtb-private2-us-east-1b</li> </ul>	
	A project-subnet-public1-us-east-1a	 project-rtb-private4-us-east-1b	
	project-subnet-private3-us-east-1a	project-rtb-private1-us-east-1a	
	project-subnet-private1-us-east-1a	rtb-0a3d97c04	
	us-east-1b	project-rtb-private3-us-east-1a	•••
	B project-subnet-public2-us-east-1b	project-rtb-public	
	B project-subnet-private2-us-east-1b		
	B project-subnet-private4-us-east-1b		

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に NAT ゲートウェイを作成する

- 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。
- 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、ルートテーブルの編集を選択します。
- プライベートサブネットのルートテーブルを更新して、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。
- パブリックサブネットの場合は、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 で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 コンソールを開き、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。[Next (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。[Create] (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た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

- 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。の VPC のプライベートサブネットを決定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 をアタッチしますステップ 1: (オプション) Amazon EKS でクラスターを作成する。
- 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 で 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、 などの既存の CodePipeline サービスロールを**cp-servicerole**、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ステップで更新します。

CodePipeline サービスロールポリシーを更新するには

- 1. https://console.aws.amazon.com/iam/ で IAM コンソール 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. などの CodePipeline サービスロールを検索しますcp-service-role。
- 4. 新しいインラインポリシーを追加します。
- 5. ポリシーエディタで、次のように入力します。
  - 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。

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



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。

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

11 \* 11 ] }, { "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" 1 } } }, { "Effect": "Allow", "Action": "ec2:CreateNetworkInterfacePermission", "Resource": "\*", "Condition": { "ArnEquals": { "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: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"
                    Т
                }
            }
        }
    ],
    "Version": "2012-10-17"
}
```

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 コンソールを開き、クラスターに移動します。
- 2. [リモートアクセス] タブを選択してください。
- 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。
- IAM プリンシパル ARN で、 など、 アクション用に更新したロールを入力しますcp-servicerole。[Next (次へ)] を選択します。
- ステップ 2: アクセスポリシーの追加ページで、ポリシー名で、 などのアクセス用の管理ポリ シーを選択しますAmazonEKSClusterAdminPolicy。[Add policy]を選択します。[Next (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、管理ポリシーではなく最小特権でポリシーのアクセス 許可の範囲を絞り込むには、代わりにカスタムポリシーをアタッチします。

6. レビューページで、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<u>https://kubernetes.io/docs/reference/kubectl/quick-</u> reference/.com または https://wwwhttps://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. 既存の GitHub リポジトリを作成または使用します。
- 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。

3. リポジトリのルートレベルに ファイルを追加します。

# 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。
- [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。
- [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。
- 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で、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。

### Deploy configuration type

Please select deploy configuration type.

0	Helm
	Helm configuration type

### O Kubectl

Kubectl configuration type

Helm release name

Enter the helm release name.

my-release

### Helm chart location

Enter folder location of helm chart.

nginx-chart

### Override for helm values files - optional

Enter comma-separated helm values files in helm chart location.

### Kubernetes namespace - optional

You can provide a name for the Kubernetes namespace to override the default.

### Subnet IDs

Specify the subnet IDs that your compute action will use.

▼

- a. デプロイ設定タイプで、Helm を選択します。
- b. Helm チャートの場所に、などのリリース名を入力しますmy-release。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. [Next (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。

Pipeline execution ID: <u>0483d26d-5000-</u> 2	4
	1 Alexandre
Source	6
<u>GitHub (via GitHub App)</u>	
Succeeded - <u>3 minutes ago</u>	
<u>11412f6c</u>	
View details	
<u>11412+6c</u> 🖾 Source: Opdate neim chart values.yami	
Disable transition	
Disable transition	
Disable transition     Deploy Succeeded	
Disable transition Disable transition ODeploy Succeeded Pipeline execution ID: 0483d26d-5000-	
Disable transition Disable transition ODeploy Succeeded Pipeline execution ID: 0483d26d-5000	
Disable transition Disable transition ODeploy Succeeded Pipeline execution ID: 0483d26d-5000 Deploy	
Disable transition Disable transition Deploy Succeeded Deploy Deploy Amazon FKS [2]	
Disable transition Disable transition Deploy Succeeded Deploy Deploy Amazon EKS	

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、アクションの出力を表示します。

# 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 (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、「<u>コマン</u> ドアクションリファレンス」を参照してください。

### A 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。「<u>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</u> 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

 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 App 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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。
- [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。
- [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。
- 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、「<u>サービスロールのポリ</u> 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. ソースプロバイダーで、GitHub (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。

(i) 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
<pre>echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}</pre>

[Next (次へ)]	を選択します。
-------------	---------

Add source stage	Build - optional			
Step 3				
Add build stage	Build provider Choose the tool you want to use to run build commands and specify artifacts for your build action.			
Step 4 Add deploy stage	O Commands O Other build providers			
Step 5	Commands			
Review	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.			
	Is echo hello world echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}			
	Input artifacts			
	Choose an input artifact for this action. Learn more			
	▼			
	SourceArtifact X Defined by: Source			
	No more than 100 characters			

10. ステップ 5: テストステージを追加し、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。

Previous

Skip build stage

Next

Cancel

[Next (次へ)] を選択します。

11. ステップ 6: デプロイステージを追加し、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。

[Next (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。

FAC OV

### Input artifacts

Choose an input artifact for this action. Learn more [

▼	
SourceArtifact X 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 CodeBuild.	r this action will incur separ
ls	
echo hello	
echo pipeline Execution Id is #{codepipeline.PipelineExecutionId}	
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 of the variables of the var	ation. Learn more [
compute	
Variables	
Specify the names of the variables in your environment that you want to export.	
AWS_DEFAULT_REGION Add variable	
Output artifacts Choose a name for the output of this action. CodePipeline will create the output artifact for your pipeline artifact store.	
Name	
Files	
Add to file	
paths	

## 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 [2	Input	Amazon S3
Output variables		
Кеу		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

	Feetingenet 1			THADED TOURS IN WALL I
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、「<u>構文での glob</u> パターンの使用」を参照してください。

### A 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、「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection</u> (<u>Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマ</u> 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 Management Console 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 Management Consoleにサインインします。
- 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 Management Console のメ インページが表示されます。
- 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、<u>CloudShell でサポートされているリージョ</u> 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。

- 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

 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。

ステップ 1: CloudShell を開いてリポジトリを複製する

- 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 (<u>http://console.aws.amazon.com/codesuite/codepipeline/home</u>) 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。
- [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。
- [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。
- [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、「<u>Add</u> permissions to the the CodePipeline service role」を参照してください。

 [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. ソースプロバイダーで、GitHub (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. Webhook イベントで、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、「」と入力しますrelease\*。

▲ 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
```

- I. [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: テストステージを追加し、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。

 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

 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 origi	git push origin release-1				
Developer Tools CodePipeline Source • CodeCommit Artifacts • CodeArtifact Build • CodeBuild Deploy • CodeDeploy Pipeline • CodePipeline Getting started Pipeline History Settings Settings	Developer Tools > CodePipeline > Pipelines > compipeline I Compipeline A Notify ▼ Edit Stop execution Clone pipeline Release change Source Succeeded Pipeline execution ID: 6544c70c-a557-419d-8729-2e824526c137 Source © GitHub (Version 2) Ed Source © GitHub (Version 2) Ed Source © GitHub (Version 2) Ed Source © GitHub (Version 2) Ed Source Succeeded - Just new 49366bda View in AWS CodeStarSourceConnection Ed				
AWS CloudShell  Cloudshell-user@ip-10-4-40-128 repo]5 13  Cloudshell-user@ip-10-4-40-128 repo]5 14  Cloudshell-user@ip-10-4-40-128 repo]5 14	Disable transition	▲ Actions ▼ C ②			
[cloudshell-user@jp.10-4-40-128 MyGitHubb Forach 'release-branch' velase-branch' [cloudshell-user@jp.10-4-40-128 MyGitHubb matter 'elesterionanth [cloudshell-user@jp.10-4-40-128 MyGitHub Username for 'https://github.com' fotil 0 (dollan 0), reused 0 (dolta 0), pr fotil 0 (dollan 0), reused 0 (dolta 0), pr fotil 0 (dollan 0), reused 0 (dolta 0), pr for fotil 0 (dollan 0), reused 0 (dolta 0), pr for fotil 0 (dollan 0), reused 0 (dolta 0), pr for fotil 0 (dollan 0), reused 0 (dolta 0), pr for fotil 0 (dollan 0), reused 0 (dolta 0), pr for fotil 0 (dollan 0), reused 0 (dolta 0), pr for fotil 0 (dollan 0), reused 0 (dolta 0), pr form fotil fotil 0 (dollan 0), reused 0 (dolta 0), pr	sep15 git checkout release-branch origin/release-branch'. eepo15 git push origin release-1 49005 git push origin release-1 *; choreuned 0 po.git **	ļ			

# 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: パイプラインを開始するためのプルリクエストの ブランチ名をフィルタリングする (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、「<u>プッシュおよびプルリクエスト</u> <u>イベントタイプのフィルターを追加する (CLI)</u>」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、「<u>構文での glob パターンの使用</u>」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 Management Console を設定する と、GitHub 認証情報で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 (<u>http://console.aws.amazon.com/codesuite/codepipeline/home</u>) 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。
- [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。
- [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、「<u>Add</u> permissions to the the CodePipeline service role」を参照してください。

 [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。

[Next (次へ)] を選択します。

- 8. [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. ソースプロバイダーで、GitHub (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。

dit: Triggers			Cancel	Done
For source action: <b>Source</b>				Remove
Filters				
Pull request Events: Created Closed Revised	(غُ			
Include branches: main*		+ Add filter		
2	×			
		+ Add trigger		

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。

[Next (次へ)] を選択します。

- ステップ 4: ビルドステージを追加する、ビルドプロバイダーで を選択しますAWS CodeBuild。[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。 「<u>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</u>」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: テストステージを追加し、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。

[Next (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、もうー 度スキップを選択して警告メッセージを受け入れます。[Next (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

- 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。

Succeeded Succeeded Pipeline execution ID: <u>7e1becf6-fc7d-4</u>	4b5-9157-56e9d7982a93		
Source <u>GitHub (Version 2)</u> [2] Succeeded - <u>Just now</u> <u>030c6b39</u> [2] View details <u>030c6b39</u> [2] Source: Update README.	md for PR		
Disable transition			
Build In progress Pipeline execution ID: <u>7e1becf6-fc7d-4</u>	4b5-9157-56e9d7982a93		

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。

Developer Tools > CodePipeline > Pipelines > new-github > Execution history						
Exec ସ	ution history Info			Rerun Stop exect	tion View details	Release change
	Execution ID	Status	Trigger	Started	Duration	Completed
0	61986255	⊘ Succeeded	PullRequest 5 MERGED From repository/branch: /MyGitHubRepo /feature-branch 2 To repository/branch: /MyGitHubRepo /main 2	Feb 7, 2024 6:26 PM (UTC-8:00)	5 minutes 31 seconds	Feb 7, 2024 6:32 PM (UTC-8:00)
0	b9614702	⊘ Succeeded	PullRequest 5 CREATED From repository/branch: /MyGitHubRepo /feature-branch [2] To repository/branch: /MyGitHubRepo /main [2]	Feb 7, 2024 6:26 PM (UTC-8:00)	4 minutes 7 seconds	Feb 7, 2024 6:30 PM (UTC-8:00)
0	09c14335	⊘ Succeeded	Webhook - connection/40d122c4-23fb- 48bf- a08f-1cd9	Feb 5, 2024 1:19 AM (UTC-8:00)	2 days 16 hours	Feb 7, 2024 5:38 PM (UTC-8:00)

# 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 (<u>http://console.aws.amazon.com/codesuite/codepipeline/home</u>) 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。
- [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。
- [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。
- [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、「<u>Add</u> permissions to the the CodePipeline service role」を参照してください。  [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。

 [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。

[Next (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。

[Next (次へ)] を選択します。

- 10. ステップ 4: ビルドステージを追加するで、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. [Next (次へ)] を選択します。
- 11. ステップ 5: テストステージを追加し、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。

[Next (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、もうー 度スキップを選択して警告メッセージを受け入れます。[Next (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。

パイプラインを作成するときに、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、「<u>CodePipeline にクロスリージョ</u> 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 Management Console。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。

- [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。
- 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 <u>YAML</u> 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「<u>CodeDeploy AppSpec ファイルリ</u> 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 <u>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</u> <u>トールする</u>」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. https://console.aws.amazon.com/iam/ で IAM コンソール 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]を選択します。
- 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。
- 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]を選択します。

インスタンスを起動するには

- 1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
- 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。
- [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 として説明され、「無料利用枠対 象」というラベルが付いており、[クイックスタート] の下にあります。)
- 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. Launch instance (インスタンスの起動)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。
- [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)。
- 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. https://console.aws.amazon.com/iam/ で IAM コンソール 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]を選択します。
- [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。[Next (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. [Next (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。

- 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. [Create application] 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. [Create application] を選択します。

CodeDeploy でデプロイグループを作成するには

- 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。
- [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<u>説明されている信頼とアクセス許可を使用し</u> 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。

A 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。

- 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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。
- [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。
- [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、「<u>AWS CodePipeline のクォータ</u>」を参照して ください。

- 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。
- [ステップ 3: ソースステージを追加する] の [ソースプロバイダー] で、[Amazon S3] を選 択します。[バケット] に、「<u>ステップ 1: アプリケーションの S3 バケットを作成する</u>」 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

[Next step]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。

[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、[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。[Next (次へ)] を選択します。
- 10. ステップ 5: テストステージを追加し、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。

[Next (次へ)] を選択します。

 ステップ 6: デプロイステージを追加し、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は、デフォルトでパイプライン AWS リージョン と同じ になります。[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。
- 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

- 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。
- [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。
- [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、「<u>CodePipeline の概念</u>」を参照 してください。

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

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 ンドを実行して変更を加えてパイプラインを更新します。

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. <u>https://console.aws.amazon.com/codedeploy</u> で、CodeDeploy コンソールを開きます。
- [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。
- [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。
- [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。

- [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] ボックスから 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

aiting: MyFirst	Pipeline		Delete Cancel Save
Edit: Source			Edit stage
Source S3	٩		
		+ Add stage	
Edit: Deploy			Edit stage
Deploy CodeDeploy	٩		

- [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。
- [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。
- 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。[Save] 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、または Unix マシンのターミナルから、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。

 [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詳細に可視化できます。

パイプラインの3番目のステージを作成するには

 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
    }
}
```

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

- [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、[Deploy] 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。

A 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。

 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。
- 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。

Disable transition		×
You are about to disable the transition between "Source" and	l "Producti	on".
Disabling reason Explain why you are disabling the transition.		
Disabling transition while I troubleshoot the failure		
	Cancel	Disable

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。

- 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。
- 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、「<u>デプロイグループの設定を変更する</u>」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

<u>チュートリアル: 4 ステージのパイプラインを作成する</u>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

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

 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、<u>CodeCommit のセットアップ</u>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. CodeCommit コンソールを <u>https://console.aws.amazon.com/codecommit/</u>://https//https//
- 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。
- [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。<u>ステップ 2: CodeCommit リポジトリにサ</u> <u>ンプルコードを追加する</u>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。

「CodeCommit リポジトリを作成」

- 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。
- 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。
- 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。

- 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 <u>YAML</u> 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「<u>CodeDeploy AppSpec ファイルリ</u> <u>ファレンス</u>」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. <u>SampleApp\_Linux.zip</u>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		
	<pre># appspec.yml</pre>		
	<pre># index.html</pre>		
	# LICENSE.txt		
	# scripts		
	<pre># install_dependencies</pre>		
	<pre># start_server</pre>		
	# 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

コードのダウンロード、コミット、プッシュする

API バージョン 2015-07-09 155

 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. https://console.aws.amazon.com/iam/ で IAM コンソール 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]を選択します。
- [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] (次へ: 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。
- 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]を選択します。

インスタンスを起動するには

- 1. Amazon EC2 コンソール (https://console.aws.amazon.com/ec2/) を開きます。
- 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。
- [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。
- 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 を選択し、ソースタイプで My 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. Launch instance (インスタンスの起動)を選択します。
- [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、<u>アプリケーション</u>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. <u>https://console.aws.amazon.com/iam/</u>でIAM コンソール 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]を選択します。
- [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。[Next (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. [Next (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. [Create application] 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. [Create application] を選択します。

CodeDeploy でデプロイグループを作成するには

<u>デプロイグループ</u>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。

CodeDeploy でアプリケーションを作成する

- 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。
- [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。
- 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。
- 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。
- [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。

- [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。
- 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。
- [ステップ 3: ソースステージを追加する]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。

[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、[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。[Next (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: テストステージを追加し、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。

[Next (次へ)] を選択します。

- ステップ 6: デプロイステージを追加し、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。

 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。

) Success Congratulations! The pipeline AnyCor	uccess X ongratulations! The pipeline AnyCompanyPipeline has been created.				
Developer Tools > CodePipeline	> Pipelines > AnyCompanyPip	eline			
AnyCompanyPipeli	ne	Edit View history Release change			
¢ Source		View current revisions			
Source CodeCommit C In progress - Just now	٥				
Disable transition			_		
⊖ Deploy		View current revisions			
Deploy CodeDeploy Didn't Run No executions yet	٥				

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

- 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。
- 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。[成功] が表示 されない場合は、「CodePipeline のトラブルシューティング」を参照してください。

- 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。
- [説明] タブの [パブリック DNS] でアドレス (例: ec2-192-0-2-1.uswest-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、「<u>CodePipeline の概念</u>」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

 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">
    Learn more:
    <a href="https://docs.aws.amazon.com/codepipeline/latest/</p>
userguide/">CodePipeline User Guide</a>
    <a href="https://docs.aws.amazon.com/codecommit/latest/</p>
userguide/">CodeCommit User Guide</a>
    <a href="https://docs.aws.amazon.com/codedeploy/latest/</p>
userquide/">CodeDeploy User Guide</a>
  </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

- 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。
- 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

- CodePipeline リソースをクリーンアップするには、「<u>AWS CodePipelineでパイプラインを削除</u> する」の手順に従います。
- CodeDeploy リソースをクリーンアップするには、<u>チュートリアルのデプロイリソースのクリー</u> ンアップ の手順に従います。
- CodeCommit リポジトリを削除するには、「<u>CodeCommit リポジトリの削除</u>」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、「<u>CodePipeline の概念</u>」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「<u>チュートリアル: 4 ステージのパイプラインを作成する</u>」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、「<u>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)</u>」または「<u>チュートリ</u> <u>アル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)</u>」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、「<u>CodePipeline にクロスリージョンア</u> クションを追加する」を参照してください。

#### A 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、「<u>CodePipeline の使用開始</u>」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、「<u>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</u>」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

- ・ <u>サンプルのコピーまたはクローンを</u> GitHub リポジトリに作成する
- <u>Jenkins 統合に使用する IAM</u> 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

- 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。
- リポジトリから、[Fork] を選択してサンプルリポジトリを Github アカウントのレポジトリに複 製します。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 Management Console し、<u>https://console.aws.amazon.com/iam/</u>:// www.com」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。
- [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。
- [Next: Permissions] (次へ: アクセス許可) 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、「<u>Amazon EC2 の IAM ロール</u>」、 「<u>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</u> 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、

 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、「<u>Amazon EC2 ユーザーガイド</u>」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、「<u>CodePipeline のトラブルシュー</u> <u>ティング</u>」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「<u>Jenkins のインストール</u>」と「<u>Jenkins の開始とアクセス</u>」のほか、「<u>CodePipeline との製</u> 品とサービスの統合」の「<u>details of integration with Jenkins</u>」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。

- 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。
- [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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」で CodePipeline コンソールを開きます。
- 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、「<u>AWS</u> 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。
- [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。[Next (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。
- 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: ソースステージを追加する] 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。
- [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: テストステージを追加し、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。

[Next (次へ)] を選択します。

 ステップ 6: デプロイステージを追加し、で作成した CodeDeploy アプリケーションとデプ ロイグループを再使用します<u>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</u> <u>ト)</u>。[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、パイプラインの作成を選択します。
- 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、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。

## 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) ヘテ 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

- 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。

ステージを追加する

[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。

 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。
- [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。

1 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。

[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。

 [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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> <u>codesuite/codepipeline/home</u>.com」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。
- [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。
- [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。
- 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

 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。
- 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。
- 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 認してください。

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、「<u>Amazon Simple Notification</u> 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。
- [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。

Topic name	PipelineNotificationTopic	0
Display name	Enter topic display name. Required for topics with SMS subscriptions.	0
. ,		

4. [トピックの作成]を選択してください。

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:

- 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] 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 <u>トピックのサブスクライブ</u> 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 (https://console.aws.amazon.com/cloudwatch/) を開きます。
- 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"
    1
  }
}
```

- 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. ルールの作成を選択します。
- 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、<u>ク</u> <u>リーンアップ (Amazon SNSトピックからの退会)</u> および <u>Amazon CloudWatch Events API リファレ</u> 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、<u>CodeBuild</u> 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。

A 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。 パイプラインを作成するときに、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、「<u>CodePipeline にクロスリージョ</u> <u>ンアクションを追加する</u>」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、 <u>Device Farmが提供したサンプルアプ</u> リケーションとテスト定義 を使用できます。

Note

[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。
- 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 <u>buildspec.yml</u> 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
```

 (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 Management Console し、「https://<u>https://console.aws.amazon.com/</u> codepipeline/.com」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。[Next (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、「<u>パイ</u> <u>プラインタイプ</u>」を参照してください。CodePipeline の料金については、<u>料金</u>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。

③ Note	e
2018	3 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、「 <u>CodePipeline サービスロールにアクセス許可を追加する</u> 」を参照
して	ください。
{	<pre>"Effect": "Allow", "Action": [ "devicefarm:ListProjects", "devicefarm:ListDevicePools", "devicefarm:GetRun", "devicefarm:GetUpload", "devicefarm:CreateUpload", "devicefarm:ScheduleRun" ], "Resource": "*"</pre>

- 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で、ソースプロバイダーで GitHub (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。
- I. 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。[Next (次へ)] を選択します。
- 4. ステップ 4: ビルドステージを追加するで、ビルドステージを追加します。
  - a. ビルドプロバイダーで、その他のビルドプロバイダーを選択し、 を選択しますAWS CodeBuild。[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. [Next (次へ)] を選択します。
- ステップ 5: テストステージを追加し、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。

[Next (次へ)] を選択します。

- 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、もうー 度スキップを選択して警告メッセージを受け入れます。[Next (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ー 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、 プロジェクトで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:uswest-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

Device Farm テストを使用するように CodePipeline を設定します。

- APPIUM\_WEB\_JAVA\_JUNIT
- APPIUM\_WEB\_JAVA\_TESTNG
- APPIUM\_WEB\_NODE
- APPIUM\_WEB\_RUBY
- APPIUM\_WEB\_PYTHON
- BUILTIN\_FUZZZ
- INSTRUMENTATION
- XCTEST
- XCTEST\_UI

カスタム環境ノードはサポートされていません。

- I. 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. [Save] 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: を使用して 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。

Note

#### ▲ 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。 パイプラインを作成するときに、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、「<u>CodePipeline にクロスリージョ</u> 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。
- 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 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

- 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。「<u>ステップ 1: ア</u> <u>プリケーションの S3 バケットを作成する</u>」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 Management Console し、「https://<u>https://console.aws.amazon.com/</u> codepipeline/.com」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。[Next (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、「<u>パイ</u> <u>プラインタイプ</u>」を参照してください。CodePipeline の料金については、<u>料金</u>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。

3 Note 2018 合、[ コン] す。言	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 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 ノールを開き、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 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」
を参見	照してください。
{	
	"Effect": "Allow",
	"Action": [
	"devicefarm:ListProjects",
	"devicefarm:ListDevicePools",
	"devicefarm:GetRun",
	"devicefarm:GetUpload",
	"devicefarm:CreateUpload",
	"devicefarm:ScheduleRun"
	],
	"Resource": "*"
L	

- 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: ソースステージを追加する] 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. [Next (次へ)] を選択します。
- ステップ 4: ビルドステージを追加するで、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. [Next (次へ)] を選択します。
- f. ステップ 5: テストステージを追加し、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。

[Next (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。

y-ios-pipeline	
Source Amazon S3 🖸 🔿 In progress - Just now	<b>(i)</b>
Disable transition	
Build Custom Jenkins1 (Version: 1) 🖸	١
Didn't Run No executions yet	

- 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ー 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。

- 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、 プロジェクトで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:uswest-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
- [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. [Save] 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。「<u>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</u>」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

 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. [Next (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. [Next (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。
- 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 prodexample123456 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

- 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 Management Console し、<u>https://console.aws.amazon.com/</u> codepipeline/://www.com」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。[Next (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、「<u>パイ</u> <u>プラインタイプ</u>」を参照してください。CodePipeline の料金については、<u>料金</u>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。
- 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するには、次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. [Next (次へ)] を選択します。
- [ステップ 4: ビルドステージを追加する] 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。
- ステップ 5: テストステージを追加し、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。

[Next (次へ)] を選択します。

- 5. ステップ 6: デプロイステージを追加するで、以下を完了します。
  - 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、[AWS 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. [Next (次へ)] を選択します。
- 6. ステップ 7: パイプライン設定を確認してから、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。

⊘ Source		
Source CodeCommit	١	
Succeeded - 1 minute ago 99e3b988		
Source: Added S3_template.json 99	e3b988	
Disable transition		
⊙ Deploy		
Deploy	٩	
ServiceCatalog		

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

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

 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

 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。



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。

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. [Next (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. [Next (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。
- 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 prodexample123456 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

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 Management Console し、「https://<u>https://console.aws.amazon.com/</u> codepipeline/.com」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. [Next (次へ)]を選択します。
- [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. [Next (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。

1yServiceCatalog	Pipeline
⊗ Source	
Source CodeCommit Succeeded - 1 minute ago 99e3b988	6
Source: Added S3_template.json 99e Disable transition	13b988
♥ ⊙ Deploy	
Deploy ServiceCatalog ☑	3
Source: Added S3_template.json 99e	3b988

 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

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。
- 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 : AWS CloudFormationを使用して Amazon S3 パイプラインを作成します。

# 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。 <u>ル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)</u>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

- 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。
- 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。[Next (次へ)] を選択しま す。
- [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. [Next (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」を選択し、「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",
1	"Action": [
	"events:PutRule",
	"events:PutEvents",
	"events:PutTargets",
	"events:DeleteRule",
	"events:RemoveTargets",
	"events:DescribeRule"
	],
1	"Resource": " <i>resource_ARN</i> "
}	

8. にサインイン AWS Management Console し、「https://<u>https://console.aws.amazon.com/</u> codepipeline/.com」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、 AWS 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 : AWS CloudFormationを使用して Amazon S3 パイプラインを作成します。

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。<u>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)</u>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 (<u>YAML</u> | <u>JSON</u>) 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。[Next (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. [Next (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承認」を選択し、「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 AWS CloudFormationリソースタブで、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、 AWS 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-codepipelinesourcebucket-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 AWS 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。

1 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 Management Console し、「https://<u>https://console.aws.amazon.com/</u> codepipeline/.com」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。
- [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。ブラウズを選択し、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。[Next (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. [Next (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た後、「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": [
        "cloudtrail:CreateTrail",
        "cloudtrail:DeleteTrail",
        "cloudtrail:StartLogging",
        "cloudtrail:StopLogging",
        "cloudtrail:PutEventSelectors"
    ],
    "Resource": "resource_ARN"
}
```

7. にサインイン AWS Management Console し、「https://<u>https://console.aws.amazon.com/</u> <u>codepipeline/</u>.com」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 リポジトリを作成します。
- <u>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</u>る
- ステップ 2: パイプラインを作成する
- ステップ 3: AWS 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
- ・ <u>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</u>

前提条件: AWS CloudFormation サービスロールと CodeCommit リポジト リを作成します。

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<u>チュートリ</u>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 の以下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。

- リージョンのサンプルテンプレートに移動します。たとえば、のテーブルを使用 してリージョン<u>https://docs.aws.amazon.com/documentdb/latest/developerguide/</u> <u>quick\_start\_cfn.html#quick\_start\_cfn-launch\_stack</u>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。
- 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

 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 AWS 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」で CodePipeline コンソールを開きます。
- [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。

- [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。
- [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。
- 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<u>CodePipeline パイプライン構</u> 造リファレンスを参照してください。

[Next (次へ)] を選択します。

- 8. [ステップ 3: ソースステージを追加する] 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、<u>ステップ 1: CodeCommit リポジトリを作成する</u> 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。

[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、[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。

[Next (次へ)] を選択します。

10. ステップ 5: テストステージを追加し、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。

[Next (次へ)] を選択します。

- 11. ステップ 6: デプロイステージを追加する:
  - 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. [Next (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: AWS 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 が変更セット AWS CloudFormation 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。

- [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。
- 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。

- [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。
- 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 AWS 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 AWS 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。

[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。

ステップ 4: 手動承認アクションを追加する

- 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 AWS 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。

#### A 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、<u>チュートリアル: Amazon ECR ソース、ECS</u> <u>- CodeDeploy 間のデプロイでパイプラインを作成する</u> を参照してください。

Note

このチュートリアルは、ソースアクションを持つ CodePipeline の Amazon ECS 標準デ プロイアクションを対象としています。Amazon ECSstandard 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<u>チュートリアル: CodePipeline を使用</u> して Docker イメージを構築して Amazon ECR にプッシュする (V2 タイプ)。

## 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「<u>リポジトリの作成</u>」と「<u>イメージをプッシュする</u>」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「<u>タスク定義の作成</u>」を参 照してください。

#### A 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 <u>クラスターの作成</u>と <u>サービスの作成</u>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_bulld: commands:
- echo Logging in to Amazon ECR
- awsversion
<pre>- aws ecr get-login-passwordregion \$AWS_DEFAULT_REGION   docker login</pre>
username AWSpassword-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
- REPOSITORY_URI=012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world
- 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:

AWS CodePipeline

files: imagedefinitions.json

このチュートリアルで使用する Amazon ECS サービスで、<u>前提条件</u>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。
- REPOSITORY\_URI の値 (012345678910.dkr.ecr.us-west-2.amazonaws.com/helloworld) 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。[Next (次へ)] を選択します。
- ステップ 2: パイプライン設定を選択し、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、「<u>パイプラインのタイプ</u>」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。
- 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. [Next (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. ステップ 5: デプロイステージの追加ページで、デプロイプロバイダーで 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 AWS CodePipeline 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、<u>チュートリアル: CodePipeline を使用し</u> 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、<u>チュートリアル:Amazon ECS サービスのデプロイ</u>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、「<u>Amazon ECS Blue/Green デプロイアクション用の</u> 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
- <u>ステップ 3: Application Load Balancer とターゲットグループを作成する</u>
- <u>ステップ 4: Amazon ECS クラスターとサービスを作成する</u>
- <u>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</u>

- 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<u>チュートリ</u>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

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

 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

 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 <u>Amazon ECS タスク実行 IAM ロール</u> 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

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"

- AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"

- AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"
```

例を含む AppSpec ファイルの詳細については、<u>CodeDeploy AppSpec ファイルのリファレンス</u> 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

 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、「<u>Network</u> 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 Management Console し、<u>https://console.aws.amazon.com/vpc/</u>:// www.com」で Amazon VPC コンソールを開きます。
- 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。
- [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。

- 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。
- 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 Management Console し、「https://<u>https://console.aws.amazon.com/</u> ec2/.com」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

- 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。

ステップ 3: Application Load Balancer とターゲットグループを作成する

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

- 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。
- 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。
- [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) を選択します。

- 3. [クラスターを作成]を選択してください。
- 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

を使用して AWS CLI、Amazon ECS でサービスを作成します。

 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,
```



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://createservice.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. [Create application] を選択します。

次に、CodeDeploy デプロイ グループを作成します。

- 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。
- 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. https://console.aws.amazon.com/iam/ で IAM コンソール 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]を選択します。
  - 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] (次へ: アクセス許可) 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. ロードバランサー 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。
- [Production listener port] 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。
- [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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。
- [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。
- [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、「<u>パイプライ</u> 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。
- [ステップ 3: ソースステージを追加する] 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。

[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、[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。[Next (次へ)] を選択します。
- 10. ステップ 5: テストステージを追加し、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。

[Next (次へ)] を選択します。

- 11. ステップ 6: デプロイステージを追加する:
  - 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.yam1**] 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. [Next (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。
- 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。

Luit action	
Action name Choose a name for your action	
Image	
No more than 100 characters	
Action provider	
Amazon ECR	•
Amazon ECR Repository name Choose an Amazon ECR repository as the s	ource location.
nginx 🔻	Create repository
Image tag - optional Choose the image tag that triggers your pi	peline when a change occurs in the image repository.
If an image tag is not selected, defaults to	latest
Output artifacts Choose a name for the output of this actio	n.
Output artifacts Choose a name for the output of this action MyImage	n. Remove

- 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

- 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.yam1**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。[Save] を選択します。
- 6. AWS CodePipeline ペインで、パイプラインの変更を保存を選択し、変更を保存を選択します。 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。

#### A 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、<u>デプロイアクショ</u> 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、<u>「カスタムスキルを AWS Lambda 関数とし</u> <u>てホストする</u>」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<u>チュートリ</u>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。「<u>カスタムスキルサンプルコードを取得する</u>」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。「<u>ASK CLI の</u> 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、「<u>スキルパッケージ形式</u>」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ロケールとして 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

 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
    len-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"

 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

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
```

- 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。
- 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」で CodePipeline コンソールを開きます。
- 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。
- [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。

ステップ 4: パイプラインを作成する

- 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。
- 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。
- [ステップ 3: ソースステージを追加する] 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。

[Next (次へ)] を選択します。

10. [ステップ 4: ビルドステージを追加する] 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。

[Next (次へ)] を選択します。

ステップ 5: テストステージを追加し、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。

[Next (次へ)] を選択します。

- 12. ステップ 6: デプロイステージを追加する:
  - 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。

(i) Yo Pi Ch	ou cannot skip this stage pelines must have at least two stages. Your second stage must be either a build or deployment stage. noose a provider for either the build stage or deployment stage.
Deploy	,
Deploy p	rovider
Alexa S	kills Kit 🗸
Alexa Sl	kills Kit
Alexa ski The uniqu	LI ID e identifier of the skill.
amzn1.	ask.skill.da55cd70-9eaf-4cf1-b326-i
Client ID The client page.	ID of the application registered for LWA (Login With Amazon). Look for this on the Alexa Skills Kit developer portal LWA
amzn1.	application-oa2-client.e
Client see The client LWA page	cret secret of the application registered for LWA (Login With Amazon). Look for this on the Alexa Skills Kit developer portal
Refresh t	oken h token used to request new access tokens.
1	
	Cancel Previous Next

f. [Next (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 <u>Alexa Skills Kit 開発者コンソール</u>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。

A 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、「<u>CodePipeline にクロスリージョンア</u> <u>クションを追加する</u>」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<u>チュートリ</u>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 <u>サンプル静的ウェブサイト</u>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。「<u>静的ウェブサイトを Amazon S3 でホ</u> <u>スティングする</u>」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

- 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。
- 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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。
- [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。
- [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、「<u>パイプラインのタイプ</u>」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。
- [ステップ 3: ソースステージを追加する]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。

[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、[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。

[Next (次へ)] を選択します。

10. ステップ 5: テストステージを追加し、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。

[Next (次へ)] を選択します。

- 11. ステップ 6: デプロイステージを追加する:
  - 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。

Deploy - optional				
Deploy provider				
Choose how you deploy to instances. Choose t	he provider, and the	n provide the config	uration details for that	provider.
Amazon S3			•	
Amazon S3				
Specify your Amazon S3 location.				
Bucket				
my-codepipeline-website-bucket			•	
Deployment path - optional				
Extract file before deploy The deployed artifact will be unzipped bei	fore deployment.			
Additional configuration				
	<b>c</b> 1			
mon man man	Cancel	Previous	Skip deploy	stage Mext

- d. (オプション) [既定 ACL] で、<u>既定 ACL</u> 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. [Next (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。
- 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。「<u>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)</u>」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。「<u>静的ウェブサイトを Amazon S3 でホスティングする</u>」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。

- 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。
- [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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。
- [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。
- [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: ソースステージを追加する] 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。

[Next (次へ)] を選択します。

- 8. [ステップ 4: ビルドステージを追加する] 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。
  - [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. [Next (次へ)] を選択します。
- 9. [ステップ 5: デプロイステージを追加する] 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。

dd deploy stage	5
Deploy - optional	- C
Deploy provider Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.	
Amazon S3	~
Amazon S3	5
Specify your Amazon S3 location.	3
Bucket	
my-codepipeline-website-bucket	3
S3 object key	
js-application/{datetime}.zip	5
Extract file before deploy     The deployed artifact will be unzipped before deployment.	
► Additional configuration	7

- d. (オプション) [既定 ACL] で、<mark>既定 ACL</mark> 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. [Next (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。
- 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。

Amazon S3 > my-codepipeline-website-bucket > js-applicat	lion
<b>Q</b> Type a prefix and press Enter to search. Press ESC to clear.	
▲ Upload	}
	<u> </u>
Name -	Last mog
2019-01-10_07-28-07.zip	Jan 9, 25, GMT-08
2019-01-10_07-39-57.zip	Jan 9, 20

# 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、 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 に公開するには 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。

#### A 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、「<u>CodePipeline にクロスリージョンア</u> 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。

・ <u>AWS Serverless Application Model (AWS SAM)</u> と <u>AWS Serverless Application Repository</u> をよく 理解していること。 CLI AWS Serverless Application Repository を使用して に公開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 デベロッパーガイド<u>」の「クイックスタート: アプリケーションの公開</u>」を参照してください。独自のアプリケーションを に公開するには AWS Serverless Application Repository、 AWS Serverless Application Repository、 Model デベロッパーガイド<u>の AWS SAM</u> 「CLI を使用したアプリケーションの公開」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する AWS リージョン で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 Management Console し、<u>https://console.aws.amazon.com/codepipeline/</u>:// www.com」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。[Next (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。
- [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: ソースステージを追加する] 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。[Next (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. [Next (次へ)] を選択します。

15. ステップ 5: テストステージを追加し、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。

[Next (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、もうー 度スキップを選択して警告メッセージを受け入れます。[Next (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-codepipelineserverlessrepo-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]を選択します。
- AWS CloudFormation スタックの表示を選択して AWS 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。
- 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。
- 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。

 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: 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、「<u>変数リファレ</u> 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。

- <u>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)</u>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 Management Console し、 AWS Lambda コンソールを <u>https://</u> console.aws.amazon.com/lambda/://www.com で開きます。
- [Create function (関数の作成)] を選択します。[一から作成] が選択された状態のままにしておき ます。
- [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. [Create function (関数の作成)]を選択します。
- 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。このチュートリアルで後ほどパイプライ ンのアクションを設定しますが、ここでは変数を渡したものと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。
- 最後のステップとして、<u>https://console.aws.amazon.com/iam/</u>://www.com 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。<u>AWSCodePipelineCustomActionAccess</u>。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、「<u>ステップ 2 : Lambda 関数を作成す</u> <u>る</u>」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. [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}

 [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。
- 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。<u>チュートリア</u> <u>ル: シンプルなパイプラインを作成する (S3 バケット)</u>または <u>チュートリアル: シンプルなパイプラ</u> 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、ロールを検索してから、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、「<u>CodePipeline サービスロール</u> 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 <u>ステートマシンの作成</u> 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

- [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。
- 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。
- [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. [完了] をクリックします。
- 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。
- 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:myprefix-0d9a0900-3609-4ebc-925e-83d9618fcca1

# チュートリアル: デプロイプロバイダーとして AWS AppConfig 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、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。

A 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 Management Console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。

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。

 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
```

 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。
- [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。
- [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。
- 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: ソースステージを追加する] 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。

[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、[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。

[Next (次へ)] を選択します。

10. ステップ 5: テストステージを追加し、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。

ステップ 3: パイプラインを作成する

[Next (次へ)] を選択します。

- 11. ステップ 6: デプロイステージを追加する:
  - 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. [Next (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 <u>ユーザーアクセストークン</u>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。CodeConnections は、ユーザーアクセストークンを接続に使用し ます。

#### A 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、CodeCommit リポジトリにアクセスし、ソースデータに完全クローンオ 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、「<u>CodePipeline との製品とサービス</u> <u>の統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise</u> <u>Server、GitLab.com、および GitLab セルフマネージドアクションの場合</u>」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 Management Console を設定する と、GitHub 認証情報で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 名前を付けます。README.md ファイルを作成し、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。

- 1. CodePipeline コンソール (<u>http://console.aws.amazon.com/codesuite/codepipeline/home</u>) 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。
- [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。
- [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。
- 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、「<u>Add</u> permissions to the the CodePipeline service role」を参照してください。

 [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。

[Next (次へ)] を選択します。

8. [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. ソースプロバイダーで、GitHub (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 アプリをすで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

<u>ユーザーアクセストークン</u>を作成する場合は、 AWS Connector for GitHub アプリ が既にインストール済みであることを確認し、[アプリインストール] フィールドを 空のままにします。CodeConnections は、ユーザーアクセストークンを接続に使用 します。詳細については、「<u>CodeBuild でソースプロバイダーにアクセスする</u>」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。

[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、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へ 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:</pre>
    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. [Next (次へ)] を選択します。
- 10. ステップ 5: テストステージを追加し、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。

[Next (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。[Next (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 <u>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</u> 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利用可能な 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

- <u>前提条件</u>
- 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 README.md に名前を付け、ファイルを作成し、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 (<u>http://console.aws.amazon.com/codesuite/codepipeline/home</u>) 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。
- [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。
- [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。
- 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。[Next (次へ)] を選択します。
- 8. [ステップ 3: ソースステージを追加する] 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。

[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、次の操作を行います。
  - 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
```

- I. [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. [Next (次へ)] を選択します。
- 10. ステップ 5: テストステージを追加し、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。

[Next (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。[Next (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、<u>CodeBuild GitClone のアクセス権限を CodeCommit ソースアク</u> 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 ションで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、スタックセットの作成とス タックインスタンスの作成の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 ロールの 2 AWS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、 AWS 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
- <u>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</u>
- 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

 「<u>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</u>」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。「<u>ス</u> <u>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</u>」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 Management Console し、Amazon S3 コンソールを <u>https://</u> console.aws.amazon.com/s3/://www.com で開きます。
- 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

- 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 <u>https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/</u> 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

 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

 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
```

 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。
- AWS 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 AWS 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。
- [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。
- [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。
- 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、<u>入力および出力アーティファクト</u>および<u>CodePipeline パイプライン構</u>

<u>造リファレンス</u>を参照してください。

[Next (次へ)] を選択します。

- 8. [ステップ 3: ソースステージを追加する] ページの [ソースプロバイダー] で、[Amazon S3] を選 択します。
- [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. [Next (次へ)] を選択します。
- 11. [ステップ 4: ビルドステージを追加する] 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。

[Next (次へ)] を選択します。

12. ステップ 5: テストステージを追加し、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。

[Next (次へ)] を選択します。

- 13. ステップ 6: デプロイステージを追加する:
  - 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 するためのリージョンを 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. [Next (次へ)] を選択します。

ステップ 7: 確認で、パイプラインの作成を選択します。パイプラインが表示されます。
 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。
- [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。
- パイプラインの CloudFormationStackSet AWS 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、およびイベントが AWS 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。
- [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、 AWS 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. 手動で変更を解除します。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。

- パイプライン内の AWS 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、およびイベ ントが AWS 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。
- 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。
- 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. AWS 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。
- [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、「<u>CodePipeline にクロスリージョンア</u> 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 Management Console を設定する と、GitHub 認証情報で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

(i) 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。

- 1. CodePipeline コンソール (<u>http://console.aws.amazon.com/codesuite/codepipeline/home</u>) 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。
- [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。
- [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。
- 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、「<u>Add</u> permissions to the the CodePipeline service role」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. ソースプロバイダーで、GitHub (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。

[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、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。

#### 1 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. [Next (次へ)] を選択します。
- 10. ステップ 5: テストステージを追加し、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。

[Next (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、もうー 度スキップを選択して警告メッセージを受け入れます。[Next (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。

 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。

Edit rule

varrule		
No more than 100 characters		
Rule provider		
AWS VariableCheck		٧
<b>/ariable</b> The variable with the resolved value that will be ch	necked against the provided string value.	
#{SourceVariables.FullRepositoryName}		
/ariables must use the following syntax: #{namesp	pace.variable_key}.	
/MyGitHubRepo Operator Operator for the comparison.		
<ul> <li>Equals Checks whether the variable is equal to the string value.</li> </ul>	<ul> <li>Not equals</li> <li>Checks whether the variable is not equal to the string value.</li> </ul>	<ul> <li>Contains</li> <li>Checks whether the variable contains the string value as a substring.</li> </ul>
Matches     Checks whether the variable matches a		

Choose a name for your rule		
varcheckmsg		
No more than 100 characters		
Rule provider		
AWS VariableCheck		
Variable		
The variable with the resolved value	ue that will be checked against the prov	ided string value.
#{SourceVariables.CommitN	/essage}	
Variables must use the following s	yntax: #{namespace.variable_key}.	
Value		
Value The string value to check against t	he resolved variable value.	
Value The string value to check against t Update	he resolved variable value.	
Value The string value to check against t Update	he resolved variable value.	
Value The string value to check against t Update Operator	the resolved variable value.	
Value The string value to check against t Update Operator Operator for the comparison.	the resolved variable value.	
Value The string value to check against t Update Operator Operator for the comparison. Checks whether the	he resolved variable value.	• Contains
Value The string value to check against t Update Operator Operator for the comparison. Equals Checks whether the variable is equal to the	the resolved variable value.	Contains Checks whether the variable contains the
Value The string value to check against t Update Operator Operator for the comparison. Equals Checks whether the variable is equal to the string value.	the resolved variable value.           Not equals           Checks whether the variable is not equal to the string value.	• Contains Checks whether the variable contains the string value as a substring.
Value The string value to check against t Update Operator Operator for the comparison. Equals Checks whether the variable is equal to the string value.	the resolved variable value.           Not equals           Checks whether the variable is not equal to the string value.	• Contains Checks whether the variable contains the string value as a substring.
Value The string value to check against t Update Operator Operator for the comparison. Equals Checks whether the variable is equal to the string value.	the resolved variable value.           Not equals           Checks whether the variable is not equal to the string value.	Contains Checks whether the variable contains the string value as a substring.
Value The string value to check against t Update Operator Operator for the comparison. Equals Checks whether the variable is equal to the string value. Matches Checks whether the	the resolved variable value.           Not equals           Checks whether the variable is not equal to the string value.	• Contains Checks whether the variable contains the string value as a substring.
Value The string value to check against t Update Operator Operator for the comparison. Equals Checks whether the variable is equal to the string value. Matches Checks whether the variable matches a given	the resolved variable value.           Not equals           Checks whether the variable is not equal to the string value.	• Contains Checks whether the variable contains the string value as a substring.

3. [Save]を選択します。

[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。

Source Succeeded
Pipeline execution ID: <u>1438349d-9809-4249-9621-a</u>
Source
GitHub (Version 2)
Succeeded - <u>21 minutes ago</u>
77cc2e44 🖸
View details
77cc2e44 🗹 Source: Merge pull request #5 from //feature-branch 🚥
Dicable transition
Disable transition
¥
Entry condition: 📀 Succeeded Execution ID: 1438349d
Pipeline execution ID: <u>1438349d-9809-4249-9621-</u>
Puild
AWS CodeBuild
Succeeded - 18 minutes and
View details

2. [タイムライン] タブで変数情報を表示します。

Visua	lization	Timeline	Variables Revision	ns				
Actions						View execution details	ŝ	
	Action name	Stage name	Status	Action provider	Started	Comple	eted Duration	
0	Source	Source	⊘ Succeeded	GitHub (Version 2) 🖸	4 minutes ago	4 minute ago	tes 4 seconds	
0	Build	Build	⊘ Succeeded	AWS CodeBuild	4 minutes ago	Just nov	3 minutes 31 w 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、<u>CodePipeline パイプライン構造リファレンス</u> を参照してく ださい。

トピック

- ・ <u>Amazon S3 で CodePipeline を使用する AWS CodeCommit、および Amazon S3 AWS</u> 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
- <u>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</u> に配信する
- <u>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</u>バリーする
- <u>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</u>
   <u>の AWS Lambda 継続的な配信を行う</u>
- <u>AWS 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</u>

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<u>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)</u>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。<u>チュートリアル: 4 ステージのパイプラインを作成する</u>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、ビルド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、「<u>CodePipeline と CodeBuild を</u> 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、「<u>チュートリアル: CodePipeline を使用した継続的なデプロイ」</u>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

<u>「サーバーレスアプリケーションのデプロイ</u>」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。

# AWS 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 AWS 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、<u>CodePipeline を使用した継続的デリバリー</u>」を参照してください。 AWS 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 <u>Amazon Virtual Private Cloud (Amazon VPC)</u> エンドポイントを サポートするようになりました<u>AWS PrivateLink</u>。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 ネットワーク内の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、<u>VPC ユーザー</u> ガイド参照してください。

## 可用性

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「<u>インターフェイスエンドポイントの作成</u>」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。
- 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. <u>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</u> 確認します。
- 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<u>CodePipeline の使用開始</u>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、<u>CodePipeline の概念</u>「」、<u>CodePipeline チュートリアル</u>「」、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<u>パイプライン、ステージ、アクションを作成する</u>。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パイプラインを CodePipeline で作成する
- 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、「<u>CodePipeline にクロスリージョンアクションを追加する</u>」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、<u>イメージ定義ファイルのリファレンス</u> 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、「<u>CodeCommit ソースアクションと EventBridge</u>」 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。[Next (次へ)] 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。

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、「<u>パイプラインのタイプ</u>」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、「<u>CodePipeline サービス</u> 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、「<u>変数リファレンス</u>」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、「<u>チュー</u> 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、パイプライン用に AWS リージョン 選択した のパイプライ ンに対して、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、<u>入力および出力アーティファクト</u>および<u>CodePipeline パイプライン構</u> 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. [Next (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

- [ステップ 3: ソースステージを追加する] 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、「<u>コードプッシュまたはプルリクエス</u> トイベントタイプで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、<u>CodePipeline のトラブルシューティング</u>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、<u>チュートリアル:</u> <u>CodeCommit パイプラインソースで完全なクローンを使用する</u>を参照してください。

・ Amazon S3 については:

[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。

- 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。[Next (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 <u>CodeBuild GitClone のアクセス権限を CodeCommit ソースアクションに追</u> 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 <u>CodePipeline サービスロールにアクセ</u> <u>ス許可を追加する</u>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、<u>チュートリアル: CodeCommit パイプラインソース</u> <u>で完全なクローンを使用する</u>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、<u>チュートリアル:</u> <u>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</u>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、「<u>Amazon ECS Blue/Green</u> <u>デプロイアクション用の imageDetail.json ファイル</u>」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、<u>Customizing and Configuring Elastic Beanstalk</u> <u>Environments</u> と <u>Supported Platforms</u> 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、<u>Uploading Your Application Revision</u> と <u>Prepare a</u> <u>Revision</u> を参照してください。

- 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. [Next (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、次のいずれかを実行し、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、「<u>チュートリアル: 4 ステージのパイプラインを作成する</u>」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 <u>Create a Pipeline That Uses</u> CodeBuild の指示に従います。

ビルド仕様では、CodeBuild buildspec ファイルはオプションであり、代わりにコマンドを入 力できます。ビルドコマンドを挿入 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、「」を参照してください<u>コマンドアク</u> <u>ションリファレンス</u>。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

環境変数を使用して機密情報、特に AWS 認証情報を保存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。

- 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. [Next (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場合は、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、次のいずれかを実行し、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. [Next (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、次のいずれかを実行し、次へを選択します。
  - 前のステップでビルドステージまたはテストステージを作成した場合は、デプロイステージ をスキップを選択します。

Note

このオプションは、ビルドステージまたはテストステージを既にスキップしている場合は表示されません。

[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でのみ、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。[Next (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。[Next (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、<u>CodePipeline の使用 AWS OpsWorks Stacks</u>」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、変更セットの作成または置換を選択し、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 のパイプラインに AWS CloudFormation 機能を統合する方法の詳細について は、AWS CloudFormation 「ユーザーガイド」の<u>CodePipeline を使用した継続的デリバ</u> 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れを指定する 必要があります。詳細については、「<u>Amazon ECS 標準デプロイアクション用の</u> 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、<u>Tutorial: Continuous Deployment with CodePipeline</u> 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、「<u>Amazon ECS Blue/Green デプロイアクション用の imageDetail.json</u> <u>ファイル</u>」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、<u>チュートリアル: Amazon ECR ソー</u> <u>ス、ECS - CodeDeploy 間のデプロイでパイプラインを作成する</u> 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、「<u>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</u> 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。[Next (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、「<u>チュートリアル: Amazon Alexa Skill をデプロイするパイプ</u> <u>ラインを作成する</u>」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。選択しなかった場合は、[S3 object key (S3 オブジェクトキー)] に値を入力する必 要があります。

(i) 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、<u>canned ACL</u> 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 <u>Cache-Control</u> 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"

[Next (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、<u>チュートリアル: Amazon S3 をデプロイプロバイダとして使</u> 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、「<u>ステージ障害時の自動再試行を設定する</u>」を参照してくだ さい。
- 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. [Next step]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、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、「<u>CodePipeline でパイプラインと詳細を表示する</u>」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、「<u>CodePipeline でパイプラインを編集す</u> <u>る</u>」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、パイプライン構造を定義する JSON ファイルを 作成し、 --cli-input-jsonパラメータを使用して 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、<u>CodePipeline パイプライン構造リファレンス</u>および CodePipeline <u>API Reference</u> の <u>create-pipeline</u> 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、createpipeline 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:

<u>CodePipeline の使用開始</u> で CodePipeline 用に作成したサービスロールの ARN が必要です。createpipeline 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、「<u>CodePipeline サービスロールを</u> <u>作成する</u>」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。

- 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。

```
{
            "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、<u>リソースのタグ付け</u> 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。

- 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、<u>CodePipeline にク</u> 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

 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。

- 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、「<u>CodePipeline でパ</u> 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、<u>CodeCommit ソースに対する</u> <u>EventBridge ルールを作成する (CLI)</u> 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<u>EventBridge と を使用</u> 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 のソースアクションを指定する か、CodeConnections でサポートされているサードパーティリポジトリ (GitHub など)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 AWS 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://<u>https://github.com/aws/codepipeline-starter-templates</u>.comiter でリ 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 AWS 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。
- AWS 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: 作成オプションを選択する

- 1. にサインイン AWS Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。[Next (次へ)] 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。

静的テンプレートからパイプラインを作成する

- 1. [ステップ 2: テンプレートを選択する] 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。

Choose creation option	Step 2 of 4		
Choose template	Category		
Step 3 Choose source	O Deployment	Continuous Integration	O Automation
Configure template	Template		
	O Push to ECR Build and push a new container image to Amazon ECR	O Deploy to ECS Fargate Deploy a ECR image to Amazon ECS Fargate cluster	<ul> <li>Deploy to CloudFormatic Deploy your cloud formation template</li> </ul>

	Category		
Step 3 Choose source Step 4	O Deployment	• Continuous Integration	O Automation
Configure template	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	Category		
Choose source	O Deployment	O Continuous Integration	• Automation
Configure template	Template		
	<ul> <li>Schedule a python build pipeline         Build a python project         periodically         </li> </ul>	<ul> <li>Schedule a gradle build pipeline Build a gradle project periodically</li> </ul>	<ul> <li>Schedule a nodejs build pipeline Build a nodejs project periodically</li> </ul>

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、「<u>コードプッシュまたはプルリクエス</u> トイベントタイプで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、<u>CodePipeline のトラブルシューティング</u>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、<u>チュートリ</u> <u>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</u>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、「<u>Amazon ECS Blue/Green</u> 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、<u>Customizing and Configuring Elastic Beanstalk</u> <u>Environments</u> と <u>Supported Platforms</u> 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、<u>Uploading Your Application Revision</u> と <u>Prepare a</u> 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。

Choose template	Template Details
Step 3	
Choose source	ConnectionArn The CodeConnections ARN for your Docker container source repository.
Step 4 Configure template	arn:aws:codeconnections:us-east-1:
	FullRepositoryId The full repository ID to use with your CodeConnections connection.
	'/MyGitHubRepo
	BranchName The branch name to use with your CodeConnections connection.
	main
	<b>CodePipelineName</b> The CodePipeline pipeline name that will deploy your cfn template.
	DeployToCloudFormationService2
	StackName The CloudFormation Stack Name that you want to create and/or update.
	DeployToCloudFormationService2
	TemplatePath The path in your source repository to the CloudFormation template to create and/or update your Stack
	template.yaml
	OutputFileName The path the output from the CloudFormation stack update will be written to.
	output.json

- 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、 AWS 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、 「<u>CodePipeline でパイプラインと詳細を表示する</u>」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、「<u>CodePipeline でパイプラインを編集する</u>」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、完了する必要があるステージやアクションなど、 AWS CodePipeline 実行するリ リースプロセスについて説明します。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、<u>パイプライ</u> 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。クロス リージョンアクションの詳細については、「<u>CodePipeline にクロスリージョンアクションを追加す</u> 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。を使用 してパイプライン AWS CLI 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。CodeCommit パイプラインの作成または更新の詳細については、「<u>CodeCommit</u> <u>ソースに対する EventBridge ルールを作成する (CLI)</u>」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、「<u>Amazon S3 ソースに対す</u> <u>る EventBridge ルールを作成する (CLI)</u>」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、「<u>パ</u> <u>イプラインを手動で開始する」</u>を参照してください。

#### パイプラインを編集するには

1. にサインイン AWS Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

- [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、「<u>パイプライン</u> 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、「<u>変数リファレンス</u>」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、「<u>チュー</u> 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。

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 などの GitLab セルフマネージドへの接続を作成する手順 を参照してくださいGitHub コネクション。

- 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 ルールと証 AWS CloudTrail 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、証跡は削除されず、データイベントも削除されま 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。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。

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 <u>CodeBuild で CodePipeline を使用して、コードをテスト</u> 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 認証情報は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。

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、「」の手順に従ってパイプラインを手動で開始する、AWS CLI を使用して変更を手動でリリースします。

### 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、「<u>パ</u> イプラインを手動で開始する」を参照してください。 ▲ 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編集できますが、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

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。

 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、「<u>CodePipeline パイプライン構造リファレンス</u>」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": [
                    ſ
                        "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、「<u>CodePipeline で</u> 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。

- 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 に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。
- 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、「<u>CodeCommit ソースに対する EventBridge ルールを作成する (CLI)</u>」を参照し てください。
  - Amazon S3 ソースの場合、「」で説明されているように、CloudWatch Events ルールと AWS CloudTrail 証跡を作成する必要があります<u>EventBridge と を使用する Amazon S3 ソー</u> スアクションへの接続 AWS CloudTrail。
- 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。

A 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、startpipeline-execution コマンドを実行します。詳細については、「<u>パイプラインを手動で開始す</u> 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、「<u>CodePipeline パイプライン構造リファレン</u> 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。

Pipe	lines Info	♦ Notify ▼ Vie	w history Release cl	hange Delete pipeline	Create pipeline
Q					< 1 > 🕲
	Name	Latest execution status	Latest execution started	Most recent executions	
0	Pipeline-trigger (Type: V2   Execution mode: SUPERSEDED)	Succeeded	2 days ago	$\oslash \odot \odot \odot \ominus \ominus$ View details	5
0	check1 (Type: V2   Execution mode: SUPERSEDED)	⊗ Failed	2 days ago	🛞 🛞 🛞 ⊗ 🛇 View details	5
0	<b>tr-pi2</b> (Type: V2   Execution mode: QUEUED)	⊖ Stopped	19 days ago	) 🙁 View details	
0	Pipeline-Stack (Type: V1   Execution mode: SUPERSEDED)	⊗ Failed	2 months ago	🛞 🛞 🛞 View details	
0	Pipeline-ChangeSet (Type: V2   Execution mode: QUEUED)	⊗ Failed	2 months ago	🛞 🛞 🛞 View details	

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。

Most recent execu	tions			×
Trigger StartPipelineExecution	- assumed-role,		2	A
Pipeline execution ID	Status	Last updated		
7cb97af6	Θ In progress	51 minutes ago		
Trigger				
StartPipelineExecution	- assumed-role/			
Pipeline execution ID	Status	Last updated		
b289be6e	⊘ Succeeded	1 hour ago		
Trigger				
Webhook - connection/	40d122c4-23fb-48	3bf-a08f-	2	
Pipeline execution ID	Status	Last updated		
049c2110	Succeeded Succeeded	3 months ago		
Trigger				
Webhook - connection/	40d122c4-23fb-48	8bf-a08f	2	
Pipeline execution ID	Status	Last updated		
3dcaf66a	Succeeded Succeeded	4 months ago		
Trigger				
Webhook - connection/	40d122c4-23fb-48	Bbf-a08f-	2	
				· ·
				Done

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。

Source Succeeded Pipeline execution ID: <u>b289be6e-e25d-4f61-a5c3-821028cfe4d3</u>	
Source <u>GitHub /Version 2</u> } Succeeded - <u>1 minute 200</u> 2de85792 View details	
2de8579a 🖾 Source: Add files via upload	0
Disable transition	
Build Succeeded Pipeline execution ID: <u>b289be6e-e25d-4f61-a5c3-821028cfe4d3</u>	
Build AWS CodeBuild Succeeded - Just now View details	
2de8579a 🖾 Source: Add files via upload	

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 AWS CloudFormation アクションで使用できます。

- 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。
- 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。
- 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。

 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 AWS CloudFormation アクションで使用できます。

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。

Action execution details Action name: Build Status: Succeeded		×
Summary Logs Configuration		
Status	Last updated	
⊘ Succeeded	1 minute ago	
Action execution ID		
850739e4-13ef-4de8-a721-32c87727a1c7		
Message -		
Execution details		
View in CodeBuild 🗹		

Done

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。

Summary	Logs	Configuration		
Succeeded			Start time: 3 minutes ago	Current phase: COMPLETED
Showing the	last 51 lines	of the build log. <u>Vie</u>	w entire log	
			∧ Show previous log	gs
1 [Contain 2 [Contain 3 [Contain 5 [Contain 6 [Contain 7 [Contain 8 [Contain 9 [Contain 10 [Contain 11 Installi 12 13 [Contain 14 cop 15 insta 16	er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 ying : nod lled : v12	1/10 19:23:33.8421 1/10 19:23:34.0434 1/10 19:23:35.2327 1/10 19:23:35.2337 1/10 19:23:35.2345 1/10 19:23:35.2345 1/10 19:23:35.2364 1/10 19:23:35.4352 1/10 19:23:35.4352 1/10 19:23:36.8980 version 12 1/10 19:23:36.8980 e/12.22.12 .22.12 (with npm 6.	20 Waiting for agent ping 25 Waiting for DOWNLOAD_SOURCE 26 Phase is DOWNLOAD_SOURCE 27 CODEBUILD_SRC_DIR=/codebuild 39 YAML location is /codebuild 30 YAML location is /codebuild 30 Setting HTTP client timeout 31 Processing environment varia 32 Setting 'nodejs' runtime y 33 Running command echo "Insta 34 Running command n \$NODE_12_Y .14.16)	d/output/src180370599/src /readonly/buildspec.yml name: install to higher timeout for S3 source ables version '12' based on manual selections lling Node.js version 12"
<pre>17 [Contain 18 [Contain 19 [Contain 22850a87 20 [Contain 21 [Contain 22 [Contain 23 [Contain 24 [Contain</pre>	er] 2024/0 er] 2024/0 b0f4 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0 er] 2024/0	1/10 19:24:09.75334 1/10 19:24:09.75480 1/10 19:24:09.79169 1/10 19:24:09.82224 1/10 19:24:09.822260 1/10 19:24:09.82271 1/10 19:24:09.82271 1/10 19:24:09.82271	46 Moving to directory /codebu 55 Unable to initialize cache 77 Configuring ssm agent with 49 Successfully updated ssm ag 59 Registering with agent 16 Phases found in YAML: 2 23 INSTALL: 0 commands 27 PRE_BUILD: 2 commands	ild/output/src180370599/src download: no paths specified to be cached target id: codebuild:f79dc603-3eb0-48ff-970e- ent configuration
24 [Contain	er] 2024/0	1/10 19:24:09.82272	27 PRE_BUILD: 2 commands	D

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。

Action execution of Action name: Build Sta	letails atus: Succeeded			×
Summary Log	s Configuration	_		
Variable namespace	BuildVariables			
Input artifact	SourceArtifact			
Output artifact	BuildArtifact			
ProjectName	cb-porject			
				Done

# 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。

- 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。
- 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。

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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。

#### 🗗 e1a7e739-t211-420e-aet9-ta7837666968



### 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 の delete-pipeline コマンドを使用してパイプライン AWS CLI 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. にサインイン AWS Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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. [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)

を使用してパイプライン AWS CLI を手動で削除するには、delete-pipeline 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

 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パイプラインを CodePipeline で作成する

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 キーには、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、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。<u>Amazon EC2 インスタンスロール</u> 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 <u>Amazon EC2 インスタンスロール</u>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 Management Console を使用して にサインインし、 AWS KMS コンソールを 開きます。
- 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。
- 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。
- [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/222222-3333333-4444-556677EXAMPLE*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 Management Console を使用して にサインインし、「https:// console.aws.amazon.com/s3/.com で Amazon S3 コンソールを開きます。
- 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確認する 10 桁の乱数です (例: codepipeline-useast-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。

 [Bucket Policy Editor]ウィンドウに以下のポリシーを入力します。これにより、AccountBはパ イプラインアーティファクトにアクセスできるようになり、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作成された場合は、AccountBで出力アーティファクトを追加するこ とができます。

次の例では、*AccountB*の ARN は、*012ID\_ACCOUNT\_B*です。Amazon S3 バケットの ARN は 次のとおりです。*codepipeline-us-east-2-1234567890* 。これらの ARN を、アクセスを 許可するアカウントの ARN、および Amazon S3 バケットの ARN に置き換えます。

```
{
 "Version": "2012-10-17",
"Id": "SSEAndSSLPolicy",
"Statement": [
{
 "Sid": "DenyUnEncryptedObjectUploads",
 "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:PutObject",
 "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
 "Condition": {
 "StringNotEquals": {
"s3:x-amz-server-side-encryption": "aws:kms"
}
  }
},
{
 "Sid": "DenyInsecureConnections",
 "Effect": "Deny",
  "Principal": "*",
  "Action": "s3:*",
  "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
 "Condition": {
 "Bool": {
    "aws:SecureTransport": false
 }
 }
 },
ſ
  "Sid": "",
```

```
"Effect": "Allow",
   "Principal": {
  "AWS": "arn:aws:iam::012ID ACCOUNT B:root"
 },
  "Action": [
            "s3:Get*",
            "s3:Put*"
        ],
  "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
},
 {
  "Sid": "",
   "Effect": "Allow",
   "Principal": {
       "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
  },
  "Action": "s3:ListBucket",
  "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890"
}
  ]
}
```

- 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 Management Console で にサインインし、「https://<u>https://</u> console.aws.amazon.com/iam/.com 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。

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

{

```
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": [
"arn:aws:iam::012ID_ACCOUNT_B: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 [<u>Amazon</u> <u>EC2 インスタンスロール</u>] 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 Management Console で にサインインし、「https:// console.aws.amazon.com/iam/.com 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。
- [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。
- 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):

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "s3:Get*"
       ],
       "Resource": [
         "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
       ]
     },
     {
       "Effect": "Allow",
       "Action": [
         "s3:ListBucket"
       ],
       "Resource": [
         "arn:aws:s3:::codepipeline-us-east-2-1234567890"
       ]
     }
   ]
 }
```

- 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. 2 つ目のポリシーを作成します。 AWS KMS ここで、 arn:aws:kms:useast-1:012ID\_ACCOUNT\_A:key/222222-3333333-4444-556677EXAMPLE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。

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
           "kms:DescribeKey",
           "kms:GenerateDataKey*",
           "kms:Encrypt",
           "kms:ReEncrypt*",
           "kms:Decrypt"
          ],
        "Resource": [
           "arn:aws:kms:us-
east-1:012ID_ACCOUNT_A:key/222222-3333333-4444-556677EXAMPLE"
          ٦
      }
   ]
}
```

▲ 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 Management Console で にサインインし、「https:// console.aws.amazon.com/iam.com 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。[ロールの作成]を選択します。
- [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」で、CodePipeline (AWS AccountAccountA)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。

 [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。

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

```
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
           "codedeploy:CreateDeployment",
           "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。

```
{
   "Version": "2012-10-17",
   "Statement": [
     {
       "Effect": "Allow",
       "Action": [
         "s3:GetObject*",
         "s3:PutObject",
         "s3:PutObjectAcl"
       ],
       "Resource": [
         "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
       1
     }
   ]
}
```

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. リポジトリの [アクセス許可] タブで、[ポリシー名]から AmazonS3ReadOnlyAccess を探して、 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)

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。

 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:useast-1:012ID\_ACCOUNT\_A:key/222222-3333333-4444-556677EXAMPLE 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/222222-3333333-4444-556677EXAMPLE",
    "type": "KMS"
    }
},
```

 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"
                   }
               ٦
```

}

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。

 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、 updatepipeline 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

 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、「<u>パイプラインを</u>手動で開始する」を参照してください。

2. AccountA AWS Management Console で にサインインし、「https://<u>http://</u> <u>console.aws.amazon.com/codesuite/codepipeline/home</u>.com 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。
- 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、コードが変更されるたびにパイプラインを開始するなど、完全なend-to-end の継続的配信をサポートします。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 AWS 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)。	「 <u>CodeCommit ソースを使用してポー</u> <u>リングパイプラインを移行する</u> 」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)。	「 <u>イベントに対応した S3 ソースを使</u> <u>用してポーリングパイプラインを移行</u> <u>する</u> 」を参照してください。
Amazon S3	EventBridge と AWS CloudTrail 証跡。	「 <u>S3 ソースと CloudTrail 証跡を使用</u> <u>してポーリングパイプラインを移行す</u> <u>る</u> 」を参照してください。
GitHub (GitHub ア プリ経由)	Connections (推奨)	「 <u>GitHub (OAuth アプリ経由) ソース</u> <u>アクションのポーリングパイプライン</u> <u>を接続に移行する</u> 」を参照してくださ い。
GitHub (OAuth アプ リ経由)	ウェブフック	「 <u>GitHub (OAuth アプリ経由) ソース</u> アクションのポーリングパイプライン <u>をウェブフックに移行する</u> 」を参照し てください。

### A Important

GitHub (viaOAuth アプリ) アクションを使用するパイプラインなど、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 PollForSourceChangesパ ラメータを明示的に 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、「<u>AWS CodePipeline のクォータ</u>」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

アカウント内のポーリングパイプラインの表示

```
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' <<< "$LIST_PIPELINES_RESPONSE")</pre>
    NEXT TOKEN=$(jg -r '.nextToken' <<< "$LIST PIPELINES RESPONSE")</pre>
    if [ "$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))' <<< "$PIPELINE")</pre>
        if [ ! -z "$HAS_POLLABLE_ACTIONS" ];
        then
            POLLING_PIPELINES+=("$pipline_name")
            PIPELINE_EXECUTIONS=$(aws codepipeline list-pipeline-executions --
pipeline-name $pipline_name --region $REGION)
```

```
LAST_EXECUTION=$(jq -r '.pipelineExecutionSummaries[0]' <<<</pre>
 "$PIPELINE_EXECUTIONS")
            if [ "$LAST_EXECUTION" != "null" ];
                then
                    LAST_EXECUTED_TIMESTAMP=$(jq -r '.startTime' <<<</pre>
 "$LAST EXECUTION")
                   LAST_EXECUTED_DATE="$(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()))
```

- 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。パイプラインのソース設定で PollForSourceChanges パラメータが省略されている場合、CodePipeline は デフォルトでリポジトリにポーリングしてソースの変更を確認します。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: <u>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</u> ソール)
- CLI: <u>ポーリングパイプラインを移行する (CodeCommit ソース) (CLI)</u>
- AWS CloudFormation: <u>ポーリングパイプラインの移行 (CodeCommit ソース) (AWS</u> <u>CloudFormation テンプレート)</u>

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、「<u>CodeCommit ソースアクションと EventBridge</u>」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、「<u>パイプライン、ステージ、アクションを作成する</u>」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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。

- [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。[Update] (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。[Update] (更新) を選択します。

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。 [Save and continue] を選択します。

 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

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

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

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。

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"
},
```

 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 startpipeline-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

- 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。

```
{
    "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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
          "Effect": "Allow",
          "Action": [
              "codepipeline:StartPipelineExecution"
        ],
        "Resource": [
              "arn:aws:codepipeline:us-west-2:80398EXAMPLE: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

 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、 AWS 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

- (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、オーバーライドを設 定します。
  - Source この例ではactionName、は、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る動的値です。
  - COMMIT\_ID この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、ソースイベント変数から派生しています。

ポーリングパイプラインの移行 (CodeCommit ソース) (AWS 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

AWS 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 AWS CloudFormation テンプレートを更新して EventBridge ルールを作成するには

- テンプレートのでResources、AWS::IAM::Role AWS 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 AWS CloudFormation 、 は EventBridge のアクセス許可を作成できます。このリソースは AWS 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": [
                  "",
                  Ε
                     "arn:aws:codepipeline:",
                     {
                      "Ref": "AWS::Region"
                    },
                     ":",
```

```
{
    "Ref": "AWS::AccountId"
    },
    ":",
    {
        "Ref": "AppPipeline"
    }
]
....
```

 テンプレートのでResources、AWS::Events::Rule AWS 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 AWS CloudFormation 、 は イベントを作成できます。このリソースは AWS 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
```

```
"EventRule": {
  "Type": "AWS::Events::Rule",
  "Properties": {
    "EventPattern": {
      "source": [
        "aws.codecommit"
      ],
      "detail-type": [
        "CodeCommit Repository State Change"
      ],
      "resources": [
        {
          "Fn::Join": [
            "",
            Ε
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
```

```
],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
          "main"
        ]
      }
    },
    "Targets": [
      {
        "Arn": {
          "Fn::Join": [
            "",
            Ε
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
          ]
        },
        "Id": "codepipeline-AppPipeline"
      }
    ]
  }
},
```

- (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、オーバーライドを設定し ます。
  - Source この例ではactionName、は、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る動的値です。
  - COMMIT\_ID この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、ソースイベント変数から派生しています。
  - BranchName およびの出力変数Value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、 AWS 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。
- テンプレートをアップロードし、AWS 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
```

```
{
    "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
    }
 1
},
```

## Example

これらのリソースを で作成すると AWS 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": [
                                 "",
                                 Ε
                                      "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": [
                         "",
                         Ε
                             "arn:aws:codecommit:",
                             {
                                  "Ref": "AWS::Region"
                             },
                             ":",
                             {
                                 "Ref": "AWS::AccountId"
                             },
                             ":",
                             {
                                 "Ref": "RepositoryName"
                             }
                         ]
                     ]
                 }
            ],
            "detail": {
                 "event": [
                     "referenceCreated",
                     "referenceUpdated"
                 ],
                 "referenceType": [
                     "branch"
                 ],
                 "referenceName": [
                     "main"
                 ]
            }
```

```
},
        "Targets": [
            {
                 "Arn": {
                     "Fn::Join": [
                         "",
                         Ε
                             "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 AWS CloudFormation を移行する場合に推奨される方法で す。 (i) 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、「<u>S3 ソースと</u> <u>CloudTrail 証跡を使用してポーリングパイプラインを移行する</u>」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)
- AWS CloudFormation: <u>S3 ソースと CloudTrail</u> 証跡 (AWS 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、「<u>パイプライン、ス</u> <u>テージ、アクションを作成する</u>」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

- 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。

"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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "codepipeline:StartPipelineExecution"
        ],
            "Resource": [
               "arn:aws:codepipeline:us-west-2:80398EXAMPLE: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

 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/Rolefor-MyRule"

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

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。

 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、amzn-s3-demosource-bucket という名前のバケットの 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。

M 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-pipelineexecution 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 (AWS 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。

AWS 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
. . .
```

```
"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

- テンプレートのでResources、AWS::IAM::Role AWS 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 AWS 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 AWS 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 ] ]
. . .
```

```
"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": [
                    "",
                    Γ
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                       {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                       {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]
. . .
```

AWS::Events::Rule AWS 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 AWS CloudFormation 、 は イベントを作成できます。このリソースは AWS 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": [
            "",
            Ε
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
          ]
        },
        "RoleArn": {
          "Fn::GetAtt": [
            "EventRole",
            "Arn"
```



- 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、 AWS 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、 AWS 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

```
{
   "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

AWS 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 AWS 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
```

```
{
    "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": {
                         "Fn::Join": [
                             "",
                             Ε
                                 {
                                     "Fn::GetAtt": [
                                          "CodePipelineArtifactStoreBucket",
                                          "Arn"
                                     ]
                                 },
                                 "/*"
                             ]
                         ]
                    },
                     "Condition": {
                         "StringNotEquals": {
                             "s3:x-amz-server-side-encryption": "aws:kms"
                         }
                    }
```

```
},
                {
                     "Sid": "DenyInsecureConnections",
                     "Effect": "Deny",
                     "Principal": "*",
                     "Action": "s3:*",
                     "Resource": {
                         "Fn::Join": [
                             "",
                             Ε
                                 {
                                      "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": [
                                      "",
                                      Ε
                                          "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": {
                           "Fn::Join": [
                               "",
                               Γ
                                   "arn:aws:codepipeline:",
                                   {
                                        "Ref": "AWS::Region"
                                   },
                                   ":",
                                   {
                                       "Ref": "AWS::AccountId"
                                   },
                                   ":",
                                   {
                                       "Ref": "AppPipeline"
                                   }
                               ]
                           ]
```



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)
- AWS CloudFormation: S3 ソースと CloudTrail 証跡 (AWS 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、およびバケット 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、<u>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</u>」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-sourcebucket

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

 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

- 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。

```
{
    "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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "codepipeline:StartPipelineExecution"
        ],
            "Resource": [
               "arn:aws:codepipeline:us-west-2:80398EXAMPLE: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

 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

- (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、オーバーライドを設 定します。
  - Source この例ではactionName、は、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る動的値です。
  - S3\_OBJECT\_VERSION\_ID この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、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

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。

 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、amzn-s3-demosource-bucket という名前のバケットの 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"
},
```

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。

A 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-pipelineexecution コマンドを使用します。 S3 ソースと CloudTrail 証跡 (AWS 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、「<u>証跡のデータイベント</u>」と「<u>管理イベント</u> のログ記録」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。

AWS 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

```
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": {
```

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

- テンプレートの でResources、 AWS::IAM::Role AWS 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 AWS 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 AWS 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 ] ]
...
```

```
"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": [
                     "",
                     Г
                       "arn:aws:codepipeline:",
                       {
                         "Ref": "AWS::Region"
                       },
                       ":",
                       {
                         "Ref": "AWS::AccountId"
                       },
                       ":",
                       {
                         "Ref": "AppPipeline"
                       }
                     ]
                   1
. . .
```

 AWS::Events::Rule AWS 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 AWS CloudFormation 、 は イベントを作成できます。このリソースは AWS CloudFormation スタックに追加されます。

YAML

EventRule: Type: AWS::Events::Rule Properties: EventPattern:



```
"EventRule": {
   "Type": "AWS::Events::Rule",
   "Properties": {
      "EventPattern": {
        "source": [
           "aws.s3"
      ],
      "detail-type": [
           "AWS API Call via CloudTrail"
      ],
      "detail": {
           "eventSource": [
           "s3.amazonaws.com"
```





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
```

```
"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、オーバーライドを設定し ます。
  - Source この例ではactionName、は、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る動的値です。

- S3\_OBJECT\_VERSION\_ID この例ではrevisionType、は、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る動的値です。
- この例の revisionValue、 < revisionValue> は、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、 AWS 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、 AWS 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

#### A 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、 「<u>PollForSourceChanges パラメータの有効な設定</u>」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

```
{
    "Name": "SourceAction",
    "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "S3"
    },
    "OutputArtifacts": [
        {
            "Name": "SourceOutput"
        }
    ],
    "Configuration": {
            "S3Bucket": {
            "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
               "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
                "Same": {
```

```
"Ref": "SourceBucket"
},
"S30bjectKey": {
    "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 AWS 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。(「<u>の制限 AWS CloudTrail</u>」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

```
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": [
              "",
              Γ
                 {
                   "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": [
                       "",
                       Γ
                         {
                           "Fn::ImportValue": "SourceBucketARN"
                         },
                         "/",
                         {
                           "Ref": "SourceObjectKey"
                         }
                       ]
                     ]
                   }
                ]
              }
            ],
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": false
          }
        ],
        "IncludeGlobalServiceEvents": true,
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
 }
}
. . .
```

### Example

AWS 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 AWS 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
```

```
"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": [
                                 "",
                                 Ε
                                     {
```

```
"Fn::GetAtt": [
                                          "CodePipelineArtifactStoreBucket",
                                          "Arn"
                                      ]
                                  },
                                  "/*"
                             ]
                         ]
                     },
                     "Condition": {
                         "StringNotEquals": {
                             "s3:x-amz-server-side-encryption": "aws:kms"
                         }
                     }
                },
                 {
                     "Sid": "DenyInsecureConnections",
                     "Effect": "Deny",
                     "Principal": "*",
                     "Action": "s3:*",
                     "Resource": {
                         "Fn::Join": [
                             "",
                             Ε
                                  {
                                      "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"
            1
        },
        "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": [
                                     "",
                                     Ε
                                          "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": [
                                 "",
                                  Ε
                                      {
                                          "Fn::GetAtt": [
                                              "SourceBucket",
                                              "Arn"
                                          ]
                                      },
```



```
"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移行して、外部リポジトリの接続を使用できます。 これは、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に推奨される変更検 出方法です。

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、変更検出が自動化さ れるようにGitHub (GitHub アプリ経由) アクションを使用するようにパイプラインを変更することを お勧めします AWS CodeConnections。接続の使用の詳細については、「<u>GitHub コネクション</u>」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。
- 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 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。
- 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]を選択してください。

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。

③ 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、<u>Bitbucket、GitHub、GitHub Enterprise Server、または</u> <u>GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。</u>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、<u>チュートリアル: CodeCommit パイプラ</u> <u>インソースで完全なクローンを使用する</u>をご覧ください。

- 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。
- [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。

#### A Important

AWS CLI または を介して作成された接続 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 AWS 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

 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 --providertype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 ウェブフックに移行する

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。このウェブフックへの移行は、GitHub (OAuth アプリ経由) アクション専用です。

- コンソール: ポーリングパイプラインをウェブフックに移行する (GitHub (OAuth アプリ経由) ソー スアクション) (コンソール)
- CLI: <u>ポーリングパイプラインをウェブフックに移行する (GitHub (OAuth アプリ経由) ソースアク</u> ション) (CLI)
- AWS CloudFormation: <u>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)</u> ソースアクション) (AWS 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ウェブフックに移行する (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) 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、「<u>パイプライン、ス</u> 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

- [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 にウェブフックを作成してソースの変更を検出することを示すメッセー 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。[Update] (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。

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。[Save and continue] を選択します。

 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ウェブフックに移行する (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、「<u>パイプライン、ステージ、アクションを作成</u> 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 AWS 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 AWS 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、「<u>PollForSourceChanges パラメータの有効な設定</u>」を参照してく ださい。

 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。「mywebhook」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/
trigger111111111EXAMPLE111111111111111111,
        "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、<u>リソースのタ</u> <u>グ付け</u>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

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。

 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、 updatepipeline 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-pipelineexecution 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) (AWS 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。 で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。

AWS 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"
```



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

認証情報の保存 AWS Secrets Manager には を使用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、「<u>動的な参照を使用してテンプレート値を指定する</u>」を参照してくださ い。

A 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、 AWS CloudFormation テンプレートで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 AWS 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、Pol1ForSourceChanges 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 AWS 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、 「<u>PollForSourceChanges パラメータの有効な設定</u>」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"
        },
    ....
```

AWS::CodePipeline::Webhook AWS 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、「<u>https://</u> 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"
            1
        },
        "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、 AWS 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、AWS 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

A 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 AWS 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: "\$.ret"
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
0AuthToken:
{{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"
                             },
```



# 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た AWS 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<u>チュートリアル: AWS</u> 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、「<u>CodePipeline サービスロールを管理する</u>」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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。
- 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 AWS 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。

 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。

```
{
    "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

 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

 JSON ポリシーをファイルに貼り付けます。「」で指定されている最小サービスロールポリシー を使用します<u>CodePipeline サービスロールポリシー</u>。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と各アク 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、「」を参照してくださ い<u>CodePipeline サービスロールにアクセス許可を追加する</u>。

Resource フィールドのリソースレベルまでスコープダウンして、アクセス許可を可能な限り スコープダウンしてください。

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作成されたポリシー TrustPolicy 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 て作成されたポリシー MyRole を使用します。

aws iam put-role-policy --role-name MyRole --policy-name RolePolicy --policydocument 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 AWS 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<u>AWS タグ付け戦略</u>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、「<u>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</u>」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、<u>リソースのタグ付け</u>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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。
- [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. [送信]を選択します。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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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. [送信]を選択します。

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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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. [送信]を選択します。

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、「<u>パイプライン、ステージ、アクションを作成する</u>」 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、「<u>AWS Command Line Interfaceのインストール</u>」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:accountid: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:uswest-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、tagresource 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:accountid: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:accountid:MyPipeline --tag-keys Project IscontainerBased

パイプラインにタグ付けする (CLI)

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 Management Console し、「https://<u>https://console.aws.amazon.com/</u> codepipeline/.com」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。
- 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。

- [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すでに設定している場合は、ターゲットタイプを選択す るで、チャットアプリケーション (Slack) で Amazon Q Developer または SNS トピックを選 択します。ターゲットの選択で、クライアントの名前 (チャットアプリケーションで Amazon Q Developer で設定された 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で既に設定された 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クライアントで Amazon Q Developer 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 (Slack) で Amazon Q Developer 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コンソールで Amazon Q Developer が開きます。詳細については、「Configure Integrations Between Notifications and Amazon Q Developer in chat applications」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。
- 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、「<u>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</u>」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 の統合を設定して、Amazon Chime チャットルームまたは Slack チャネルに通知を送信することもできます。詳細については、「Configure Integration Between Notifications and Amazon Q Developer in chat applications」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。

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。

 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 が 123456789012 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-notificationrule 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。

```
{
```

通知ルールの作成

```
"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?	ステップ
CloudTrail リソー スを使用した 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。	<u>パイプライン、ステージ、アクショ</u> <u>ンを作成する</u> および <u>EventBridge と</u> <u>を使用する Amazon S3 ソースアク</u> <u>ションへの接続 AWS CloudTrail</u> 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 クションを作成する場合は、これ らのリソースも作成および管理し ます。	<u>パイプライン、ステージ、アクショ ンを作成する</u> および <u>イベントに対</u> してソースを有効にした Amazon S3 <u>ソースアクションへの接続</u> 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?	ステップ
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。	「 <u>Bitbucket Cloud への接続</u> 」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。	<u>パイプライン、ステージ、アクショ</u> <u>ンを作成する</u> および <u>CodeCommit</u> <u>ソースアクションと EventBridge</u> 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。	「 <u>パイプライン、ステージ、アク</u> <u>ションを作成する</u> 」および「 <u>Amazon</u> <u>ECR ソースアクションと EventBrid</u> <u>ge リソース</u> 」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。	<u>GitHub コネクション</u> 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。	「 <u>GitHub Enterprise Server 接続</u> 」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。	「 <u>GitLab.com への接続</u> 」を参照して ください。
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。	「 <u>GitLab セルフマネージドの接続</u> 」 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、「<u>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</u>」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、<u>CodeStarSourceConnection</u> (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージ <u>ドアクションの場合</u>) 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、「<u>CodeCommit ソースアクションと EventBridge</u>」 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
- イベントに対してソースを有効にした Amazon S3 ソースアクションへの接続
- EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail
- <u>CodeCommit ソースアクションと EventBridge</u>

## 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 [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] ページを使用し、[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 <u>Amazon ECR ソースアクションリファレンス</u> で使用し、<u>パイプ</u> 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。<u>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 (AWS CloudFormation テンプレート)</u>の内、いずれかのの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 (AWS 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 <u>https://console.aws.amazon.com/events/</u>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。
- [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"
]
}
```

### 1 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. [Save]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。

### 1 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

- 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 に与え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。

- 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。
  - ・ [追加の設定] を展開します。

ターゲット入力の設定 で、入力トランスフォーマーの設定 を選択します。

ダイアログウィンドウで、自分の を入力します。入力パスボックスに、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に、次のキーと値のペアを入力します。

{ "sourceRevisions": { "actionName": "Source", "revisionType": "IMAGE\_DIGEST", "revisionValue": "<revisionValue>" } }

・[確認]を選択してください。

- 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、「<u>Amazon</u> 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

- 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。

```
{
    "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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "codepipeline:StartPipelineExecution"
        ],
            "Resource": [
```

"arn:aws:codepipeline:us-west-2:80398EXAMPLE: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

 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:uswest-2:80398EXAMPLE:TestPipeline,RoleArn=arn:aws:iam::80398EXAMPLE:role/Role-for-MyRule

- (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、オーバーライドを設 定します。
  - Source この例ではactionName、は、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る動的値です。
  - IMAGE\_DIGEST この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、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 (AWS CloudFormation テンプレート)

AWS 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 AWS CloudFormation テンプレートを更新して EventBridge ルールを作成するには

- テンプレートのでResources、AWS::IAM::Role AWS 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。

```
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}"

}

}

}

}

...
```

 テンプレートのでResources、AWS::Events::Rule AWS 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。

```
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"
                         ]
                     },
```



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、[<u>Amazon ECR Events と EventBridge</u>] または [<u>Amazon Elastic Container Registry</u> <u>Events</u>] を参照してください。

- (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、オーバーライドを設定し ます。
  - Source この例ではactionName、は、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る動的値です。
  - IMAGE\_DIGEST この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、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、 AWS 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、AWS 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。

# イベントに対してソースを有効にした Amazon S3 ソースアクショ ンへの接続

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のない 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、「」の手順を参照するか、「」を参照してください<u>イベン</u> 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 AWS 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、「<u>S3 ソースと</u> <u>CloudTrail 証跡を使用してポーリングパイプラインを移行する</u>」を参照してください。

- CLI: <u>S3 ソースと CloudTrail</u> 証跡を使用してポーリングパイプラインを移行する (CLI)
- AWS CloudFormation: <u>S3 ソースと CloudTrail</u> 証跡 (AWS 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 S3 ソースを有効にしてパイプラインを作成する (CLI)

EventBridge の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、「」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。

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

- 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。

```
{
    "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。

 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

 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

#### A 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

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。

 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、amzn-s3-demosource-bucket という名前のバケットの 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"
},
```

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。

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。

A 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-pipelineexecution 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 (AWS 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。

以下のステップを使用して、イベントベースの変更検出用の Amazon S3 ソースを使用してパイプラ インを作成します。

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。

AWS 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。

```
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

- テンプレートの でResources、 AWS::IAM::Role AWS 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 AWS 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 AWS CloudFormation スタックに 追加されます。

```
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": [
                     "",
                     Ε
                       "arn:aws:codepipeline:",
                       {
                         "Ref": "AWS::Region"
                       },
                       ":",
                       {
                         "Ref": "AWS::AccountId"
                       },
                       ":",
                       {
                         "Ref": "AppPipeline"
                       }
                     ]
                   1
. . .
```

AWS::Events::Rule AWS 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 AWS CloudFormation 、 は イベントを作成できます。このリソースは AWS 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": [
               "",
               Ε
                 "arn:aws:codepipeline:",
                 {
                   "Ref": "AWS::Region"
                 },
                 ":",
                 {
                   "Ref": "AWS::AccountId"
                 },
                 ":",
                 {
                   "Ref": "AppPipeline"
                 }
               ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
               "EventRole",
               "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},
. . .
```

- 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、 AWS 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、 AWS 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。

	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

AWS 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 AWS 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。

```
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

{

```
"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": {
                "Fn::Join": [
                     "",
                     Γ
                         {
                             "Fn::GetAtt": [
                                 "CodePipelineArtifactStoreBucket",
                                 "Arn"
                             ]
                         },
                         "/*"
                     ]
                ]
            },
            "Condition": {
                "StringNotEquals": {
                     "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": {
                "Fn::Join": [
                     "",
                     Ε
                         {
                             "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": [
                                     "",
                                      Γ
                                          "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": {
                           "Fn::Join": [
                               "",
                                Ε
                                    "arn:aws:codepipeline:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    ":",
                                    {
                                        "Ref": "AWS::AccountId"
                                    },
                                    ":",
                                    {
                                        "Ref": "AppPipeline"
                                    }
                               ]
                           ]
                       },
                       "RoleArn": {
                           "Fn::GetAtt": [
                                "EventRole",
                               "Arn"
                           ]
                       },
                       "Id": "codepipeline-AppPipeline"
                   }
              ]
          }
      }
  }
  }
```

}

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ステップを示します。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 手順を使用します<u>イベントに対</u> 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、「<u>イベン</u> <u>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</u>」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 [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] ページを使用し、[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 <u>Amazon S3 ソースアクションリファレンス</u> で使用し、<u>パイプラ</u> 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 (AWS 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、「<u>CloudTrail の</u> 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、「<u>コンソールで証跡を作成する</u>」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。
- [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。

- [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。
- [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。
- 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。ルールの作成を選択します。
- 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。[Next (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 myfiles.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。

{

Amazon S3 ソースに対する EventBridge ルールを作成する (コンソール)

```
"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. [Next (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):
  - 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 に与え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。
- 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。
  - [追加の設定] を展開します。

ターゲット入力の設定で、入力トランスフォーマーの設定を選択します。

ダイアログウィンドウで、自分の を入力します。入力パスボックスに、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に、次のキーと値のペアを入力します。



- [確認] を選択します。
- 14. [Next (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、<u>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</u>」を参照してくだ さい。

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-sourcebucket

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

 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

- 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。

```
{
    "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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "codepipeline:StartPipelineExecution"
        ],
            "Resource": [
               "arn:aws:codepipeline:us-west-2:80398EXAMPLE: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

 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"

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

- (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、オーバーライドを設 定します。
  - Source この例ではactionName、は、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る動的値です。
  - S3\_OBJECT\_VERSION\_ID この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、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

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。

 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、amzn-s3-demosource-bucket という名前のバケットの 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"
},
```

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。

A 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-pipelineexecution 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 (AWS CloudFormation テンプレート) Amazon S3

AWS 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

- テンプレートの でResources、 AWS::IAM::Role AWS 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 : : Ro1e リソースを追加する AWS 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 AWS 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": [
                  "",
                  Γ
```

```
"arn:aws:codepipeline:",
    {
        "Ref": "AWS::Region"
    },
     ":",
    {
        "Ref": "AWS::AccountId"
    },
        ":",
        {
            "Ref": "AppPipeline"
        }
        ]
    ]
....
```

 AWS::Events::Rule AWS 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 AWS CloudFormation 、 は イベントを作成できます。このリソースは AWS 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": [
               "",
               Γ
                 "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"
        }
    }
....
```

- (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、オーバーライドを設定し ます。
  - Source この例ではactionName、は、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る動的値です。
  - S3\_OBJECT\_VERSION\_ID この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、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:
<pre>revisionValue: "\$.detail.object.version-id</pre>
InputTemplate:
sourceRevisions:
actionName: Source
revisionType: S3_OBJECT_VERSION_ID
revisionValue: '< <u>revisionValue</u> >'

- 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、 AWS 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。
- 更新されたテンプレートをアップロードし、 AWS 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

#### A 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、 「<u>PollForSourceChanges パラメータの有効な設定</u>」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:	
-	
Ν	ame: SourceAction
A	ctionTypeId:
	Category: Source
	Owner: AWS
	Version: 1
	Provider: S3
0	utputArtifacts:
	- Name: SourceOutput
C	onfiguration:
	S3Bucket: !Ref SourceBucket
	S3ObjectKey: !Ref SourceObjectKey
	PollForSourceChanges: false
R	unOrder: 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 AWS 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"
                1
              }
            },
            {
              "Sid": "AWSCloudTrailWrite",
              "Effect": "Allow",
```

```
"Principal": {
            "Service": [
              "cloudtrail.amazonaws.com"
            ]
          },
          "Action": "s3:PutObject",
          "Resource": {
            "Fn::Join": [
              "",
              Ε
                 {
                  "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": [
                        "",
                        Γ
                          {
                            "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 [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] ページを使用し、[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 (AWS CloudFormation テンプレート)の内、いずれかのの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 (AWS 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)

\Lambda 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。
- 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。ルールの作成を選択します。
- 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。[Next (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。
- [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

- 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):
  - 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 に与え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。
- 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。
  - [追加の設定] を展開します。

ターゲット入力の設定 で、入力トランスフォーマーの設定 を選択します。

ダイアログウィンドウで、自分の を入力します。入力パスボックスに、次のキーと値のペア を入力します。

{"revisionValue": "\$.detail.image-digest",
"branchName": "\$.detail.referenceName"}

・ テンプレートボックスに、次のキーと値のペアを入力します。

• [確認]を選択します。

- 13. [Next (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、「<u>Amazon</u> 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

- 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。

```
{
    "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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "codepipeline:StartPipelineExecution"
        ],
            "Resource": [
               "arn:aws:codepipeline:us-west-2:80398EXAMPLE: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

 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、 AWS 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"
```

- 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

- (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、オーバーライドを設 定します。
  - Source この例ではactionName、は、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る動的値です。
  - COMMIT\_ID この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、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": {
               "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions": {
                "sourceRevisions"; {
```



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

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。

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"
},
```

 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。

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 startpipeline-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 (AWS CloudFormation テンプレート)

AWS 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 AWS CloudFormation テンプレートを更新して EventBridge ルールを作成するには

- テンプレートのでResources、AWS::IAM::Role AWS 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 AWS CloudFormation 、 は EventBridge のアクセス許可を作成できます。このリソースは AWS 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:
```

```
ユーザーガイド
```

```
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"
    }
]
....
```

 テンプレートのでResources、AWS::Events::Rule AWS 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 AWS CloudFormation 、 は イベントを作成できます。このリソースは AWS 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": [
            "",
            Ε
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
```

} ], "detail": { "event": [ "referenceCreated", "referenceUpdated" ], "referenceType": [ "branch" ], "referenceName": [ "main" ] } }, "Targets": [ { "Arn": { "Fn::Join": [ "", Ε "arn:aws:codepipeline:", { "Ref": "AWS::Region" }, ":", { "Ref": "AWS::AccountId" }, ":", { "Ref": "AppPipeline" } ] ] }, "RoleArn": { "Fn::GetAtt": [ "EventRole", "Arn" ] }, "Id": "codepipeline-AppPipeline" }

- (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、オーバーライドを設定し ます。
  - Source この例ではactionName、は、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る動的値です。
  - COMMIT\_ID この例ではrevisionType、は、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る動的値です。
  - この例の revisionValue、 < revisionValue> は、ソースイベント変数から派生しています。
  - BranchName およびの出力変数Value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、 AWS 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。
- テンプレートをアップロードし、AWS 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
    }
 ]
},
```

# CodeConnections 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 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、「<u>CodeCommit ソースアクションと EventBridge</u>」 を参照してください。

トピック

- Bitbucket Cloud への接続
- GitHub コネクション
- GitHub Enterprise Server 接続
- <u>GitLab.com への接続</u>
- 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

## 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および 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。

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 間で共有接続を使用できます AWS アカウント。「<u>別のアカウントと共有されている接続を使用する</u>」を参照してくださ い。

Note

#### 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、 「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドア</u> 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 <u>Bitbucket Cloud への</u> <u>接続を作成する (コンソール)</u> 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、<u>Bitbucket Cloud への接続を作成する (CLI)</u> 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 <u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアク</u> <u>ションの場合</u>)で使用し、<u>パイプラインを作成する (CLI)</u>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。[<u>接</u> 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。

Create Bitbucket connection	
Connection name	
	Connect to Bitbucket

[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。

Bitbucket connection	settings Info			
Connection name				
a-connection				
Bitbucket apps Bitbucket apps create a link for y	your connection with Bitbucket. To st	tart, ins	stall a new app and save this connection.	
Q		or	Install a new app	

- 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]を選択してください。作成された接続が接続リストに表示されます。

onnect to Bitbuck	et				
Bitbucket connection sett	ings Info				
Connection name					
MyConnection					
Bitbucket apps Bitbucket apps create a link for your cor	nection with Bitbucket. To	start, in	stall a new app and save t	his connection.	
Q ari:cloud:bitbucket::app/{c26	d1f3 - 🗙	or	Install a new app		
					Connect

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。
- [パイプライントリガー] で、アクションが CodeConnections アクションである場合は、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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」 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、<u>Bitbucket、GitHub、GitHub Enterprise Server、または</u> <u>GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。</u>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 を介して作成された接続 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 AWS 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

 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 --providertype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 --connectionname 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、「<u>Update a pending connection</u>」 を参照してください。
- 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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」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"
                                  1
                             }
                         }
                     ]
                }
            }
        1
```

## 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 間で共有接続を使用できます AWS アカウント。「<u>別のアカウントと共有されている接続を使用する</u>」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、 「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドア</u> <u>クションの場合</u>」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、GitHub (GitHub App 経由) 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 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、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、「<u>チュートリアル: CodeCommit</u> 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。[<u>接</u> <u>続を作成する]</u> 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。ソースページのソースプロバイダーで、GitHub (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 ダーで、GitHub (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。

Create a connection Info	
Create GitHub App connection	
Connection name githubc-connectid	
	Connect to GitHub

GitHub への接続を作成するには

- [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。

GitHub connection settings Info		
Connection name		
githubc-connection		
<b>SitHub Apps</b> SitHub Apps create a link for your connection with GitHub. To start, ir	istall a new app and save th	is connection.

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 アプリをすで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

<u>ユーザーアクセストークン</u>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。CodeConnections は、ユーザーアクセストークンを接続に使用します。

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。

- GitHub 用 AWS コネクタのインストールページで、デフォルトのままにして、インストールを 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]を選択してください。

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。
- [パイプライントリガー] で、アクションが CodeConnections アクションである場合は、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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」 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、<u>Bitbucket、GitHub、GitHub Enterprise Server、または</u> <u>GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。</u>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、<u>チュートリアル:</u> 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。

A Important

AWS CLI または を介して作成された接続 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 AWS 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

 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 --providertype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、「<u>Update a pending connection</u>」 を参照してください。
- 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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」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"
                                 1
                             }
                         }
                     ]
                }
```

]

## 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および 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。

(i) 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 間で共有接続を使用できます AWS アカウント。「<u>別のアカウントと共有されている接続を使用する</u>」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、 「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドア</u> <u>クションの場合</u>」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 <u>GitHub Enterprise Server への接続を作成する (コンソール)</u> 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、<u>GitHub Enterprise Server (CLI) への接続を作成します。</u>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 <u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアク</u> <u>ションの場合</u>)で使用し、<u>パイプラインを作成する (CLI)</u>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。[<u>接</u> 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 ヘ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。

GitHub Enterprise Server への接続を作成する (コンソール)

#### ▲ Important

AWS CodeConnections リリースで既知の問題があるため、 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。

- 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。



- [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。
- [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。
- [パイプライントリガー] で、アクションが CodeConnections アクションである場合は、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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」 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。

A Important

AWS CLI または を介して作成された接続 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 AWS 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

{

 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-host コマンド AWS CLI を実行し、接続--provider-endpoint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。

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

 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:codestarconnections: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。
- 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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」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および 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 間で共有接続を使用できます AWS アカウント。「<u>別のアカウントと共有されている接続を使用する</u>」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、 「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドア</u> <u>クションの場合)</u>」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 <u>GitLab.com への接続を作成する (コンソール)</u> 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、<u>GitLab.com への接続を作成する (CLI)</u> 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 <u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアク</u> <u>ションの場合</u>)で使用し、<u>パイプラインを作成する (CLI)</u>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。[<u>接</u> 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。

GitLab.com への接続
### 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。

 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。

Developer Tools > <u>Connections</u> > Create connection

Create a connection Info

Create GitLab connection Info	
Connection name	
Tags - optional	
	Connect to Citlish
	Connect to GitLab

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。

# Authorize codestar-connections to use your account?

An application called codestar-connections is requesting access to your GitLab account. This application was created by Amazon AWS. Please note that this application is not provided by GitLab and you should verify its authenticity before allowing access.

This application will be able to:

Access the authenticated user's API

Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.

• Read the authenticated user's personal information Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username,

public email, and full name. Also grants access to read-only API endpoints under /users.

Read Api

Grants read access to the API, including all groups and projects, the container registry, and the package registry.

- Allows read-only access to the repository Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- Allows read-write access to the repository Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).

Authorize

Deny

- 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。

 [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 groupname/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、<u>https://</u> <u>docs.gitlab.com/ee/api/projects.html#get-single-project</u> 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、<u>https://docs.gitlab.com/ee/user/</u> 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。

- [パイプライントリガー] で、アクションが CodeConnections アクションである場合は、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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」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、<u>Bitbucket、GitHub、GitHub Enterprise Server、または</u> <u>GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。</u>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、<u>チュートリアル:</u> 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 を介して作成された接続 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 AWS 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

 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 --providertype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"
}

- 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、「<u>Update a pending connection</u>」
   を参照してください。
- 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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」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。



# 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および 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 間で共有接続を使用できます AWS アカウント。「<u>別のアカウントと共有されている接続を使用する</u>」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、 「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドア</u> 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 <u>GitLab セルフマネージドへの接続を作成する (コンソール)</u> 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 <u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアク</u> <u>ションの場合</u>)で使用し、<u>パイプラインを作成する (CLI)</u>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、<u>https://</u> 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、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。

 スコープダウンされたアクセス許可: api のみで GitLab 個人アクセストークン (PAT) を作 成済みである必要があります。詳細については、<u>https://docs.gitlab.com/ee/user/profile/</u> <u>personal\_access\_tokens.html</u> 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、「<u>ホストの作成</u>」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)

以下のステップを使用して、CodePipeline コンソールで GitLab セルフマネージドリポジトリ用の接 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。
  - 接続で、プロバイダーへの接続を既に作成している場合は、接続を選択し、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

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。

1 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。
- 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。[個人アクセストークンを提供] で、範囲 を絞ったアクセス許可「api」のみを GitLab PAT に付与します。

Note PAT を作成して使用できるのは管理者のみです。

[Continue] (続行) を選択します。

et up myno	stgt				
Provide personal	access token				
To set up GitLab self-n to have the following s	nanaged, provide your j coped-down permissio	personal access toke ons only: api.	n from GitLab. The	personal access toke	n is required
				Cancel	Continu

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。
- [パイプライントリガー] で、アクションが CodeConnections アクションである場合は、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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」 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 を介して作成された接続 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 AWS 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

 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-host コマンド AWS CLI を実行し、接続--provider-endpoint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。

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

 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。
- 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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」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。



# 別のアカウントと共有されている接続を使用する

を使用して共有接続を作成および管理できます AWS RAM。これにより、サードパーティーのリポ ジトリ AWS アカウント にアクセスするために 間の接続を共有できます。これにより、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。

- ・ 設定の 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。[<u>接続を作成する</u>] を参照し てください。
- ・を使用してリソース共有を設定します AWS RAM。「<u>と接続を共有する AWS アカウント</u>」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の編集ページを使用して、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、「<u>CodeConnections を使用して</u> 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」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、「<u>CodeCommit ソースアクションと EventBridge</u>」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次のとおりです。
    - Git タグ

含めるまたは除外する

・ブランチ

含めるまたは除外する

• ブランチ + ファイルパス

含めるまたは除外する

- ・プルリクエスト
  - 有効なフィルターの組み合わせは次のとおりです。
    - ・ブランチ

含めるまたは除外する

• ブランチ + ファイルパス

含めるまたは除外する

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	真
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、「<u>構文での glob パターンの使用</u>」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、<u>4: 競合する 2 つのプッ</u>シュフィルタータイプを持つトリガーには、と が含まれません 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) トリガー設定が交差する場合、プルリクエストをマージすると 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可能性があります。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 ジは終了イベントです。詳細については、「<u>プロバイダー別のトリガーのプルリクエストイベン</u> ト」を参照してください。

# プロバイダー別の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、プロバイダー別の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		
Open - 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、Opened Git イベントが発生 します。	プルリクエス トを作成する と、Opened Git イベントが発生 します。	プルリクエス トを作成する と、Opened Git イベントが発生 します。	プルリクエス トを作成する と、Opened 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 ストリビジョン が公開されたと きにパイプライ ンをトリガーし ます。	更新を発行する と、更新された Git イベントが発 生します。	更新を発行する と、更新された Git イベントが発 生します。	更新を発行する と、更新された Git イベントが発 生します。	更新を発行する と、更新された Git イベントが発 生します。		
Closed - このオ プションは、ブ ランチ/ファイル	Bitbucket でプ ルリクエスト をマージする	プルリクエスト をマージまた は手動で閉じる	プルリクエスト をマージまた は手動で閉じる	プルリクエスト をマージまた は手動で閉じる		

	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			
パスのプルリク エストが閉じら れたときにパイ プラインをトリ ガーします。	と、クローズド 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、クローズド Git イベントは発 生しません。	と、クローズド Git イベントが発 生します。	と、クローズド Git イベントが発 生します。	と、クローズド 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、トリガー設定の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 オペレーションを使用します。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 オペレー ションを使用します。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、それらの間で AND オペレーションが使用されます。つまり、完 全一致のみがパイプラインを開始します。ブランチ名など、インクルード/除外内の名前は、OR オ ペレーションを使用します。1つにmainブランチが含まれ、1つにブランチを除外するなど、2つ のプッシュフィルターの間に競合がある場合、デフォルトは除外です。次のリストは、Git 設定オブ ジェクトの各部分のオペレーション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、「」を参 照してください<u>triggers</u>。 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 オペレーション)

プルリクエストなどの単一のフィルタータイプでは、フィルターを組み合わせることができ、これら のフィルターは AND オペレーションを使用します。つまり、完全一致のみがパイプラインを開始し ます。次の例は、2 つの異なるフィルター (filePaths と)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branches。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ます。これによ り、refs/heads/feature/triggers 指定されたブランチに影響がない場合でも、パイプライン が AND として開始されます。

次の例は、上記の設定のトリガーのイベントを示しています。このイベントは、ブランチはフィルタ リングできますが、ファイルパスはフィルタリングできません。

```
{
    "ref": "refs/heads/feature/triggers",
    ...
```

```
ユーザーガイド
```

```
"commits": [
    {
        ...
        "modified": [
           "src/Main.java"
        ]
        ...
    }
    ]
}
```

Example 2: 包含と除外は、それらの間で AND オペレーションを使用します

単一のプルリクエストイベントタイプのブランチなどのトリガーフィルターは、含める と除外する の間に AND オペレーションを使用します。これにより、複数のトリガーが同じパイプラインの実 行を開始するように設定できます。次の例は、プッシュイベントの設定オブジェクトに単一のフィ ルタータイプ (branches) を持つトリガー設定を示しています。Includes および Excludesオペ レーションは AND になります。つまり、ブランチが除外パターン (例feature-branchの など) と 一致する場合、 インクルードも一致しない限り、パイプラインはトリガーされません。[含める] パ ターンが一致すると (main ブランチの場合など)、パイプラインはトリガーされます。

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しても、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 (AND オペレーション) を含むトリガー

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 オペレーションを使用します。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を除外した 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 がLICENSE.txt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。Includes および Excludesオ

ペレーションは AND'd になり、競合はデフォルトで除外されます。たとえば、featurebranchブランチ (プルリクエストに含まれる) のプルリクエストイベントの場合、featurebranchブランチはプッシュイベントタイプから除外されるため、デフォルトは除外されます。

次の例では、

README MD ファイルパス (付属) のmainブランチ (付属) 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 シュしても、パイプラインはトリガーされません。

含まれているブランチで、README . MDファイルパス (含まれ ている) 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) を編集してもパイプラインはトリガーされません。

feature-branch ブランチでは、README.MDファイルパス (プッシュイ ベントに含まれる) のプルリクエストを閉じても、プッシュイベントタイ プでfeature-branchブランチを除外として指定するため、パイプライン はトリガーされません。そのため、競合はデフォルトで除外になります。

次の画像は、構成を示しています。

lit: Triggers				
For source action: Source				
Filters				
Push	0	Pull request	Granted	0
Exclude branches:		Include branche	s: feature-branch	
feature-branch, feature-b 🚥		Exclude branche	25:	
Include file paths: README.md		feature-branch-	2, main	
Evolution file method LICENCE but				

### 設定の JSON の例を次に示します。

```
"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": [
```



Example 4: 競合する 2 つのプッシュフィルタータイプを持つトリガーには、 と が含まれません

次の図は、 タグ release-1 (含まれる) でフィルタリングするように を指定す 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、ブランチをフィルタリングする main (含まれる) ように指定さ れ、feature\*ブランチへのプッシュを開始しない (含まれない) ように指定されます。

### 次の例では

 2 つのイベントタイプが AND'd になるため、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しても、パイプラインはトリガーされません。

> main ブランチ (2 番目のプッシュフィルターに付属) から リリースをプッシュすると、パイプラインが開始されます。

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、の包含と除外の設定を示しています。

Edit: Triggers					Cancel Done
For source action: Source					Remove
Filters					
Push Include tags: release-1	© ⊘ ×	Push Include branches: main* Exclude branches: feature*	0	+ Add filter	
		Ø	×		

設定の JSON の例を次に示します。



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ときにトリガーが設定されます

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1つのブランチを定義します。一方、 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。

Action execution details X Action name: Source Status: Succeeded					
Summary Inpu	nt Output				
Output artifact	SourceArtifact 🖪				
AuthorDate	2024-11-08T00:16:03Z				
AuthorDisplayName					
AuthorEmail	r@amazon.com				
Authorid					
BranchName	main				
CommitId	e87b1678ec5c50b2addf20f213cc7				
CommitMessage	Dockerfile created online with Bitbucket				
ConnectionArn	arn:aws:codestar-connections:us-west-2: connection/cdacd948-8633-4409-a4e f-				
FullRepositoryName	'dk-bitbucket-repo				
ProviderType	Bitbucket				
	Done				

# 次のアクション出力例は、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。

Action execution deta Action name: Source Statu:	ils ×						
Summary Input	Output						
Output artifact	SourceArtifact [						
AuthorDate	2024-10-23T19:12:43Z						
AuthorDisplayName							
AuthorEmail	@amazon.com						
Authorld	{c26d1f33-2013-46e8-b193 }						
CommitId	32b96d37d12c53680a16029cc3						
CommitMessage	README.md edited online with Bitbucket						
ConnectionArn	am:aws:codestar-connections:us-west-2: k:connection/cdacd948-8633-4409 -a4						
DestinationBranchName	feature-branch						
FullRepositoryName	/dk-bitbucket-repo						
ProviderType	Bitbucket						
PullRequestId	6						
PullRequestTitle	Feature branch 2						
SourceBranchName	feature-branch-2						
	Done						

# フィルターなしで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 などの CodePipeline で CodeStarSourceConnection アクションを使用する接続を持つソースアクションに対して設定できま す。

トリガーの追加 (コンソール)

1. にサインイン AWS Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。

- [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。
- [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。トリガーを追加するには、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でトリガーを 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BranchNameフィールドは単一のブランチを定義し、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、<u>5: デフォルトのアクション設定</u> BranchName が手動起動に使用されるときにトリガーが設定されます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)

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、「<u>プロバイダー別のトリガーのプルリクエストイベント</u>」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、「<u>プッシュおよびプルリクエストイベント</u> <u>タイプのフィルターを追加する (CLI)</u>」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 (コンソール)

- 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 (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 (AWS CloudFormation テ ンプレート)

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 (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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

- [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. Push の Filter type 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「<u>構文での glob パターンの使用</u>」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「<u>構文での glob パター</u> 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の場合は、プルリクエストを開く先のターゲット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「<u>構文での glob パ</u> 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. Push の Filter type 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「<u>構文での glob パターンの使用</u>」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「<u>構文での glob パター</u> 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の場合は、プルリクエストを開く先のターゲット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「<u>構文での glob パ</u> <u>ターンの使用</u>」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 トリガー設定を設定します。

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<u>「トリガー</u>」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 (AWS CloudFormation テンプレート)

でパイプラインリソースを更新 AWS 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<u>「トリ</u> 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、 AWS CloudFormation 「ユーザーガイド」の「2 <u>つのステージとトリガー設定を持つパイプライン</u>」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。

変更検出をオフにするトリガーの追加 (コンソール)

1. にサインイン AWS Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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 リガーを追加するには、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。
- 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、ステージを手動でまたは自動的に AWS CLI 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、「<u>ソースアクションと変更検出方法</u>」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。
- 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、「<u>変数リファレンス</u>」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

 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-pipelineexecution 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
```

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、<u>パイプラインを</u> <u>表示する (コンソール)</u>および<u>パイプラインの詳細と履歴を表示する (CLI)</u>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 (<u>https://console.aws.amazon.com/events/</u>) 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。

- 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、「<u>ルールのスケ</u> 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。

パイプライン ARN は、コンソールの [設定] に表示されます。「<u>パイプラインの ARN と</u> 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。

- 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

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。

スケジュールに基づいたパイプラインの開始

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 MyRu1e2 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

- 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
```

- 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。

```
{
    "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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "codepipeline:StartPipelineExecution"
        ],
            "Resource": [
               "arn:aws:codepipeline:us-west-2:80398EXAMPLE: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に EventBridge revisionValueの 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、、<u>Amazon ECR ソースアクションと EventBridge リソース</u>、<u>イベントに対</u> 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または 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<u>CodeCommit ソースアクションと</u> EventBridge。

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 AllowOverrideForS30bjectKey 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。
- 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。

<ul> <li>Source revisi</li> <li>A source revisi</li> </ul>	ion override on is the version wit	th all the changes to	your application code, o	r source artifact,
for the pipelin changes that y	e execution. Choose you want to run in th	the source revision, ne pipeline execution	or version of your source	e artifact, with the
Source				
Commit ID				
Q				

### 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

 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 れづいしお n Type の値は COMMIT\_ID | IMAGE\_DIGEST | S3\_OBJECT\_VERSION\_ID | S3\_OBJECT\_KEY です。

次の例では、codecommit-pipeline という名前のパイプラインを通じて、指定した変更の 実行を開始します。次のコマンドは、ソースアクション名を Source、リビジョンタイプを COMMIT\_ID に、コミット ID を 78a25c18755ccac3f2a9eec099dEXAMPLE に設定します。

```
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、<u>パイプラインを</u> <u>表示する (コンソール)</u>および<u>パイプラインの詳細と履歴を表示する (CLI)</u>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)、実行を停止するのを AWS CodePipeline 待ちます。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まで待たずに実行を 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。

Marning

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。

#### 1 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、<u>インバウンド実行を停止します (コンソール)。</u>を参照してくだ さい。

- 1. にサインイン AWS Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」で CodePipeline コンソールを開きます。
- 2. 次のいずれかを行います:

(i) 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。

Stop execution		×
Select execution Choose the pipeline execution you wan	t to stop.	
Choose a stop mode for the execu If you choose to stop and wait, and you state, you can choose to abandon. Stop and wait Wait until all in-progress actions are complete.	tion change your mind while your execution is still in a stop Stop and abandon Don't wait until the in- progress actions are complete. Warning: This option can lead to failed actions.	ping
Stop execution comments - option	nal	
Cancel Stop		

- 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。

⑥ 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、無効なステージ遷移段階で保留中の実行はありません。

1. にサインイン AWS Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。

- インバウンド実行を停止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。
  - [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-pipelineexecution 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 --pipelineexecution-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</mark> 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、無効なステージ遷移段階で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 stoppipeline-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:

インバウンド実行 (CLI) を停止します。

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ために、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。

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、「<u>パイプライン実行モードを設定または変更する</u>」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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。

eveloper Tools > CodePipeline > Pipelines > rbtest > Execution history         Execution history Info         Rerun       Stop execution					Release change	
٩						<1 > ⊚
	Execution ID	Status	Source revisions	Trigger	Started Duration	Completed
0	33bdf70c Rollback	⊘ Succeeded	<b>Source</b> – <u>73ae512c</u> : Added README.txt	-	Apr 16, 2024 2:51 AM 1 second (UTC-7:00)	Apr 16, 2024 2:51 AM (UTC-7:00)
0	3f658bd1 Rollback	⊘ Succeeded	<b>Source</b> – <u>73ae512c</u> : Added README.txt	-	Apr 16, 2024 2:16 AM 1 second (UTC-7:00)	Apr 16, 2024 2:16 AM (UTC-7:00)
0	4f47bed9	⊘ Succeeded	<b>Source</b> – <u>73ae512c</u> : Added README.txt	StartPipelineExec ution	Apr 16, 2024 2:14 AM 5 seconds (UTC-7:00)	Apr 16, 2024 2:14 AM (UTC-7:00)
0	eb7ebd36 Rollback	⊘ Succeeded	<b>Source</b> – <u>73ae512c</u> : Added README.txt	-	Apr 16, 2024 2:00 AM 1 second (UTC-7:00)	Apr 16, 2024 2:00 AM (UTC-7:00)

- 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。

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。

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、<u>インバウンドの実行(コンソール)を表示します。</u> 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、無効なステージ遷移段階で保留中の実行はありません。

1. にサインイン AWS Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

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:

インバウンドの実行(コンソール)を表示します。

• [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。

6793d5e5 Source: Edited documentdb_full_stack.yml	
Johnund execution ID: 29d5a641-4323-	View summary
Disable transition	
Approval Pending	
Pipeline execution ID: baf5ecbd-04d2	
Approval (1)	5
Manual approval	

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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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. 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。

<b>"Source" current revisions</b> These are the source artifact changes that started the current pipeline execution.	×
Source Amazon S3 version id: ZqY_zLkx	
4e49e3b7-7dcc-4d93-b91a-	
Done	2

# 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

- 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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。

- 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。
- 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。
- [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、<u>list-pipeline-executions</u> 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、<u>get-pipeline-execution</u> 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、getpipeline-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、<u>get-</u> 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、getpipeline-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

 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 <u>get-pipeline-state</u> 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。

 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-s83I-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-s831-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": [
                 ſ
                      "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",
                "IternalExecutionUrl": "IternalExecutionUrl": "IternalExecut
```

```
"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、「<u>パイプライン実行の仕組み</u>」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、「<u>PARALLEL モードから変更したパイプラインに、以</u> 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、「<u>PARALLEL モードのパイプラインは、QUEUED モードまたは</u> <u>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</u> <u>す。</u>」を参照してください。

A 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。

MyPipeline	♦ Notify ▼ Edit	Stop execution	Clone pipeline	Release change	
Pipeline type: V2 Execution mode: SUPERSEDED					
Source Succeeded Pipeline execution ID: <u>3ff0e57c-e595-407c-8668-</u>					
Source GitHub (Version 2)					
Succeeded - <u>1 minute ago</u> <u>77cc2e44</u>					Ø
View details					
77cc2e44 Z Source: Merge pull request #5 from /feature-bu	ranch (•••)				
Disable transition					
Build In progress					
Pipeline execution ID: <u>3ff0e57c-e595-407c-866</u> ?					
Build					

## 次のビューは、CodePipeline の QUEUED モードを示しています。

MyPipeline Edit Stop execution Clone pipeline Release change	
Pipeline type: V2 Execution mode: QUEUED	
Source Succeeded Pipeline execution ID: 100f7c0e-4545-485a-88ea-	
Source	
GitHub (Version 2) ☑ Succeeded - Just now 77cc2e44 ☑	0
View details         77cc2e44       Source: Merge pull request #5 from         /feature-branch	
Disable transition	]
Build In progress Pipeline execution ID: 100f7c0e-4545-485a-88ea-	
Build AWS CodeBuild	

- 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 モードパイプラインに追加すること はできません。

<ul> <li>Source Succeeded</li> <li>Source Sittlub (Version 2) [2]</li> <li>Succeeded - Just now View details</li> <li>Build In progress</li> </ul>		
Source GitHub (Version 2) [2] Succeeded - Just now View details Build In progress Build AWS CodeBuild On progress - Just now	Source Succeeded	
Gittlub (Version 2) [2] © Succeeded - Just now View details © Build In progress Build AWS CodeBuild @ In progress - Just now	Source	
Succeeded - Just now View details	GitHub (Version 2)	
View details     Image: Second Sec	Succeeded - Just now	
Build In progress          Build         AWS CodeBuild         In progress - Just now	View details	
Build In progress Build AWS CodeBuild In progress - Just now		
Build In progress          Build         AWS CodeBuild         In progress - Just now		
Build In progress Build AWS CodeBuild In progress - Just now		
Build In progress Build AWS CodeBuild In progress - Just now		
Build <u>AWS CodeBuild</u> in progress - <u>Just now</u>		
Build       AWS CodeBuild       In progress - Just now	) Build In progress	
AWS CodeBuild	) Build In progress	
○ In progress - <u>Just now</u>	) <b>Build</b> In progress Build	
	) Build In progress Build AWS CodeBuild	
View details	Build In progress Build <u>AWS CodeBuild</u>	

#### 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 始し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/SUPERSE D から QUEUED へ	<ul> <li>進行中のアクションが完了 すると、アクティブな実行 はキャンセルされます。</li> <li>保留中の実行はキャンセル されます。</li> </ul>	[キャンセル済み] などのパイ プライン状態は、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 SUPERSEDE D へ	<ul> <li>進行中のアクションが完了 すると、アクティブな実行 はキャンセルされます。</li> <li>保留中の実行はキャンセル されます。</li> </ul>	[キャンセル済み] などのパイ プライン状態は、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 ヘ/QUEUED から PARALLEL ヘ	<ul> <li>進行中のアクションが完了 すると、アクティブな実行 はキャンセルされます。</li> <li>保留中の実行はキャンセル されます。</li> </ul>	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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。

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。

 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",
            "type": "S3",
            "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"
    }
}
```

 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 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。

A 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。

アーティファクトが上書きされても、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、失敗したアクションがあ る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。

「<u>ステージ障害時の自動再試行を設定する</u>」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

Retry stage Retry failed actions	
build	
AWS CodeBuild	C
Succeeded - 12 minutes and	
Details	
View in AWS CodeBuild	
$\checkmark$	
Deploy	(
AWS CodeDeploy	
S Failed - Just now	
Details	
View in AWS CodeDeploy	

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。

 ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、<u>retry-stage-</u> <u>execution</u> 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" }

- 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 <u>get-pipeline-state</u> 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"
            }
        },
        ſ
            "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:

A 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、「<u>CodePipeline でパイプラインと詳細を表示する</u>」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

Add entry condition 🔻	Add success cond	ition 🔻	Add failure conc	lition	
+ Add action group					
Build	(i)				
AWS CodeBuild		+	- Add action		
	P Enable auto	omatic roll	oack on stage failure		
	Enable auto	omatic retr	y on stage failure 🗸		
+ Add action group	None				
utomated stage configuration	on: Enable auto	omatic retr	y on stage failure 🔺	Retry mode:	Retry failed stage

 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。

Source	
GitHub (Version 2)	
Succeeded - <u>1 minute ago</u>	
<u>d4002ba3</u>	
View details	
d4002ba3 🖸 Source: Making an update to Update README.md	
Disable transition	
Build Failed Auto retry attempt View retry metadata	
Start rollback Retry stage Retry failed acti	ions
Pipeline execution ID: <u>ee4b9da8-62b4-</u>	
Build	
AWS CodeBuild	
S Failed - <u>1 minute ago</u>	
View details	
d4002ba3 🖸 Source: Making an update to Update README.md	

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。

Retry stage metada	ta	×
Pipeline Execution Id	ee4b9da8-62b4-	
Latest Retry Trigger	AutomatedStageRetry	
Auto Retry Attempt	1	
		Done

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、障害発生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、 <u>パイプライン、ステージ、アクションを作成する</u>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。 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 (AWS CloudFormation)

AWS 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: SourceS30bjectKey
            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、「AWS CloudFormation ユーザーガイド」で「StageDeclaration」の「<u>OnFailure</u>」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。

「<u>ステージの自動ロールバックを設定する</u>」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。

Source <u>AWS CodeCommit</u> Succeeded - <u>Just now</u> <u>10cb9a83</u> View details		
10cb9a83 Source: update Disable transition		
deploys3 Succeeded  Pipeline execution ID: d1b8bf31-1d2f-41	<u>133-98f8-6a104fee1b4f</u>	Start rollback

- 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。



Roll	back to				Start rollback
	Execution ID	Source revisions	Started	Duration	Completed
0	07c0ff1d	<b>Source</b> – <u>04b7e235</u> : Edited README.txt	4 days ago	5 seconds	4 days ago
0	f15b38f7	<b>Source</b> – <u>f823f4a3</u> : Edited README.txt	5 days ago	5 seconds	5 days ago
0	4f47bed9	<b>Source</b> – <u>73ae512c</u> : Added README.txt	8 days ago	5 seconds	8 days ago
0	fcd61d8b	Source – <u>73ae512c</u> : Added README.txt	8 days ago	7 seconds	8 days ago

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、「<u>ステージを手動でロールバックする</u>」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)

 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-pipelineexecutions 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1ist-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。

 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、成功したパイプライン実行がパイプライン ステージ内に既に存在する場合にのみ、パイプラインステージが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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> <u>codepipeline/home</u> 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。

201

- 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。

Edit: deploys3			Cancel	Delete Done
+ Add action group	]			
s3deploy Amazon S3 ☑	١	+ Add action		
	∠ ×			
+ Add action group				
Configure automatic rol	lback on stage failure			

#### 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、 <u>パイプライン、ステージ、アクションを作成する</u>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<u>CodePipeline でパイプラインを編集す</u> <u>る</u>。

 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ロールバックを設定します。

```
{
                "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": "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 (AWS CloudFormation)

AWS 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、Release 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。

```
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: SourceS30bjectKey
            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、「AWS CloudFormation ユーザーガ イド」で「StageDeclaration」の「<u>OnFailure</u>」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。
| Exec | ution history        | Info        |   | Rerun Sto  | p execution                            | View details | Release change                         |
|------|----------------------|-------------|---|--|--|--------------|--|
| Q    | Ĩ                    |             |   |  |  |              | < 1 > 🔘                                |
|      | Execution<br>ID      | Status      | Source<br>revisions                           | Trigger  | Started                                | Duration     | Completed                              |
| 0    | 5cd064ca<br>Rollback | 🛞 Failed    | Source –<br>04b7e235:<br>Edited<br>README.txt | Automated<br>Rollback<br>FailedPipeli<br>neExecution<br>Id -<br>b2e77fa5 | Apr 24, 2024<br>12:19 PM<br>(UTC-7:00) | 1 second     | Apr 24, 2024<br>12:19 PM<br>(UTC-7:00) |
| 0    | b2e77fa5             | ⊗ Failed    | Source –<br><u>10cb9a83</u> :<br>update       | StartPipelin<br>eExecution   | Apr 24, 2024<br>12:19 PM<br>(UTC-7:00) | 5 seconds    | Apr 24, 2024<br>12:19 PM<br>(UTC-7:00) |
| 0    | 5efcfa68<br>Rollback | ⊘ Succeeded | Source –<br>04b7e235:<br>Edited<br>README.txt | ManualRoll<br>back -   | Apr 24, 2024<br>12:16 PM<br>(UTC-7:00) | 2 seconds    | Apr 24, 2024<br>12:16 PM<br>(UTC-7:00) |
| 0    | d1b8bf31             | ⊘ Succeeded | Source –<br><u>10cb9a83</u> :<br>update       | StartPipelin<br>eExecution   | Apr 24, 2024<br>12:14 PM<br>(UTC-7:00) | 6 seconds    | Apr 24, 2024<br>12:14 PM<br>(UTC-7:00) |

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。

Developer Tools > CodePipeline	> Pipelines > rbtest > Execut	tion history > 01ccf			
Pipeline executior	n: 01cc	Rerun Stop execution	A Previous execution	Next execution	>
Execution summary					
Status Succeeded	Started 1 hour ago	Completed 1 hour ago	Duration 1 second		
Trigger ManualRollback -	Ċ				
Latest action execution message Deployment Succeeded	2				
Pipeline execution ID  D 01ccf652-ab11-4d4b-898c-	9473ef8521ba				
Execution type ROLLBACK					
Target pipeline execution ID  f15b38f7-20bf-4c9e-94ed-2	2535ee02				0
Visualization Timeline	Variables Revisions				
O Source Didn't Run					
Source 3 AWS CodeCommit					
🕞 Didn't Run					
No executions yet					
⊘ deploys3 Succeeded				Start rollback	

# 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、<u>get-pipeline-</u> <u>execution</u> 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 --pipelineexecution-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"
        7
    }
}
```

#### 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。「<u>ステージの条件を設定する</u>」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「<u>条件</u>」、「<u>RuleTypeId</u>」、「<u>RuleExecution</u>」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。

入力条件の作成 - CloudWatchAlarm ルールの例 (コンソール)

- 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)

入力条件が満たされない場合にステージをスキップするようにステージの入力条件を設定できます。 条件が失敗すると、結果が適用されて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、「<u>ステージ条件に</u> 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。

dd entry condition ▼	Add	success condition <b>v</b>	Ac	ld failure condition
Before stage entry		Delete condition		
Result: Skip MyEntrySkipRule	<b>(</b> )			
AWS VariableCheck		+ Add rule		

- 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. Variable で、GitHub (GitHub App 経由) ソースアクションを持 つ#{SourceVariables.FullRepositoryName}パイプラインなど、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。

Reason: Job failed. Job I Intry conditions with Ski	99df30f7-634c-44ba-100 100 100 100 100 100 100 100 100 100	Kerlew
Deploy Skipped Pipeline execution ID: 1	b8096-35a1-	
Deploy	٥	
Amazon S3		

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。

cution ID: 082755	62-de97-456e-		
ndition type: Be	eforeEntry Result: SKIP Status: 😣 F	ailed	
Rule states	Rule Configuration		
Name	Rule Execution ID	Status	Reason
	2d28d804-20d3-4357-8ebe-	Failed	Job failed. Job ID d51899c9-44f4-499c-

### 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、<u>パイプライン、ステー</u> <u>ジ、アクションを作成する</u>「」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、Deploy という名前のステージに対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",
                "Extract": "false",
                "
                "actions": "1"
                "BucketName": "MyBucket",
                "Extract": "false",
                "
                "state
                "state
               "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                "state
                      "state
                "state
                "state
                "state
                "statee
                "statee
                "statee
                "statee
                "statee
                 "statee
```

```
"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「<u>SuccessConditions</u>」を参照してください。

# 入力条件の作成 (CFN)

AWS 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、「AWS CloudFormation ユーザーガイド」で 「StageDeclaration」の「<u>AWS::CodePipeline::Pipeline BeforeEntryConditions</u>」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。
- 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。

3deploy	0	
Imazon 53 🗹		+ Add action
	0×	Enable automatic rollback on stage failure
		Enable automatic retry on stage failure
+ Add action	group	None 🗸
itomated stage	configuration:	None

- 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、<u>パイプライン、ス</u> <u>テージ、アクションを作成する</u>「」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、Deploy という名前のステージに対して失敗時の条件を設定します。

```
"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": [
        ſ
            "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「<u>FailureConditions</u>」 を参照してください。

#### 失敗時の条件の作成 (CFN)

AWS 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。成功時に、ステージはルールを実行し、結果を適用します。

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、「AWS 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「<u>条件</u>」、「<u>RuleTypeId</u>」、「<u>RuleExecution</u>」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)

- 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。
- [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、「<u>cron 式と rate 式を使用</u>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。

View details	
Disable transition	
O Deploy () Succeeded	Start rollback
Pipeline execution ID: <u>8a6fe20b-9670-4a41-a5a8-be04d67</u>	<u>50d1c</u>
Amazon S3	
Succeeded - Just now	
View details	
d34def6d Source: Added simpleCov-report.json	
OnSuccess condition: 💮 In progress Execution ID:	8ae Override condition Review
Reason: Waiting for the time window to open. This is es	timated to happen in >1 week.

### 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、<u>パイプライン、ステー</u> <u>ジ、アクションを作成する</u>「」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<u>CodePipeline でパイプラインを編集する</u>。

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、MyDeploymentRule という名前のルールを使用し、Deploy という名前のステージに対して 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": "MyAlarmRule",
                         "ruleTypeId": {
                             "category": "Rule",
                             "owner": "AWS",
                             "provider": "CloudWatchAlarm",
                             "version": "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 「<u>SuccessConditions</u>」を参照してください。

#### 成功時の条件を作成する (CFN)

AWS CloudFormation を使用して成功時の条件を設定するには、 0nSuccess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、「AWS CloudFormation ユーザーガ イド」で「StageDeclaration」の「<u>OnFailure</u>」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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> 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。

Disable transition	
Deploy      In progress      Pipeline execution ID: <u>8a6fe20b-9670-4a41-a5a8-be04d6750d1c</u>	<b>Ø</b>
Deploy <u>Amazon S3</u> Succeeded - <u>3 minutes ago</u> <u>View details</u> <u>d34def6d</u> Source: Added simpleCov-report.json	
OnSuccess condition:       In progress       Execution ID:       8ad       Override condition       Review         Reason:       Waiting for the time window to open. This is estimated to happen in >1 week.       Review	

2. 上書きする条件の横にある [上書き] を選択します。

Disable transition		
Opploy Succeeded Pipeline execution ID: <u>8a6fe20b-9670-4a41-a5a8-be04d6750d1c</u>	Start rollback	0
Deploy <u>Amazon S3</u> <sup>[2]</sup> Succeeded - <u>5 minutes ago</u> View details		
d34def6d Source: Added simpleCov-report.json		
OnSuccess condition: V Overridden Execution ID: 8a6fe20b	Review	

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。

Condition execution	det	ails		
xecution ID: 8a6fe20b-9670-4	a41-a	5a8-be04d6750d1c		
Condition type: OnSucces	s Re	sult: FAIL Status: 🗸 Överridden		
lule states:				
Name		Rule Execution ID	Status	Reason
		20508000 b50d 4008 07fb 5047578056bb	Abandonod	

Done

# 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 に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、AWS CloudFormation またはコマンドラインを使用して追加できます。
  - AWS 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、「<u>ActionTypeDeclaration</u>」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。

アクションタイプをリクエストする

- 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"
      },
```

アクションタイプをリクエストする

```
"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:

 Attach an action type definition file in JSON format. This includes the schema for the action type.
 A list of test accounts for the allowlist which can access the new action type [{account, account\_name}]
 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%
 List of AWS ##### where your action will be available
 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。
- 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。
- [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。

- 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。手順はアクションのカテゴリ、ウェブサイト、および構成によって異なりますが、通常、 顧客を [アクションの編集]ページへ送り返す完了アクションが含まれます。
- 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し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、<u>パイプライン、ステージ、アクションを作成する</u> 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

 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>"
            ٦
        },
        "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。

 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。  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、<u>ステップ 2: アクションタイプ定義ファイルを作</u> 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構築、テスト、デプロイの設定に役立 っ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:



CodePipeline

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<u>CodePipeline の使用開始</u>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

 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、「<u>CodePipeline API リファレンス</u>」の「<u>ActionTypeSettings</u>」および 「<u>CreateCustomActionType</u>」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、「<u>CodePipeline パイプライン構造リファレンス</u>」を参照してくださ い。

- 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

 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}/"
    }
}
```

#### 1 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

- 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、「<u>Amazon EC2 Linux インス</u> 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

- 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。
- 「<u>一時的なアクセス権を付与するシナリオ</u>」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。
- 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
- 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、「<u>IAM ユーザーガイド</u>」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ときに、使用する次のいずれかを AWS アカウント で作成します。

ユーザーが の AWS 外部とやり取りする場合は、プログラムによるアクセスが必要です AWS Management Console。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マチック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。

プログラマチックアクセス権 を必要とするユーザー	目的	方法
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)	ー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 にを設定する AWS IAM Identity Center」を参照して ください。 ・ AWS SDKs、ツール、API については、SDK および AWS APIs「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	ー時的な認証情報を使用 して、 AWS CLI、 AWS SDKs、または AWS APIs。	「IAM <u>ユーザーガイド」の</u> <u>「 AWS リソースでの一時的</u> <u>な認証情報</u> の使用」の手順に 従います。
IAM	(非推奨)	使用するインターフェイスの 指示に従ってください。

プログラマチックアクセス権 を必要とするユーザー	目的	方法
	長期認証情報を使用して、 AWS CLI、AWS SDKs、また は AWS APIs。	<ul> <li>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。</li> <li>AWS SDKs「SDK とツー ルリファレンスガイド」の 「長期認証情報を使用した 認証」を参照してください。 AWS SDKs</li> <li>API AWS APIs「IAM ユー ザーガイド」の「IAM ユー ザーのアクセスキーの管 理」を参照してください。</li> </ul>

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。

```
{
  "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/"
      ]
    }
  ]
```

# } ③ 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。
- 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。
- 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。
- 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。
- 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<u>CodePipeline API</u> 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 に必要です 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 <u>PutObject</u> が暗号化を使用するよう設定する必要があります。現在、 AWS キー管理サービス (AWS KMS) のみが encryption. AWS KMS uses でサポートされています 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、<u>SDK を使用した Amazon S3 AWS Key Management Service での の指定 AWS SDKs</u>」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、<u>CodePipeline の概念</u>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。

 <u>CodePipeline のサンプルジョブワーカー</u>: 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。

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。

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": "MvApp"
             }
               ],
                "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、 updatepipeline 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、<u>リソースのタ</u> 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、「<u>CodePipeline でカスタムアクションを作成および追加</u> 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、「<u>AWS Command Line Interfaceのインストール</u>」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:accountid: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:accountid*: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:uswest-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:accountid: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

<u>AWS Lambda</u> 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。

#### A 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AWS 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 Labda 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<u>「 の制限</u>」を参照してくださ い。

A 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。

パイプラインで Lambda 関数を呼び出す

#### 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、

「<u>CodePipeline チュートリアル</u>」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

- <u>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)</u>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。
- 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。
- Amazon EC2 コンソールのインスタンスの [Details] タブで、[Public 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 1ambda : 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 Management Console し、<u>https://console.aws.amazon.com/iam/</u>:// www.com」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。

```
{
  "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。

[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。

[ポリシーの作成]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。
- [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 Management Console し、 AWS Lambda コンソールを <u>https://</u> console.aws.amazon.com/lambda/://www.com で開きます。
- 2. [関数]ページで、[関数の作成]を選択します。

In the second secon

[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。

- [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、<u>ジョブの詳細</u>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、<u>JSON イ</u> 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);
        }
    });
};
```

- [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. [Save]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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> <u>codepipeline/home://www.com」</u>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。
- [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。

 [+ 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<u>「プログラミングモデル</u>」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

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。

ステップ 4 : Lambda 関数でパイプラインをテストする

 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。

⊘ Source				View current re	evisions
Source CodeCommit Succeeded - 1 minute ago dbb74de	١				
Source: Added codepipeline-codecomn	nit-events.yn	nl dbb74de			
Disable transition					
⊘ Deploy				View current re	evisions
Deploy CodeDeploy Succeeded - 1 minute ago Details	١				
Source: Added codepipeline-codecomn	hit-events.yn	nl dbb74de			
Disable transition					
> LambdaStage				View current re	evisions
MyLambdaAction1	١	MyLambdaAction2	١	MyLambdaAction3	٤
In progress - 1 minute ago		In progress - 1 minute ago		In progress - 1 minute ago	

Lambda 関数の実験が完了したら、料金が発生する可能性を避けるために、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、<u>CodePipeline でパイプラインを編集する</u>、<u>CodePipeline でパイプラインを作成します。</u>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、<u>GetJobDetails API</u> 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": "MIICiTCCAfICCQD6m7oRw0uX0jANBgkghkiG9w
```
```
ØBAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTA1dBMRAwDgYDVQQHEwdTZ
 WF0dGx1MQ8wDQYDVQQKEwZBbWF6b24xFDASBqNVBAsTC01BTSBDb25zb2x1MRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTA1dBMRAwDgYDVQQHEwdTZWF0dGx1MQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC01BTSBDb25zb2x1MRIwEAYDVQQDEw1UZXN0Q21sYWMx
 HzAdBqkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wqZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEI03IyNoH/f0wYK8m9TrDHudUZq3qX4waLG5M43q7Wqc/MbQ
 ITx0USQv7c7ugFFDzQGBzZswY6786m86gpEIbb30hjZnzcvQAaRHhdlQWIMm2nr
 AqMBAAEwDQYJKoZIhvcNAQEFBQADqYEAtCu4nUhVVxYUntneD9+h8Mq9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZqXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp3780D8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 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 AWS 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。 AWS 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 AWS CloudFormation、、Amazon S3、および CodePipeline のA11owアクセス 許可が必要です。

```
{
    "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": "*"
        }
    ]
}
```

AWS 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、「<u>CodePipeline でパイプ</u> 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、このサンプルに独自の AWS CloudFormation テンプレートを置き換えることができます。独自のテンプレートを使用する場合は、 AWS 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."
```

- [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. [Save] を選択します。
- 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。

- スタックの名前
- AWS 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、「<u>CodePipeline でのマニュア</u> 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、<u>ARNs) と AWS のサービス 名前空間</u>」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。

アクセス権限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。

・ 以下のユーザーとグループ AWS IAM Identity Center:

アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「<u>権限設定を</u> 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「<u>サード</u> <u>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</u>」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「<u>IAM</u> 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。詳細については「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセ ス権限の追加」を参照してください。

#### 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 Management Console し、<u>https://console.aws.amazon.com/iam/</u>:// www.com」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]をクリックします。

```
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、[Visual] 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 visual 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. [ポリシーの作成]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 Management Console 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。

```
{
    "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 Management Console し、<u>https://console.aws.amazon.com/iam/</u>:// www.com」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</mark>リファレンス」を参照してください。
- 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。

サービスロールへの Amazon SNS アクセス許可の付与

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、[Visual] 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 visual 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 Management Console、テンプ レートで使用できる JSON または YAML ポリシー AWS 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 AWS CloudFormation、「 AWS CloudFormation ユーザーガイド」の<u>AWS Identity and Access Management 「リソースタイプのリファレンス</u>」 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。
- [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 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<u>AWS Identity and Access</u> Management 「リソースのタグ」を参照してください。
- 11. [Create Policy (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、<u>Amazon SNS アクセス権限を CodePipeline サービスロールに付与</u> 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、 AWS CLI 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。
- 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. [Save] 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、「<u>パイプライン、ステージ、アクションを作成す</u>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、「<u>CodePipeline パイプライン構造</u> 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、「<u>CodePipeline パイプライン構造</u> <u>リファレンス</u>」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。

Review Action name: Approval Status: Waiting for approval				
Details Revisions				
Trigger				
StartPipelineExecution - assumed-role/				
Comments about this action				
Comments for reviewer/approver				
URL for review				
https://review-url 🗹				
Decision				
• Approve Approving will resume the pipeline execution.				
<ul> <li>Reject</li> <li>Rejecting will stop the pipeline execution with a failed status.</li> </ul>				
Comments - optional  Preview markdown Learn more				
Comments from reviewer/approver				
Cancel Submit				

#### [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。

Review Action name:	Approval Status: Waiting for approval		×	
Details	Revisions			
Source Source				
Revision <u>2de8579a</u> I Add files via upload				
		Cancel	Submit	

- 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. [送信]を選択します。

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。

 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

 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"
                    }
                ſ
//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、<u>put-approval-result</u> 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://approvalstageapproved.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 [<u>Amazon SQS キューへの Amazon SNS</u> <u>メッセージの送信]</u> と [<u>Amazon SNS 通知を使った Lambda 関数の呼び出し]</u> 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。

{

手動の承認通知の JSON データ形式

```
"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 に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構築、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<u>AWS CodePipeline のクォータ</u>。

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 AWS CloudFormation を追加できます。

(i) 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 リージョンがあることに注意してください。

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、「<u>CodePipeline パイプライン構造リファ</u> 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は、アクショ ンの作成先のリージョンにデフォルトのアーティファクトバケットが CodePipeline によって設定さ れます。 AWS CLI、 AWS 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 AWS リージョンアクションと同じリー 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、<u>PutJobSuccessResult</u> と <u>PutJobFailureResult</u> 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。

パイプラインに含まれているクロスリージョンアクションがステージの一部である場合、 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、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含む新しいパイプ 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。

コンソールでパイプラインステージのクロスリージョンアクションを作成するには、アクション プロバイダーと、そのプロバイダーのリージョンで作成したリソースを一覧表示する [リージョン] フィールドを選択します。クロスリージョンアクションを追加すると、CodePipeline は、このアク 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、「<u>CodePipeline パイプライン構造リファレンス</u>」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 (<u>http://console.aws.amazon.com/codesuite/codepipeline/home</u>) 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。
- 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。
- [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. [Save]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. [Save]を選択します。

パイプラインステージの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 (<u>https://console.aws.amazon.com/codesuite/codepipeline/home.</u>) 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。
- [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. [Save] 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 (<u>http://console.aws.amazon.com/codesuite/codepipeline/home</u>) 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。
- [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. [Save]を選択します。

# パイプラインにクロスリージョン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しておく必要があります。単ー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。##### B は、CodeDeploy が使用する 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
- CodeDeploy アプリケーションや、リージョンをまたがるデプロイアクションのデプロイグループ など、アクションに必要なリソースが ##### B にあります。

パイプラインにクロスリージョンアクションを追加する (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

 ##### 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。

 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 アクションを関連付けた新しいステージを追加します。以下の JSON サンプルは、プロバイ ダーを CodeDeploy とするクロスリージョンデプロイアクションを関連付けたデプロイステー ジを、新しいリージョン us-east-1 に追加します。

```
{
               "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
                   }
```

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 マップを追加します。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、mystorage-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"
    }
},
```

 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": {

```
ユーザーガイド
```

```
"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。

A 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 startpipeline-execution コマンドを使用します。

6. パイプラインを更新したら、クロスリージョンのアクションはコンソールに表示されます。

500.00	(1)	
CodeCommit		
Succeeded - 29 days ago dbb74de1		
Source: Added codepipeline-codec	ommit-events.yml dbb74de1 🖸	
Disable transition		
⊙ Deploy		View current revisions
Deploy	0	
Cross-region action	÷	
Succeeded - 29 days ago		

## パイプラインにクロスリージョンアクションを追加する (AWS

## CloudFormation)

を使用して AWS CloudFormation 、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 AWS CloudFormation

 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:
Туре: Мар
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

 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 マップを以下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 eucentral-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
```

- 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、 AWS 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、 AWS 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 AWS 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<u>チュートリアル: AWS</u> 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、<u>例: 手動承認で変数を使用する</u> 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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。
- [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。
- 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。

Configuration		×
Variable namespace	SourceVariables	
Output artifact	SourceArtifact	
BranchName	main	
PollForSourceChanges	false	
RepositoryName	repo	
		Done

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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」で CodePipeline コンソールを開きます。
- 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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」で CodePipeline コンソールを開きます。
- 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

 「<u>パイプライン、ステージ、アクションを作成する</u>」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 --cliinput-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、<u>create-</u> 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

 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、<u>create-</u> 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 Management Console し、「https://<u>http://console.aws.amazon.com/</u> codesuite/codepipeline/home.com」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。
- 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。

Output variables		
Кеу	Value	
AuthorDate	2019-10-29T03:32:21Z	
BranchName	master	
CommitId	8cf40f22b935b306f06d214517e98aet	
CommitMessage	Added LICENSE.txt	
CommitterDate	2019-10-29T03:32:21Z	
RepositoryName	repo	

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。

Action configuration		Show resolved configuration
EnvironmentVariables [{"name":"Execution_ID","value":"# {codepipeline.PipelineExecutionId]","type":"PLAINTEXT"}, {"name":"Commit_ID","value":"#[SourceVariables.CommitId]","type":"PLAINTEXT"}]	ProjectName dk-var-build-proj	

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。

Action configuration		Show resolved configuration
EnvironmentVariables [{"name":"Execution_ID","value":"ab9f6ead-a64c-4fd5-b6aa- 3bf ","type":"PLAINTEXT"}, {"name":"Commit_ID","value":"8cf40f22b935b306f06d214517e <sup>4</sup> ","type" :"PLAINTEXT"}]	ProjectName var-build-proj	

### 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 output Variables 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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

- 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。
- [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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。

- 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。

例:CodeBuild 環境変数で BranchName 変数を使用する

Build logs Phase details Reports	Environment variables Build details	Resource utilization
Name	Value	Туре
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 Management Console し、<u>http://console.aws.amazon.com/codesuite/</u> codepipeline/home://www.com」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。
- 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。

ා Deploy	View current revisions
Deploy (j) CodeDeploy	
Succeeded - 15 minutes ago Details	7440
Disable transition	
<b>⊙</b> Production	View current revisions
Deploy 🔅	
CodeDeploy	
Succeeded - 14 minutes ago Details	

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。

Deploy	View current revisions
Deploy	
CodeDeploy	
Succeeded - 3 minutes ago Details	
ource: Added codepipeline-codecommit-events.yml dbb	74de
F Enable transition	
Production	View current revisions
Deploy (j	
CodeDeploy	
Succeeded - 2 minutes ago	
Details	
	<i>v</i> v

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

 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 --stagename Staging --transition-type Inbound --reason "My Reason"

このコマンドは何も返しません。

 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

 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 <u>enable-stage-transition</u> 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 --stagename Staging --transition-type Inbound

このコマンドは何も返しません。

 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。

# 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 CodePipeline に代わって行われた API コールをキャプチャし、ログファイルを Amazon S3 バ ケットに配信します。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
- <u>を使用した CodePipeline API コールのログ記録 AWS CloudTrail</u>
- <u>CodePipeline CloudWatch メトリクス</u>

# 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、softwareas-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「Amazon EventBridge とは」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、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す るよう、イベントソースとして CodePipeline を使用するイベントパターンでルールを作成するこ とが可能です。
- 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

- <u>詳細タイプ</u>
- 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、"detailtype": "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	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。
の状態変更	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。

AWS CodePipeline

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 能亦更	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。
悲変史	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	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。
<u> </u>	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 useast-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,
```

}

```
ユーザーガイド
```

```
"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。

1 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",
```
### 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,
```

```
ユーザーガイド
```

```
"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"
                }
            }
        1,
        "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",
```

```
ユーザーガイド
```

```
"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"
```



## 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
}
```

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。
- ナビゲーションペインで [Rules (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。
- 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

- 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):
  - 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 に与え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。
- 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、「<u>Amazon</u> 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

 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、「<u>ソースアクションと変更検出方法</u>」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 を使用して CloudWatch Events リソースを AWS CloudFormation セットアップした場合は、パイプラインのセットアップ時に指定したバケットに CloudTrail ファイルがあります。
- 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、AWS リージョン パイプラインが作成された のプレースホル ダーバケットに保存されます。

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、<u>CloudTrail ログファイルの取得と表示</u> 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-pla ceholder-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-pla ceholder-bucket-us-east-1	us-east-1
米国西部 (北カリフォルニア)	codepipeline-cloudtrail-pla ceholder-bucket-us-west-1	us-west-1
米国西部 (オレゴン)	codepipeline-cloudtrail-pla ceholder-bucket-us-west-2	us-west-2
カナダ (中部)	codepipeline-cloudtrail-pla ceholder-bucket-ca-central-1	ca-central-1
欧州 (フランクフルト)	codepipeline-cloudtrail-pla ceholder-bucket-eu-central-1	eu-central-1
欧州 (アイルランド)	codepipeline-cloudtrail-pla ceholder-bucket-eu-west-1	eu-west-1
欧州 (ロンドン)	codepipeline-cloudtrail-pla ceholder-bucket-eu-west-2	eu-west-2
欧州 (パリ)	codepipeline-cloudtrail-pla ceholder-bucket-eu-west-3	eu-west-3
欧州 (ストックホルム)	codepipeline-cloudtrail-pla ceholder-bucket-eu-north-1	eu-north-1
アジアパシフィック (香港)	codepipeline-cloudtrail-pla ceholder-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-pla ceholder-bucket-ap-south-2	ap-south-2
アジアパシフィック (ジャカル タ)	codepipeline-cloudtrail-pla ceholder-bucket-ap-southeas t-3	ap-southeast-3

AWS CodePipeline

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-pla ceholder-bucket-ap-southeas t-4	ap-southeast-4
アジアパシフィック (ムンバ イ)	codepipeline-cloudtrail-pla ceholder-bucket-ap-south-1	ap-south-1
アジアパシフィック (大阪)	codepipeline-cloudtrail-pla ceholder-bucket-ap-northeas t-3-prod	ap-northeast-3
アジアパシフィック (東京)	codepipeline-cloudtrail-pla ceholder-bucket-ap-northeas t-1	ap-northeast-1
アジアパシフィック (ソウル)	codepipeline-cloudtrail-pla ceholder-bucket-ap-northeas t-2	ap-northeast-2
アジアパシフィック (シンガ ポール)	codepipeline-cloudtrail-pla ceholder-bucket-ap-southeas t-1	ap-southeast-1
アジアパシフィック (シド ニー)	codepipeline-cloudtrail-pla ceholder-bucket-ap-southeas t-2	ap-southeast-2
アジアパシフィック (東京)	codepipeline-cloudtrail-pla ceholder-bucket-ap-northeas t-1	ap-northeast-1
カナダ (中部)	codepipeline-cloudtrail-pla ceholder-bucket-ca-central-1	ca-central-1
欧州 (フランクフルト)	codepipeline-cloudtrail-pla ceholder-bucket-eu-central-1	eu-central-1

AWS CodePipeline

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-pla ceholder-bucket-eu-west-1	eu-west-1	
欧州 (ロンドン)	codepipeline-cloudtrail-pla ceholder-bucket-eu-west-2	eu-west-2	
欧州 (ミラノ)	codepipeline-cloudtrail-pla ceholder-bucket-eu-south-1	eu-south-1	
欧州 (パリ)	codepipeline-cloudtrail-pla ceholder-bucket-eu-west-3	eu-west-3	
欧州 (スペイン)	codepipeline-cloudtrail-pla ceholder-bucket-eu-south-2	eu-south-2	
欧州 (ストックホルム)	codepipeline-cloudtrail-pla ceholder-bucket-eu-north-1	eu-north-1	
欧州 (チューリッヒ)*	codepipeline-cloudtrail-pla ceholder-bucket-eu-central-2	eu-central-2	
イスラエル (テルアビブ)	codepipeline-cloudtrail-pla ceholder-bucket-il-central-1	il-central-1	
中東 (バーレーン)*	codepipeline-cloudtrail-pla ceholder-bucket-me-south-1	me-south-1	
中東 (UAE)	codepipeline-cloudtrail-pla ceholder-bucket-me-central-1	me-central-1	
南米 (サンパウロ)	codepipeline-cloudtrail-pla ceholder-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、「<u>CloudTrailイベント履歴でのイベントの表示</u>」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。追跡により、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
- 「<u>複数のリージョンからCloudTrailログファイルを受け取る</u>」および「<u>複数のアカウントから</u> 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。イベントは任意ソースからの単一リクエストを表し、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の情報を含みます。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",

```
ユーザーガイド
```

```
"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 でメトリクスを使用して CodePipeline パイプラインを測定できます。次のメトリクス 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 でパイプラインで 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、アカウント内のすべてのパイプラインを対象としています。 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<u>「使用可能なメトリ</u> <u>クス</u>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個別のカウントが含まれ 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。たとえば、S3 ソースアクションが最初に失敗し、1 回実行さ 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

- パイプラインのエラー: AWS Elastic Beanstalk で設定されたパイプラインは次のようなエラーメッ セージを返します。「デプロイに失敗しました。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:「DescribeEvents」アクセス許可がない場合、AWS Elastic Beanstalk デプロイ アクションで設定したパイプラインは、失敗ではなくハングアップ状態になります。
- 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 1 つの 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したが AWS Elastic Beanstalk、アプリケーション URL は 404 が見つかりませんエラーを報告します
- <u>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</u>
- <u>Bitbucket、GitHub、GitHub Enterprise Server、または GitLab.com に接続するための CodeBuild</u> <u>GitClone アクセス許可を追加します。</u>
- ・ 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。
- <u>GitHub (OAuth アプリ経由) ソースアクション: リポジトリリストには異なるリポジトリが表示されます</u>
- <u>GitHub (GitHub App 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</u>
- <u>Amazon S3 エラー: CodePipeline サービスロール <ARN> により、S3 バケット <BucketName> に</u> 対する S3 アクセスが拒否されました。
- <u>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。</u>

- 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」
- 実行モードを QUEUED または PARALLEL モードに変更したパイプラインは、実行制限に達する と失敗します
- <u>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</u> 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
- <u>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</u> 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で失敗する No such file
- EKS デプロイアクションがcluster unreachableエラーメッセージで失敗する
- 別の問題があるため問い合わせ先を教えてください。

パイプラインのエラー: AWS Elastic Beanstalk で設定されたパイプ ラインは次のようなエラーメッセージを返します。「デプロイに 失敗しました。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 オペレーションを含むがこれに限定されない、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、「<u>CodePipeline サービスロールにアクセス許可を追加する</u>」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、<u>パイプラインを手動で開始する</u>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: 「DescribeEvents」アクセス許可がない場合、 AWS Elastic Beanstalk デプロイアクションで設定したパイプライ ンは、失敗ではなくハングアップ状態になります。

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、「<u>CodePipeline サービスロールにアクセス許可を追加する</u>」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、<u>パイプラインを手動で開始する</u>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、「<u>AWS の 管理ポリシー AWS CodePipeline</u>」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 1 つの 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: 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、パイプラインと バケットが異なる AWS リージョンで作成されている場合に失敗します。

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 ロイされましたが AWS Elastic Beanstalk、アプリケーション URL は 404 が見つかりません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 アクションで AWS CodeStar 接続を使用する場合、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。
- Git 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。

Git 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することができま 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、UseConnection のアク セス許可が action フィールドに指定され、接続 ARN が Resource フィールドに指定されたポリ シーを作成します。

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 を確認するには、パイプラインを開き、ソースアクションの (i) のア イコンをクリックします。CodeBuild サービスロールポリシーに接続 ARN を追加します。

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

 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。

- 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 [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。

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、Resource フィール ドに接続 ARN を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "codestar-connections:UseConnection",
            "Resource": "insert connection ARN here"
        }
    ]
}
```

[JSON] タブで、ポリシーを貼り付けます。

- [ポリシーの確認] 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。
- 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。

Atta Crea	Ch ate p	Permis	SIONS			2
Filter	r pol	icies ~	Q connection			Showing 1 result
		Policy na	ame 👻	Ту	pe	Used as
	•	conn	ection-permissions	Cu	ustomer managed	None

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

- 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。
- 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 (CodeDep1oyToECS アクション) に必要なアーティファクトです。CodeDep1oyToECS 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。そのファイルが既に存在する場合は、圧縮サイズが 3 MB 未満であることを確認してください。

# GitHub (OAuth アプリ経由) ソースアクション: リポジトリリストに は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 App 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、<u>https://docs.github.com/en/</u> <u>free-pro-team@latest/github/setting-up-and-managing-organizations-and-teams/permission-levels-</u> 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手順については、「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。 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

- 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。 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。
- [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。

```
{
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:ListBucket".
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890",
            "Condition": {
                "StringLike": {
                    "aws:userid": "AROAEXAMPLEID:*"
                }
            }
        },
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
```

```
"Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "StringNotEquals": {
                     "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "Bool": {
                     "aws:SecureTransport": false
                }
            }
        }
    ]
}
```

詳細については、<u>https://aws.amazon.com/blogs/security/writing-iam-policies-how-to-grant-</u> access-to-an-amazon-s3-bucket/のステップ を参照してください。

5. [Save] を選択します。

編集済みのポリシーを適用したら、<u>パイプラインを手動で開始する</u> 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。releasebranch-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 AWS CloudFormation を使用して、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、 「<u>EventBridge と を使用する Amazon S3 ソースアクションへの接続 AWS CloudTrail</u>」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、「<u>Amazon</u> 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、「<u>CodeCommit ソースアクションと EventBridge</u>」 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

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

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。

実行モードを QUEUED または PARALLEL モードに変更したパイ 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、「<u>CodePipeline の概念</u>」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 モードの更 新時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、「<u>CodePipeline の概念</u>」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、「<u>CodePipeline の概念</u>」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 パイプラインは、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、開始しないことがあり ます。これは、CodeConnections 接続では、変更されたファイルを解決できないためです。ファイ ルパスでフィルタリングするようにトリガーの Git 設定をセットアップした場合、パイプラインは、 ソースリポジトリでのフィルターを含むブランチの作成直後に開始しません。ファイルパスでのフィ ルタリングの詳細については、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを</u> 追加する」を参照してください。

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 ファイルまで取得します。したがって、 ファイルパスでフィルタリングするようにトリガーの Git 設定をセットアップした場合、ファイル数 が 100 を超えると、パイプラインは開始し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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」 を参照してください。

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で失敗する No such file

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、そのインスタンスでアクションは失敗し、残り のインスタンスはデプロイに失敗します。

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 であるデプロ イのログに、次のようなエラーメッセージが表示されますmyRepo/postScript.sh。

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:デプロイアクションはパイプラインで失敗します。

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。
- 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。
- 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cluster unreachableエラーメッ セージで失敗する

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:デプロイアクションはパイプラインで失敗します。

解決方法: 既存のロールを使用する場合、CodePipeline 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 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。

- CodePipeline 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、「」を参照してください<u>サービスロールのポ</u> リシーのアクセス許可。
- 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 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。

## 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 織の要件を満たすために built である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 とお客様の間の責任共有です。<u>責任共有モデル</u>では、これをクラウドのセ キュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ AWS は、AWS のサービス で実行され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、安全に使用できるサービス AWS も提供しま す。「AWS」 コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセ キュリティの有効性をテストおよび検証しています。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、AWS のサービス「コンプライアンスプログラムによる対象 範囲内」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 によって決まり AWS のサービス 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や規制といった他の要因 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と 保護 AWS のサービス に役立つ他の 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
- AWS CodePipelineのためのアイデンティティおよびアクセス管理
- CodePipeline でのロギングとモニタリング
- <u>のコンプライアンス検証 AWS CodePipeline</u>
- 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
- セキュリティに関するベストプラクティス

## でのデータ保護 AWS CodePipeline

責任 AWS <u>共有モデル</u>、 でのデータ保護に適用されます AWS CodePipeline。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、 <u>データプライバシー</u> に関するよくある質問を参照してください。欧州でのデータ保護の詳細については、AWS セキュリ ティブログに投稿された AWS 責任共有モデルおよび GDPR のブログ記事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 Identity Center 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、それぞれのジョブを遂行するために必要な権限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 が必須で、TLS 1.3 をお勧めします。
- 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 CodePipeline AWS CLIまたは他の AWS のサービス を使用する場合も同様で す。 AWS SDKs タグ、または名前に使用される自由記述のテキストフィールドに入力したデータ は、請求または診断ログに使用される場合があります。外部サーバーに URL を提供する場合、そ のサーバーへのリクエストを検証できるように、認証情報を URL に含めないことを強くお勧めしま す。 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
- <u>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</u> <u>跡する</u>

## インターネット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。 っまり、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、「<u>CodePipeline 用に Amazon S3 に保存したアー</u> <u>ティファクトのサーバー側の暗号化を設定する</u>」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ローテー ションできます。

#### A 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。

A 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ロー 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。

```
{
    "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、<u>「サーバー側の暗号化を使用したデータの保護</u>」 および<u>「 (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</u>」を参照してください。 詳細については AWS KMS、「 <u>AWS Key Management Service デベロッパーガイド</u>」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
- AWS 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 Management Console 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。
- 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、「<u>AWS</u> 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。

AWS 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

AWS 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パイプラインを CodePipeline で作成する」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、「<u>キーローテーション</u>」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、<u>AWS 「 Secrets Manager ユーザーガイド」の「Secrets Manager とは</u>AWS 」を参 照してください。

AWS 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、「<u>Secrets Manager のシー</u> <u>クレット</u>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、「<u>Webhook リソースの設</u> 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 AWS CloudFormation テンプレートに追加する方法について は、<u>https://aws.amazon.com/blogs/security/how-to-create-and-retrieve-secrets-managed-in-aws-</u> secrets-manager-using-aws-cloudformation-template/ を参照してください。

# AWS CodePipelineのためのアイデンティティおよびアクセス管理

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

- 対象者
- 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の管理
- 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
- <u>CodePipeline 許可リファレンス</u>
- CodePipeline サービスロールを管理する

## 対象者

AWS Identity and Access Management (IAM) の使用方法は、CodePipeline で行う作業によって異なります。

[Service user] (サービスユーザー) - CodePipeline サービスを使用してジョブを実行する場合は、必要なアクセス許可と認証情報を管理者が用意します。さらに多くの CodePipeline 機能を使用して作業を行う場合は、追加のアクセス許可が必要になることがあります。アクセスの管理方法を理解すると、管理者に適切なアクセス許可をリクエストするのに役に立ちます。CodePipeline の機能にアクセスできない場合は、AWS CodePipeline ID とアクセスのトラブルシューティング を参照してください。

サービス管理者 - 社内の CodePipeline リソースを担当している場合は、通常、CodePipeline への フルアクセスがあります。サービスのユーザーが CodePipeline のどの機能やリソースにアクセスす るかを決めるのは管理者の仕事です。その後、IAM 管理者にリクエストを送信して、サービスユー ザーの権限を変更する必要があります。このページの情報を点検して、IAM の基本概念を理解して ください。会社で CodePipeline を使用して IAM を利用する方法の詳細については、<u>が IAM と AWS</u> CodePipeline 連携する方法 を参照してください。

IAM 管理者 - 管理者は、CodePipeline へのアクセスを管理するポリシーの作成方法の詳細について 確認する場合があります。IAM で使用できる CodePipeline アイデンティティベースのポリシーの例 を表示するには、AWS CodePipeline アイデンティティベースポリシーの例 を参照してください。

## 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。として、IAM ユーザーとして 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によって、認証(にサイン イン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインイ ンできます。 AWS IAM Identity Center (IAM Identity Center) ユーザー、会社のシングルサインオン 認証、Google または Facebook 認証情報は、フェデレーション ID の例です。フェデレーティッド ID としてサインインする場合、IAM ロールを使用して、前もって管理者により ID フェデレーション が設定されています。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引 き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインイ ンできます。へのサインインの詳細については AWS、「 AWS サインイン ユーザーガイド<u>」の「 へ</u> のサインイン AWS アカウント方法」を参照してください。 AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインイ ンターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストを暗号化して署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。リクエストに自分 で署名する推奨方法の使用については、「IAM ユーザーガイド」の「<u>API リクエストに対するAWS</u> Signature Version 4」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。たと えば、 では、アカウントのセキュリティを高めるために多要素認証 (MFA) を使用する AWS ことを お勧めします。詳細については、「AWS IAM Identity Center ユーザーガイド」の「<u>多要素認証</u>」お よび「IAM ユーザーガイド」の「IAM のAWS 多要素認証」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、アカウント内のすべての およびリソースへの AWS のサービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウ ント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサイ ンインすることでアクセスできます。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。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実 行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストに ついては、「IAM ユーザーガイド」の「<u>ルートユーザー認証情報が必要なタスク</u>」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、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内の ID です。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を保有する IAM ユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAM ユーザーでの長期的な認証情報が必要な特定のユースケースがある場合は、アクセスキーをローテーション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「長期的な認証情報を必要とするユースケースのためにアクセスキーを定期的にローテーションする」を参照してください。

IAM グループは、IAM ユーザーの集団を指定するアイデンティティです。グループとしてサインイ ンすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できま す。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。 例えば、IAMAdmins という名前のグループを設定して、そのグループに IAM リソースを管理する許 可を与えることができます。

アイデンティティを使用した認証

ユーザーは、ロールとは異なります。ユーザーは1人の人または1つのアプリケーションに一意に 関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユー ザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。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 AWS アカウント を持つ 内の ID です。これは IAM ユーザーに 似ていますが、特定のユーザーには関連付けられていません。で IAM ロールを一時的に引き受ける には AWS Management Console、ユーザーから IAM ロール (コンソール) に切り替える ことができ ます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタ ム URL を使用します。ロールを使用する方法の詳細については、「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 ロールと一時的な認証情報は、次の状況で役立ちます:

- フェデレーションユーザーアクセス フェデレーティッド ID に許可を割り当てるには、ロール を作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID は ロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロール 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 Identity Center を使用する場合は、許可セッ トを設定します。アイデンティティが認証後にアクセスできるものを制御するため、IAM Identity Center は、権限セットを IAM のロールに関連付けます。アクセス許可セットの詳細については、 「AWS IAM Identity Center User Guide」の「Permission sets」を参照してください。
- ・一時的な IAM ユーザー権限 IAM ユーザーまたはロールは、特定のタスクに対して複数の異なる
   権限を一時的に IAM ロールで引き受けることができます。
- クロスアカウントアクセス IAM ロールを使用して、自分のアカウントのリソースにアクセスすることを、別のアカウントの人物 (信頼済みプリンシパル) に許可できます。クロスアカウントアクセス権を付与する主な方法は、ロールを使用することです。ただし、一部の では AWS のサービス、(プロキシとしてロールを使用する代わりに) リソースに直接ポリシーをアタッチできます。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。
- クロスサービスアクセス 一部の は他の の機能 AWS のサービス を使用します AWS のサービ ス。例えば、あるサービスで呼び出しを行うと、通常そのサービスによって Amazon EC2 でアプ リケーションが実行されたり、Amazon S3 にオブジェクトが保存されたりします。サービスで

アイデンティティを使用した認証

は、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこ れを行う場合があります。

- 転送アクセスセッション (FAS) IAM ユーザーまたはロールを使用してアクションを実行する AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行する ことで、別のサービスの別のアクションがトリガーされることがあります。FAS は、を呼び出 すプリンシパルのアクセス許可を AWS のサービス、ダウンストリームサービス AWS のサービ ス へのリクエストをリクエストする と組み合わせて使用します。FAS リクエストは、サービス が他の AWS のサービス またはリソースとのやり取りを完了する必要があるリクエストを受け 取った場合にのみ行われます。この場合、両方のアクションを実行するためのアクセス許可が必 要です。FAS リクエストを行う際のポリシーの詳細については、「<u>転送アクセスセッション</u>」 を参照してください。
- サービスロール サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、IAM 内からサービスロールを作成、変更、削除することができま す。詳細については、「IAM ユーザーガイド」の「AWS のサービスに許可を委任するロールを 作成する」を参照してください。
- サービスにリンクされたロール サービスにリンクされたロールは、にリンクされたサービス ロールの一種です AWS のサービス。サービスは、ユーザーに代わってアクションを実行する ロールを引き受けることができます。サービスにリンクされたロールはに表示され AWS アカ ウント、サービスによって所有されます。IAM 管理者は、サービスリンクロールのアクセス許 可を表示できますが、編集することはできません。
- Amazon EC2 で実行されているアプリケーション IAM ロールを使用して、EC2 インスタンスで 実行され、AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を 管理できます。これは、EC2 インスタンス内でのアクセスキーの保存に推奨されます。EC2 イン スタンスに AWS ロールを割り当て、そのすべてのアプリケーションで使用できるようにするに は、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロ ファイルにはロールが含まれ、EC2 インスタンスで実行されるプログラムは一時的な認証情報を 取得できます。詳細については、「IAM ユーザーガイド」の「<u>Amazon EC2 インスタンスで実行</u> されるアプリケーションに IAM ロールを使用して許可を付与する」を参照してください。

## ポリシーを使用したアクセスの管理

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 AWS 、アイデンティティまたはリソースに関連付けられているときにアクセス許可を 定義する のオブジェクトです。 は、プリンシパル (ユーザー、ルートユーザー、またはロールセッ ション) 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。ポリシーでの権限によ り、リクエストが許可されるか拒否されるかが決まります。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構造と内容の詳細については、 「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 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。

デフォルトでは、ユーザーやロールに権限はありません。IAM 管理者は、リソースで必要なアク ションを実行するための権限をユーザーに付与する IAM ポリシーを作成できます。その後、管理者 はロールに IAM ポリシーを追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの許可を定義します。例え ば、iam:GetRole アクションを許可するポリシーがあるとします。そのポリシーを持つユーザー は、 AWS Management Console、、 AWS CLIまたは AWS API からロール情報を取得できます。

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ポリシーは、IAM ユーザーグループ、ユーザーのグループ、ロールなど、 アイデンティティにアタッチできる JSON 許可ポリシードキュメントです。これらのポリシーは、 ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。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「<u>カスタマー管理ポリ</u> 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、さらにインラインポリシーまたはマネージドポリシーに分類 できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれてい ます。管理ポリシーは、内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロン ポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシー が含まれます。マネージドポリシーまたは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「<u>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</u>」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、リソースに添付する JSON ポリシードキュメントです。リソース ベースのポリシーには例として、IAM ロールの信頼ポリシーや Amazon S3 バケットポリシーがあげ られます。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。ポリシーがアタッチされているリソースの 場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーに よって定義されます。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。プ リンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含める ことができます AWS のサービス。

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、一般的でない追加のポリシータイプをサポートしています。これらのポリシータイプで は、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 アクセス許可の境界は、アイデンティティベースポリシーによって IAM エンティティ (IAM ユーザーまたはロール) に付与できる権限の上限を設定する高度な機能です。エンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。Principalフィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「IAM ユーザーガイド」の「IAM エンティティのアクセス許可の境界」を参照してください。
- サービスコントロールポリシー (SCPs) SCPsは、の組織または組織単位 (OU) の最大アクセス 許可を指定する JSON ポリシーです AWS Organizations。 AWS Organizations は、ビジネスが所 有する複数の AWS アカウント をグループ化して一元管理するためのサービスです。組織内のす べての機能を有効にすると、サービスコントロールポリシー (SCP) を一部またはすべてのアカウ ントに適用できます。SCP は、各 を含むメンバーアカウントのエンティティのアクセス許可を制 限します AWS アカウントのルートユーザー。Organizations と SCP の詳細については、「AWS Organizations ユーザーガイド」の「<u>サービスコントロールポリシー (SCP)</u>」を参照してくださ い。
- リソースコントロールポリシー (RCP) RCP は、所有する各リソースにアタッチされた IAM ポリ シーを更新することなく、アカウント内のリソースに利用可能な最大数のアクセス許可を設定する ために使用できる JSON ポリシーです。RCP は、メンバーアカウントのリソースのアクセス許可 を制限し、組織に属しているかどうかにかかわらず AWS アカウントのルートユーザー、を含む ID の有効なアクセス許可に影響を与える可能性があります。RCP をサポートする のリストを含む Organizations と RCP の詳細については、AWS Organizations RCPs<u>「リソースコントロールポリ</u> シー (RCPs」を参照してください。AWS のサービス
- セッションポリシー セッションポリシーは、ロールまたはフェデレーションユーザーの一時的な セッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果として セッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポ

リシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もありま す。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。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、「IAM ユーザーガイド」の「IAM <u>AWS のサービス と連携する</u>」 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 要素にはポリシー内のアクセスを許可または拒否するために使用できる アクションが記述されます。ポリシーアクションの名前は通常、関連付けられた AWS API オペレー ションと同じです。一致する API オペレーションのない許可のみのアクションなど、いくつかの例 外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追 加アクションは依存アクションと呼ばれます。

このアクションは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ポリシー で使用されます。 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、<u>IAM ユーザーガイド</u>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が適用されるオブジェクトを指定します。ステートメ ントにはResource または NotResource 要素を含める必要があります。ベストプラクティスとし て、<u>アマゾン リソースネーム (ARN)</u>を使用してリソースを指定します。これは、リソースレベルの 許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限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": "\*"

CodePipeline のリソースとオペレーション

CodePipeline 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。CodePipeline 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に対応していま 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「Amazon リソースネーム (ARN) と AWS <u>のサービス 名前空間</u>」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	<pre>arn:aws:codepipeline:region:account:pipeline-name /stage- name /action-name</pre>
カスタムアクション	arn:aws:codepipeline: <i>region:account</i> :actionty pe: <i>owner/category/provider/version</i>
全ての CodePipeline リ ソース	arn:aws:codepipeline:*
特定のリージョンの特 定アカウントが所有す るすべての CodePipel ine 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、CodePipeline では、イベントパターンおよびルールは完全に一致し

ています。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。

CodePipeline には、リソースレベルのアクセス許可に対応する API コールの機能があります。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、CodePipeline 許可リファレンス 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 CodePipeline API コールは、複数のリソース (例:GetPipeline) を受け入れます。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"]

が IAM と AWS CodePipeline 連携する方法

API バージョン 2015-07-09 935

CodePipeline には、CodePipeline リソースを操作できる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、「<u>CodePipeline 許可リファレンス</u>」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 (または Condition ブロック) を使用すると、ステートメントが有効な条件を指定 できます。Condition 要素はオプションです。イコールや未満などの <u>条件演算子</u> 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。

1つのステートメントに複数の Condition 要素を指定する場合、または 1 つの Condition 要素に 複数のキーを指定する場合、 AWS では AND 論理演算子を使用してそれらを評価します。1 つの条 件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ス テートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば IAM ユーザーに、IAM ユーザー 名がタグ付けされている場合のみリソースにアクセスできる権限を付与することができます。詳細 については、「IAM ユーザーガイド」の「<u>IAM ポリシーの要素: 変数およびタグ</u>」を参照してくださ い。

AWS は、グローバル条件キーとサービス固有の条件キー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<u>AWS 「グローバル条件コンテキスト</u> <u>キー</u>」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<u>AWS 「グローバル</u> 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、「<u>例: 特定のリージョンへのアクセスの制限</u>」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、<u>IAM ユーザーガイド</u>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、<u>「で定義され</u> 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、<u>AWS CodePipeline アイデンティ</u> ティベースポリシーの例 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、<u>AWS CodePipeline リソースベースの</u> 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 <u>条件要素</u> 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、<u>リソース</u> 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、「<u>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</u>」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

ー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。

#### サービス役割

CodePipeline は、<u>サービスロール</u>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。サービス役割は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 Management Console、 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「<u>JSON タブでのポリシーの作成</u>」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、「<u>別の AWS アカウントのリソースを使用するパイプラインを CodePipeline で作成す</u> <u>る</u>」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
- CodePipeline コンソールを使用するために必要なアクセス許可
- CodePipeline 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を実行すると、 AWS アカウントに 料金が発生する可能性があります。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「<u>AWS マネージドポリシー</u>」または「<u>ジョブ機</u> 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「<u>IAM でのポリシーとアクセス許可</u>」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 AWS 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 Access Analyzer を使用して IAM ポリシーを検証し、安全で機能的な権限を確保する IAM Access Analyzer 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するようにします。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「<u>IAM Access Analyzer でポリシーを</u> 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「<u>IAM でのセキュリ</u> 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

CodePipeline 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: ビルドプロジェクト
- 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。

- 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、次の操作を実行できます。

- アカウントのユーザーまたはグループにアクセス権限ポリシーをアタッチする CodePipeline コ ンソールのパイプラインを表示するアクセス権限を付与するために、ユーザーが所属するユーザー またはグループにアクセス許可の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。
- 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。
- 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「<u>アクセス管理</u>」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。

```
{
    "Version": "2012-10-17",
    "Statement" : [
    {
        "Effect" : "Allow",
        "Action" : [
            "codepipeline:EnableStageTransition",
            "codepipeline:DisableStageTransition"
        ],
        "Resource" : [
            "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/*"
        ]
    }
]
```

以下の例では、アカウント (111222333444) のポリシーを示します。このポリシーでは、ユーザー は CodePipeline コンソールのパイプライン MyFirstPipeline 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、CodePipeline 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、「<u>マネージドポリシーの使用</u>」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。

```
{
  "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": {
        "StringLike": {
          "codestar-notifications:NotificationsForResource": "arn:aws:codepipeline:*"
        }
      }
    }
  ],
  "Version": "2012-10-17"
}
```

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。

```
{
    "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。

{			
	"Ve	ersio	n": "2012-10-17",
	"St	atem	ent": [
		{	
			"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 ロールを作成する方法、CodePipeline の IAM ポリシーステートメントの例を調べる方法の詳細については、「<u>カスタマーマネージドポリ</u> シーの例」を参照してください。

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、<u>リソースのタグ付け</u>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、「<u>タグを使用したアクセスコントロール</u>」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 と連携する
- 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。

```
{
  "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。

```
{
  "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。

```
{
  "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。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codepipeline:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": ["Project"]
        }
      }
    }
  ]
}
```

CodePipeline コンソールを使用するために必要なアクセス許可

CodePipeline コンソールで CodePipeline 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。

アイデンティティベースのポリシーの例

- AWS CodeCommit
- CodeBuild
- AWS 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 CodePipeline コ ンソールを使用できること、および <u>AWS の 管理ポリシー AWS CodePipeline</u> マネージドポリシー 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 CodePipeline 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。

CodePipeline 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提供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<u>カスタ</u> 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。

A 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 プリンシパルがチャットアプリケーションで Amazon Q Developer 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。
- 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 AWS 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。
- 1ambda 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 プリンシパルにステートマシンの表示を許可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。

{ {	
"Statement": [	
{	
"Action": [	
"codepipeline:*".	
"cloudformation:DescribeStacks",	
"cloudformation:ListStacks",	
"cloudformation:ListChangeSets",	
"cloudtrail:DescribeTrails",	
"codebuild:BatchGetProjects",	
"codebuild:CreateProject",	
"codebuild:ListCuratedEnvironmentImages",	
"codebuild:ListProjects",	
"codecommit:ListBranches",	
"codecommit:GetReferences",	
"codecommit:ListRepositories",	
"codedeploy:BatchGetDeploymentGroups",	
"codedeploy:ListApplications",	
"codedeploy:ListDeploymentGroups",	
<pre>"ec2:DescribeSecurityGroups",</pre>	
"ec2:DescribeSubnets",	
"ec2:DescribeVpcs",	
"ecr:DescribeRepositories",	
"ecr:ListImages",	
"ecs:ListClusters",	
"ecs:ListServices",	
"elasticbeanstalk:DescribeApplications",	
"elasticbeanstalk:DescribeEnvironments",	
"iam:ListRoles",	
"iam:GetRole",	
"lambda:ListFunctions",	
"events:ListRules",	
"events:ListTargetsByRule",	
"events:DescribeRule",	
"opsworks:DescribeApps",	
"opsworks:DescribeLayers",	
"opsworks:DescribeStacks",	
"s3:ListAllMyBuckets",	
"sns:ListTopics",	
"codestar-notifications:ListNotificationRules",	
"codestar-notifications:ListTargets",	
"codestar-notifications:ListTagsforResource",	
"codestar-notifications:ListEventTypes",	

```
"states:ListStateMachines"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Sid": "CodePipelineAuthoringAccess"
},
{
    "Action": [
        "s3:GetObject",
        "s3:ListBucket",
        "s3:GetBucketPolicy",
        "s3:GetBucketVersioning",
        "s3:GetObjectVersion",
        "s3:CreateBucket",
        "s3:PutBucketPolicy"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:s3::*:codepipeline-*",
    "Sid": "CodePipelineArtifactsReadWriteAccess"
},
{
    "Action": [
        "cloudtrail:PutEventSelectors",
        "cloudtrail:CreateTrail",
        "cloudtrail:GetEventSelectors",
        "cloudtrail:StartLogging"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:cloudtrail:*:*:trail/codepipeline-source-trail",
    "Sid": "CodePipelineSourceTrailReadWriteAccess"
},
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:iam::*:role/service-role/cwe-role-*"
    ],
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": [
                "events.amazonaws.com"
            ]
```

```
}
    },
    "Sid": "EventsIAMPassRole"
},
{
    "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": [
                "codepipeline.amazonaws.com"
            ]
        }
    },
    "Sid": "CodePipelineIAMPassRole"
},
{
    "Action": [
        "events:PutRule",
        "events:PutTargets",
        "events:DeleteRule",
        "events:DisableRule",
        "events:RemoveTargets"
    ],
    "Effect": "Allow",
    "Resource": [
        "arn:aws:events:*:*:rule/codepipeline-*"
    ],
    "Sid": "CodePipelineEventsReadWriteAccess"
},
{
    "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:*"
                }
            }
        },
        {
            "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
            "Effect": "Allow",
            "Action": [
                "sns:CreateTopic",
                "sns:SetTopicAttributes"
            ],
            "Resource": "arn:aws:sns:*:*:codestar-notifications*"
        },
        {
            "Sid": "CodeStarNotificationsChatbotAccess",
            "Effect": "Allow",
            "Action": [
                "chatbot:DescribeSlackChannelConfigurations",
                "chatbot:ListMicrosoftTeamsChannelConfigurations"
            ],
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

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、「<u>AWSCodePipeline\_ReadOnlyAccess</u>」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。

```
{
    "Statement": [
        {
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:GetPipelineExecution",
                "codepipeline:ListPipelineExecutions",
                "codepipeline:ListActionExecutions",
                "codepipeline:ListActionTypes",
                "codepipeline:ListPipelines",
                "codepipeline:ListTagsForResource",
                "s3:ListAllMyBuckets",
                "codestar-notifications:ListNotificationRules",
                "codestar-notifications:ListEventTypes",
                "codestar-notifications:ListTargets"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Action": [
                "s3:GetObject",
                "s3:ListBucket",
                "s3:GetBucketPolicy"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3::*:codepipeline-*"
        },
        {
            "Sid": "CodeStarNotificationsReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
```

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、 「<u>AWSCodePipelineApproverAccess</u>」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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:GetPipelineExecution",
                "codepipeline:ListPipelineExecutions",
                "codepipeline:ListPipelines",
                "codepipeline:PutApprovalResult"
                "Statement"
                "Codepipeline:PutApprovalResult"
                "Codepipeline:PutApprovalResult"
```

```
],
"Effect": "Allow",
"Resource": "*"
}
]
}
```

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したり、ビルドまたはテストアクション用に Jenkins リソースを統合したり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、「<u>AWSCodePipelineCustomActionAccess</u>」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。

```
{
    "Statement": [
        {
            "Action": [
                "codepipeline:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

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、 AWS 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。この管理ポリシーが適用されているユーザーは、通知の Amazon SNS トピックの作成と管 理、トピックへの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の 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、「<u>AWS CodeStar Notifications の Identity and Access Management</u>」 を参照してください。

AWS CodePipelineAWS 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 [<u>ドキュメント履歴</u>] 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
<u>AWSCodePipeline_FullAccess</u> - 既存のポリシーの更新	CodePipeline は、ListStack s を AWS 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
<u>AWSCodePipeline_FullAccess</u> - 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	2023 年 6 月 21 日

変更	説明	日付
	セス許可が追加されました。 詳細については、「 <u>CodePipel</u> <u>ine のマネージドポリシーと通</u> <u>知</u> 」を参照してください。	
AWSCodePipeline_FullAccess および <u>AWSCodePipeline_Re</u> adOnlyAccess マネージドポリ シー - 既存のポリシーの更新	CodePipeline は、チャット アプリケーション で Amazon Q Developer を使用する追加 の通知タイプをサポートする ために、これらのポリシーに アクセス許可を追加しまし たchatbot:ListMicros oftTeamsChannelCon figurations 。	2023 年 5 月 16 日
AWSCodePipelineFullAccess - 廃止	このポリシーは AWSCodePi peline_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

AWS CodePipeline

変更	説明	日付
AWSCodePipelineRea dOnlyAccess - 廃止	このポリシーは AWSCodePi peline_ReadOnlyAcc ess 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	2022 年 11 月 17 日
	した。詳細については、「 <u>AWS の 管理ポリシー AWS</u> <u>CodePipeline</u> 」を参照してく ださい。	
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 CodePipeline アクションのアクセス権限を付与するユーザーポリ シー例を示しています。これらのポリシーは、CodePipeline 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、「<u>CodePipeline コンソールを使用するた</u> めに必要なアクセス許可」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 8: 別のアカウントに関連付けられた AWS リソースをパイプラインで使用する

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。

```
{
    "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。

```
{
    "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。

```
{
    "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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline/
Staging/*"
```

		}				
	1					
h	-					
3						

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。

```
{
    "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

Jenkins を使用するためにパイプラインを設定して、ビルドまたはテストを行う場合は、統合用に別の ID を作成し、Jenkins と CodePipeline の統合に必要な最小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。

"Statement": [

{

{	
"Effect": "Allow",	
"Action": [	
"codepipeline:Acknowled	lgeJob",
"codepipeline:GetJobDet	ails",
"codepipeline:PollForJo	bs",
"codepipeline:PutJobFai	llureResult",
"codepipeline:PutJobSuc	cessResult"
],	
"Resource": "*"	
}	
],	
"Version": "2012-10-17"	
}	
-	

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、「<u>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</u> ス」を参照してください。

以下の例では、アカウント (80398EXAMPLE) のポリシーを示します。このポリシーでは、ユーザー は CodePipeline コンソールのパイプライン MyFirstPipeline 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、CodePipeline 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。

```
{
    "Statement": [
        {
          "Effect": "Allow",
          "Action": [
          "codepipeline:GetPipeline",
          "codepipeline:GetPipelineState",
          "codepipeline:ListActionTypes",
```



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、11111111111 AWS アカウントのユーザーが 80398EXAMPLE 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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::1111111111111:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

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 <u>11111111111</u> AWS アカウントCrossAccountPipelineViewersで作成されたポリシーを 示しています。

例 8: 別のアカウントに関連付けられた AWS リソースをパイプラインで使用する

ユーザーが別の AWS アカウントのリソースを使用するパイプラインを作成できるようにするポリ シーを設定できます。これを行うには、パイプライン (AccountA) を作成するアカウントと、パイプ ラインで使用するリソースを作成したアカウント (AccountB) の両方にポリシーおよびロールを設定 する必要があります。また、クロスアカウントアクセスに使用する AWS Key Management Service には、 でカスタマーマネージドキーを作成する必要があります。詳細およびステップごとの例につ いては、「<u>別の AWS アカウントのリソースを使用するパイプラインを CodePipeline で作成する</u>」 および「<u>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</u> る」を参照してください。

次の例は、パイプラインアーティファクトを保存するために使用される S3 バケットに対して AccountA によって設定されたポリシーを示しています。このポリシーは、AccountB へのアクセス を許可します。次の例では、AccountB の ARN は 012ID\_ACCOUNT\_B です。S3 バケットの ARN は codepipeline-us-east-2-1234567890 です。これらの ARN を、アクセスを許可するアカウン トおよび S3 バケットの ARN に置き換えます。

```
}
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": false
                }
            }
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
            },
            "Action": [
                "s3:Get*",
                "s3:Put*"
            ],
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::012ID_ACCOUNT_B:root"
            },
            "Action": "s3:ListBucket",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890"
        }
    ]
}
```

以下の例では、AccountA が設定したポリシーを示します。このポリシーは、AccountB がロール を継承できるようにします。このポリシーは、CodePipeline (CodePipeline\_Service\_Role) のサービスロールに適用する必要があります。IAM のロールにポリシーを適用する方法について は、<u>「ロールの修正」</u>を参照してください。次の例では、 012ID\_ACCOUNT\_B は AccountB の ARN です。

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": [
            "arn:aws:iam::012ID_ACCOUNT_B:role/*"
        ]
    }
}
```

以下の例で示しているのは、AccountB で設定したポリシーを、CodeDeploy の<u>EC2 インスタンス</u> <u>ロール</u>に適用しています。このポリシーでは、パイプラインアーティファクト (*codepipeline-us-east-2-1234567890*)を保存するために AccountA によって使用される S3 バケットへのアク セスを付与します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:Get*"
            ],
            "Resource": [
                "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                 "arn:aws:s3:::codepipeline-us-east-2-1234567890"
            ]
        }
    ]
}
```

次の例は、 のポリシーを示しています AWS KMS 。ここで、 *arn:aws:kms:useast-1:012ID\_ACCOUNT\_A:key/222222-3333333-4444-556677EXAMPLE*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。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey*",
                "kms:Encrypt",
                "kms:ReEncrypt*",
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-
east-1:012ID_ACCOUNT_A:key/222222-3333333-4444-556677EXAMPLE"
            ٦
        }
    ]
}
```

次の例では、AccountB によって作成された IAM ロール (CrossAccount\_Role) のインラインポリ シーを示しています。このポリシーは、AccountA のパイプラインで必要な CodeDeploy アクション にアクセスできるようにするものです。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
               "codedeploy:CreateDeployment",
               "codedeploy:GetDeployment",
               "codedeploy:GetDeploymentConfig",
               "codedeploy:GetApplicationRevision",
               "codedeploy:RegisterApplicationRevision"
        ],
        "Resource": "*"
    }
}
```

]

}

次の例では、AccountBによって作成された ロール (CrossAccount\_Role) のインラインポリシーを 示しています。このポリシーは、入力アーティファクトダウンロードし、出力アーティファクトを アップロードするために、S3 バケットにアクセスできるようにします。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject*",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                 "arn:aws:s3:::codepipeline-us-east-2-1234567890/*"
            ]
        }
    ]
}
```

リソースへのクロスアカウントアクセスのパイプラインを編集する方法の詳細については、「<u>ステッ</u> プ 2: パイプラインを編集する」を参照してください。

# 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-useast-2-1234567890 です。この ARN を S3 バケットの ARN に置き換えます。

```
{
    "Version": "2012-10-17",
    "Id": "SSEAndSSLPolicy",
    "Statement": [
        {
            "Sid": "DenyUnEncryptedObjectUploads",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "Condition": {
                "StringNotEquals": {
                    "s3:x-amz-server-side-encryption": "aws:kms"
                }
            }
        },
        {
            "Sid": "DenyInsecureConnections",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::codepipeline-us-east-2-1234567890/*",
            "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

トラブルシューティング

- 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
- 自分の 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 Management Console 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。

ー部の 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、既存のロールをそのサービスに渡すことができます。そのためには、サービスにロー ルを渡す権限が必要です。

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 Identity Center 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 Identity Center 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「<u>IAM アイデンティ</u> 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。

自分の 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、<u>が IAM と AWS CodePipeline 連携する方法</u>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<u>「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</u>」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「<u>IAM でのクロスアカウントのリソースへのアクセス</u>」を参照してください。

# CodePipeline 許可リファレンス

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 CodePipeline 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。

CodePipeline API オペレーションおよびコミットされたコードのアクションで必要なアクセス権限

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/version
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:

CodePipeline 許可リファレンス

## アクションの種類

arn:aws:codepipeline:region:account:actiontype:owner/category/provider/version
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/version
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/version

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/version
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/version

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/version

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

CodePipeline 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。CodePipeline 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 サービスロールポリシー

CodePipeline 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 のアクセス許可が含まれています。サービスロールステートメントを編集して、使用しないリソース へのアクセスを削除または追加できます。CodePipeline が各アクションに使用する最低限必要なア 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。

```
{
  "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": "AllowS30bjectAccess",
      "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 の CodePipeline 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 呼び出しアクションのサポート が追加されました。「 <u>CodePipeline 呼び出し</u> <u>アクションのサービスロールポリシーのアクセ</u> <u>ス許可</u> 」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 。「 <u>EC2 デプロイアクションのサービスロー</u> <u>ルポリシーのアクセス許可</u> 」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。 「 <u>サービスロールのポリシーのアクセス許可</u> 」 を参照してください。	2025 年 2 月 20 日
Amazon Elastic Container Registry ECRBuildA ndPublish アクションのサポートが追加さ れました。「 <u>サービスロールのアクセス許可:</u> <u>ECRBuildAndPublish アクション</u> 」を参照 してください。	2024 年 11 月 22 日
Amazon Inspector InspectorScan アクショ ンのサポートが追加されました。「 <u>サービス</u>	2024 年 11 月 22 日

AWS のサービス	CodePipeline のサポート日付
<u>ロールのアクセス許可: InspectorScan アク</u> <u>ション</u> 」を参照してください。	
コマンドアクションのサポートが追加されまし た。「 <u>サービスロールのアクセス許可: コマン</u> <u>ドアクション</u> 」を参照してください。	2024 年 10 月 3 日
AWS CloudFormation アクションのサポートが 追加されました。「 <u>サービスロールのアクセス</u> 許可: CloudFormationStackSet アクショ ン」および「 <u>サービスロールのアクセス許可:</u> <u>CloudFormationStackInstances アク</u> ション」を参照してください。	2020年12月30日
CodeCommit のフルクローン出力アーティファ クト形式のアクションのサポートが追加され ました。「 <u>サービスロールのアクセス許可:</u> <u>CodeCommit アクション</u> 」を参照してくださ い。	2020 年 11 月 11 日
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。「 <u>サービスロールのアクセ</u> <u>ス許可: CodeCommit アクション</u> 」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。「 <u>サービスロールのアクセス許</u> <u>可: AppConfig アクション</u> 」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。「 <u>サービスロールのアクセス</u> <u>許可: StepFunctions アクション</u> 」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。「 <u>サービスロールの</u> <u>アクセス許可: CodeConnections アクション</u> 」 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。「 <u>サービスロールのアクセス許可: S3</u> <u>デプロイアクション</u> 」を参照してください。	2019 年 1 月 16 日
CodeDeployToECS アクションアクションの サポートが追加されました。「 <u>サービスロール</u> <u>のアクセス許可: CodeDeployToECS アクショ</u> <u>ン</u> 」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。「 <u>サービスロールのアクセス許可:</u> <u>Amazon ECR アクション</u> 」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。「 <u>サービスロールのアクセス許</u> <u>可: Service Catalog アクション</u> 」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。「 <u>サービスロールのアクセス許</u> <u>可: AWS Device Farm アクション</u> 」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。「 <u>サービスロールのアクセス許可:</u> <u>Amazon ECS 標準アクション</u> 」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	CodePipeline
-----	--------------

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。「 <u>サービスロールのアクセス許可:</u> <u>CodeCommit アクション</u> 」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。「 <u>サービスロールのアクセス許</u> <u>可: AWS OpsWorks アクション</u> 」を参照して ください。	2016 年 6 月 2 日
AWS CloudFormation アクションのサポートが 追加されました。「 <u>サービスロールのアクセス</u> <u>許可: AWS CloudFormation アクション</u> 」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。「 <u>サービスロールのアクセス許</u> <u>可: CodeBuild アクション</u> 」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。「 <u>サービスロールのアクセス許</u> <u>可: アクションをElasticBeanstalk デプロ</u> <u>イする</u> 」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。「 <u>サービスロールのアクセス許可:</u> <u>AWS CodeDeploy アクション</u> 」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。「 <u>サービスロールのアクセス許可: S3</u> 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 Management Console し、<u>https://console.aws.amazon.com/iam/</u> 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、<u>https://</u> <u>docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege</u> 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、「<u>を使用した</u> 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

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、<u>AWS のサービス 「コンプライアンスプログラムによる範囲内</u>」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、<u>AWS 「Compliance</u> ProgramsAssurance」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細について は、<u>「Downloading Reports in AWS Artifact</u>」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。 は、コンプライアン スに役立つ以下のリソース AWS を提供します。

- セキュリティのコンプライアンスとガバナンス これらのソリューション実装ガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスの機能をデプロイする 手順を示します。
- HIPAA 対応サービスのリファレンス HIPAA 対応サービスの一覧が提供されています。すべてが HIPAA AWS のサービス の対象となるわけではありません。
- <u>AWS コンプライアンスリソース</u> このワークブックとガイドのコレクションは、お客様の業界や 地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド コンプライアンスの観点から責任共有モデルを理解 します。このガイドは、複数のフレームワーク (米国国立標準技術研究所 (NIST)、Payment Card Industry Security Standards Council (PCI)、国際標準化機構 (ISO) を含む) のセキュリティコント ロールを保護し、そのガイダンスに AWS のサービス マッピングするためのベストプラクティス をまとめたものです。

- 「デベロッパーガイド」の「ルールによるリソースの評価」 この AWS Config サービスは、リ ソース設定が内部プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価 します。AWS Config
- <u>AWS Security Hub</u> これにより AWS のサービス、内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールの一覧については、<u>Security Hub のコントロールリファレンス</u>を参照してください。
- Amazon GuardDuty 不審なアクティビティや悪意のあるアクティビティがないか環境をモニタ リングすることで AWS アカウント、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS のサービス を検出します。GuardDuty を使用すると、特定のコンプライアンスフレームワー クで義務付けられている侵入検知要件を満たすことで、PCI DSS などのさまざまなコンプライア ンス要件に対応できます。
- <u>AWS Audit Manager</u> これにより AWS のサービス、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

## 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イルオーバーするアプリケーショ ンとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一 または複数のデータセンターインフラストラクチャよりも可用性が高く、フォールトトレラントで、 スケーラブルです。

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、<u>AWS「 グローバルインフラスト</u> <u>ラクチャ</u>」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、<u>AWS「 クラウドセキュリティ</u>」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 AWS 環境を設計するには、「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。

また、リクエストにはアクセスキー ID と、IAM プリンシパルに関連付けられているシークレットア クセスキーを使用して署名する必要があります。または<u>AWS Security Token Service</u> (AWS STS) を 使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

# セキュリティに関する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 AWS 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、「<u>Jenkins 統合に使用する IAM ロールを作成する</u>」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。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。また、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も選択します。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、「<u>DevOps パイプラインの例</u>」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
- ・ <u>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</u>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「<u>PipelineDeclaration</u>」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では、new-github という名前のパイプライ ンSource、、manual、および という名前のステージBuild、および GitHub からのアクション (GitHub アプリ経由)、手動承認、および CodeBuild アクションプロバイダーを示しています。

new-github	👃 Notify 💌	Edit	Stop execution	Clone pipeline	Release change	
Pipeline type: V2 Execution m	ode: SUPERSEDED					
Pipeline execution ID: 60a18b	a0-b0d6-4a57-					
Source						
GitHub (Version 2)						
Succeeded - <u>1 minute ago</u>						
77cc2e44						
View details						
77cc2e44 12 Source: Moree pull re	quast #5 from . /fac	turo brooch				
77002044 El Source: Merge putre	quest #5 from / rea	iture-branch				
Disable transition	]					
	J					
⊘ manual Succeeded					Start rollback	
Pipeline execution ID: 60a18b	a0-b0d6-4a57					ŏ
						ŏ
Approval						
Manual approval						
Approved - Just now						
View details						
77cc2e44 🗹 Source: Merge pull re	quest #5 from /fea	ature-branch				
Disable transition	]					
	-					
Build In progress						
Pipeline execution ID: 60a18b	a0-b0d6-4a57-9aa2-					
Build						
AWS CodeBuild						
View dotails						
77cc2e44 🖸 Source: Merge pull re	quest #5 from/fea	ature-branch				

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。

Editing: new-github	Delete Cancel Save
Edit: Pipeline properties	Edit
Pipeline type V2	
Execution mode SUPERSEDED	
Edit: Variables Pipeline type V2 required	Edit variables
Name Default value Descript	tion
No variables No variables defined at the pipeline level in this pipeline.	
Edit: Triggers	Edit triggers
For source action: <b>Source</b>	
Filters Pull request Events: Created Revised Closed Include branches: master*	
Edit: Source	Edit stage
+ Add stage	
Edit: manual	Cancel Delete Done
Add entry condition     Add success condition     Add failure condition       + Add action group	
<b>、</b> ピック	
<u>パイプライン宣言</u>	
ステージ宣言	

・ <u>アクションの宣言</u>

- 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「<u>PipelineDeclaration</u>」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"
            }
        ],
        "<u>triggers</u>": [
            {
                 "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「<u>StageDeclaration</u>」 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、「<u>パイプライン実行モードを設定または変更す</u> る」を参照してください。

A 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、「<u>変数リファレンス</u>」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、「<u>チュートリアル: パイプラインレベル</u> 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、「<u>CodeConnections を使用して</u> 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」を参照してください。

詳細と詳細な例については、「」を参照してください<u>トリガーとフィルタリングを使用してパイプラ</u> インを自動的に開始する。

フィルタリングでは、「<u>構文での glob パターンの使用</u>」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。

A 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、パイプライン構造リファレンスの <u>pollingDisabledAt</u> 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、<u>「変</u> <u>更検出方法</u>」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"
                                 1
                             },
                             "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"
                         ]
                     }
                 }
             ]
        }
    }
],
```

include および exclude 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)

include および exclude 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)
 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 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)
 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。

- ・非アクティブなパイプラインは、30日間実行しないとポーリングが無効になります。
- 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 <u>PipelineMetadata</u> 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

ステージ宣言

{
 "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",
                              "<u>rules</u>": [
                                  {
                                       "name": "MyLambdaRule",
```



### 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、「<u>ルール構造リファ</u> 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": [
                             {
```



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、「<u>プロバイダー</u> <u>タイプ別の有効な設定パラメータ</u>」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&gt;_<hh>-<mm>- <ss></ss></mm></hh></mm></yyyy>

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ー意識 別子です。この形式の UUID (すべての 桁は 16 進形式): <8 桁>-<4 桁>-4 桁>- <4 桁>-<12 桁>
		js-application/54a60075-b96a-4bf3-90 13-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、「<u>変数リファ</u> 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 revisionValueで 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 ソースを有効にした Amazon S3 ソースアクションへの接続または 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CodeCommit ソースアクションと EventBridge。

# 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、入力アーティファクト構造が含まれます (アクションカテゴリでサポートされ ている場合)。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、入力アーティファクト構造が含まれます (アクションカテゴリでサポートされ ている場合)。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": [		
{		
"МуАрр"		
}		
],		

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、「<u>アクション構造</u> <u>リファレンス</u>」と、「<u>プロバイダータイプ別の有効な設定パラメータ</u>」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、「<u>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</u>」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 revisionValueで 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 ソースを有効にした Amazon S3 ソースアクションへの接続または 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CodeCommit ソースアクションと EventBridge。

▲ 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、パイプライン構造リファレンスの <u>pollingDisabledAt</u> 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 ー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、厳密なシーケンス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、「<u>アクション構造リファレンス</u>」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、「<u>アクション構造リファレン</u> 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、<u>CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise</u> <u>Server、GitLab.com、および GitLab セルフマネージドアクションの場合)</u> 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	<u>CodeCommit ソースアク</u> 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	<u>CodeStarSourceConn</u> <u>ection (Bitbucket</u> <u>Cloud、GitHub、GitHu</u> <u>b Enterprise Server、Gi</u> <u>tLab.com、およびGitLab</u> セルフマネージドアク ションの場合)
ビルド	Amazon ECR ECRBuildA ndPublish アクション	V2 のみ	<u>ECRBuildA</u> ndPublish 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	<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	カスタム Jenkins	V1, V2	<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>

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	<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
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	<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	カスタム BlazeMeter	V1, V2	<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	サードパーティー GhostInsp ector		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	カスタム Jenkins		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	サードパーティー Micro Focus StormRunner Load		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	サードパーティー Nouvola		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	サードパーティー Runscope		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>

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
	AWS CloudFormation		AWS CloudFormation デ プロイアクションリファ レンス
	CodeDeploy		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	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		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	Amazon ECS (Blue/Green) (こ れは CodeDeployToECS ア クションです)		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	Amazon EKS アクション	V2 のみ	<u>???</u>
	Elastic Beanstalk		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	AWS AppConfig		AWS AppConfig デプロ <u>イアクションリファレン</u> <u>ス</u>
	AWS OpsWorks		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>

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
	Service Catalog		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	Amazon Alexa		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
	カスタム XebiaLabs		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
承認	手動		<u>アクションタイプ別の有</u> <u>効な入力/出力アーティ</u> <u>ファクトの数</u>
Invoke	CodePipeline 呼び出しアク ション		<u>AWS CodePipeline アク</u> <u>ションリファレンスを呼</u> <u>び出す</u>
	AWS Lambda		<u>AWS Lambda アクショ</u> <u>ンリファレンスを呼び出</u> <u>す</u>
	AWS Step Functions		<u>AWS Step Functions ア</u> <u>クションリファレンスを</u> <u>呼び出す</u>
	InspectorScan		<u>Amazon Inspector</u> <u>InspectorScan 呼び</u> <u>出しアクションリファレ</u> <u>ンス</u>
コンピュー ティング	コマンドアクション		<u>コマンドアクションリ</u> <u>ファレンス</u>

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、「<u>CodePipeline アクションタイプとの統合</u>」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 AWS 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", "PollForSourceCha nges": "false", "RepositoryName": "my- repo"

ソース	作成方法	JSON 構造の出力の設定例
	パイプラインは CLI または で作成され AWS CloudFormation、 PollForSourceChang es パラメータは JSON 出力には表示されま 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。	<pre>"S3Bucket": "my-bucke t", "S3ObjectKey":     "object.zip", "PollForSourceChanges" : "false"</pre>
	パイプラインは CLI または で作成され AWS CloudFormation、 PollForSourceChang es パラメータは JSON 出力には表示されま せんが、は .2 に設定されますtrue。	"S3Bucket": "my-bucke t", "S3ObjectKey": "object.zip"
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。	<pre>"Owner": "MyGitHubA ccountName ", "Repo": " MyGitHubR epositoryName " "PollForSourceCh anges": "false", "Branch": " main" "OAuthToken": " ****"</pre>
	パイプラインは CLI または で作成され AWS CloudFormation、 PollForSourceChang es パラメータは JSON 出力には表示されま せんが、 は .2 に設定されますtrue。	<pre>"Owner": "MyGitHubA ccountName ", "Repo": "MyGitHubR epositoryName ", "Branch": " main", "OAuthToken": " ****"</pre>

ソース	作成方法	JSON 構造の出力の設定例	
	2 Pol1ForSourceChanges が JSON 構造または AWS 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 「 <u>変更検出方法</u> 」を参照してください。	リソースの詳細については、	

# 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

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	CodeStarS ourceConn ection	0	1
AWS	ビルド	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 CodePipeline

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	デプロイ	S3	1	0
AWS	デプロイ	CloudForm ation	0~10	0~1
AWS	デプロイ	CodeDeploy	1	0
AWS	デプロイ	ElasticBe anstalk	1	0
AWS	デプロイ	0psWorks	1	0
AWS	デプロイ	ECS	1	0
AWS	デプロイ	ServiceCa talog	1	0
AWS	Invoke	Lambda	0~5	0~5
ThirdParty	デプロイ	AlexaSkil lsKit	1~2	0
Custom	ビルド	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 デプロイフ 「」を参照してくださ	マクションパラメータに関連する例を含 い <u>Amazon S3 デプロイアクションリフ</u>	む詳細については、 <u>ァレンス</u> 。
Amazon S3 (ソースアク ションプロ バイダー)	Amazon S3 ソースアク 「 <u>Amazon S3 ソースフ</u>	フションパラメータに関連する例などの <u>アクションリファレンス</u> 」を参照してく	詳細については、 ださい。
Amazon ECR	Amazon ECR パラメー <u>ソースアクションリフ</u>	-タに関連する例などの詳細については <u>ァレンス</u> 」を参照してください。	√ <sup>r</sup> Amazon ECR
CodeCommi t	CodeCommit パラメー <u>ソースアクションリフ</u>	·タに関連する例などの詳細については、 <u>ァレンス</u> 」を参照してください。	<sup>∟</sup> CodeCommit
Bitbucket 、GitHub (GitHub アプリ経 由)、GH ES、GitLab の CodeStarS ourceConn ection アク ション	アクション設定の例を <u>メータ</u> 。	含む詳細については、「」を参照してく	ください <u>設定パラ</u>

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
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、「 <u>GitHub (OAuth アプリ</u> <u>経由) ソースアクションリファレンス</u> 」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 CloudForm ation	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 <u>AWS CloudFormation デプロイアクションリファレンス</u> 。		
CodeBuild	CodeBuild パラメータに関連する詳細な説明と例については、「 <u>AWS CodeBuild</u> <u>ビルドおよびテストアクションリファレンス</u> 」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、「 <u>AWS</u> <u>CodeDeploy デプロイアクションリファレンス</u> 」を参照してください。		
AWS Device Farm	AWS Device Farm パラメータに関する詳細な説明と例については、「」を参照し てください <u>AWS Device Farm テストアクションリファレンス</u> 。		
AWS Elastic Beanstalk	ElasticBe anstalk	ApplicationName	必須
		EnvironmentName	必須
AWS Lambda	AWS Lambda パラメー ださい <u>AWS Lambda ア</u>	-タに関連する例を含む詳細については <u>^クションリファレンスを呼び出す</u> 。	、「」を参照してく
AWS OpsWorks Stacks	OpsWorks	Stack	必須
		Layer	オプションです。
		Арр	必須
Amazon ECS	Amazon ECS パラメー <u>Elastic Container Servi</u> い。	·タに関連する詳細な説明と例について( <u>ce デプロイアクションリファレンス</u> 」	よ、「 <u>Amazon</u> を参照してくださ

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 ECS と CodeDeplo y (Blue/Gre en)	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、「 <u>Amazon ECS および CodeDeploy ブルー/グリーンデプロイアク</u> <u>ションリファレンス</u> 」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
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 ( <i>MyJenkins</i> <i>ProviderName</i> 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 EKSデプロイアクションリファレンス
- <u>Amazon S3 デプロイアクションリファレンス</u>
- 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
- <u>AWS CloudFormation デプロイアクションリファレンス</u>
- AWS 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
- <u>CodeStarSourceConnection (Bitbucket Cloud、GitHub、GitHub Enterprise Server、GitLab.com、</u> 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 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、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 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。インスタンスには 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か ら失敗したインスタンスが最初に選択されます。アクションが以前に失敗した場合など、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
- 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: EC2
- バージョン:1

設定パラメータ

InstanceTagKey

必須: はい

など、Amazon EC2 で作成したインスタンスのタグキーName。

InstanceTagValue

必須: はい

など、Amazon EC2 で作成したインスタンスのタグ値my-instances。

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、インストール済みである必要が あります。

#### Note

インスタンスを作成するときは、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。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アクセス許可を使用して、デフォルトのロールを 作成するか、既存のロールを更新します。

TargetDirectory

必須: はい

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。

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を既に作成している必要があり 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戻されます。

## 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、 アクションの編集ページの例を示しています。

×

#### Instance type

Choose the instance type that you want to deploy to. Supported types are Amazon EC2 instances or AWS Systems Manager (SSM) managed nodes. You must have already created, tagged, and installed the SSM agent on all instances.

T

EC2

You can add one group of tags for EC2 instances to this deployment group.

#### **One tag group:** Any instance identified by the tag group will be deployed to.

ľ	Key	
ſ	Q Name	×

#### Value

...

Q my-instances

### **Matching instances**

#### 2 unique matched instances.

#### Click here for details [ ]

#### Target directory

Specify the location of the target directory you want to deploy to. Use an absolute path like /home/ec2-user/deploy.

#### /home/ec2-user/testhelloworld

#### PreScript - optional

Path to the executable script file that runs BEFORE the Deploy phase. It should start from the root directory of your uploaded source artifact. Use an absolute path like uploadDir/preScript.sh.

#### PostScript

Path to the executable script file that runs AFTER the Deploy phase. It should start from the root directory of your uploaded source artifact. Use an absolute path like uploadDir/postScript.sh.

test/script.sh

#### Advanced

#### Max batch - optional

Specify the number or percentage of targets that can deploy in parallel.

targets

○ percentage

#### 設備の要は、figm 1 to the number of instances you have

# 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場合、CodePipeline サービスロールには以下のアクセス許可 が必要です。アクセス許可は最小特権で適切にスコープダウンされます。

```
{
    "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": [
                "*"
            ٦
        },
        {
            "Sid": "StatementForLogs",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
```

```
"logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:{{region}}:{{AccountId}}:log-group:/aws/codepipeline/
{{pipelineName}}:*"
            ]
        },
        {
            "Sid": "StatementForElasticloadbalancing",
            "Effect": "Allow",
            "Action": [
                "elasticloadbalancing:DeregisterTargets",
                "elasticloadbalancing:RegisterTargets"
            ],
            "Resource": [
                "arn:aws:elasticloadbalancing:{{region}}:{{AccountId}}:targetgroup/
[[targetGroupName]]/*"
            1
        },
        {
            "Sid": "StatementForSsmOnTaggedInstances",
            "Effect": "Allow",
            "Action": [
                "ssm:SendCommand"
            ],
            "Resource": [
                "arn:aws:ec2:{{region}}:{{AccountId}}:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/{{tagKey}}": "{{tagValue}}"
                }
            }
        },
        {
            "Sid": "StatementForSsmApprovedDocuments",
            "Effect": "Allow",
            "Action": [
                "ssm:SendCommand"
            ],
            "Resource": [
                "arn:aws:ssm:{{region}}::document/AWS-RunPowerShellScript",
                "arn:aws:ssm:{{region}}::document/AWS-RunShellScript"
```

) } }

CloudWatch Logs 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、「<u>CodePipeline コン</u> <u>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</u>」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 Logs 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、「<u>CodePipeline コン</u> <u>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</u>」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。

#### ユーザーガイド

# 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",
```

# 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。
- <u>EC2 デプロイアクションがエラーメッセージで失敗する No such file</u> 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、「<u>チュートリアル: Amazon ECR ソース、ECS - CodeDeploy 間のデプロイでパイ</u> 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 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。

## 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 revisionValueで 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 ソースを有効にした Amazon S3 ソースアクションへの接続または 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CodeCommit ソースアクションと EventBridge。

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

- <u>アクションタイプ</u>
- 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)
- <u>関連情報</u>

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、「<u>Amazon ECS Blue/Green デプロイアクション用の imageDetail.json</u> 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。

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、「」を参照してください<u>Amazon ECR ソースアクションリファ</u> レンス。

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
 このチュートリアルでは、サンプルアプリ仕様ファイル、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、CodeCommit とAmazon ECS インスタンスにデプロ イする 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 Docker ファイル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。そのアクションの詳細については、「」を参照してください<u>Amazon ECR ソースアクションリ</u>ファレンス。

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 を構築し、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。

A 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る 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 Docker ファイルの場所。必要に応じて、ルートレベルにない場合 は、代替の docker ファイルの場所を指定できます。

### 1 Note

の値が指定DockerFilePathされていない場合、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

イメージに使用されるタグ。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:イメージの構築に必要な 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、ポリシーステートメントに以下を追加し 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",
			<pre>"ecr-public:PutImage",</pre>
			<pre>"ecr-public:BatchCheckLayerAvailability",</pre>
			"sts:GetServiceBearerToken"
			],
			"Resource": "PublicECR_Resource_ARN"
		},	
		{	
		c	"Effect": "Allow".
			"Action": [
			"sts:GetServiceBearerToken"
			1.
			"Resource". "*"
		3	
	1	J	
ı	Т		
ſ			

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、「」を参照してください<u>ECRBuildAndPublish ビルドアク</u> ションリファレンス。

# 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 を使用して Docker イメージを構築して Amazon ECR にプッシュ する (V2 タイプ) – 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の変更を検出し、CodeDeployを使用して Amazon ECS クラスターとロードバランサーにトラフィックをルーティングしてデプロイします。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、「<u>Amazon Elastic Container Service デプロイアクションリファレンス</u>」を 参照してください。

トピック

- <u>アクションタイプ</u>
- 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、<u>こちら</u>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ロケーションに 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ロケーションに 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、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、「<u>Amazon ECS</u> 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、CodeDep1oyToECS 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。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCodeDeployDeploymentActions",
      "Action": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetDeployment"
      ],
      "Resource": [
        "arn:aws:codedeploy:*:{{customerAccountId}}:deploymentgroup:
[[ApplicationName]]/*"
      ],
      "Effect": "Allow"
    },
    {
      "Sid": "AllowCodeDeployApplicationActions",
      "Action": [
```

```
"codedeploy:GetApplication",
    "codedeploy:GetApplicationRevision",
    "codedeploy:RegisterApplicationRevision"
  ],
  "Resource": [
    "arn:aws:codedeploy:*:{{customerAccountId}}:application:[[ApplicationName]]",
    "arn:aws:codedeploy:*:{{customerAccountId}}:application:[[ApplicationName]]/*"
  ],
  "Effect": "Allow"
},
{
  "Sid": "AllowCodeDeployDeploymentConfigAccess",
  "Action": [
    "codedeploy:GetDeploymentConfig"
  ],
  "Resource": [
    "arn:aws:codedeploy:*:{{customerAccountId}}:deploymentconfig:*"
  ],
  "Effect": "Allow"
},
{
  "Sid": "AllowECSRegisterTaskDefinition",
  "Action": [
    "ecs:RegisterTaskDefinition"
  ],
  "Resource": [
    "*"
  ],
  "Effect": "Allow"
},
{
  "Sid": "AllowPassRoleToECS",
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": [
    "arn:aws:iam::{{customerAccountId}}: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、「<u>Amazon ECS タスク実行 IAM ロール</u>そして<u>タスクの IAM ロール</u>」を参照し てください。

上記の例に示すように、 iam: PassedToService 条件の サービスのリストecstasks.amazonaws.comに を追加することもできます。

# 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 <u>る</u> - 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、「<u>チュートリアル: CodePipeline を使用した継続的デプロイ</u>」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、「<u>Amazon ECS とは</u>」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、<u>Amazon ECS および CodeDeploy ブ</u> 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、「<u>入力アー</u> <u>ティファクト</u>」および「<u>Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイ</u> <u>ル</u>」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、「<u>Amazon ECS 標準デプロイアク</u> 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。

```
{
  "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:*:{{customerAccountId}}:service/[[clusters]]/*"
      ]
```

```
},
    { "Sid": "ECSTagResource",
      "Effect": "Allow",
      "Action": [
        "ecs:TagResource"
      ],
      "Resource": [
        "arn:aws:ecs:*:{{customerAccountId}}:task-definition/[[taskDefinitions]]:*"
      ],
      "Condition": {
        "StringEquals": {
          "ecs:CreateAction": [
            "RegisterTaskDefinition"
          ٦
        }
      }
    },
    { "Sid": "IamPassRolePermissions",
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": [
        "arn:aws:iam::{{customerAccountId}}: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「<u>タグ付け承認のタイムライン</u>」を参照してください。

タスクに IAM ロールを使用するには、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、「<u>Amazon ECS タスク実行 IAM ロール</u>そして<u>タスクの IAM ロール</u>」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。

#### ユーザーガイド

# 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<u>チュートリアル: CodePipeline を使用して Docker イメージを構築して Amazon ECR にプッシュする (V2 タイプ)</u>については、「」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、このチュートリアルでは、Docker イメージをビルドして 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。

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、<u>チュートリアル: Amazon ECR ソー</u> <u>ス、ECS - CodeDeploy 間のデプロイでパイプラインを作成する</u>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、「<u>Amazon</u> ECS とは」を参照してください。

# Amazon Elastic Kubernetes Service 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、イメージをイメージリポジトリに 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。

Note

### ▲ 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actionRoleArnから 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)

デプロイのグラフの場所。

HelmValuesFiles

必須: いいえ (Helm タイプでのみオプション)

デプロイのグラフの場所。

#### 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、サービスのコンテナ名を説明するテキストファイル、イメージ とタグ。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。

#### 名前空間

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指定して、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で Kubernetes マニフェストファ イルまたは Helm チャートを検索します。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。 たとえば、指定されたオブジェクトキーは になりますmy-chart-0.1.0.tgz。

## 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

などのキーと値の環境変数ペアのキー\$IMAGE\_TAG。

#### 値

キーと値のペアの値。などv1.0。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。たとえば、パイプラインには で ECR イメージを作成する ECRBuildAndPublish アクションがあり\${codepipeline.PipelineExecutionId}、EKS ア クションは を環境変数の値\${codepipeline.PipelineExecutionId}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 が実行されるときは、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、サービスロールに次 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ときは、EKS クラスター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 IAM EKS クラスター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、サービスロールに次 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 がアクションを実行すると、CodePipeline は次のよう にパイプラインの名前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、サービスロールに次 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、アクセス許可をリソースレベルに 絞り込みます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "eks:DescribeCluster"
            ],
            "Resource": "arn:aws:eks:*:YOUR_AWS_ACCOUNT_ID: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:*: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、「<u>CodePipeline コン</u> <u>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</u>」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、「」 のチュートリアルの例を参照してください<u>ステップ 4: CodePipeline サービスロールのアクセスエン</u> トリを作成する。

#### ユーザーガイド

# 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<u>チュートリアル: CodePipeline を使用して Amazon EKS にデプ</u>ロイするについては、「」を参照してください。

# 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、静的ウェブサイトのホスティングやアーカイブ用に Amazon S3 バケットにファイルをデプロイします。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。指定しないと、ホ ストされている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時に圧 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 <u>Cache-Control</u> 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。

入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

S3 デプロイアクションをサポートするには、ポリシーステートメントに以下を追加します。

```
{
  "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": "{{customerAccountId}}"
        }
      }
    }
  ]
}
```

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
### 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。
- <u>Amazon S3 ソースアクションリファレンス</u> 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

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たときに、パイ プラインをトリガーします。

#### Note

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。

Amazon S3 バケットを作成して、アプリケーションファイルのソース場所として使用できます。

Note

ソースバケットを作成するときは、バケットでバージョニングを有効にしてください。既存 の Amazon S3 バケットを使用する場合は、「<u>バージョニングの使用</u>」を参照して、既存の バケットでバージョニングを有効にします。

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 S3 ソースバケット に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。

(i) 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 revisionValueで 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 ソースを有効にした Amazon S3 ソースアクションへの接続または 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CodeCommit ソースアクションと EventBridge。 Amazon S3 アクションを使用して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、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ておく必要があります。

#### 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。

トピック

- <u>アクションタイプ</u>
- 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。

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

AllowOverrideForS30bjectKey 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 S30bjectKey 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、「<u>ソースリビジョンオーバーラ</u> イドでパイプラインを開始する」を参照してください。

▲ Important

AllowOverrideForS30bjectKey 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、AllowOverrideForS30bjectKey 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、 「<u>S3 ソースと CloudTrail</u>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「<u>S3</u> <u>ソースと CloudTrail</u>証跡 (AWS 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、ポリシーステートメントに以下を追加します。

```
{
    "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": "{{customerAccountId}}"
        }
      }
    }
  ]
}
```

# 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、サンプルアプリケーション仕様ファイル、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場合、CodePipeline サービスロールポリシーには、最小限の 特権でアクセスを維持するために、リソースレベルまで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。

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Action": [
                "appconfig:StartDeployment",
                "appconfig:StopDeployment",
                "appconfig:GetDeployment"
            ],
            "Resource": [
             "arn:aws:appconfig:*:{{customerAccountId}}:application/[[Application]]",
             "arn:aws:appconfig:*:{{customerAccountId}}:application/[[Application]]/*",
             "arn:aws:appconfig:*:{{customerAccountId}}: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

```
"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。

- <u>AWS AppConfig</u> 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: デプロイプロバイダーとして AWS AppConfig 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、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。

# AWS CloudFormation デプロイアクションリファレンス

AWS 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 AWS 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。スタックおよび変更セットで実行できる AWS 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 AWS 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 AWS 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

- <u>アクションタイプ</u>
- 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 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 AWS 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 AWS 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 AWS 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 AWS 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 AWS 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 AWS CloudFormation を更新しま す。

AWS 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。この方法で AWS CloudFormation アク ションを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よ うに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。<u>アクションの宣言</u>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 [<u>UpdateStack</u>] 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 AWS 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

AWS 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 AWS 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

AWS 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1MB です。CodePipeline の AWS 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 [AWS 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、 AWS 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、<u>設定プロパティ(JSON オブジェクト)</u>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、「<u>AWS CloudFormation アーティファクト</u>」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、AWS 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、「<u>AWS CloudFormation アー</u> <u>ティファクト</u>」を参照してください。
  - AWS 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、 AWS CloudFormation スタックの「出力」セクションの内容が含まれています。

AWS 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、<u>出力</u>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。

AWS 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、「<u>変数リファレンス</u>」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、 「<u>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</u> 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: AWS CloudFormation アクション

CodePipeline がアクションを実行する場合、CodePipeline サービスロールポリシーには、最小限の 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。たとえば、ポリシーステートメントに以下を追加します。

```
{
  "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:*:{{customerAccountId}}:stack/[[cfnDeployStackNames]]/
*"
       ]
    },
    {
      "Sid": "ValidateTemplate",
      "Effect": "Allow",
      "Action": [
        "cloudformation:ValidateTemplate"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowIAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::{{customerAccountId}}: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、 AWS 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。
- ・ <u>AWS CloudFormation API リファレンス</u> AWS CloudFormation API リファレンスの <u>CreateStack</u> パラメータは、テンプレートのスタックパラメータを記述します AWS CloudFormation。

# AWS 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 AWS 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、1 つの AWS 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 AWS 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。

AWS 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 AWS CloudFormation StackSets と統合するには、2 つの異なるアクションタイプを 一緒に使用します。

- 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 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、<u>「Stacksets AWS CloudFormation による信頼されたアクセスの有効化</u>」を参照してく ださい。

### トピック

- Stack AWS 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
- ・ <u>サービスロールのアクセス許可: CloudFormationStackSetアクション</u>
- ・ サービスロールのアクセス許可: CloudFormationStackInstancesアクション
- <u>スタックセットオペレーションのアクセス許可モデル</u>
- テンプレートパラメータのデータタイプ
- 関連情報

## Stack AWS 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。

- 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。
- 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

 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

スタックインスタンス	リージョン	ステータス
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<u>ベストプラクティス</u>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。スタックセットを 1 つの AWS リージョンで作成した 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

CloudFormationStackSet アクション

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ロケーション。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。

#### 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。 AWS 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。<u>例</u> <u>CloudFormationStackSet アクションの設定</u> 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

CloudFormationStackSet アクション

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、「<u>スタックセットオペレーションのアクセス許可モデル</u>」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: AWS 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

AWS 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

AWS 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 AWS 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、 AWS 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。

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。

```
[
"11111222222","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可能性のあ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 AWS CloudFormation 停止します。リージョンでオペ レーションが停止した場合、 AWS 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 AWS 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 AWS CloudFormation 切り下げられます。切り捨てると ゼロになる場合、 AWS CloudFormation は代わりに数値を 1 に設定します。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、 AWS 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"

]

Γ

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、 「<u>テンプレートパラメータのデータタイプ</u>」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::ParameterOverridessFileName 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。

FailureTolerancePercentage

必須: いいえ

このスタックオペレーションが失敗する可能性のあ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 AWS CloudFormation 停止します。リージョンでオペ レーションが停止した場合、 AWS 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 AWS 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 AWS CloudFormation 切り下げられます。切り捨てると ゼロになる場合、 AWS CloudFormation は代わりに数値を 1 に設定します。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、 AWS 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: '11111222222'
   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

AWS 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",
```

{



# 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

AWS 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

AWS 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アクセス許可を使用すると、AWS Organizations によって管理されるアカウ 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<u>AWS 「Organizations で信頼されたアクセスを有</u> 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「<u>https://docs.aws.amazon.com/AWSCloudFormation/latest/</u> UserGuide/stacksets-bestpractices.html」を参照してください。
- <u>AWS CloudFormation API リファレンス</u> 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 リファレンスの以下の 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。
- <u>DescribeStackSetOperation</u> 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 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は 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ことができます。詳細については、「<u>AWS CodeBuild ユーザーガイド</u>」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 buildspec ファイルを検索し、解凍されたバージョンのこのアー ティファクトを含む ディレクトリで 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。

バッチビルドが有効になっているパイプラインの例については、<u>CodePipeline と CodeBuild の統</u> 合」および「バッチビルド」を参照してください。

#### BuildspecOverride

必須: いいえ

このビルドでのみ、ビルドプロジェクトで定義された最新のもの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。

この値が設定されている場合、次のいずれかになります。

- インライン buildspec 定義。詳細については、<u>Buildspec</u>構文の構文リファレンスを参照してく ださい。
- 組み込みCODEBUILD\_SRC\_DIR環境変数の値または S3 バケットへのパスを基準とする 代替 buildspec ファイルへのパス。バケットは、ビルドプロジェクト AWS リージョン と同じ にある必要があります。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、または空の文字列に設定されている場合、ソースコードにはルートディレ 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

環境変数を使用して、機密情報、特に AWS 認証情報を保存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<u>EnvironmentVariable</u>」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。

### 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 ファイルを検索し、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 buildspec ファイルで定義されているアーティファクトをパイプライン内の後続のアクションで使用できるようにするために使用できます。1つの出力アーティファクトのみが定義されている場合、このアーティファクトは buildspec ファイルの artifactsセクションに直接定義できます。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 ファイル内のアーティファクト 識別子と一致する必要があります。

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、<u>CodeBuild アクションの出力変数</u> 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。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild",
        "codebuild:BatchGetBuildBatches",
        "codebuild:StartBuildBatch"
      ],
      "Resource": [
        "arn:aws:codebuild:*:{{customerAccountId}}:project/[[ProjectName]]"
      ],
      "Effect": "Allow"
    }
  ]
}
```

# 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"}]'
    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\"}]"
            },
            "OutputArtifacts": [
```



### 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。

- <u>AWS CodeBuild ユーザーガイド</u> CodeBuild アクションを使用したパイプラインの例について は、「<u>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</u>」 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、[<u>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</u> 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、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、CodeBuild と 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します AWS Device Farm。
- <u>CodeBuild のビルド仕様リファレンス</u> このリファレンストピックでは、CodeBuild buildspec ファイルを理解するための定義と例を示します。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すると、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ことを簡素化できます。

### 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。という名前の s3-pipeline-test (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 (ターゲット) パイプラインmy-s3-pipeline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。たとえば、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、変数をシリアル化された JSON 文字列として指定します。パイプライン変数の使用の 詳細については、「 PipelineVariable」を参照してください。 CodePipeline

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。



次の図は、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、という名前のパイプラインs3-pipeline-test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。という名前のs3-pipeline-testパイプラインmy-s3-pipelineの実行が完了すると、アクションによって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zq8mjNYEexample。

Action name Choose a name for your action			
Invoke-pipeline			
No more than 100 characters			
Action provider			
AWS CodePipeline		•	
Region			
United States (N. Virginia)		•	
Pipeline name Choose a pipeline that you have already created in the AWS Coo task.	dePipeline console. Or create a pipeline	in the AWS CodePipelin	e console and then ret
Q my-s3-pipeline		XC	
Source Revision Overrides - optional Choose the action name, revision type, and revision value for yo CodePipeline.	our CodePipeline source revisions. In th	e value field, you can ref	erence variables gener
Action name F	Revision type		
Source	S3_OBJECT_VERSION_ID	•	
zq8mjNYE		Remove	
Add source revision override			
		variables generated by	CodePipeline.
Variables - optional Choose the key and value for your CodePipeline pipeline variabl	les. In the value field, you can reference	e variables generated by	
Variables - optional Choose the key and value for your CodePipeline pipeline variab Add pipeline variable Variable namespace - optional Choose a namespace for the output variables from this action. You configuration. Learn more	les. In the value field, you can reference You must choose a namespace if you w	ant to use the variables t	this action produces in
Variables - optional Choose the key and value for your CodePipeline pipeline variab Add pipeline variable Variable namespace - optional Choose a namespace for the output variables from this action. Ye configuration. Learn more	les. In the value field, you can reference	ant to use the variables t	this action produces in
Variables - optional Choose the key and value for your CodePipeline pipeline variab Add pipeline variable Variable namespace - optional Choose a namespace for the output variables from this action. Your configuration. Learn more 2	les. In the value field, you can reference	ant to use the variables t	this action produces ir <b>Cancel</b>
Variables - optional Choose the key and value for your CodePipeline pipeline variab Add pipeline variable Variable namespace - optional Choose a namespace for the output variables from this action. V configuration. Learn more	les. In the value field, you can reference	ant to use the variables t	this action produces in Cancel
Variables - optional Choose the key and value for your CodePipeline pipeline variab Add pipeline variable Variable namespace - optional Choose a namespace for the output variables from this action. Your configuration. Learn more 2	les. In the value field, you can reference	ant to use the variables t	this action produces in Cancel
Variables - optional Choose the key and value for your CodePipeline pipeline variab Add pipeline variable Variable namespace - optional Choose a namespace for the output variables from this action. N configuration. Learn more []	les. In the value field, you can reference	ant to use the variables t	this action produces in Cancel

• 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と、CodePipeline サービスロールポリシーには アクセ スcodepipeline:StartPipelineExecution許可が必要になり、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ます。

```
{
    "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。

<u>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</u> – このセクションでは、ソースリビジョンを使用してパイプラインを手動で開始するか、EventBridge イベント入力トランスフォーマーを使用して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。

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 revisionValueで を使用することもできます。ここで、 revisionValueはオブジェク

Note

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、、<u>Amazon ECR ソースアクションと EventBridge リソース</u>、<u>イベントに対して</u> <u>ソースを有効にした Amazon S3 ソースアクションへの接続</u>または 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<u>CodeCommit ソースアクションと</u> <u>EventBridge</u>。

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

アクションタイプ

- 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、<u>ポーリングパイプライ</u> <u>ンを移行する (CodeCommit ソース) (CLI)</u> または <u>ポーリングパイプラインの移行 (CodeCommit</u> <u>ソース) (AWS CloudFormation テンプレート)</u>を参照してください。

A 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 <u>CodeBuild GitClone のアクセス権限を CodeCommit ソースアクションに追</u> <u>加します。</u>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 <u>CodePipeline サービスロールにアクセス</u> 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、<u>チュートリアル:CodeCommit パイプラインソー</u> <u>スでフルクローンを使用する</u> 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場合、CodePipeline サービスロールポリシーには、最小限の 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。たとえば、ポリシーステートメントに以下を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "codecommit:CancelUploadArchive",
                "codecommit:GetBranch",
                "codecommit:GetCommit",
                "codecommit:GetRepository",
                "codecommit:GetUploadArchiveStatus",
                "codecommit:UploadArchive"
            ],
            "Resource": [
                "arn:aws:codecommit:*:{{customerAccountId}}:[[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、サンプルアプリケーション仕様ファイル、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、<u>Amazon ECS および CodeDeploy ブルー/グリーンデプロイア</u> <u>クションリファレンス</u>を参照してください。

トピック

- <u>アクションタイプ</u>
- 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
- ・ <u>サービスロールのアクセス許可: AWS CodeDeploy アクション</u>
- アクションの宣言
- <u>関連情報</u>

# 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、[<u>CodeDeploy AppSpec ファイルリファレンス</u>] 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。

アクションタイプ

# 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。

```
{
  "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:*:{{customerAccountId}}:application:
[[codedeployApplications]]",
        "arn:aws:codedeploy:*:{{customerAccountId}}:deploymentgroup:
[[codedeployApplications]]/*"
      ٦
    },
    {
      "Effect": "Allow",
      "Action": [
        "codedeploy:GetDeploymentConfig"
      ],
      "Resource": [
        "arn:aws:codedeploy:*:{{customerAccountId}}: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

## 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。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、「<u>コードプッシュまたはプルリクエストイベントタイプでトリガーを追加する</u>」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、 「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドア</u> <u>クションの場合</u>」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)

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プロバイダー経由)

### 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)

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 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、<u>Connections アクセス許可リファレンス</u>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、 「<u>CodePipeline サービスロールにアクセス許可を追加する</u>」を参照してください。

欧州 (ミラノ) で接続を使用するには AWS リージョン、以下を実行する必要があります。

- 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: 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。

▲ 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。

例: some-user/my-repo

### ▲ 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、<u>Bitbucket、GitHub、GitHub Enterprise Server、ま</u> 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、<u>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</u>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、<u>CodePipeline のトラブルシューティング</u>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、<u>チュートリ</u> <u>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</u>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: CodeConnections アクション

CodeConnections の場合、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"
],
    "Resource": "resource_ARN"
},
```

接続に関する IAM アクセス許可の詳細については、「<u>Connections アクセス許可リファレンス</u>」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:codestarconnections: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 AWS 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。

- <u>AWS::CodeStarConnections::Connection</u> AWS CodeStar Connections リソースの AWS CloudFormation テンプレートリファレンスには、 AWS CloudFormation テンプレート内の接続の パラメータと例が記載されています。
- <u>AWS CodeStar Connections API リファレンス</u> 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。「<u>Bitbucket Cloud への接続</u>」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。「<u>GitHub コネクショ</u> 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。「<u>GitHub</u> 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actionRoleArnか らを追加します。

コマンドアクションに関する考慮事項

- 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、「CodePipeline 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、「<u>CodePipeline コン</u> <u>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</u>」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、「<u>ビルド環境コンピュー</u> 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。このアクションで使用できる 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<u>「ビルド環境コンピューティングモードとタイプ</u> のリファレンス」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

ー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。たとえ 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「<u>ビルド環境の環境変数</u>」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 $\times$ Action name Choose a name for your action Commands No more than 100 characters Action provider Commands • 67 Input artifacts Choose an input artifact for this action. Learn more [ ¥ SourceArtifact $\times$ 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 Туре Secrets Manager • Remove Add environment variable VPC ID Specify the VPC ID that your compute action will use. Q Additional configuration Environment type, compute Environment Type LINUX\_CONTAINER Ŧ Compute 3 GB memory, 2 vCPUs O 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

### Vpcld

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

+-

などのキーと値の環境変数ペアのキーBEDROCK\_TOKEN。

#### 値

キーと値のペアの値。 などapiTokens:bedrockToken。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 の場合SECRETS\_MANAGER、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。

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 を使用すると AWS CLI、プ レーンテキストの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。

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、「」を参照してください設定パラメータ。

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権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

コマンドをサポートするには、ポリシーステートメントに以下を追加します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:{{region}}:{{customerAccountId}}:log-group:/aws/
codepipeline/{{pipelineName}}",
                "arn:aws:logs:{{region}}:{{customerAccountId}}:log-group:/aws/
codepipeline/{{pipelineName}}:log-stream:*"
            ٦
        }
    ]
}
```

# 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。

<u>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 (V2 タイプ)</u>
 – 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、<u>AWS Device Farm で</u> 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

### АррТуре

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。

- i0S
- 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、プロジェクトで使用できるデバイスプール 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:uswest-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

設定パラメータ
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場合、CodePipeline サービスロールポリシーには、最小限の 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。たとえば、ポリシーステートメントに以下を追加します。

```
{
    "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
```

```
{
    "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:@EXAMPLE-
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 ルビルド仕様ファイルとサンプルアプリケーションを提供します。
- <u>チュートリアル: を使用して iOS アプリをテストするパイプラインを作成する AWS Device Farm</u>
   ニ 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 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 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 環境にデプロイします。

トピック

- <u>アクションタイプ</u>
- <u>設定パラメータ</u>
- 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: アクションをElasticBeanstalkデプロイする
- 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 で作成したアプリケーションの名前。

#### EnvironmentName

必須: はい

Elastic Beanstalk で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。各環境が実行するのは一度に 1 つのアプリケーションバージョン

だけですが、同じアプリケーションバージョンや複数の異なるアプリケーションバージョンを多数の環境で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: アクションをElasticBeanstalkデプロ イする

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:*",
        "sas:*",
        "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、<u>https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege</u>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
```

```
{
    "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脆弱性の検出と修正を自動化 します。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リポジトリの 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。

#### A 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。アクションを実行すると、個別の料金が発生します AWS CodeBuild。

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)スキャンの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重要度の脆弱性の数。

HighThreshold

CodePipeline がアクションを失敗させる必要がある、ソースで見つかった重要度の高い脆弱性の数。

MediumThreshold

CodePipeline がアクションを失敗させるソースで見つかった中程度の重要度の脆弱性の数。

LowThreshold

CodePipeline がアクションを失敗させるソースで見つかった重要度の低い脆弱性の数。

#### 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 X 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.

T

#### 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

T

## 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 リポジトリ用である場合、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、ポリシーステートメントに以下を追加します。

```
{
    "Effect": "Allow",
    "Action": "inspector-scan:ScanSbom",
    "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
           "ecr:GetDownloadUrlForLayer",
           "ecr:BatchGetImage",
        "
}
```

```
"ecr:BatchCheckLayerAvailability"
],
"Resource": "resource_ARN"
},
```

さらに、 コマンド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。



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、<u>JSON イベントの例</u>を 参照してください。Lambda 関数の実装の一部として、<u>PutJobSuccessResult API</u> または <u>PutJobFailureResult API</u> 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 <u>PutJobSuccessResult API</u> 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": [
        ſ
            "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、<u>ジョブ</u>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": "1111111-abcd-1111-abcd-11111abcdef",
        "accountId": "1111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                "FunctionName": "MyLambdaFunction",
                "UserParameters": "input_parameter"
                "Second Second Second
```

```
}
            },
            "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:kev/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: セッションのシークレットアクセスキー。
- 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。

- <u>AWS CloudFormation ユーザーガイド</u> パイプラインの Lambda アクションと AWS CloudFormation アーティファクトの詳細については、<u>CodePipeline Pipelines でのパラメータ</u> <u>オーバーライド関数の使用</u>」、「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。
- <u>CodePipeline のパイプラインで AWS Lambda 関数を呼び出す</u> 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

AWS 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。

#### スタック

必須: はい

AWS OpsWorks アプリ。アプリケーションは、デプロイして実行するコードを表します。 レイヤー

必須: いいえ

AWS OpsWorks スタック。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
```

```
{
    "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。

<u>AWS OpsWorks ユーザーガイド</u> – 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

ソースロケーションの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": [
```

```
ユーザーガイド
```

```
"cloudformation:ValidateTemplate"
],
"Resource": "resource_ARN"
}
```

# 設定ファイルの種類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。

- <u>Service Catalog ユーザーガイド</u> 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、パイプラインを作成して設定し、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「<u>Lambda クォータ</u>」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": [
```

```
ユーザーガイド
```

```
"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-1:1112222333:stateMachine:HelloWorld-
StateMachine
   ExecutionNamePrefix: my-prefix
```

```
{
    "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-1:11122223333:stateMachine:HelloWorld-
StateMachine
ExecutionNamePrefix: my-prefix
Input: '{"action": "test"}'
```

```
{
    "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
```

```
{
    "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。

- <u>AWS Step Functions デベロッパーガイド</u> 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
- <u>コマンド</u>
- DeploymentWindow
- Lambdalnvoke
- 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、「<u>ステージ条件はどのよ</u> 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

CloudwatchAlarm

- 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 する前にアラームの状態が OK に変わるまで 20 分かかるように設定します。

### ルール設定の例

YAML

rules:

ルールタイプ
```
- 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。

 <u>失敗時の条件の作成</u> – 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

条件を作成するときに、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、「<u>ステージ条件はどのように機能し</u> ますか?」を参照してください。

CodeBuild ルールを使用して、ビルドプロジェクトの正常な実行が、ビルド実行が beforeEntry 条件 に対して成功するなどのルール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 のルールのみを使用できますVariableCheck。

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スコープダウンします。

```
{
    "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 buildspec ファイルを検索し、解凍されたバージョンのこのアー ティファクトを含むディレクトリで 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。

バッチビルドが有効になっているパイプラインの例については、<u>CodePipeline と CodeBuild の統</u> 合」および「バッチビルド」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 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。

(i) Note

環境変数を使用して機密情報、特に AWS 認証情報を保存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<u>EnvironmentVariable</u>」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。

#### ユーザーガイド

### ルール設定の例

#### YAML

#### 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「<u>条件</u>」、「<u>RuleTypeld</u>」、 「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、「<u>ステージ条件はどのように機能</u> しますか?」を参照してください。

Commands ルールを使用して、成功したコマンドがルール基準を満たす条件を作成できます。たと えば、beforeEntry 条件に対して成功するコマンドの出力やファイルパスなどです。

Note

スキップ結果で設定された beforeEntry 条件では、 LambdaInvokeおよび のルールのみを使 用できますVariableCheck。

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、次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、ビルドプロジェ クトを関連付けたり作成したりすることなく、仮想コンピューティングインスタンスでシェル環境 コマンドを許可します。

#### 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、アクションによって実行 されるビルドはCodeBuild のアカウントのビルド制限に帰属します。コマンドルールによって実行 されるビルドは、その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、そのロールを使用して実行時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、「<u>CodePipeline コン</u> <u>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</u>」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 で成功したコマンドルールのログを 表示します。

	Condition execution details Execution ID: 9ace7b55-da5d-426d-8451-fe	3e92b1c1e6		×
	Condition type: BeforeEntry Result:	FAIL Status: 🕑 Succeeded		
	Rule states Rule Configurati	on		
	Name Rule Execution	ID	Status Re	ason
	cmdsrule 05f51200-cf09	-4d	Succeeded Log	<u>gs 🖸</u>
				Done
•	2024-11-04T15:47:07.928Z	[Container] 2024/11/04 15:47:07	.469537 Entering phase BUILD	
•	2024-11-04T15:47:07.928Z	[Container] 2024/11/04 15:47:07	.514998 Running command echo "he	llo world"
Þ	2024-11-04T15:47:07.928Z	hello world		
Þ	2024-11-04T15:47:07.928Z			
Þ	2024-11-04T15:47:07.928Z	[Container] 2024/11/04 15:47:07	.522410 Phase complete: BUILD Sta	ate: SUCCEE
Þ	2024-11-04T15:47:07.928Z	[Container] 2024/11/04 15:47:07	.522428 Phase context status code	e: Message:
Þ	2024-11-04T15:47:07.928Z	[Container] 2024/11/04 15:47:07	.565724 Entering phase POST_BUILD	)
•	2024-11-04T15:47:07.928Z	[Container] 2024/11/04 15:47:07.	.566944 Phase complete: POST_BUI	D State: S

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 がルールを実行すると、CodePipeline は次のようにパイプラインの名前を使用してロ 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、「<u>CodePipeline コン</u> <u>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</u>」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 ルールを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。

以下の詳細は、 コマンドルールに使用されるデフォルトのコンピューティングを示しています。 詳細については、「CodeBuild ユーザーガイド」の「<u>ビルド環境のコンピューティングモードお</u> <u>よびタイプ</u>」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「<u>条件</u>」、「<u>RuleTypeld</u>」、 「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、「<u>ステージ条件はどのよ</u> うに機能しますか?」を参照してください。

- トピック
- <u>ルールタイプ</u>
- 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。
- 「-」文字は範囲を指定するために使用します。例えば、時間フィールドの「10-12」は「10
   時、11 時、12 時」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、<u>https://docs.oracle.com/middleware/1221/wcs/tag-ref/</u> <u>MISC/TimeZones.html</u> 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

#### 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「<u>条件</u>」、「<u>RuleTypeld</u>」、 「<u>RuleExecution</u>」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、「<u>ステージ条件はどのように</u> 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。

 <u>失敗時の条件の作成</u> – 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、「<u>ステージ条件はどのよう</u> 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

#### 演算子

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、「j<u>ava.util.regex.Pattern</u>」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。

- <u>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</u> 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。
- <u>変数リファレンス</u> 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「<u>条件</u>」、「<u>RuleTypeld</u>」、 「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、<u>AWS CLI リファレンス</u> 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。

- 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。
- 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、 <u>アクションタイ</u> 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。

Integration

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。

山秋のために、 仄の衣に と フの ヒノルの付取と小しよ 9	比較のために、	次の表に 2	! つのモデルの特徴を示しま	ミす。
--------------------------------	---------	--------	----------------	-----

	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 力が軽減されます。	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。

- 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。
    - タイプ: 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:
- Lambda 関数を更新して CodePipeline からの入力を処理します。 ・ 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": "$3",
                    "s3Location": {
                         "bucketName": "inputBucket",
                         "objectKey": "inputKey"
                    }
                }
            }
        ],
        "outputArtifacts": [
            {
                "name": "output-art-name",
                "location": {
                    "type": "$3",
                    "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 グレータがリソースでカスタマーの構築を開始した後、構築が完了するのを待たずに、 result をcontinue として返すことで、 CodePipeline にターミナル結果がないことを示し、構築の一意の ID を CodePipeline に continuation トークンとして返します。

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、 AWS CloudFormation または コマンドラインを使用します。例については、<u>アク</u> 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 に必要です AWS KMS。

アーティファクトを Amazon S3 バケットに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 <u>PutObject</u>リク エストを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<u>ジョブデータ</u>を調べ、<u>暗号化キー</u>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、<u>SDK を使用して Amazon S3 AWS Key Management Service で を指定する AWS SDKs</u>」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、「<u>CodePipeline の概念</u>」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。組み込みの との 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、 「<u>Amazon EC2 - 仮想サーバーのホスティング</u>」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、「<u>フェデレーションの管理</u>」を参 照してください。

アクセス権限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。

・ 以下のユーザーとグループ AWS IAM Identity Center:

アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「<u>権限設定を</u> 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「<u>サード</u> <u>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</u>」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「<u>IAM</u> 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。詳細については「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセ ス権限の追加」を参照してください。

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。

```
{
  "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、「<u>パイプライン、ステージ、アクションを作成する</u>」 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、<u>チュートリアル: CodePipeline を使用した Amazon ECS</u>標 <u>準デプロイ</u> を参照してください。CodePipeline の Amazon ECS 標準デプロイアクションと ECRBuildAndPublish アクションを使用する別のチュートリアルの例については、「」を参照して ください<u>チュートリアル: CodePipeline を使用して Docker イメージを構築して Amazon ECR に</u> <u>プッシュする (V2 タイプ)</u>。
- 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、「<u>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</u> 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

- <u>Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル</u>
- 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。

+-	值
名前	#####
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。

Ε

```
{
    "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。

- 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ビルドスペック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、[<u>チュートリアル: CodePipeline を使用した継続的デプロイメント</u>] 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、「<u>Amazon ECS および CodeDeploy ブルー/グリー</u> 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、<u>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</u> 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。

+-	值
ImageURI	image_URI

imageDetail.json

以下に、イメージ URI が ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-imagerepo@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:EXAMPLE11223344556677889900bfea42ea2d3b8a1ee8329ba7e68694950afd3",
    "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": [
        "lmageTags": [
        "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。

- 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にイメージ定義ファイルを出力するコマン ドをビルドスペック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、「<u>変数の操作</u>」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。

ー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。

#### A 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 AWS 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<u>チュートリアル: AWS</u> 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、<u>例: 手動承認で変数を使用する</u> 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、「<u>パイプラインアクション</u> <u>で使用できる変数</u>」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 revisionValueで 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 ソースを有効にした Amazon S3 ソースアクションへの接続または 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CodeCommit ソースアクションと EventBridge。

# 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、「<u>パイプライン、ステージ、アクションを作成する</u>」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、「<u>チュー</u> 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": "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
- AWS 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
- <u>GitHub アクション出力変数 (GitHub (OAuth アプリ経由) アクション)</u>
- <u>S3 アクションの出力変数</u>

#### 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	PipelineE xecutionId	8abc75f0-fbf8-4f4c- bfEXAMPLE	<pre>#{codepip eline.Pip elineExec utionId}</pre>

### 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-wes t-2.amazonaws.com/ecs-repo: latest	#{SourceVariables. ImageURI}
RegistryId	EXAMPLE12233	#{SourceVariables. RegistryId}
RepositoryName	my-image-repo	#{SourceVariables. RepositoryName}

# AWS CloudFormation StackSets アクション出力変数

### AWS 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-1 111-2222-2bbb-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	<pre>#{SourceVariables. RepositoryName}</pre>

CodeStarSourceConnection アクションの出力変数

**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
AuthorDate	2019-10-29T03:32:21Z	#{SourceVariables. AuthorDate}
BranchName	開発	#{SourceVariables. BranchName}
CommitId	exampleb01f91b31	#{SourceVariables. CommitId}

変数キー	値の例	変数構文例
CommitMessage	バグを修正 (最大サイズ 100 KB)	#{SourceVariables. CommitMessage}
ConnectionArn	arn:aws:codestar-connection s:region: <i>account-id</i> :connecti on/ <i>connection-id</i>	#{SourceVariables. ConnectionArn}
FullRepositoryName	Username/GitHubRepo	<pre>#{SourceVariables. FullRepositoryName}</pre>

GitHub アクション出力変数 (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}

GitHub 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

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、 AWS 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

AWS CloudFormation 変数

変数キー	変数構文例
AWS 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。	#{DeployVariables. StackName}
CREATE_UPDATE	

#### 変数キー

変数構文例

- 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、「<u>AWS CloudForm</u> <u>ation デプロイアクションリファレンス</u>」を参照してくださ い。AWS CloudFormation 出力変数を使用するパイプライン に AWS 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<u>チュートリアル: AWS CloudForm</u> <u>ation 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</u> 成する。

#### 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。CodeBuild 環境変数を設定するには、CodePipeline で CodeBuild アク ションを編集するか、ビルド仕様に環境変数を追加します。	<pre>#{BuildVariables.EnvVar}</pre>
CodeBuild ビルドスペックに指示を追加して、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。 <u>env/</u> <u>exported-変数</u> の AWS CodeBuild ユーザーガイド を参照して ください。	

# 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 <u>PutJobSuccessResult</u> <u>API</u> 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。	#{TestVariables.te stRunId}
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、 <u>チュートリアル: Lambda 呼び出しアクショ</u> <u>ンで変数を使用する</u> を参照してください。	

# 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。以下に例を示します。

- \*. 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、myfile.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が含まれる移行ガイドについては、「<u>ポーリングパイプ</u> 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。

# 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、<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドアク</u> <u>ションの場合)</u>リソースにバックアップされた GitHub アプリベースの認証を使用します。これは GitHub 組織内に AWS CodeStar Connections アプリケーションをインストールし、GitHub でアク セスを管理できるようにします。
- 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、 「<u>CodePipeline との製品とサービスの統合</u>」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、「<u>CodeStarSourceConnection (Bitbucket</u> <u>Cloud、GitHub、GitHub Enterprise Server、GitLab.com、および GitLab セルフマネージドア</u> <u>クションの場合</u>」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、<u>https://</u> <u>docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps</u> を参照して ください。

- CLI または CloudFormation で GitHub (GitHub アプリ経由) アクションを管理する場合、個人用アクセストークンをシークレットとして Secrets Manager に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 の GitHub (GitHub App 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 などの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、Github (OAuth アプリ経由) ではソースのみをダウンロードでき 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、「<u>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</u>」 を参照してください。

このトピックのステップを使用して、GitHub (OAuth アプリ経由) 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) 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 アプリを介して) 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
- <u>ステップ 3: GitHub のソースアクションを保存する</u>

# ステップ 1: (OAuth アプリを介して) 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。
- 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 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。

 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]を選択してください。

# 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、<u>Bitbucket、GitHub、GitHub Enterprise Server、または</u> <u>GitLab.com に接続するための CodeBuild GitClone アクセス許可を追加します。</u>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、<u>チュートリアル: CodeCommit パイプラ</u> 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。

- 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。
- [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	デフォルト値
アクションがタイムアウトです。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)	AWS 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
	<ul> <li>Note</li> <li>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。</li> <li>Lambda には、CodePipeline アク ションのタイムアウトとは関係なく</li> </ul>



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 「<u>ActionDeclaration</u>」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	デフォルト値
	他のすべてのアクション: 1 時間 ③ 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 アカウント内のリージョンあたりのパイプラ インの最大数	1,000 ③ 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 ③ 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、「 <u>ソースアクション</u> <u>と変更検出方法</u> 」を参照してくださ い。 <sup>1</sup>
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

デフォルト値

AWS アカウントのリージョンあたりのカスタムア 50 クションの数

<sup>1</sup>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、<u>ポーリングパイプラインを移行する</u> (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、<u>ポーリングパイプラインを移行する</u> (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、「<u>ポーリングパイプラインをウェブフッ</u> <u>クに移行する (GitHub (OAuth アプリ経由) ソースアクション) (コンソール)</u>」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(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 (チューリッヒ)\*

イスラエル (テルアビブ)
- 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 (スペイン)
- 欧州 (ストックホルム)
- 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 (ハイデラバード)
- 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(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)
- 南米 (サンパウロ)
- 中東 (バーレーン)\*
- 中東 (UAE)
- 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 ProjectNa me 、PrimarySource 、および Environme ntVariables )。	50 文字
アクション設定値の最大長 (たとえば、Cod eCommit アクション設定中の Repositor yName の設定の値は 1000 文字未満にする必 要があります。	1000 文字
"RepositoryName": "my-repo-name- less-than-1000-characters" )	
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: AWS 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	
AWS 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「 <u>Lambda</u> クォータ」を参照してください。	

Parameter0verrides 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 AWS 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、AWS 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、「 <u>アクションタ</u> <u>イプ別の有効な入力/出力アーティファクトの</u> <u>数</u> 」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。
	<ul> <li>・以下から開始します AWS。</li> <li>・空白文字のみで構成されている</li> <li>・末尾にスペースを使用する</li> <li>・絵文字、または以下の文字を含める:?^*[\ ~!#\$%&amp;*()&gt;&lt; "'</li> </ul>

### 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:?^\*[\ ~!#\$%&\*()><|"'</li>

### トリガー

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	<ol> <li>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。</li> <li>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。</li> <li>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。</li> </ol>
出力変数と名前空間のクォータ	<ul> <li>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。</li> <li>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。</li> <li>出力変数名では、大文字と小文字が区別されま す。</li> <li>名前空間では、大文字と小文字が区別されま す。</li> <li>使用できる文字は次のとおりです。</li> <li>小文字 (a~z)。</li> <li>大文字 (A~Z)。</li> <li>大文字 (A~Z)。</li> <li>数値 (0~9)。</li> <li>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)。</li> <li>スペースなどの他の文字は使用できません。</li> </ul>

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。	
	<ul> <li>・最大文字数は 128 文字</li> <li>・小文字 (a~z)。</li> <li>・大文字 (A~Z)。</li> <li>・数値 (0~9)。</li> <li>・特殊文字 e\]+</li> <li>スペースなどの他の文字は使用できません。</li> </ul>	
	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 アクションの (OAuth アプリ経由) に関する情報を提供しま す。

#### Note

GitHub (OAuth アプリ経由) アクションを使用することはお勧めしませんが、GitHub (OAuth アプリ経由) アクションを使用する既存のパイプラインは引き続き機能します。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 <u>GitHub</u> (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、<u>https://docs.github.com/en/developers/apps/differences-</u> 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 はウェブフックを使用して、GitHub (OAuth アプリ経由) ソースアクションを使用 してパイプラインの変更検出を管理します。

Note

で GitHub (GitHub App 経由) ソースアクションを設定する場合 AWS 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「<u>AWS::CodeStarConnections::Connection</u>」に示 すように、接続リソースを設定します AWS 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以下のセクションが含まれ ています。

GitHub (OAuth アプリ経由) ソースアクションとウェブフックをパイプラインに追加する方法については、「」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追加。

 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、「」を参照してくださいGitHub (OAuth アプリ経由) ソースアクションリファレンス。

#### 🛕 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、以下を実行します。

- CodePipeline コンソール 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」ページを使用し、GitHub プロバイダー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) アクションを使用することはお勧めしませんが、GitHub (OAuth アプリ経由) アクションを使用する既存のパイプラインは引き続き機能します。GitHub 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「<u>GitHub (OAuth アプリ経由) ソースアクションを GitHub (GitHub アプリ経由)</u> 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
- <u>出力変数</u>
- <u>アクションの宣言 (GitHub の例)</u>
- <u>GitHub (OAuth) への接続</u>
- 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

アクションタイプ

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。
- AWS 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 に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「<u>GitHub 開発者 API リファ</u> 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、「<u>ポーリングパイプライン</u> をウェブフックに移行する (GitHub (OAuth アプリ経由) ソースアクション) (CLI)」または「<u>プッ</u> 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) (AWS 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
```

```
ユーザーガイド
```

```
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 AWS 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。

- <u>AWS CloudFormation ユーザーガイド AWS::CodePipeline::Webhook</u>のリソースリファレンス これには、のリソースのフィールド定義、例、スニペットが含まれます AWS CloudFormation。
- <u>AWS CloudFormation AWS::CodeStar::GitHubRepository ユーザーガイド</u>のリソースリファレンスフィールド定義、例および AWS 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 ソースでパイプラインを作成するためのビル ド仕様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 8 日

赤	西
*	Ŧ
$\sim$	$\sim$

変更	説明	日付
<u>コンピューティングタイプ、</u> 環境タイプ、環境変数タイ <u>プのコマンドアクションに</u> フィールドを追加する	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 ョンには、環境変数の新しい 型フィールドも追加されまし た。 <u>CodePipeline Comands</u> 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
<u>デフォルトのサービスロール</u> <u>ポリシーを文書化する新しい</u> <u>トピック</u>	最小サービスロールポリシー に関する情報を追加し、Cod ePipeline 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。 <u>CodePipel</u> <u>ine サービスロールポリシー</u> 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
<u>新しいCodePipeline 呼び</u> <u>出しアクション</u>	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。 <u>CodePipel</u> ine 呼び出しアクションリファ	2025 年 3 月 14 日

	<u>レンスのアクションリファレ</u> <u>ンス</u> ページを参照してくださ い。	
新しいCodeBuild ルール	ステージ条件のCodeBuild 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 きる新しいルールに関する情 報を追加しました。 <u>CodeBuild</u> <u>ルールリファレンスの新しい</u> <u>ルールリファレンス</u> ページを 参照してください。	2025 年 3 月 14 日
<u>での接続の共有 AWS</u> <u>CodeConnections</u>	を使用して接続するパイプラ インでは AWS CodeConne ctions、共有用に設定された 接続を使用できます AWS ア カウント。で共有接続を設 定できます AWS Resource Access Manager。詳細につ いては、「別の と共有されて いる接続を使用する AWS ア カウント」を参照してくださ	2025 年 3 月 6 日

い。

<u>ソースリビジョンの EventBrid</u> ge 入力変換エントリ	CodeCommit、Amazon ECR、Amazon S3 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、revisionV alue はオブジェクトキー (S3)、コミット (CodeComm it)、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
<u>AWS CodeBuild アクションの</u> <u>ビルド仕様オーバーライド</u>	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 することもできます。 <u>AWS</u> <u>CodeBuild ビルドおよびテス</u> トアクションリファレンスの <u>アクションリファレンス</u> ペー ジを参照してください。 <u>パ</u> イプライン、ステージ、アク ションの作成も参照してくだ さい。	2025 年 3 月 3 日

<u>新しいEC2デプロイアクショ</u> 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。 <u>Amazon EC2 アクション</u> 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: <u>CodePipeline を使用して EC2</u> <u>インスタンスにデプロイ</u> す る」を参照してください。	2025 年 2 月 21 日
<u>新しいEKSデプロイアクショ</u> <u>ン</u>	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。 <u>EKS デプロイ</u> 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
<u>CodePipeline の新しい</u> <u>CloudWatch メトリクスとディ</u> <u>メンション</u>	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。 <u>CodePipeline</u> <u>CloudWatch メトリクス</u> を参照 してください。	2025 年 2 月 13 日
<u>新しいCommandsルール</u>	ステージ条件 のCommands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ルール に関する情報を追加しました 。 <u>「コマンドルールリファレ</u> 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

<u>トリガーの拡張された例とリ</u> <u>ファレンス情報</u>	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、 <u>プロバイダー別のプルリクエ</u> <u>ストイベントフィルターの説</u> 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 ベントに含まれると除外に 関する詳細情報を含むトリ ガーの例を追加しました。ト リガーの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
<u>アクションカタログの新しい</u> <u>アクション</u>	これで、ECRBuildA ndPublish および 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 および InspectorScan 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 11 月 22 日
<u>障害時のステージ再試行の新</u> <u>しい自動設定</u>	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、「 <u>ステージ</u> <u>障害時の自動再試行を設定す</u> <u>る</u> 」を参照してください。	2024 年 10 月 15 日

<u>入力条件の新しい Skip 結果</u>	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。「 <u>スキッ</u> 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、「 <u>ステージ</u> 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
<u>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</u>	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 AWS 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

<u>新しい Commands アクション</u>	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「 <u>コマンドアクショ</u> ン」と「 <u>CodePipeline サービ</u> 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、「 <u>チュートリアル:コン</u> 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
<u>条件の新しい VariableC</u> <u>heck ルール</u>	ステージ条件の VariableC heck 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 「 <u>VariableCheck</u> 」を参照して ください。チュートリアルに ついては、「 <u>チュートリアル</u> : 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

<u>GitHub の接続の更新</u>	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。「 <u>GitHub</u> 接続」と「 <u>チュートリアル:</u> <u>GitHub パイプラインソースで</u> 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
<u>パイプライン JSON リファレ</u> <u>ンスとガイドの目次を再構築</u> <u>するための更新</u>	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
<u>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</u>	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、「 <u>PutWebhook</u> 」と 「 <u>ListWebhooks</u> 」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 V2 タイプのパイプラインのス 2024 年 7 月 30 日 新しいコンテンツを追加 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。 パイプラインタイプおよび関 V2 タイプのパイプラインへの 2024 年 7 月 11 日 連機能の新しいリファレンス 移行コストを評価する新しい 情報を追加 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。

<u>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</u> 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 AllowOverrideForS3 ObjectKey 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
<u>S3 ソースアクションを更新し</u> て新しい出力変数を追加	新しい出力変数 BucketNam e 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。 「 <u>Amazon S3 ソースアクショ</u> 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
<u>CloudFormationStac</u> <u>kSet および CloudForm</u> <u>ationStackInstances</u> アクションの更新	CloudFormationStac kSet および CloudForm ationStackInstance s アクションに CallAs パ ラメータが追加されました。 「 <u>アクションリファレンス</u> <u>ページ</u> 」を参照してくださ い。	2024 年 5 月 2 日
<u>ステージレベルのロールバッ</u> 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。「 <u>ステージ</u> <u>ロールバックの設定</u> 」と「 <u>概</u> <u>念</u> 」を参照してください。	2024 年 4 月 26 日

StackSets アクションと Step	StackSets アクションと Step	2024 年 3 月 27 日
Functions アクションを利用可	Functions アクションが、Co	
能なリージョンの更新	dePipeline 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。「 <u>AWS</u>	
	CloudFormation StackSets 7	
	<u>クションリファレンス</u> 」と	
	「 <u>AWS Step Functions アク</u>	
	<u>ションリファレンス</u> 」を参照	
	してください。	
マネージドポリシーの更新	AWS 管理ポリシーが更新さ	2024 年 3 月 15 日
	れAWSCodePipeline_Fu	
	llAccess ました。 <mark>AWS</mark>	
	<u>CodePipelineのAWS マネージ</u>	
	<u>ドポリシー</u> を参照してくださ	
	い。	
手動承認アクションの設定可	手動承認アクションの新し	2024 年 2 月 15 日
	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、「<u>クォータ</u>」を参

照してください。

ブランチとファイルパスによ V2 タイプのパイプラインの 2024 年 2 月 8 日 るトリガーフィルタリングの プルリクエストステータス、 サポート 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。 新しいパイプライン実行モー パイプライン実行モード 2024 年 2 月 8 日 ドのサポート 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。

<u>アクションの詳細を表示した</u> 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、「 <u>パイプラ</u> <u>インの表示 (コンソール)」、</u> 「 <u>パイプラインのアクション</u> <u>の詳細の表示</u> 」、および「 <u>パ</u> <u>イプラインの承認アクション</u> <u>の管理</u> 」を参照してくださ い。	2024年1月10日
<u>GitLab セルフマネージドのサ</u> <u>ポート</u>	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、「 <u>GitLab セルフマネージ</u> <u>ドとの接続</u> 」を参照してくだ さい。	2023 年 12 月 28 日
<u>CloudFormationStac</u> <u>kSet および CloudForm</u> <u>ationStackInstances</u> アクションの更新	CloudFormationStac kSet および CloudForm ationStackInstance s アクションに Concurren cyMode パラメータが追加 されました。「 <u>アクションリ</u> <u>ファレンスページ</u> 」を参照し てください。	2023 年 12 月 19 日

<u>CodePipeline の AWS Device</u> <u>Farm アクションパラメータの</u> 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、「 <u>AWS Device Farm</u> <u>アクションリファレンス</u> 」を 参照してください。	2023 年 12 月 18 日
<u>CodePipeline の AWS</u> <u>CloudFormation アクションの</u> 詳細なエラーメッセージのサ ポートが追加されました	AWS 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、 「 <u>AWS CloudFormation アク</u> ションリファレンス」を参照 してください。	2023 年 12 月 15 日
<u>CodePipeline のソースリビ</u> <u>ジョンオーバーライドでパイ</u> <u>プラインを起動するための更</u> <u>新</u>	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、「 <u>ソースリビジョンオ</u> <u>ーバーライドでパイプライン</u> <u>を開始する</u> 」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 CodePipeline は、アジアパシ 2023年11月13日 ジョン」を参照してくださ フィック (ハイデラバード)、 アジアパシフィック (ジャカル い。 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。「<u>イベントプ</u> レースホルダーバケットリフ アレンス」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 9 日 Amazon EventBridge のイベン Amazon EventBridge で、更 トフィールドの更新 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。
CodePipeline での新しいパイ CodePipeline で、パイプライ 2023 年 10 月 24 日 プラインタイプ V2 パイプラ ンのタイプを選択できるよう になりました。V2 タイプの イン、Git タグでのトリガー、 パイプラインで、Git タグで およびパイプライン変数の更 パイプラインを開始するため 新 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。 CodePipelineで失敗したス CodePipeline でステージが失 2023 年 10 月 17 日 敗した場合、パイプラインを 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 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。

<u>GitLab グループのサポート</u>	GitLab グループとやり取り 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、 「 <u>GitLab との接続</u> 」を参照し てください。	2023 年 9 月 15 日
<u>CodePipeline が GitLab.com</u> 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、「 <u>GitLab connectio</u> ns」および「 <u>CodeStarS</u> <u>ourceConnection アクション</u> 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
<u>CloudFormationStac</u> <u>kInstances アクションの</u> 更新	CloudFormationStac kInstances アクション に RegionConcurrencyT ype パラメータが追加 されました。CloudForm ationStackInstances アクションについては、 <u>アク</u> 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

<u>CloudFormationStac</u> <u>kSet アクションの更新</u>	CloudFormationStac kSet アクションに RegionConcurrencyT ype パラメータが追加 されました。CloudForm ationStackSet アクショ ンについては、 <u>アクションリ</u> <u>ファレンスページ</u> を参照して ください。	2023 年 7 月 24 日
<u>マネージドポリシーの更新</u>	AWS 管理ポリシーが更新さ れAWSCodePipeline_Fu llAccess ました。「 <u>AWS</u> <u>CodePipelineのAWS マネージ</u> <u>ドポリシー</u> 」を参照してくだ さい。	2023 年 6 月 21 日
<u>ポーリングパイプラインの移</u> <u>行手順の更新</u>	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、「 <u>ポーリング</u> <u>パイプラインをイベントベー</u> <u>スの変更検出の使用に移行す</u> <u>る</u> 」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\_Fu llAccess と AWSCodePi peline\_ReadOnlyAcc ess 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、「<u>AWS CodePipel</u> <u>ineAWS 管理ポリシーの更新</u>」を参照してください。

マネージドポリシーの更新

AWS 管理ポリシー AWSCodePipelineFul lAccess および AWSCodePi pelineReadOnlyAcce ss は廃止されました。 AWSCodePipeline\_Fu llAccess および AWSCodePipeline\_Re adOnlyAccess ポリシー を使用してください。「<u>AWS</u> <u>CodePipeline での AWSマ</u> <u>ネージドポリシーの更新</u>」を 参照してください。

2022年11月17日

2023年5月16日

<u> (CloudTrail を使用する手順の</u> <u>更新</u>	S3 AWS 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 AWS CloudForm ation」、「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
<u>新しくサポートされた Snyk</u> <u>との統合</u>	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 [ <u>Snyk アクションリファ</u> レンス] と [統合] を参照してく ださい。	2021 年 6 月 10 日
<u>新しくサポートされる欧州 (ミ</u> 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。「 <u>制限</u> 」および「 <u>AWS の</u> <u>サービス のエンドポイント</u> 」 トピックが更新されました。	2021 年 1 月 27 日

<u>変更の検出は、接続ソースア</u> 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 CodeStarSourceConn ection ソースアクション を更新します。[CodeStarS ourceConnection 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
<u>CodePipeline が AWS</u> <u>CloudFormation StackSets デ</u> <u>プロイアクションをサポート</u> 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 AWS CloudFormation StackSets を使用するパイプライン を作成する」では、AWS 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。「 <u>AWS CloudFormation</u> 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
<u>新しくサポートされるアジア</u> <u>パシフィック (香港) リージョ</u> <u>ン</u>	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。「 <u>制限</u> 」 および「 <u>AWS のサービス の エンドポイント</u> 」トピックが 更新されました。	2020 年 12 月 22 日

<u>CodePipeline で更新された</u> <u>EventBridge イベントパターン</u> 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
<u>CodePipeline でインバウンド</u> <u>実行のパイプラインを表示し</u> <u>ます</u>	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、 [ <u>イ</u> <u>ンバウンド実行の表示 (コン</u> <u>ソール)</u> ] と [ <u>インバウンド実行</u> <u>のステータス表示 (CLI)</u> ] を参 照してください。	2020 年 11 月 16 日
<u>CodePipeline の CodeCommi</u> <u>t ソースアクションは、完全な</u> 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

<u>CodePipeline が GitHub およ び GitHub Enterprise Server へ</u> の接続をサポート	接続を使用して、GitHub、Git Hub 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、 <u>GitHub (OAuth</u> 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	2020年9月30日
	る」を参照してください。	
<u>CodeBuild アクションは、</u> でのバッチビルドの有効化 <u>をサポートします。AWS</u> <u>CodePipeline</u>	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	2020 年 7 月 30 日

い。

AWS CodePipeline が AWS 新しいチュートリア AppConfig デプロイアクショ ル「チュートリアル: AWS ンをサポートするようになり AppConfig をデプロイプロバ イダーとして使用するパイプ ました 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。 AWS CodePipeline が Amazon AWS GovCloud (米国西部)の 2020年6月2日 プライベート Amazon VPC エ VPC in AWS GovCloud (米国 西部)をサポートするようにな ンドポイント AWS CodePipel ine を介して に直接接続でき りました 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。

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	CodePipeline で、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作成で きるようになりました。新し いチュートリアル <u>「チュート</u> 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。「 <u>AWS Step</u> 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
<u>接続の表示、一覧表示、更新</u>	コンソールでは、接続を一覧 表示、削除、更新できます。[ <u>CodePipeline の接続を一覧表</u> <u>示する]</u> を参照してください。	2020 年 5 月 21 日
<u>Connections が CLI での接続</u> <u>リソースのタグ付けをサポー</u> 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。 <u>Connections</u> IAM アクセス許可リファレン <u>ス</u> を参照してください。	2020 年 5 月 6 日
<u>CodePipeline が AWS</u> GovCloud (米国西部) で利用可 <u>能になりました</u>	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、「 <u>クォータ</u> 」を参照し てください。	2020 年 4 月 8 日

<u>クォータのトピックでは、</u> <u>どの CodePipeline Service</u> <u>Quotas が設定可能かを示しま</u> <u>す</u>	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。 <u>AWS</u> <u>CodePipeline のクォ</u> ータ」を 参照してください。	2020年3月12日
<u>Amazon ECS のデプロイアク</u> <u>ションのタイムアウトは設定</u> 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
<u>新しいトピックで、パイプラ</u> <u>イン実行を停止する方法につ</u> 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

<u>CodePipeline は接続をサポー</u> <u>トします</u>	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、B itbucket 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、 「 <u>CodePipeline での接続の使</u> <u>用</u> 」を参照してください。	2019 年 12 月 18 日
<u>セキュリティ、認証、アクセ</u> <u>スコントロールのトピックを</u> 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、「 <u>セキュリティ</u> 」 を参照してください。	2019 年 12 月 17 日
<u>新しいトピックで、パイプラ</u> <u>インで変数を使用する方法に</u> <u>ついて説明</u>	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。「 <u>変</u> 数の操作」および「 <u>変数</u> 」を 参照してください。	2019 年 11 月 14 日

<u>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</u> 「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、Dev Ops 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。

<u>CodePipeline</u> 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、 「<u>通知ルールを作成する</u>」を 参照してください。

2019 年 11 月 5 日

2019年11月11日

<u>CodePipeline で利用できる</u> <u>CodeBuild の環境変数</u>	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。 「 <u>パイプラインを作成する (コ</u> 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
<u>新しいリージョン</u>	CodePipeline が欧州 (ストッ クホルム) で利用可能になりま した。「 <u>制限</u> 」および「 <u>AWS</u> <u>のサービス のエンドポイン</u> <u>ト</u> 」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

ユーザーガイド

<u>でリソースにタグを追加で</u> きるようになりました AWS <u>CodePipeline</u>

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]、[CodePipel ine 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 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 できるようになりました AWS **CodePipeline** 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。 AWS CodePipeline は、へ AWS Serverless Applicati 2019年3月8日 のアプリケーションの発行を on Repositoryに対してサー サポートするようになりまし バーレスアプリケーションを た。 AWS Serverless Applicati 発行するパイプラインが、 CodePipeline で作成できる on Repository 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 Applicati on 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**<sub>•</sub>

2019年3月20日

 AWS CodePipeline がコン
 AWS CodePipelin

 ソールでクロスリージョンア
 ルでクロスリージョン

 クションをサポートするよう
 レでクロスリージョンを管理でき

 になりました
 ションを管理でき

 ウションを追加、約
 されて、パイプラ

 されて、パイプラ
 なる AWS リージ

 ションを追加、約
 削除する手順が方

 す。トピック [パ
 作成する]、[パイ

 集する]、[CodeFipeline が Amazon
 デプロイアクション

 S3 デプロイをサポートするよ
 デプロイアクショ

 うになりました
 ゲロスアクショ

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。 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

2019 年 1 月 16 日

2019年2月14日

<u>アル: デプロイプロバイター</u> <u>として Amazon S3 を使用し</u> <u>たパイプラインを作成する</u>] 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。「<u>CodePipeline の</u> <u>パイプライン構造リファレン</u> <u>ス</u>」のトピックも更新されて います。

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 <u>「チュートリアル:</u> 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て から、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。 「 <u>CodePipeline のパイプラ</u> 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	2018 年 12 月 6 日

<u>を使う]</u>を参照してください。

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 [CodePipel ine 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 <u>「クロスリー</u> ジョンアクションの追加」に は、AWS CLI または AWS 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

<u>AWS CodePipeline が Service</u> <u>Catalog と統合されるようにな</u> 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

2018年6月30日

<u>AWS CodePipeline ユーザー</u> ガイドの更新通知が RSS で利 <u>用可能に</u>

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 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。

## 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、Cod ePipeline 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、「 <u>変更</u> <u>の検出に Webhook を使用するように GitHub パイプライ</u> <u>ンを設定する</u> 」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、Cod ePipeline は Amazon CloudWatch Events ルールと、Amaz on 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、「 <u>ソースアクション</u> <u>と変更検出方法</u> 」を参照してください。 <u>チュートリアル: シンプルなパイプラインを作成する (S3</u> <u>バケット)</u> 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。 <u>パイプライン、ステー</u> 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、「 <u>CodePipeline でパイプラインを編集す</u> <u>る</u> 」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。 「 <u>AWS CodePipeline のクォータ</u> 」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、 <u>CodePipeline との製品とサービスの統合、パイプラ</u> イン、ステージ、アクションを作成する、 <u>CodePipeline パ</u> 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、Cod ePipeline 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、「 <u>ソースアクションと変更検</u> 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。	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、 「 <u>CodePipeline イベントのモニタリング</u> 」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。Pol IForSourceChanges 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、「 <u>CodePipeline で</u> 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。 「 <u>AWS CodePipeline のクォータ</u> 」トピックおよび「 <u>リー</u> <u>ジョンおよびエンドポイント</u> 」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。 「 <u>AWS CodePipeline のクォータ</u> 」トピックおよび「 <u>リー</u> <u>ジョンおよびエンドポイント</u> 」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、「 <u>CodePipeline でパイプラインと詳</u> 細を表示する」、「 <u>CodePipeline 許可リファレンス</u> 」、 「 <u>AWS CodePipeline のクォータ</u> 」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	「 <u>Nouvola</u> 」が「 <u>テストアクションの統合</u> 」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 <u>CodePipeline との製品とサービスの統</u> 合、 <u>CodePipeline でパイプラインを編集する</u> 、および <u>CodePipeline パイプライン構造リファレンス</u> は、このサ ポートを反映するように更新されました 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、「 <u>CodePipel</u> ine 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 <u>AWS Identity and Access Managemen</u> <u>t (IAM)</u> 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	2017 年 2 月 8 日
	トからのアーティファクトの配置と取得、ハイフラインへ の AWS 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。	
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。「 <u>AWS CodePipeline のクォータ</u> 」トピック および「 <u>リージョンおよびエンドポイント</u> 」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。「 <u>AWS CodePipeline のクォータ</u> 」トピッ クおよび「 <u>リージョンおよびエンドポイント</u> 」トピックは 更新されています。	2016 年 7 月 12 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。	2016 年 12 月 1 日
	トピック <u>CodePipeline との製品とサービスの統合</u> 「」、 <u>パ イプライン、ステージ、アクションを作成する</u> 「」、「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 <u>CodePipeline パイプライン構造リファレ</u> ンスされました AWS CodeBuild。	
	CodePipeline AWS 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「 <u>CodePipeline との製品とサービスの統合</u> 」は更新 され、この新しいサポートが反映されています。	
	<u>CodePipeline との製品とサービスの統合</u> 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。	
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。「 <u>AWS CodePipeline のクォータ</u> 」トピックおよ び「 <u>リージョンおよびエンドポイント</u> 」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部として AWS 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 <u>CodePipeline との製品とサービスの統合「」、パ</u> 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 <u>CodePipeline パイプライン構造リファレ</u> ンスされました AWS CloudFormation。	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。「 <u>AWS CodePipeline の</u> <u>クォータ</u> 」トピックおよび「 <u>リージョンおよびエンドポイ</u> <u>ント</u> 」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。「 <u>AWS CodePipel</u> <u>ine のクォータ</u> 」トピックおよび「 <u>リージョンおよびエン</u> <u>ドポイント</u> 」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。「 <u>AWS CodePipeline のクォータ</u> 」ト ピックおよび「 <u>リージョンおよびエンドポイント</u> 」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	「 <u>パイプライン、ステージ、アクションを作成する</u> 」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	「 <u>手動の承認アクションをステージに追加する</u> 」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「 <u>パイプラインのモニタリング</u> 」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「 <u>手動の承認アクションをステージに追</u> 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。「 <u>AWS CodePipeline のクォータ</u> 」ト ピックおよび「 <u>リージョンおよびエンドポイント</u> 」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	<u>パイプライン、ステージ、アクションを作成する</u> 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、「 <u>CodePipeline パイプ</u> ライン構造リファレンス」「Authentication and Access Control」「 <u>CodePipeline との製品とサービスの統合</u> 」 「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 AWS 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「 <u>チュートリアル:シンプルなパイプライ</u> <u>ンを作成する (CodeCommit リポジトリ)</u> 」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」「 <u>CodePipeline との製品とサービスの統合」「チュートリ</u> <u>アル:4ステージのパイプラインを作成する</u> 」「 <u>CodePipel</u> <u>ine のトラブルシューティング</u> 」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「 <u>CodePipeline のパイプラインで AWS</u> <u>Lambda 関数を呼び出す</u> 」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「 <u>CodePipeline との製品とサービスの統</u> <u>合</u> 」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「 <u>CodePipeline との製品と</u> <u>サービスの統合</u> 」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、「 <u>チュートリアル:4ステージのパ</u> <u>イプラインを作成する</u> 」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。「 <u>AWS CodePipeline のクォータ</u> 」ト ピックが更新されました。リンクが「 <u>リージョンおよびエ</u> <u>ンドポイント</u> 」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「 <u>CodePipeline 用に Amazon S3 に保存し</u> 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「 <u>別の AWS アカウントのリソースを使用するパイプ</u> ラインを CodePipeline で作成する」が追加されました。 「Authentication and Access Control」、「 <u>例 8: 別のアカ</u> ウントに関連付けられた AWS リソースをパイプラインで 使用する」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	「 <u>CodePipeline でカスタムアクションを作成および追加</u> <u>する</u> 」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

変更	説明	変更日
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、 <u>CodePipeline のトラブルシューティング</u> のトピック が更新されました。	2015 年 8 月 11 日
トピックの更新	「 <u>Service role for CodePipeline</u> 」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	「 <u>CodePipeline のトラブルシューティング</u> 」トピックが追 加されました。「 <u>チュートリアル: 4 ステージのパイプラ</u> <u>インを作成する</u> 」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	「 <u>チュートリアル: シンプルなパイプラインを作成する</u> ( <u>S3 バケット)</u> 」および「 <u>チュートリアル: 4 ステージのパ</u> <u>イプラインを作成する</u> 」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「 <u>チュートリアル: シンプルなパイプラインを作成す</u> <u>る (S3 バケット)</u> 」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「 <u>AWS CodePipel</u> <u>ine のクォータ</u> 」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、「<u>CodePipeline パイプライン構造リファレンス</u>」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	AWS CloudForm ation 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	<u>ステージ障害</u> <u>時の自動再試</u> 行を設定する	RetryConf iguration		
2024 年 10 月 15 日	beforeEntry 条件の結果を スキップする	<u>スキップ</u> <u>結果と</u> VariableC <u>heck ルー</u> ルを使用した 入力条件の作 成 (コンソー ル)	FailureCo nditions		
2024 年 10 月 3 日	新しい Compute カ テゴリのコマ ンドアクショ ン	<u>コマンドアク</u> <u>ションリファ</u> <u>レンス</u>	ActionDec laration		
2024 年 8 月 28 日	ステージ条件 の beforeEnt ry タイプ	<u>ステージの条</u> <u>件を設定する</u>	BeforeEnt ryConditions	AWS::Code Pipeline: :Pipeline BeforeEnt ryConditions	<u>beforeEntry</u>

AWS	CodePipe	line
-----	----------	------

表の更新日	機能	CodePipeline ユーザーガイ ド	CodePipeline API ガイド	AWS CloudForm ation 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	<u>ステージの条</u> <u>件を設定する</u>	<u>SuccessCo</u> nditions	AWS::Code Pipeline: :Pipeline SuccessCo nditions	<u>onSuccess</u>
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	<u>ステージロー</u> <u>ルバックの設</u> 定	<u>RollbackS</u> tage	<u>onFailure</u>	<u>onFailure</u>
2024 年 2 月 8 日	PARALLEL および QUEUED の 実行モード	<u>パイプライン</u> <u>実行モードを</u> 設定または変 更する	PipelineE xecution	Execution Mode	<u>実行モード</u>
2023 年 11 月 17 日	ソースの上書 き	<u>ソースリビ</u> ジョンオー バーライドで パイプライン を開始する	SourceRev isionOverride	該当なし	該当なし
2023 年 10 月 24 日	パイプライン のタイプ	<u>適切なパイプ</u> <u>ラインのタイ</u> <u>プの選択</u>	PipelineD eclaration	<u>PipelineType</u>	<u>enum</u> PipelineType
2023 年 10 月 24 日	パイプライン レベルの変数	<u>チュートリア</u> <u>ル: パイプラ</u> <u>インレベルの</u> 変数を使用す る	<u>PipelineV</u> <u>ariable</u>	AWS::Code Pipeline: :Pipeline VariableD eclaration	<u>パイプライン</u> 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	AWS CloudForm ation 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	<u>トリガーと</u> 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:パイプラ インを開始す ストの ブランチ名を フィルタリン グする (V2 タ イプ)	ActionDec laration	AWS::Code Pipeline: :Pipeline PipelineT riggerDec laration	[ <u>Trigger]</u> (ト リガー)
2023 年 10 月 17 日	ステージを再 試行する	<u>失敗したス</u> <u>テージまた</u> <u>は失敗したア</u> <u>クションのス</u> <u>テージ再試行</u> の設定	ActionDec laration	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。