ユーザーガイド

AWS CodePipeline

API バージョン 2015-07-09

AWS CodePipeline: ユーザーガイド

Table of Contents

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段階を迅速にモデル化して設定できます。 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 CodePipeline を自動化します。の料金については CodePipeline、「 の料金」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 は、ソフトウェアの構築、テスト、本番環境へのデプロイを自動化する継続的デリバリーサービスです。

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

継続的統合 は、チームのメンバーがバージョン管理システムを使用し、頻繁にマスターブランチなどの同じ場所に作業を統合するソフトウェア開発のプラクティスです。各変更は、可能な限り迅速に統合エラーを検出するために構築され、検証されます。継続的な統合は、ソフトウェアのリリースプ

ロセス全体を本番稼働まで自動化する継続的な配信と比較して、コードの自動構築とテストに重点を置いています。

詳細については、「 での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: によるソフトウェアデリバリーの加速 DevOps」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。

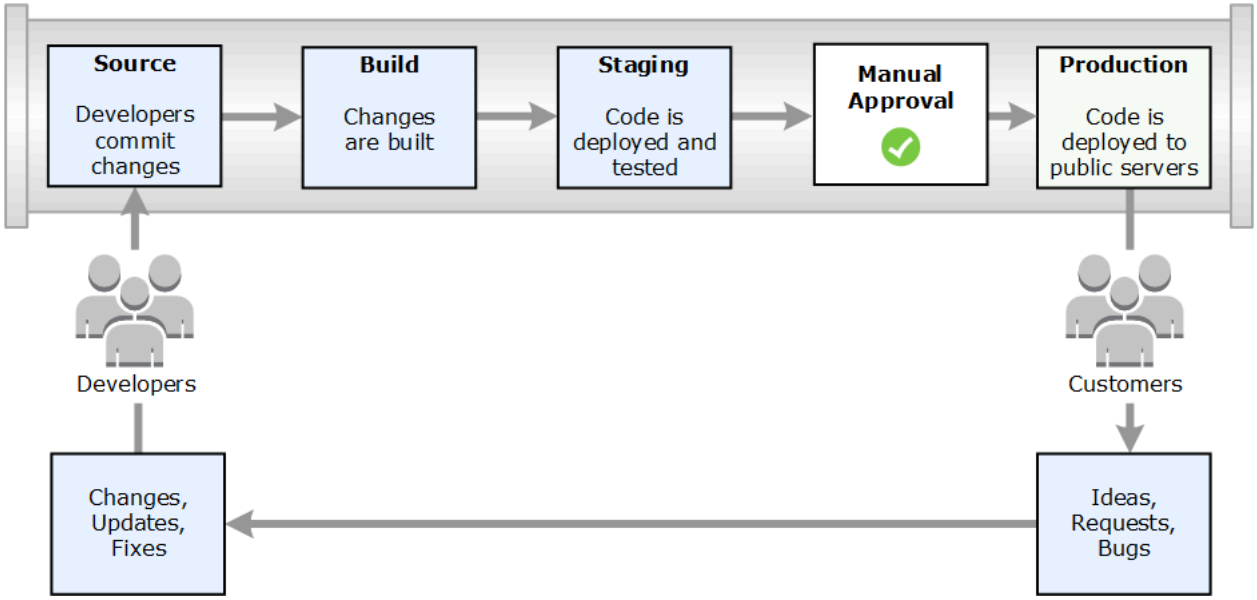
で何ができますか CodePipeline？

CodePipeline を使用すると、クラウドでアプリケーションを自動的に構築、テスト、デプロイできます。具体的な内容は以下のとおりです：

- リリースプロセスを自動化: 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エンドツーエンドで CodePipeline 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。
- 一貫したリリースプロセスを確立する: すべてのコード変更に対して一貫した一連のステップを定義します。 CodePipeline は、条件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 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 CodePipeline、「」を参照してくださいとの製品とサービスの統合 CodePipeline。
- 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。

の簡単な説明 CodePipeline

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



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 は変更を自動的に検出します。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、 は統合テストやロードテストなど、より多くのテスト CodePipeline 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ると、 はテスト済みおよび承認済みのコードを本番稼働用インスタンスに CodePipeline デプロイします。

CodePipeline は、 CodeDeploy、、または を使用して EC2 インスタンスにアプリケーションをデプロイ CodePipeline できます AWS OpsWorks Stacks。また AWS Elastic Beanstalk、Amazon ECS を使用して サービス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。デベロッパーは、 で提供される統合ポイントを使用して、ビルドサービス、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 CodePipeline 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。

の使用を開始するにはどうすればよいですか CodePipeline？

の使用を開始するには CodePipeline：

1. CodePipeline の概念 セクションを読んで、 の CodePipeline 仕組みについて説明します。

2. CodePipeline 「」の手順に従って、 を使用する準備をしますの開始方法 CodePipeline。
3. CodePipeline チュートリアル チュートリアルの手順に従って CodePipeline 、 を試してください。
4. CodePipeline 「」の手順に従って、新規または既存のプロジェクトに を使用しますでパイプラインを作成する CodePipeline。

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ル化と設定が簡単になります AWS CodePipeline。を使用する際に知っておくべき概念をいくつか紹介します CodePipeline。

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

では、次の用語が使用されます CodePipeline。

トピック

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

パイプライン

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。

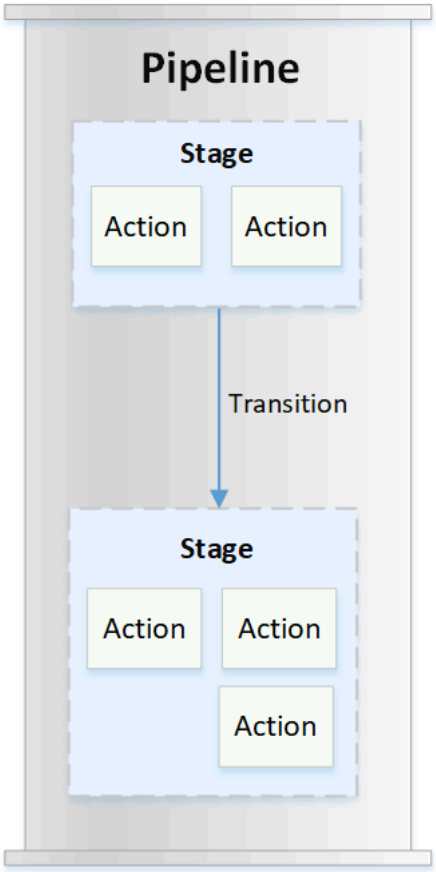
ステージ

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、アプリケーションアーティファクトに対して実行されるアクションが含まれ

ます。ソースコードはアーティファクトの例です。ステージは、ソースコードが構築され、テストが実行されるビルドステージである場合もあれば、コードをランタイム環境にデプロイするデプロイステージの場合もあります。各ステージは、連続または並列のアクションで構成されています。

Transitions

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



アクション

アクションは、アプリケーションコードに対して実行される一連の操作であり、アクションがパイプライン内で指定されたポイントで実行されるように設定されます。これには、コード変更によるソースアクション、インスタンスにアプリケーションをデプロイするためのアクションなどが含まれ

ます。たとえば、デプロイステージには、 AWS Lambdaや Amazon EC2 などのコンピューティングサービスにコードをデプロイするデプロイアクションが含まれている場合があります。

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

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

パイプライン実行

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

パイプラインは同時に複数の実行を処理できますが、パイプラインステージは一度に 1 つの実行のみを処理します。これを行うために、ステージは実行を処理している間ロックされます。2 つのパイプライン実行は、同時に同じステージを占めることはできません。占有ステージに入るのを待つ実行は、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、「インバウンド実行の仕組み」を参照してください。

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

詳細については、「」を参照してくださいPipelineExecution。

停止された実行

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

パイプラインの実行を停止する方法は 2 つあります。

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

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

失敗した実行

実行が失敗した場合、実行は停止し、パイプラインを完全に通過しません。ステータスは 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。失敗したステージは、前回の正常な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、実行がロックされたステージに入るのを待っている場合、より最近の実行が追いついて置き換えられる可能性があります。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、「」を参照してくださいSUPERSEDED モードでの実行の処理方法。

ステージオペレーション

パイプラインの実行がステージを通過すると、ステージはステージ内のすべてのアクションを完了中です。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、「」を参照してくださいSUPERSEDED モードでの実行の処理方法。

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、「」を参照してくださいStageExecution。ステージは、指定した前回の正常な実行にロールバックできます。ステージは、「」で説明されているように、障害発生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ロールバックの設定。詳細については、「」を参照してくださいRollbackStage。

アクション実行

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

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

実行タイプ

パイプラインまたはステージ実行は、標準実行またはロールバック実行のいずれかです。

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です。パイプラインのロールバックには、ロールバックするステージと、ロールバックするターゲット実行としてステージを正常に実行します。ターゲットパイプラインの実行は、再実行するステージのソースリビジョンと変数を取得するために使用されます。

アクションタイプ

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

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

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

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

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 bucket 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。 は、アーティファクトをアーティファクトストア CodePipeline にコピーし、そこでアクションがそれらを取得します。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、これはコミットです。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、パイプライン ARN をイベントルールのターゲットとして追加 CloudWatch した Amazon のイベントリソースで設定する必要があります。Amazon CloudWatch Events は、 CodeCommit または S3 ソースアクションを持つ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、GitHub.com、Enterprise Server、.com、 GitHub GitLab GitLab self-managed、Bitbucket Cloud などのサードパーティーソースプロバイダーとのパイプラインの開始に Git タグを使用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。

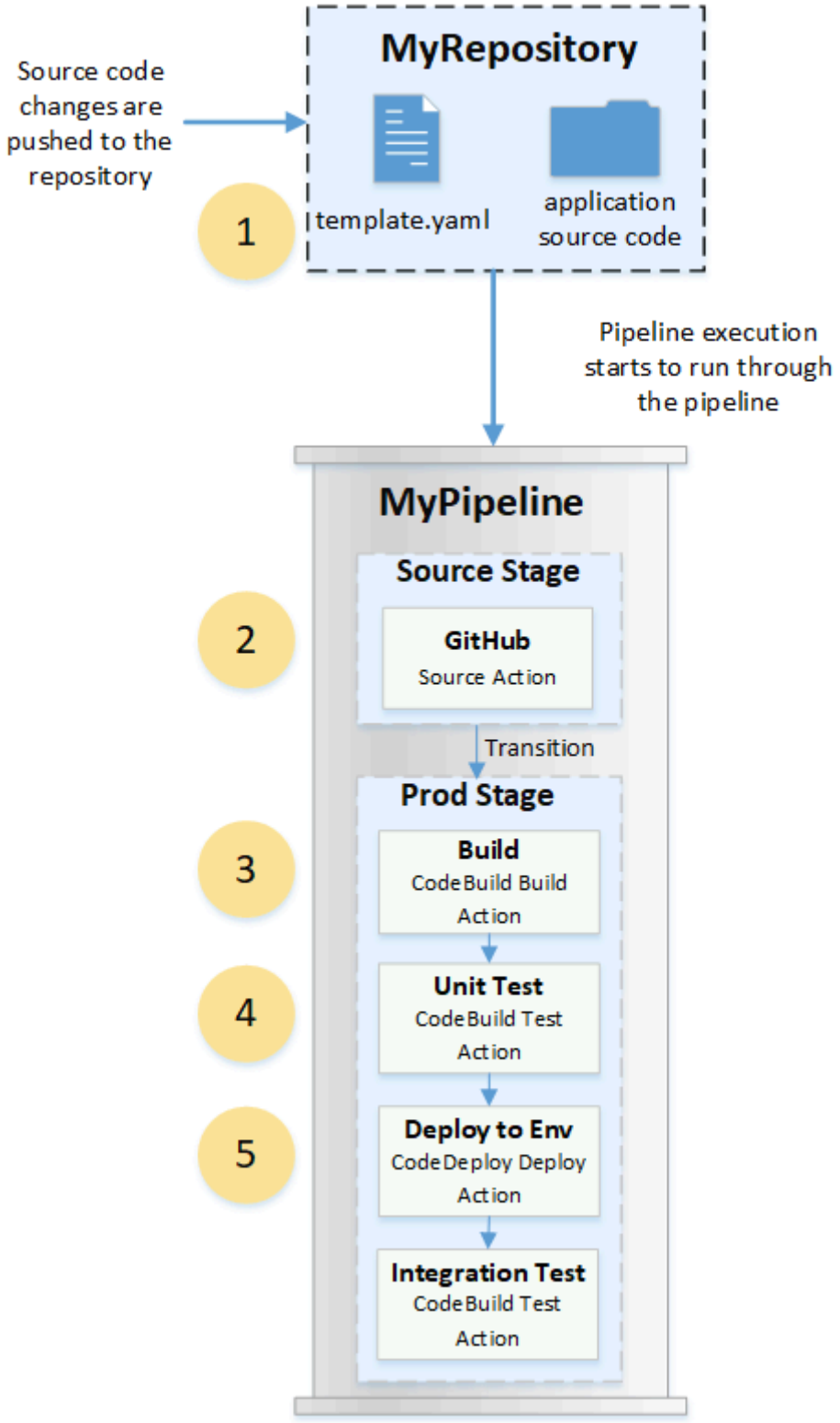
トリガーについての詳細は、「でパイプラインを開始する CodePipeline」を参照してください。Git タグをパイプラインのトリガーとして使用する手順を説明するチュートリアルについては、「チュートリアル: Git タグを使用してパイプラインを開始する」を参照してください。

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」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 ステージのパイプラインにはソースステージ と Prod という 2 番目のステージがあります。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、デベロッパーのグループが、 という GitHub リポジトリ内の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セットアップしています MyRepository。



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

1. アプリケーションのソースコードは、パイプラインの GitHub ソースアクションとして設定されたリポジトリに保持されます。デベロッパーがコミットをリポジトリにプッシュすると、 はプッシュされた変更 CodePipeline を検出し、パイプラインの実行はソースステージ から開始します。
2. GitHub ソースアクションは正常に完了します (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) 。リポジトリのアプリケーションファイルである GitHub ソースアクションによって生成された出力アーティファクトは、次のステージのアクションによって処理される入力アーティファクトとして使用されます。
3. パイプラインの実行は、ソースステージから本番ステージに移行します。Prod Stage の最初のアクションは、 で作成 CodeBuild され、パイプラインのビルドアクションとして設定されたビルドプロジェクトを実行します。ビルドタスクは、ビルド環境イメージをプルし、仮想コンテナにウェブアプリケーションをビルドします。
4. Prod ステージの次のアクションは、 で作成 CodeBuild 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ユニット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すると、ステージの最後のアクションは、 で作成 CodeBuild 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、一連の変更 CodePipeline を処理する方法の概要を説明します。 CodePipeline は、パイプラインが手動で開始されたとき、またはソースコードに変更が加えられたときに開始される各パイプライン実行を追跡します。 は、次の実行モード CodePipeline を使用して、各実行がパイプラインを通過する方法を処理します。

- SUPERSEDED モード: より最近の実行は、古い実行を上書きする可能性があります。これがデフォルトです。

- QUEUED モード: 実行はキューに入れられた順序で 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、実行は同時に実行され、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のを待ってから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。

パイプライン実行の開始方法

ソースコードを変更するか、パイプラインを手動で開始するときに実行を開始できます。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のソースリビジョンの処理方法

ソースコードの変更 (ソースリビジョン) で始まるパイプラインの実行ごとに、ソースリビジョンは次のように決定されます。

- CodeCommit ソースを持つパイプラインの場合、HEAD はコミットがプッシュされた時点で によって CodePipelineクローンされます。例えば、コミットがプッシュされ、実行 1 のパイプラインが開始されます。2 番目のコミットがプッシュされると、実行 2 のパイプラインが開始されます。

> **Note**
>
> CodeCommit ソースを持つ PARALLEL モードのパイプラインの場合、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 のクローンを作成します。詳細については、「CodeCommit または PARALLEL モードでの S3 ソースリビジョンが 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。

- S3 ソースを持つパイプラインの場合、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が更新されたときにイベントが生成され、実行 1 のパイプラインが開始されます。2 番目のバケット更新のイベントが実行された時点で、実行 2 のパイプラインが開始されます。

> **Note**
>
> S3 ソースを持つ PARALLEL モードのパイプラインの場合、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されます。詳細については、「CodeCommit または PARALLEL モードでの S3 ソースリビジョンが 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。

- Bitbucket など、接続ソースを持つパイプラインの場合、コミットがプッシュされた CodePipeline 時点で HEAD が によってクローンされます。例えば、PARALLEL モードのパイプラインの場合、コミットがプッシュされ、実行 1 のパイプラインが開始され、2 番目のパイプラインの実行では 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、「でパイプラインの実行を停止する CodePipeline」を参照してください。

パイプラインの実行を停止する方法は 2 つあります。

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

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

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

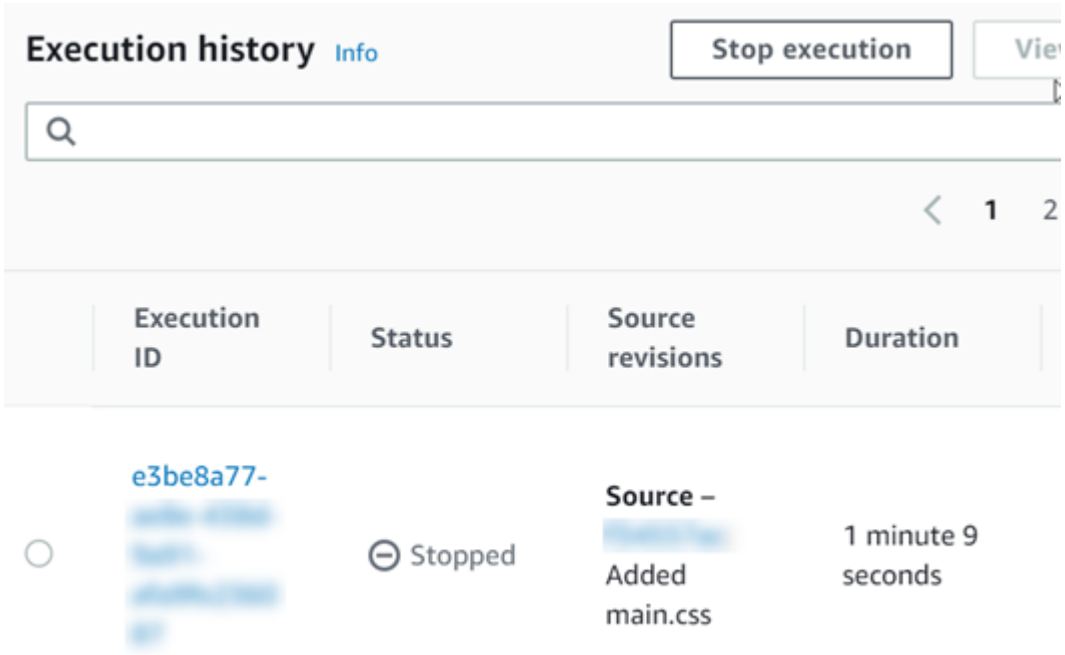
オプション 1: [Stop and wait (停止して待機)]

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

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

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

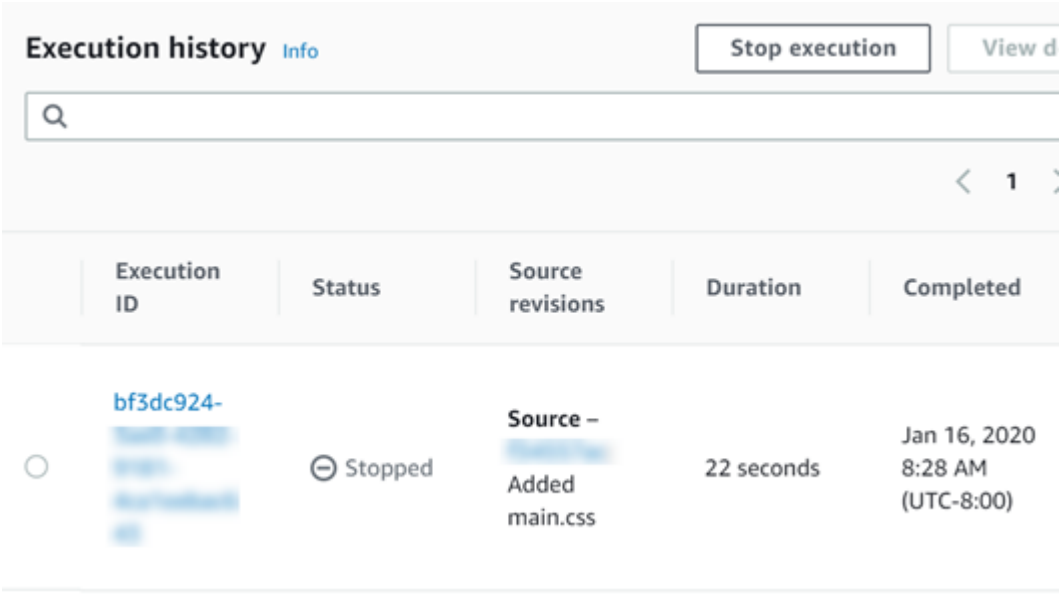
   履歴ビューでは、進行中のアクションが完了した後、実行ステータスは [停止] になります。



オプション 2: [Stop and abandon (停止して中止)]

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

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



パイプライン実行を停止するユースケース

パイプラインの実行を停止するには、[stop the wait (待機して停止)] オプションを使用することをお勧めします。このオプションは、パイプラインで障害や out-of-sequence タスクが発生する可能性を回避するため、より安全です。でアクションが中止されると CodePipeline、アクションプロバイダーはアクションに関連する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。

out-of-sequence 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大きなファイル (1GB) 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および中止することを選択した場合、アクションは で中止されますが CodePipeline、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。

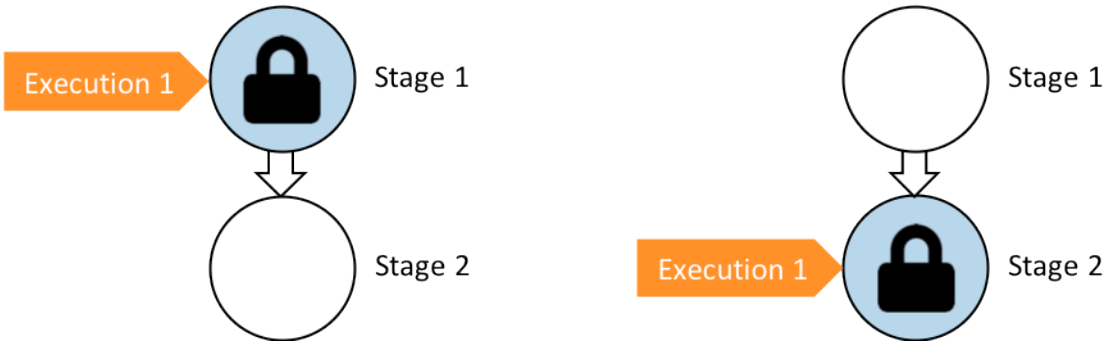
カスタムアクションがある場合は、[Stop and abandon (停止して中止)] オプションを使用できます。例えば、バグ修正のための新しい実行を開始する前に、完了する必要がない作業を含むカスタムアクションを中止できます。

SUPERSEDED モードでの実行の処理方法

実行を処理するための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。次のルールは、SUPERSEDED モードのパイプラインで実行を処理するために使用されます。

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。

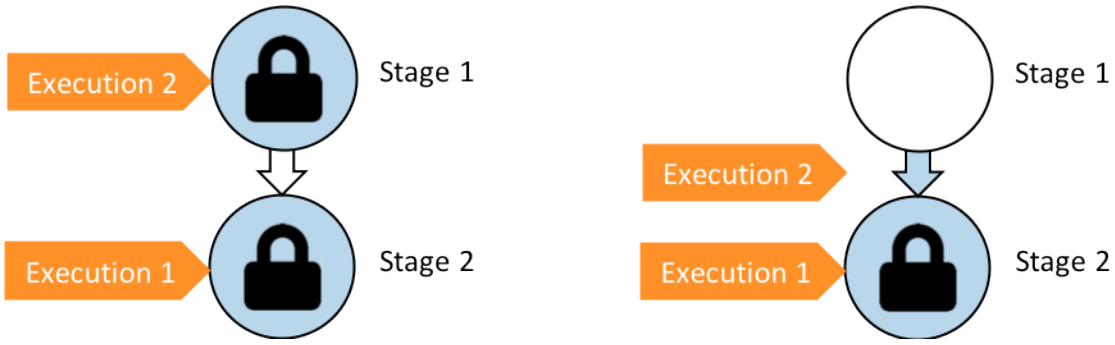


前 : Stage 1 is locked as Execution 1 enters. 後 : Stage 2 is locked as Execution 1 enters.

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。

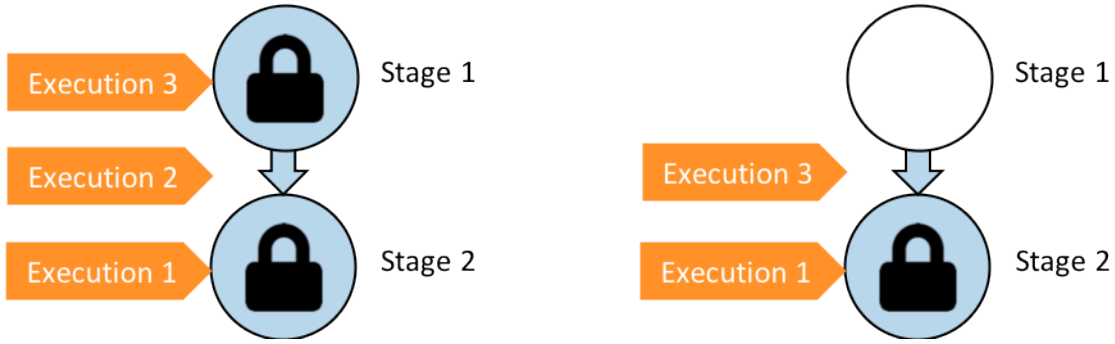
> **Note**
>
> 実行を停止する前に、ステージの前でトランジションを無効にすることをお勧めします。このようにすれば、実行が停止したためにステージのロックが解除されたときに、ステージは後続のパイプライン実行を受け付けません。



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

ルール 3: 待機中の実行は、より新しい実行に置き換えられる

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



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

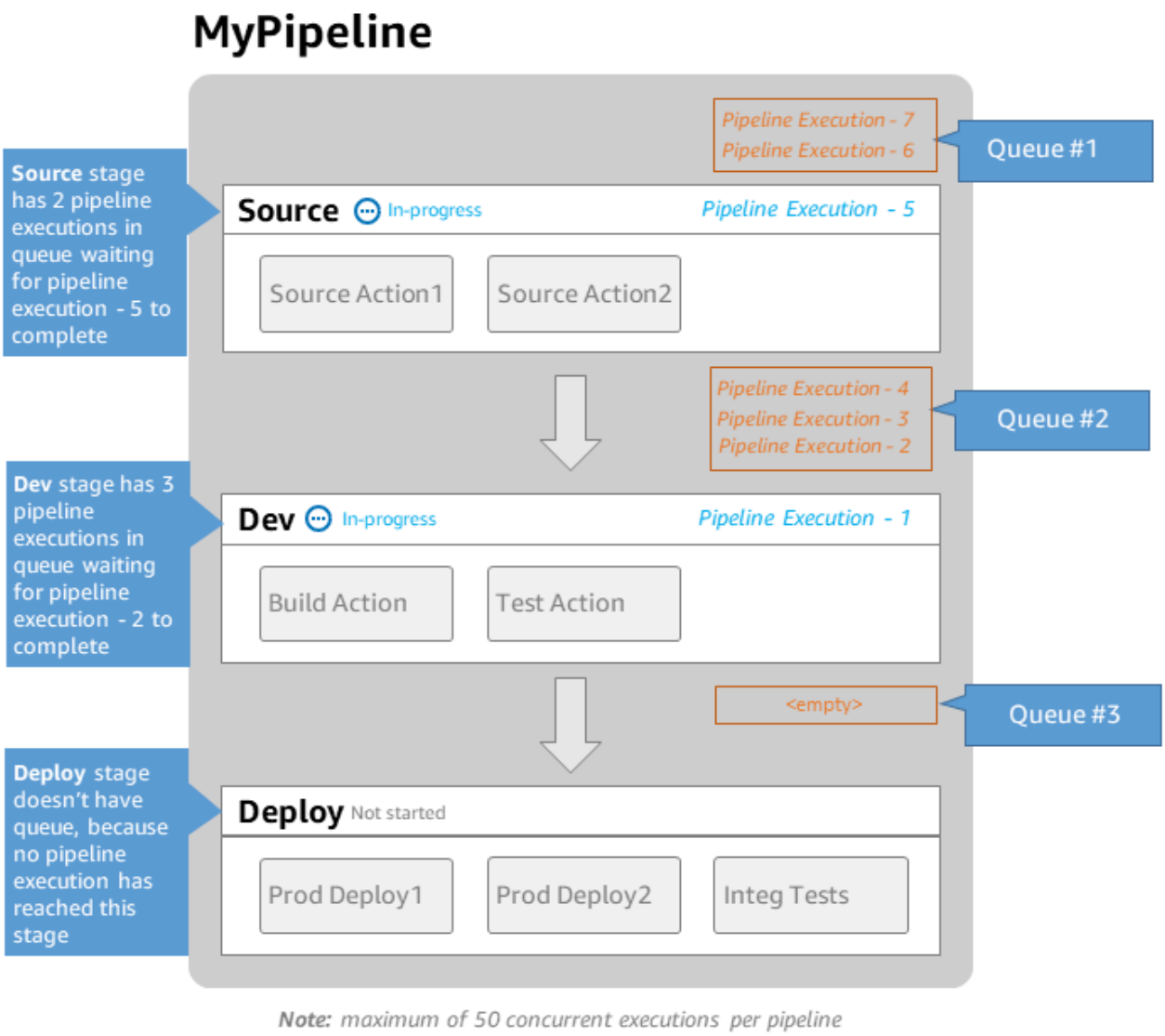
実行モードの表示と切り替えに関する考慮事項の詳細については、「」を参照してくださいパイプライン実行モードを設定または変更する。実行モードのクォータの詳細については、「」を参照してくださいのクォータ AWS CodePipeline。

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、実行の処理中にステージはロックされますが、待機中の実行は、すでに開始されている実行を上書きしません。

待機実行は、ステージに到達した順序でエントリポイントからロックされたステージに収集され、待機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持つことができます。キューに入れられた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 モードと同じままです。実行がステージを完了すると、ステージはロック解除され、次の実行の準備が整います。

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



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

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のを待ってから開始しません。キューはありません。パイプラインで並列実行を表示するには、実行履歴ビューを使用します。

各機能に独自の機能ブランチがあり、他のユーザーが共有していないターゲットにデプロイする開発環境では、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、「」を参照してくださいパイプライン実行モードを設定または変更する。実行モードのクォータの詳細については、「」を参照してくださいのクォータ AWS CodePipeline。

パイプラインのフローを管理する

パイプラインの実行のフローは、次の方法で制御できます。

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

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

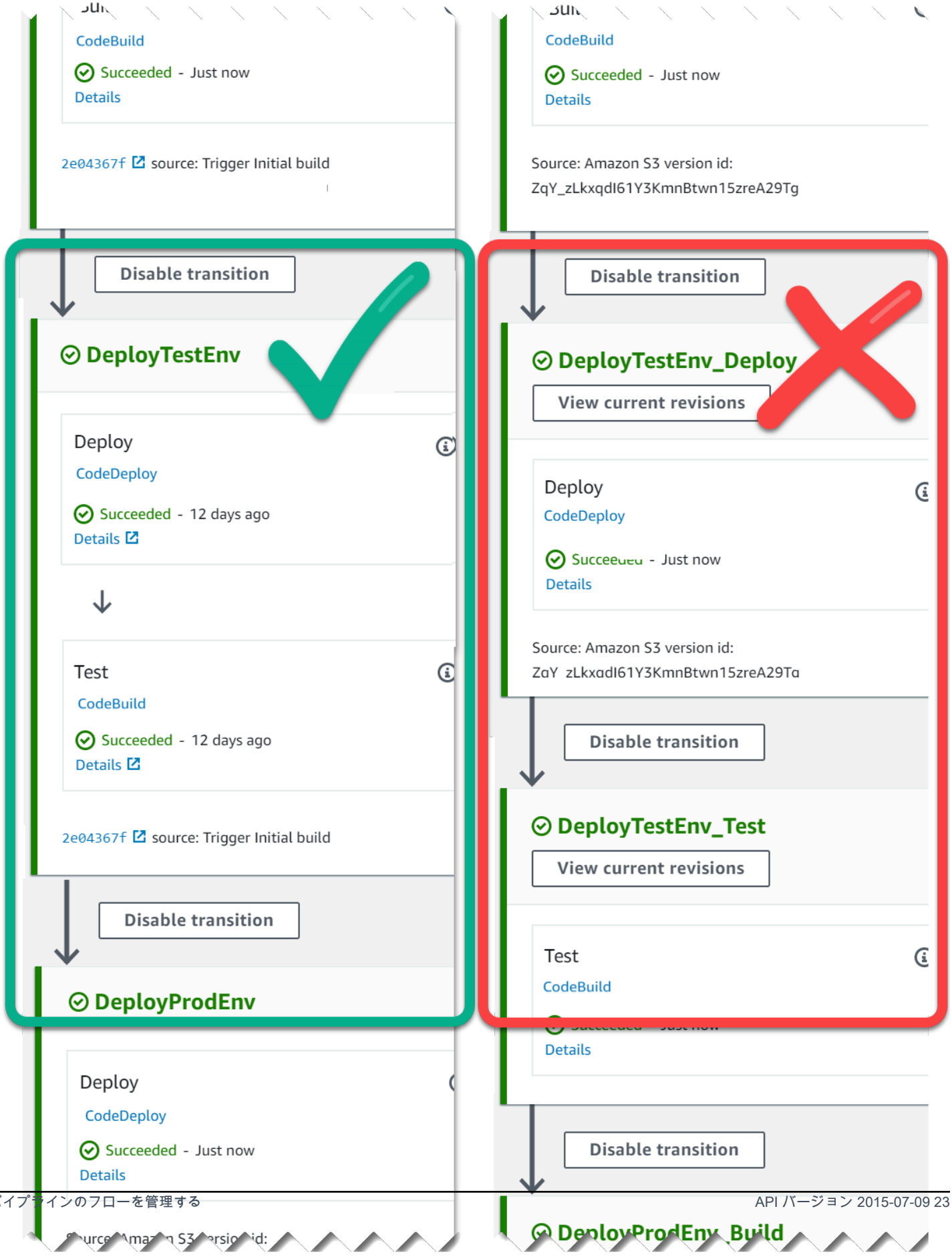
  > **Note**
  >
  > 承認アクションのあるステージは、承認アクションが承認または却下されるか、タイムアウトするまでロックされます。タイムアウトした承認アクションは、失敗したアクションと同じ方法で処理されます。

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

推奨されるパイプライン構造

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

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


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

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

インバウンド実行の仕組み

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

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

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

インバウンド実行は、以下の考慮事項に注意して動作します。

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

インバウンド実行を表示または停止できます。

入力および出力アーティファクト

CodePipeline は、開発ツールと統合して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段階で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用にアプリケーションの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) であり、ファイルは一般的に ZIP ファイルとして提供されます。例えば、アーティファクト名： SampleApp_Windows.zip の例です。ソースアクションの出力アーティファクトであるアプリケーションソースコードファイルは、ソースアクションからの出力アーティファクトであり、ビルドアクションなどの次のアクションの入力アーティファクトでもあります。別の例として、ビルドアクションでは、ソースアクションからのアプリケーションソースコードファイル (入力アーティファクト) に対して、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。アクションの 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AWS CodeBuildの CodeBuild 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、pipeline. CodePipeline zips 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、ステージのアクションタイプに応じて入力アーティファクトまたは出力アーティファクトのファイルを転送します。

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

例:

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

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

- コンソールを使用して最初のパイプラインを作成する AWS アカウント と、 は同じ と に Amazon S3 バケット CodePipeline 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、 はバケットにそのパイプラインのフォルダ CodePipeline 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保証する 12 桁の乱数です。

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

CodePipeline はアーティファクト名を切り捨てるため、一部のバケット名は類似しているように見えます。アーティファクト名が切り捨てられているように見えても、 は切り捨てられた名前のアーティファクトの影響を受けない方法でアーティファクトバケットに CodePipeline マッピングします。パイプラインは正常に動作します。これは、フォルダやアーティファクトでは問題となりません。パイプライン名には 100 文字の制限があります。アーティファクトフォルダ名は、短縮されたように見えても、パイプラインに対して依然として一意です。

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

パイプラインのタイプ

CodePipeline には、パイプラインの特徴と料金が異なる次のパイプラインタイプが用意されているため、アプリケーションのニーズに合わせてパイプラインの特徴とコストを調整できます。

- 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。

の料金については CodePipeline、「 の料金」を参照してください。

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

適切なパイプラインのタイプの選択

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

以下に、各タイプのパイプラインのユースケースと特徴をまとめました。

	V1 タイプ	V2 タイプ
特性		
ユースケース	• 標準デプロイ	• 実行時にパイプラインレベルの変数を渡すように設定したデプロイ • パイプラインが Git タグで開始されるように設定したデプロイ
アクションレベルの変数	サポート	サポート
PARALLEL 実行モード	サポート外	サポート
パイプラインレベルの変数	サポート外	サポート
QUEUED 実行モード	サポート外	サポート
パイプラインステージのロールバック	サポート外	サポート
ソースリビジョンオーバーライド	サポート外	サポート
Git タグ、プルリクエスト、ブランチ、またはファイルパスのトリガーとフィルタリング	サポート外	サポート

の料金については CodePipeline、「 の料金」を参照してください。

V1 タイプパイプラインで次のスクリプトを使用して、パイプラインを V2 タイプパイプラインに移動するコストを分析します。

パイプラインタイプのコスト分析を実行するには (スクリプト)

1. 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
```

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

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

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

def analyze_cost_in_v2(pipeline_name):
    if codepipeline.get_pipeline(name=pipeline)['pipeline']['pipelineType'] ==
 'V2':
        raise Exception("Provided pipeline is already of type V2.")
    total_action_executions = 0
    total_blling_action_executions = 0
    total_action_execution_minutes = 0
    cost = 0.0
    hasNextToken = True
    nextToken = ""

    while hasNextToken:
        if nextToken=="":
            response =
 codepipeline.list_action_executions(pipelineName=pipeline_name)
        else:
            response =
 codepipeline.list_action_executions(pipelineName=pipeline_name,
 nextToken=nextToken)
        if 'nextToken' in response:
            nextToken = response['nextToken']
        else:
            hasNextToken= False
        for action_execution in response['actionExecutionDetails']:
            start_time = action_execution['startTime']
            end_time = action_execution['lastUpdateTime']
```

```
            if (start_time < (datetime.now(timezone.utc) - timedelta(days=30))):
                hasNextToken= False
                continue
            total_action_executions += 1
            if (action_execution['status'] in ['Succeeded', 'Failed', 'Stopped']):
                action_owner = action_execution['input']['actionTypeId']['owner']
                action_category = action_execution['input']['actionTypeId']
['category']
                if (action_owner == 'Custom' or (action_owner == 'AWS' and
 action_category == 'Approval')):
                    continue

                total_blling_action_executions += 1
                action_execution_minutes = (end_time -
 start_time).total_seconds()/60
                action_execution_cost = math.ceil(action_execution_minutes) * 0.02
                total_action_execution_minutes += action_execution_minutes
                cost = round(cost + action_execution_cost, 2)

    print ("{:<40}".format('Activity in last 30 days:'))
    print ("| {:<40} | {:<10}".format('_____________________________________',
 '___________________'))
    print ("| {:<40} | {:<10}".format('Total action executions:',
 total_action_executions))
    print ("| {:<40} | {:<10}".format('Total billing action executions:',
 total_blling_action_executions))
    print ("| {:<40} | {:<10}".format('Total billing action execution minutes:',
 round(total_action_execution_minutes, 2)))
    print ("| {:<40} | {:<10}".format('Cost of moving to V2 in $:', cost - 1))

analyze_cost_in_v2(pipeline)
```

3. 特定の で選択した V1 パイプラインに対してスクリプトを実行します AWS リージョン。

   次のコマンドを実行して、PipelineCostAnalyzer.py という名前の python スクリプトを実行します。この例では、リージョンは us-west-2 です。

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

4. スクリプトからの次の出力例では、アクション実行のリスト、請求対象であったアクション実行のリスト、これらのアクション実行の合計ランタイム、パイプラインを V2 タイプに更新するコストを確認できます。

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

> **Note**
>
> この例では、最後の行の負の値は、V2 タイプのパイプラインに移動して保存する量を表します。

の開始方法 CodePipeline

を初めて使用する場合は CodePipeline、この章の手順に従ってセットアップした後、このガイドのチュートリアルに従います。

CodePipeline コンソールには、情報アイコンまたはページの任意の情報リンクから開くことができる折りたたみ可能なパネルに、役立つ情報が含まれています。

()。
このパネルは、いつでも閉じることができます。

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、 AWS アカウント を作成し、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

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。https://aws.amazon.com/ の [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、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. 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AWS Management Console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 ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」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「AWS IAM Identity Centerの有効化」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、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」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「権限設定を作成する」を参照してください。
2. 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「グループの参加」を参照してください。

ステップ 2: への管理アクセスに 管理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 CodePipeline。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**
>
> この`AWSCodePipeline_FullAccess`ポリシーには、コンソールユーザーが IAM ロールを CodePipeline または他の に渡すことを許可するアクセス許可が含まれています AWS の

サービス。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 にロールを渡します。 CodePipeline
- 変更検出用の CloudWatch イベントルールを作成し、イベントサービスロールを CloudWatch イベントに渡すことを選択する場合があります CloudWatch

詳細については、「 にロールを渡すアクセス許可をユーザーに付与する AWS のサービス」を参照してください。

> **Note**
>
> この`AWSCodePipeline_FullAccess`ポリシーは、IAM ユーザーがアクセスできるすべての CodePipeline アクションとリソースへのアクセス、および 、 CodeDeployElastic Beanstalk、または Amazon S3 を含むステージの作成など、パイプラインでステージを作成するときに実行できるすべてのアクションへのアクセスを提供します。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 ユーザーを限定された CodePipeline 一連のアクションとリソースに制限する方法の詳細については、「」を参照してください CodePipeline サービスロールからアクセス許可を削除する。

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。

- 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「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

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

- ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」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する。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 AWS CLI から CodePipeline コマンドを呼び出すには、 AWS CLI をインストールする必要があります。 CodePipeline コンソールでこのガイドのステップのみを使い始める場合は、このステップはオプションです。

をインストールして設定するには AWS CLI

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

   > **Note**
   >
   > CodePipeline は AWS CLI バージョン 1.7.38 以降でのみ動作します。インストール AWS CLI した可能性のある 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、「 をアンインストール AWS CLIする」の指示に従い、「 のインストール AWS Command Line Interface」の指示に従います。

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

   ```
   aws configure
   ```

   プロンプトが表示されたら、 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 CodePipeline。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 Secret Access Key [None]: Type your target AWS secret access key here, and then
 press Enter
Default region name [None]: Type us-east-2 here, and then press Enter
Default output format [None]: Type json here, and then press Enter
```

> **Note**
>
> IAM、アクセスキー、シークレットキーに関するさらなる詳細については、Managing Access Keys for IAM Users および How Do I Get Credentials? を参照してください。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 CodePipeline、「 AWS CodePipeline エンドポイントとクォータ」を参照してください。

ステップ 4: のコンソールを開く CodePipeline

- にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home 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。の使用を開始できます CodePipeline。の使用を開始するには CodePipeline、「」を参照してくださいCodePipeline チュートリアル。

との製品とサービスの統合 CodePipeline

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を使用して、使用する製品やサービスと統合 CodePipeline するように を設定します。

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
- との一般的な統合 CodePipeline
- 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 アクションタイプ別に整理されています。

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 アクションタイプ別に整理されており、次のソースアクションプロバイダーと統合 CodePipeline するように を設定するのに役立ちます。

トピック

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

- Bitbucket Cloud、 GitHub (バージョン 2)、 GitHub Enterprise Server、 GitLab.com、 GitLab セルフマネージドへの接続
- CodeCommit ソースアクション
- GitHub （ バージョン 1) ソースアクション

Amazon ECR ソースアクション

Amazon ECR は AWS Docker 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 を参照してください
- でパイプラインを作成する CodePipeline
- チュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する

Amazon S3 ソースアクション

Amazon S3 はインターネット用のストレージサービスです。Simple Storage Service (Amazon S3) を使用すると、いつでもウェブ上の任意の場所から任意の量のデータを保存および取得できます。バージョニングされた 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 CodePipeline ように を設定できます。

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

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、Amazon S3 ソースアクション を参照してください
- ステップ 1: アプリケーションの S3 バケットを作成する
- パイプラインを作成する (CLI)

- CodePipeline は Amazon EventBridge （ 以前の Amazon CloudWatch Events) を使用して、Amazon S3 ソースバケットの変更を検出します。との一般的な統合 CodePipeline を参照してください。

Bitbucket Cloud、 GitHub (バージョン 2)、 GitHub Enterprise Server、 GitLab.com、GitLab セルフマネージドへの接続

Connections (`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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

Bitbucket Cloud	Bitbucket Cloud リポジトリをコードのソースとして使用する CodePipeline ように を設定できます。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。接続を作成するときは、サードパーティーのコードリポジトリに AWS CodeStar アプリケーションをインストールし、接続に関連付けます。 Bitbucket Cloud の場合は、コンソールの [Bitbucket] オプションまたは CLI の `CodestarSourceConnection` アクションを使用します。Bitbucket Cloud への接続 を参照してください。 この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、 CodeBuild ダウンストリームアクションでのみ使用できます。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、GitLab.com、および GitLab セルフマネージドアクション用 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 エンタープライズクラウド	GitHub リポジトリをコードのソースとして使用する CodePipeline ように を設定できます。 GitHub アカウントと少なくとも 1 GitHub つのリポジトリを以前に作成しておく必要があります。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、 GitHub リポジトリのソースアクションを追加できます。 パイプラインがサードパーティーのコードリポジトリにアクセスできるように、接続と呼ばれるリソースを設定できます。接続を作成するときは、サードパーティーのコードリポジトリに AWS CodeStar アプリケーションをインストールし、接続に関連付けます。 コンソールの GitHub （ バージョン 2) 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。GitHub 接続 を参照してください。

	この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、 CodeBuild ダウンストリームアクションでのみ使用できます。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、GitLab.com、および GitLab セルフマネージドアクション用 を参照してください • GitHub リポジトリに接続し、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、「」を参照してくださいチュートリアル: GitHub パイプラインソースでフルクローンを使用する。 • 現在の GitHub アクションは、 のバージョン 2 のソースアクションです GitHub。バージョン 1 GitHub アクションは OAuth トークン認証で管理されます。 GitHub バージョン 1 アクションを使用することはお勧めしませんが、 GitHub バージョン 1 アクションを含む既存のパイプラインは、影響を受けずに引き続き機能します。 GitHub アプリケーションでCodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、GitLab.com、および GitLab セルフマネージドアクション用ソースアクションを管理するパイプラインで GitHub ソースアクションを使用できるようになりました。バージョン 1 GitHub アクションを使用するパイプラインがある場合は、 でバージョン 2 GitHub 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 バージョン 1 のソースアクションを GitHub バージョン 2 のソースアクションに更新する。
GitHub エンタープライズサーバー	Enterprise GitHub Server リポジトリをコードのソースとして使用する CodePipeline ように を設定できます。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。Enterprise Server リポジトリのソースアクションを追加するには、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 GitHub。 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。接続を作成するときは、サー

	ドパーティーのコードリポジトリに AWS CodeStar アプリケーションをインストールし、接続に関連付けます。 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 `CodestarSourceConnection` アクションを使用します。GitHub Enterprise Server 接続 を参照してください。 **Important** AWS CodeStar リリースで既知の問題があるため、接続は 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。 この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、 CodeBuild ダウンストリームアクションでのみ使用できます。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、GitLab.com、および GitLab セルフマネージドアクション用 を参照してください • GitHub リポジトリに接続し、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、「」を参照してくださいチュートリアル: GitHub パイプラインソースでフルクローンを使用する。
GitLab.com	GitLab.com リポジトリをコードのソースとして使用する CodePipeline ように を設定できます。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。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、 GitLab.com リポジトリのソースアクションを追加できます。

	コンソールで GitLab provider オプションを使用するか、CLI で `GitLab` provider で `CodestarSourceConnection` アクションを使用します。GitLab.com 接続 を参照してください。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、GitLab.com、および GitLab セルフマネージドアクション用 を参照してください
GitLab セルフマネージド型	GitLab セルフマネージド型インストールをコードのソースとして使用する CodePipeline ように を設定できます。以前に GitLab アカウントを作成し、セルフマネージド型 GitLab (Enterprise Edition または Community Edition) のサブスクリプションを持っている必要があります。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、 GitLab セルフマネージドリポジトリのソースアクションを追加できます。 パイプラインがサードパーティーのコードリポジトリにアクセスできるように、接続と呼ばれるリソースを設定できます。接続を作成するときは、サードパーティーのコードリポジトリに AWS CodeStar アプリケーションをインストールし、接続に関連付けます。 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 `CodestarSourceConnection` アクションを使用します。GitLab セルフマネージド型 の接続 を参照してください。 この 完全クローン作成 アクションのオプションを使用して、リポジトリの Git メタデータを参照して、ダウンストリームのアクションで Git コマンドを直接実行できるようにします。このオプションは、 CodeBuild ダウンストリームアクションでのみ使用できます。 詳細はこちら: • 設定パラメータと JSON/YAML スニペット例を表示する場合、CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、

GitLab.com、および GitLab セルフマネージドアクション用 を参照してください

- このプロバイダータイプとの接続を作成する手順については、「 GitLab セルフマネージド型 の接続」を参照してください。

CodeCommit ソースアクション

CodeCommit は、クラウド内のアセット (ドキュメント、ソースコード、バイナリファイルなど) をプライベートに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。 CodeCommit リポジトリ内のブランチをコードのソースとして使用する CodePipeline ように 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。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、 CodeCommit リポジトリに接続できます。

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 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。との一般的な統合 CodePipeline を参照してください。

GitHub (バージョン 1) ソースアクション

GitHub バージョン 1 のアクションは OAuth Apps で管理されます。利用可能なリージョンでは、パイプラインでCodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、GitLab.com、および GitLab セルフマネージドアクション用ソースアクションを使用して、 GitHub Apps で GitHub ソースアクションを管理することもできます。 GitHub バージョン 1 アクションを使用するパイプラインがある場合は、 で GitHub バージョン 2 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 バージョン 1 のソースアクションを GitHub バージョン 2 のソースアクションに更新する。

> **Note**
>
> GitHub バージョン 1 アクションを使用することはお勧めしませんが、 GitHub バージョン 1 アクションを含む既存のパイプラインは引き続き機能します。

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 アクセスの詳細については、「」を参照してくださいhttps://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps。
- バージョン 1 GitHub 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付録 A: GitHub バージョン 1 のソースアクション。

ビルドアクションの統合

以下の情報は CodePipeline アクションタイプ別に整理されており、以下のビルドアクションプロバイダーと統合 CodePipeline するように を設定するのに役立ちます。

トピック

- CodeBuild ビルドアクション
- CloudBees ビルドアクション
- Jenkins ビルドアクション
- TeamCity ビルドアクション

CodeBuild ビルドアクション

CodeBuild は、ソースコードをコンパイルし、ユニットテストを実行し、すぐにデプロイできるアーティファクトを生成するフルマネージドビルドサービスです。

パイプラインのビルドステージにビルドアクション CodeBuild として を追加できます。詳細については、「 の CodePipeline アクション設定リファレンス」を参照してくださいAWS CodeBuild。

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

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、AWS CodeBuild を参照してください。
- CodeBuildとは
- CodeBuild — フルマネージドビルドサービス

CloudBees ビルドアクション

パイプラインの 1 つ以上のアクションでコードを構築またはテストCloudBeesするために を使用する CodePipeline ように を設定できます。

詳細はこちら:

- re:INVENT 2017: Cloud First with AWS

Jenkins ビルドアクション

Jenkins CI を使用して、パイプラインの 1 つ以上のアクションでコードを構築またはテスト CodePipeline するように を設定できます。以前に Jenkins プロジェクトを作成し、そのプロジェクトの Jenkins 用 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。Plugin for Jenkins CodePipeline は、で使用するすべての Jenkins インスタンスにインストールする必要があります CodePipeline。また、Jenkins プロジェクト CodePipeline へ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

を使用して CodePipeline 、パイプラインの 1 つ以上のアクションでコードを構築およびテストTeamCityするように を設定できます。

詳細はこちら:

- TeamCity のプラグイン CodePipeline

テストアクションの統合

以下の情報は CodePipeline アクションタイプ別に整理されており、次のテストアクションプロバイダーと統合 CodePipeline するように を設定するのに役立ちます。

トピック

- CodeBuild テストアクション
- AWS Device Farm テストアクション
- Ghost Inspector テストアクション
- OpenText LoadRunner クラウドテストアクション

CodeBuild テストアクション

CodeBuild は、 クラウドのフルマネージドビルドサービスです。 はソースコードを CodeBuild コンパイルし、ユニットテストを実行し、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 CodeBuild に追加できます。詳細については、「 の CodePipelineアクション設定リファレンス」を参照してくださいAWS CodeBuild。

> **Note**
>
> CodeBuild は、必須のビルド出力アーティファクトを使用して、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 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 CodePipeline ように 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。テストが完了すると、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が更新されます。は、、、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 PhoneGap、iOS、および Fire OS アプリケーションのテスト AWS Device Farm をサポートします。Android アプリのリモートアクセスをサポートしているため、テストデバイスと直接やり取りすることができます。

詳細はこちら:

- 設定パラメータと JSON/YAML スニペット例を表示する場合、AWS Device Farm を参照してください。
- AWS Device Farmとは
- 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

Ghost Inspector CodePipeline 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するように を設定できます。

詳細はこちら:

- とのサービス統合に関する Ghost Inspector ドキュメント CodePipeline

OpenText LoadRunner クラウドテストアクション

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

詳細はこちら:

- LoadRunner と統合するためのクラウドドキュメント CodePipeline

デプロイアクションの統合

以下の情報は CodePipeline アクションタイプ別に整理されており、次のデプロイアクションプロバイダーと統合 CodePipeline するように を設定するのに役立ちます。

トピック

- Amazon S3 デプロイアクション
- AWS AppConfig デプロイアクション
- AWS CloudFormation デプロイアクション
- 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 デプロイアクション
- Service Catalog のデプロイアクション
- Amazon Alexa デプロイアクション
- CodeDeploy デプロイアクション
- XebiaLabs デプロイアクション

Amazon 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:

- でパイプラインを作成する CodePipeline
- 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。EC2 インスタンス AppConfig 、コンテナ、モバイルアプリケーション AWS Lambda、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 を使用できます。

詳細はこちら:

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

AWS CloudFormation デプロイアクション

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

AWS サーバーレスアプリケーションモデル (AWS SAM) を拡張 AWS CloudFormation して、サーバーレスアプリケーションを定義およびデプロイする簡単な方法を提供します。 AWS SAM は、Amazon API Gateway APIs関数、および Amazon DynamoDB テーブルをサポートしています。AWS と AWS SAM CodePipeline で AWS CloudFormation を使用すると、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。 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:

- CodePipeline のアクション設定リファレンス AWS CloudFormation
- による継続的デリバリー CodePipeline — CodePipeline を使用して の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 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 AWS CloudFormation 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。

CodePipeline で を使用すると 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。

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

- CloudFormationStackSet
- CloudFormationStackInstances

詳細はこちら:

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

Amazon ECS デプロイアクション

Amazon ECS は、スケーラビリティに優れた高性能なコンテナ管理サービスであり、 AWS クラウドでコンテナベースのアプリケーションを実行することができます。パイプラインを作成すると、

デプロイプロバイダとして Amazon ECS を選択できます。ソースコントロールリポジトリのコードを変更すると、パイプラインが新しい Docker イメージを作成し、コンテナレジストリにプッシュし、更新されたイメージを Amazon ECS にデプロイします。で ECS (Blue/Green) プロバイダーアクションを使用して CodePipeline 、 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 CodeDeploy。

詳細はこちら :

- Amazon ECS とは
- チュートリアル: を使用した継続的なデプロイ CodePipeline
- でパイプラインを作成する CodePipeline
- チュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する

Elastic Beanstalk デプロイアクション

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

> **Note**
>
> この機能は、アジアパシフィック (ハイデラバード)、アジアパシフィック (メルボルン)、中東 (アラブ首長国連邦)、欧州 (スペイン)、または欧州 (チューリッヒ) リージョンでは利用できません。利用可能なその他のアクションについては、「との製品とサービスの統合 CodePipeline」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
- でパイプラインを作成する CodePipeline

AWS OpsWorks デプロイアクション

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 を使用して、 のカスタム 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 AWS OpsWorks Stacks するように を設定できます 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。

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

詳細はこちら:

- CodePipeline で を使用する AWS OpsWorks Stacks
- クックブックとレシピ
- AWS OpsWorks アプリケーション

Service Catalog のデプロイアクション

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

> **Note**
>
> この機能は、アジアパシフィック (ハイデラバード)、アジアパシフィック (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、中東 (アラブ首長国連邦)、欧州 (スペイン)、欧州 (チューリッヒ)、またはイスラエル (テルアビブ) 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、「との製品とサービスの統合 CodePipeline」を参照してください。

製品テンプレートの更新とバージョンを Service Catalog CodePipeline にデプロイするように 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
- でパイプラインを作成する CodePipeline

Amazon Alexa デプロイアクション

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

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

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

詳細はこちら:

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

CodeDeploy デプロイアクション

CodeDeploy は、Amazon EC2 インスタンス、オンプレミスインスタンス、またはその両方へのアプリケーションのデプロイを調整します。 CodePipeline を使用してコードをデプロイ CodeDeploy す

るように を設定できます。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、ステージのデプロイアクションで使用する CodeDeploy アプリケーション、デプロイ、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: でアプリケーションを作成する CodeDeploy
- 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 を使用して、パイプラインの 1 つ以上のアクションにコードをデプロイXebiaLabsするように を設定できます。

詳細はこちら:

- での XL デプロイの使用 CodePipeline

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。

で手動承認リクエストを作成する場合 CodePipeline、オプションで Amazon SNS のトピックに発行して、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るようにすることができます。

詳細はこちら:

- Amazon SNS とは
- サービスロールに Amazon SNS アクセス許可を付与する CodePipeline

呼び出しアクションの統合

以下の情報は CodePipeline アクションタイプ別に整理されており、以下の呼び出しアクションプロバイダーと統合 CodePipeline するように を設定するのに役立ちます。

トピック

- Lambda 呼び出しアクション
- Snyk 呼び出しアクション
- Step Functions アクション呼び出し

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 CodePipeline に柔軟性と機能を追加するように 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:

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

Snyk 呼び出しアクション

Snyk を使用する CodePipeline ように を設定して、セキュリティの脆弱性を検出して修正し、アプリケーションコードとコンテナイメージの依存関係を更新することで、オープンソース環境を安全に保つことができます。で Snyk アクションを使用して CodePipeline 、パイプラインのセキュリティテストコントロールを自動化することもできます。

詳細はこちら:

- CodePipeline のアクション設定リファレンス Snyk アクション構造リファレンス
- Snyk AWS CodePipeline を使用して で脆弱性スキャンを自動化する

Step Functions アクション呼び出し

Step Functions では、ステートマシンの作成と設定ができます。Step Functions 呼び出しアクションを使用してステートマシンの実行をトリガー CodePipeline するように を設定できます。

詳細はこちら:

- CodePipeline のアクション設定リファレンス AWS Step Functions

- チュートリアル: パイプライン AWS Step Functions で呼び出しアクションを使用する

との一般的な統合 CodePipeline

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

Amazon CloudWatch	Amazon CloudWatch は リソースをモニタリングします AWS 。 詳細はこちら: • Amazon とは CloudWatch
Amazon EventBridge	Amazon EventBridge は、定義したルール AWS のサービス に基づいての変更を検出し、変更が発生した AWS のサービス ときに指定された 1 つ以上の でアクションを呼び出すウェブサービスです。 • 何かが変更されたときにパイプラインの実行を自動的に開始する — Amazon で設定されたルールでターゲット CodePipeline として を設定できます EventBridge。これにより、別のサービスが変更されたときに自動的にパイプラインが開始されるように設定されます。 詳細はこちら: • Amazon とは EventBridge • でパイプラインを開始する CodePipeline. • CodeCommit ソースアクションと EventBridge • パイプラインの状態が変化したときに通知を受け取る — 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化を検出して対応するための EventBridge ルールを設定できます。 詳細はこちら: • CodePipeline イベントのモニタリング • チュートリアル: パイプラインの状態変更に関する E メール通知を受信するように CloudWatch イベントルールを設定する
AWS Cloud9	AWS Cloud9 は、ウェブブラウザからアクセスするオンライン IDE です。この IDE では、リッチなコード編集エクスペリエンスを実現しており、複数のプログラミング言語、ランタイムデバッガ、組み込み

	ターミナルがサポートされています。バックグラウンドでは、Amazon EC2 インスタンスは AWS Cloud9 開発環境をホストします。詳細については、『AWS Cloud9 ユーザーガイド』を参照してください。 詳細はこちら: • AWS Cloud9のセットアップ
AWS CloudTrail	CloudTrail は、 AWS アカウントによって、または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。コンソールからの CodePipeline API コール、 からの CodePipeline コマンド AWS CLI、および API からの CodePipeline API コールをキャプチャ CloudTrail するように を設定できます。 詳細はこちら: • を使用した CodePipeline API コールのログ記録 AWS CloudTrail
AWS CodeStar 通知	パイプラインの実行開始時など、重要な変更をユーザーに知らせるための通知を設定できます。詳細については、「通知ルールの作成」を参照してください。

AWS Key Management Service	AWS KMS は、データの暗号化に使用される暗号化キーの作成と管理を容易にするマネージド型サービスです。デフォルトでは、 CodePipeline を使用して Amazon AWS KMS 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。 Amazon S3 詳細はこちら: • あるアカウントからソースバケット、アーティファクトバケット、サービスロール、別の AWS アカウントの CodeDeploy リソースを使用するパイプラインを作成するには AWS 、カスタマー管理の 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ためのアカウントポリシーとロールを設定する必要があります。詳細については、「別の AWS アカウントのリソース CodePipeline を使用するパイプラインを に作成する」を参照してください。 • 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 、カスタマー管理の 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、「 を使用して AWS CloudFormation スタック CodePipeline を別のアカウントにデプロイする方法」を参照してください。 • パイプラインの S3 アーティファクトバケットのサーバー側の暗号化を設定するには、デフォルトの AWS マネージド KMS キーを使用するか、カスタマーマネージド KMS キーを作成し、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、「の Amazon S3 に保存されているアーティファクトのサーバー側の暗号化を設定する CodePipeline」を参照してください。 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、包括的なリストまたは例の内容の推奨とはみなされません。 AWS は、外部コンテンツの内容または正確性について責任を負いません。

トピック

- 統合の例: ブログ投稿

統合の例: ブログ投稿

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

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

  2017 年 3 月発行

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

  CodeCommit、 CodePipeline、および CodeDeploy のサービスを使用して CodeBuild、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、構築、および Amazon EC2 Linux インスタンスのセット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

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

  開発ブランチ、テストブランチ、製品ブランチを別々のデプロイグループにデプロイするために、ゼロ CodePipeline からセットアップする方法について説明します。 CodeDeployとともに IAM ロール、 CodeDeploy エージェント、および を使用および設定する方法について説明します CodePipeline。

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

- DevSecOps を使用した の実装 CodePipeline

  で CI/CD パイプラインを使用して、予防および検出セキュリティコントロール CodePipeline を自動化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、本番稼働段階でセキュリティチェックを実行して AWS 、アカウントのセキュリティ体制を改善する方法について説明します。

2017 年 3 月発行

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

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

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、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 AWS のサービス 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 、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 デプロイのスケーリング

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

2016 年 11 月公開

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

でアプリケーションの CodePipeline パイプラインに継続的デリバリーを実装する方法について説明します AWS Elastic Beanstalk。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 を自動化 CodePipeline 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

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

AWS Identity and Access Managementを使用して AWS CodePipeline のパイプラインへのクロスアカウントアクセスのプロビジョニングを自動化する方法について説明します。 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 および を使用して、ASP.NET Core アプリケーション用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 段階のパイプラインを作成する方法について説明します AWS CodePipeline チュートリアル: 4 ステージのパイプラインを作成する。

2016年 3 月発行

- で AWS CodePipeline パイプラインをモックする 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アクションとステージを設計時に視覚化できる 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

- CodePipeline を使用した 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のパイプラインで AWS Lambda 関数を呼び出す CodePipeline。

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 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 CodePipeline。

2016 年 1 月投稿

- CodePipeline によるプロビジョニング AWS CloudFormation

  CodePipeline を使用して で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 AWS CloudFormation。

発行日：2015年12月

- から CodePipeline へのデプロイカスタムアクション AWS OpsWorks の使用と AWS Lambda

  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 CodePipeline。

  2015 年 7 月発行

CodePipeline チュートリアル

のステップを完了したらの開始方法 CodePipeline、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。

ウィザードを使用して、 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 CodeDeploy デプロイするパイプラインを作成します。 Amazon EC2 ウィザードを使用して 2 ステージのパイプラインを作成したら、ステージ 3 を追加します。	チュートリアル: シンプルなパイプラインを作成する (S3 バケット) を参照してください。
リポジトリから Amazon Linux を実行している Amazon EC2 インスタンス CodeDeploy にサンプルアプリケーションをデプロイ CodeCommit するために を使用する 2 段階のパイプラインを作成します。 Amazon EC2	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) を参照してください。
最初のチュートリアルで作成した 3 ステージのパイプラインにビルドステージを追加します。新しいステージでは、Jenkins を使用してアプリケーションをビルドします。	チュートリアル: 4 ステージのパイプラインを作成する を参照してください。
パイプライン、ステージ、またはアクションの実行状態が変更されるたびに通知を送信する CloudWatch イベントルールを設定したい。	チュートリアル: パイプラインの状態変更に関する E メール通知を受信するように CloudWatch イベントルールを設定する を参照してください。
CodeBuild と を使用して Android アプリを構築してテストする GitHub ソースを使用してパイプラインを作成したい AWS Device Farm。	チュートリアル: で Android アプリを構築してテストするパイプラインを作成する AWS Device Farm を参照してください。
AWS Device Farmで iOS アプリをテストする Amazon S3 ソースでパイプラインを作成します。	チュートリアル: で iOS アプリケーションをテストするパイプラインを作成する AWS Device Farm を参照してください。

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 を参照してください。
サンプルテンプレートを使用して、 AWS CloudFormation コンソールを使用してシンプルなパイプライン (Amazon S3、またはソースを使用) を作成したい。 CodeCommit GitHub	チュートリアル: でパイプラインを作成する AWS CloudFormation を参照してください。
Amazon ECR リポジトリから Amazon ECS クラスター CodeDeploy とサービスへのイメージのブルー/グリーンデプロイに と Amazon ECS を使用する 2 段階のパイプラインを作成します。	チュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する を参照してください。
サーバーレスアプリケーションを AWS Serverless Application Repositoryに継続的に発行するパイプラインを作成します。	チュートリアル: サーバーレスアプリケーションを に発行するパイプラインを作成する AWS Serverless Application Repository を参照してください。

他の ユーザーガイドの以下のチュートリアルでは、他の AWS のサービス をパイプラインに統合するためのガイダンスを提供します。

- AWS CodeBuild ユーザーガイドで を使用するパイプラインを作成する CodeBuild
- AWS OpsWorks ユーザーガイドの CodePipeline での の使用 AWS OpsWorks Stacks
- ユーザーガイドの「 を使用した継続的な配信 CodePipelineAWS CloudFormation」
- AWS Elastic Beanstalk デベロッパーガイド で Elastic Beanstalk の使用を開始する
- を使用して継続的なデプロイパイプラインを設定する CodePipeline

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、ソースアクションが Git タグトリガータイプに設定されている GitHub リポジトリに接続するパイプラインを作成します。コミット時に Git タグが作成されると、パイプラインが開始されます。この例では、タグ名の構文に基づいてタグをフィルタ処理するパイプラインの作成方法を示しています。glob パターンを使用したフィルタ処理の詳細については、「構文での glob パターンの使用」を参照してください。

このチュートリアルでは、 CodeStarSourceConnectionアクションタイプ GitHub を使用して 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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

トピック

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 認証情報を用意します。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: CloudShell リポジトリを開いてクローンを作成する

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の 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。

2. 上部のナビゲーションバーで、 AWS アイコンを選択します。 AWS Management Console のメインページが表示されます。
3. 上部のナビゲーションバーで icon AWS CloudShell . CloudShell opens を選択します。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、 でサポートされているリージョン CloudShellにいることを確認してください。このチュートリアルは、米国西部 (オレゴン) リージョンにいることを前提としています。

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

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

6. プロンプトが表示されPasswordたら、 GitHub アカウントUsernameと 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、次のアクションを使用してパイプラインを作成します。

- GitHub リポジトリとアクションへの接続があるソースステージ。
- 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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソール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: パイプラインの設定を選択する] の [パイプライン名] に「**MyGitHubTagsPipeline**」と入力します。

4. [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。

5. [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**
>
> 代わりに既存の CodePipeline サービスロールを使用する場合は、サービスロールポリシーに `codestar-connections:UseConnection` IAM アクセス許可を追加している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、「サービスロール にアクセス許可を追加する CodePipeline 」を参照してください。

6. [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] をクリックします。

7. ステップ 2: [Add source stage] (ソースステージの追加) ページで、ソースステージを追加します。

   a. ソースプロバイダー で、 GitHub （ バージョン 2) 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 ソースアクションの接続を作成または管理するには、「」を参照してくださいGitHub 接続。

   c. リポジトリ名 で、リポジトリの名前 GitHubを選択します。

   d. [パイプライントリガー] で、[Git タグ] を選択します。

      [Include] フィールドに「`release*`」と入力します。

[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。

> **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] をクリックします。

8. [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] がデフォルトでパイプラインリージョンになることを許可します。

   b. [プロジェクトを作成] を選択します。

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

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

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

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

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

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

```
version: 0.2
#env:
```

```
  #variables:
     # key: "value"
     # key: "value"
  #parameter-store:
     # key: "value"
     # key: "value"
  #git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
    #commands:
      # - command
      # - command
  #pre_build:
    #commands:
      # - command
      # - command
  build:
    commands:
      -
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
     - '*'
    # - location
  name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 「続行」を選択します CodePipeline。これによりコンソール CodePipelineに戻り、ビルドコマンドを使用して設定を行う 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] をクリックします。

9. [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] ページで、[Skip deploy stage (デプロイステージのスキップ)] を選択し、[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] をクリックします。

10. [Step 5: Review (ステップ 5: 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て Git タグを指定したら、リポジトリ内のコミットにタグを付ける GitHub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。タグ名のリリースは、のリリースの概念には適用されないことに注意してください GitHub。

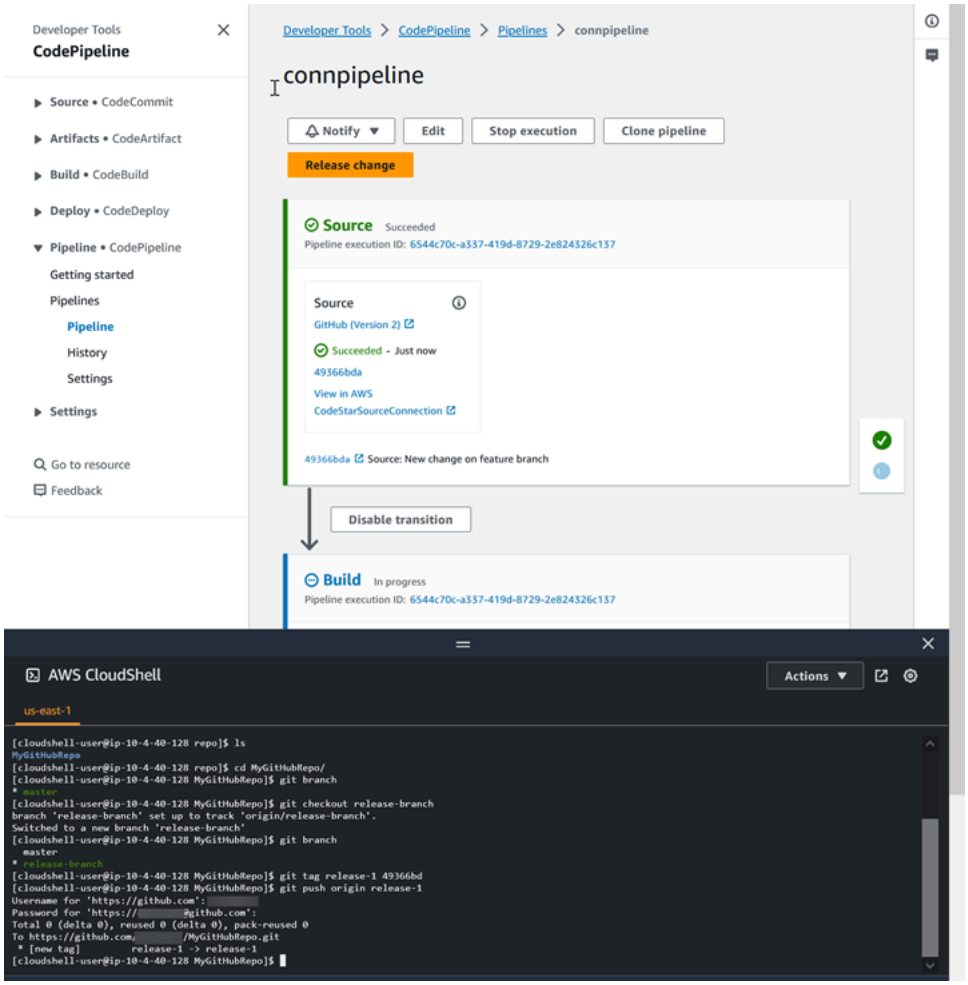
1. コピーしたコミット ID のうち、タグを付けるものを参照します。各ブランチのコミットを表示するには、 CloudShell ターミナルで次のコマンドを入力して、タグ付けするコミット IDsをキャプチャします。

```
git log
```

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

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

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



ステップ 4: 変更をリリースしてログを表示する

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

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

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

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

このチュートリアルでは、.com リポジトリに接続するパイプラインを作成します GitHub。このパイプラインでは、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が設定されます。指定されたブランチに対して指定され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されます。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を作成する方法を示します。トリガーの使用の詳細については、「」を参照してくださいパイプライン JSON でのトリガーフィルタリング (CLI)。glob 形式の正規表現パターンによるフィルタリングの詳細については、「」を参照してください構文での glob パターンの使用。

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

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 認証情報を用意します。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: 指定されたブランチのプルリクエストを開始する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。

- GitHub.com リポジトリと アクションに接続するソースステージ。
- 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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソール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: パイプラインの設定を選択する] の [パイプライン名] に「**MyFilterBranchesPipeline**」と入力します。
4. [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。
5. [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**
   >
   > 代わりに既存の CodePipeline サービスロールを使用する場合は、サービスロールポリシーに `codeconnections:UseConnection` IAM アクセス許可を追加している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、「サービスロールにアクセス許可を追加する CodePipeline 」を参照してください。

6. [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] をクリックします。
7. ステップ 2: [Add source stage] (ソースステージの追加) ページで、ソースステージを追加します。

   a. ソースプロバイダー で、 GitHub （ バージョン 2) を選択します。

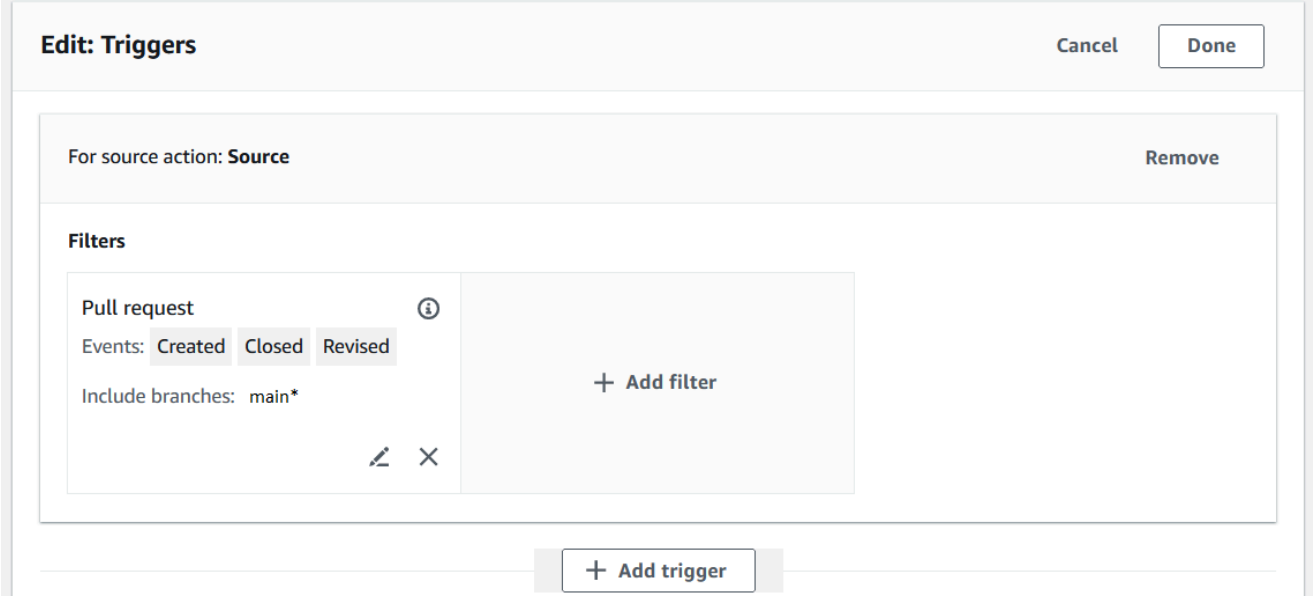
b. 接続 で、既存の接続を選択するか、新規の接続を作成します。 GitHub ソースアクションの接続を作成または管理するには、「」を参照してくださいGitHub 接続。

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て、作成、更新、またはクローズされたプルリクエストに対してイベントが発生するようにします。

ブランチ のインクルード フィールドに、 と入力しますmain*。



> **Important**
>
> このトリガータイプで始まるパイプラインは WebhookV2 イベント用に設定され、Webhook イベント (すべてのプッシュイベントの変更検出) を使用してパイプラインを起動しません。

[次へ] をクリックします。

8. ビルドステージの追加 で、ビルドプロバイダー で を選択しますAWS CodeBuild。[リージョン] がデフォルトでパイプラインリージョンになることを許可します。「」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されます。

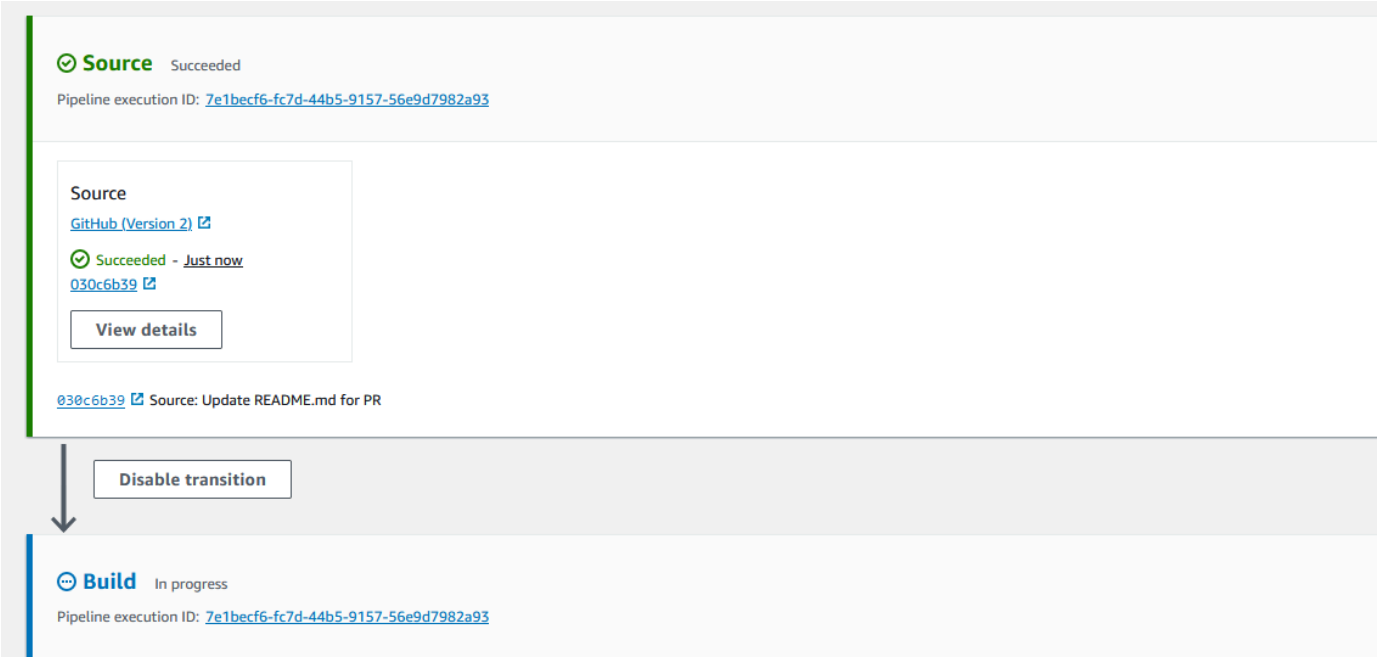
9. [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] ページで、[Skip deploy stage (デプロイステージのスキップ)] を選択し、[スキップ] を選択して警告メッセージを受け入れます。[次へ] をクリックします。
10. [Step 5: Review (ステップ 5: 確認)] で、[パイプラインの作成] を選択します。

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

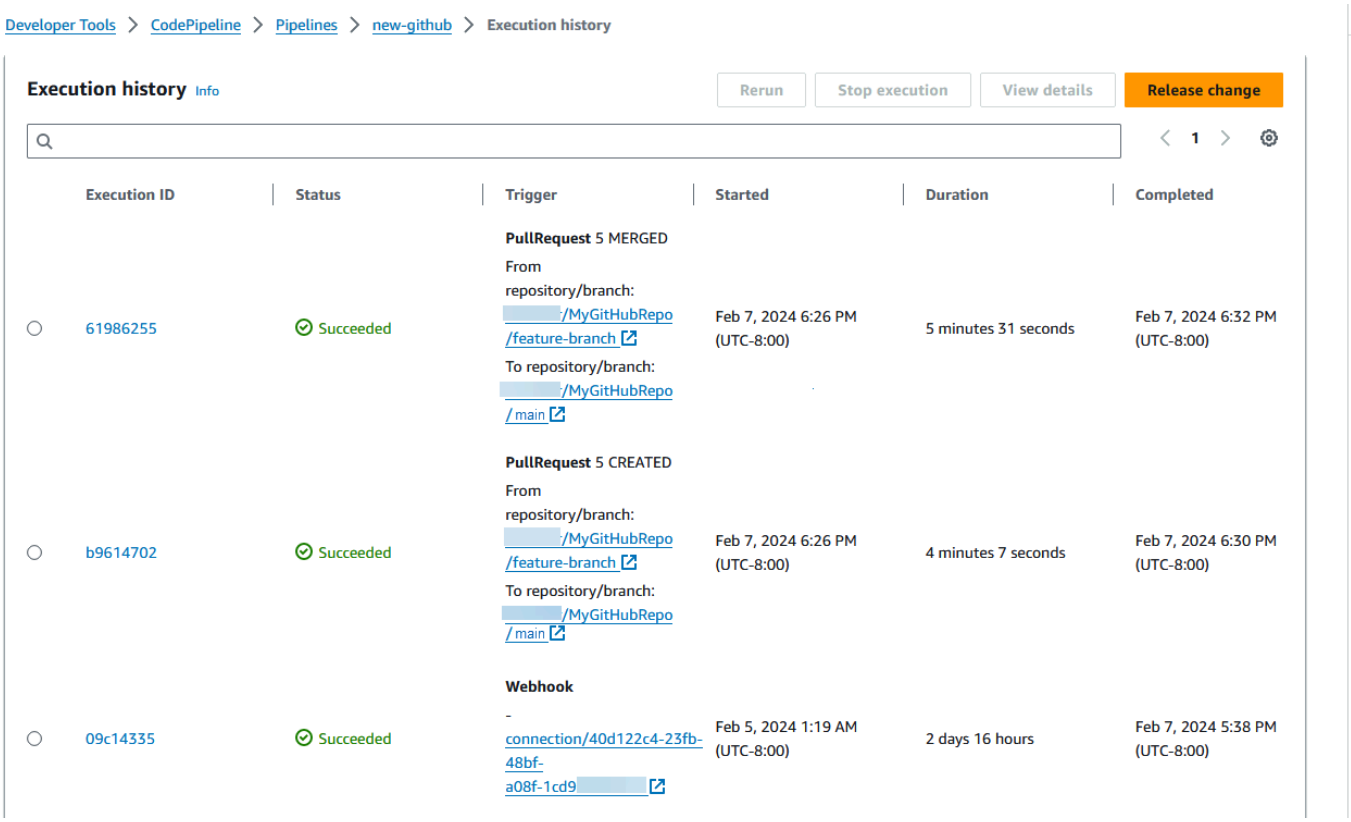
このセクションでは、プルリクエストを作成してマージします。これにより、パイプラインが開始され、開いているプルリクエストに対して 1 つの実行と、閉じているプルリクエストに対して 1 つの実行が行われます。

プルリクエストを作成してパイプラインを開始するには

1. GitHub.com で、機能ブランチで README.md に変更を加え、`main`ブランチにプルリクエストを送り、プルリクエストを作成します。のようなメッセージで変更をコミットしますUpdate `README.md for PR`。
2. パイプラインは、プルリクエストのソースメッセージを PR の Update README.md として示すソースリビジョンで始まります。



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



チュートリアル: パイプラインレベルの変数を使用する

このチュートリアルでは、パイプラインレベルで変数を追加し、変数値を出力する CodeBuild ビルドアクションを実行するパイプラインを作成します。

トピック

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、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 リポジトリに接続するソースステージ。
- 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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソール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: パイプラインの設定を選択する] の [パイプライン名] に「**MyVariablesPipeline**」と入力します。
4. [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。
5. [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**
   >
   > 代わりに既存の CodePipeline サービスロールを使用する場合は、サービスロールポリシーに `codeconnections:UseConnection` IAM アクセス許可を追加していることを確認してください。 CodePipeline サービスロールの手順については、 CodePipeline「サービスロール にアクセス許可を追加する」を参照してください。

6. [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。

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

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

[次へ] をクリックします。

8. ステップ 2: [Add source stage] (ソースステージの追加) ページで、ソースステージを追加します。

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

   b. [リポジトリ名] と [ブランチ名] で、リポジトリとブランチを選択します。

   [次へ] をクリックします。

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] がデフォルトでパイプラインリージョンにな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. 「続行」を選択します CodePipeline。これによりコンソール CodePipelineに戻り、ビルドコマンドを使用して設定を行う 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の名前空間 `variables` と、ステップ 5 でパイプライン用に作成されたパイプラインレベルの変数 `timeout` に基づきます。

j. [次へ] をクリックします。

10. [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] ページで、[Skip deploy stage (デプロイステージのスキップ)] を選択し、[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] をクリックします。

11. [Step 5: Review (ステップ 5: 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、バージョニングされた S3 バケットと 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 段階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。

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、ソースアクションの pipeline. AWS resources 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。パイプラインを作成するのと同じ AWS リージョンに常に作成する必要があります。例えば、米国東部 (オハイオ) リージョンでパイプラインを作成する場合、 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、「でクロスリージョンアクションを追加する CodePipeline」を参照してください。

開始する前に、「の開始方法 CodePipeline」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する
- ステップ 3: でアプリケーションを作成する CodeDeploy
- ステップ 4: で最初のパイプラインを作成する CodePipeline
- (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: のステージ間の移行を無効または有効にする CodePipeline
- 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. で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。
6. [アクセス許可] タブは、デフォルト設定のままにします。S3 バケットおよびオブジェクトへのアクセス許可に関する詳細については、「ポリシーでのアクセス許可の指定」を参照してください。

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

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

      - を使用して Amazon Linux インスタンスにデプロイする場合は CodeDeploy、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。
      - を使用して Windows Server インスタンスにデプロイする場合は CodeDeploy、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、 でデプロイするための以下のファイルが含まれています CodeDeploy。

      - `appspec.yml` – アプリケーション仕様ファイル (AppSpec ファイル) は、デプロイを管理する CodeDeploy ために が使用する YAML 形式のファイルです。 AppSpec ファイルの詳細については、「 ユーザーガイド」のCodeDeploy AppSpec 「ファイルリファレンスAWS CodeDeploy 」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。複数の CodeDeploy デプロイライフサイクルイベントごとに 1 つのファイルが次のように書き込まれます。
        - (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する

> **Note**
>
> このチュートリアルでは、Amazon EC2 Windows インスタンスを作成するサンプル手順を示します。Amazon EC2 Linux インスタンスを作成するサンプルステップについては、「ステップ 3: Amazon EC2 Linux インスタンスを作成し、 CodeDeploy エージェントをインストールする 」を参照してください。作成するインスタンスの数の入力を求められたら、2 つのインスタンスを指定します。

このステップでは、サンプルアプリケーションをデプロイする Windows Server Amazon EC2 インスタンスを作成します。このプロセスの一環として、インスタンスへの 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持つインスタンスロールを作成します。 CodeDeploy エージェントは、インスタンスを CodeDeploy デプロイで使用できるようにするソフトウェアパッケージです。また、インスタンスがアプリケーションをデプロイするために CodeDeploy エージェントが使用するファイルを取得し、SSM によるインスタンスの管理を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. [ロールを作成] 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。

6. **AmazonSSMManagedInstanceCore** という名前のマネージドポリシーを検索して選択します。[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。

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

ステップ 3: でアプリケーションを作成する CodeDeploy

では CodeDeploy、アプリケーションは、デプロイするコードの名前の形式の識別子です。 はこの名前 CodeDeploy を使用して、リビジョン、デプロイ設定、デプロイグループの正しい組み合わせがデプロイ中に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ときに、このステップで作成する 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. [ロールを作成] を選択します。
4. [信頼されたエンティティを選択] で、[AWS のサービス] を選択します。[Use case] (ユースケース) で、[CodeDeploy] を選択します。表示されたオプションCodeDeployから選択します。[次へ] をクリックします。`AWSCodeDeployRole` マネージドポリシーはロールにアタッチ済みです。
5. [次へ] をクリックします。
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。

でアプリケーションを作成するには CodeDeploy

1. https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。
2. アプリケーションページが表示されない場合は、 AWS CodeDeploy メニューでアプリケーション を選択します。
3. [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。

でデプロイグループを作成するには CodeDeploy

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

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

6. AWS Systems Manager でのエージェント設定 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 SSM エージェントですでに設定されており、 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: で最初のパイプラインを作成する CodePipeline

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: パイプラインの設定を選択する] の [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、「のクォータ AWS CodePipeline」を参照してください。

4. 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。
5. [Service role (サービスロール)] で、次のいずれかの操作を行います。
   - IAM で新しいサービスロールを作成 CodePipeline することを許可する新しいサービスロールを選択します。
   - 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。
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. [Step 2: Add source stage (ステップ 2: ソースステージの追加)] ページの [ソースプロバイダ] で、[Amazon S3] を選択します。[バケット] に、「ステップ 1: アプリケーションの S3 バケットを作成する」で作成した S3 バケットの名前を入力します。S3 オブジェクトキーで、ファイルパスの有無にかかわらずオブジェクトキーを入力し、必ずファイル拡張子を含めます。たとえば、`SampleApp_Windows.zip` の場合、次の例に示すように、サンプルファイル名を入力します。

   ```
   SampleApp_Windows.zip
   ```

   [次のステップ] を選択します。

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

   [次へ] をクリックします。
8. [Step 3: Add build stage] (ステップ 3: ビルドステージを追加する) 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。[次へ] をクリックします。
9. ステップ 4: デプロイステージ を追加、デプロイプロバイダー で を選択しますCodeDeploy 。リージョンフィールドのデフォルトはパイプライン AWS リージョン と同じです。[アプリケーション名] に `MyDemoApplication` を入力するか、更新ボタンを選択してリストからそのアプリケーション名を選択します。[デプロイグループ] に「**`MyDemoDeploymentGroup`**」と入力するか、リストからデプロイグループを選択して [次へ] を選択します。

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

10. [ステップ 5: 確認] で情報を確認し、[パイプラインの作成] を選択します。
11. パイプラインの実行が開始されます。 CodePipeline サンプルがデプロイ内の各 Amazon EC2 インスタンス CodeDeployにウェブページをデプロイすると、進行状況、成功、失敗のメッセージを表示できます。

お疲れ様でした。でシンプルなパイプラインを作成しました CodePipeline。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。
- これらの変更を で EC2 インスタンスにデプロイするデプロイステージ CodeDeploy。

ここで、結果を確認します。

パイプラインが正常に実行されたことを確認するには

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

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

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

(オプション) ステップ 5: 別のステージをパイプラインに追加する

次に、パイプラインに別のステージを追加して、 を使用してステージングサーバーから本番稼働用サーバーにデプロイします CodeDeploy。まず、 CodePipelineDemoApplication の に別のデプロイグループを作成します CodeDeploy。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 を使用して JSON ファイル内のパイプラインの構造 AWS CLI を取得して手動で編集し、 update-pipeline コマンドを実行してパイプラインを変更で更新します。

トピック

- で 2 番目のデプロイグループを作成する CodeDeploy
- 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

で 2 番目のデプロイグループを作成する CodeDeploy

> **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。これは、エラーが にどのように表示されるかを表示できないように設計されています CodePipeline。

で 2 番目のデプロイグループを作成するには CodeDeploy

1. https://console.aws.amazon.com/codedeploy で CodeDeploy コンソールを開きます。
2. [アプリケーション] を選択し、アプリケーションのリストで [MyDemoApplication] を選択します。
3. [デプロイグループ] タブを選択して、[Create deployment group (デプロイグループの作成)] を選びます。
4. [Create deployment group (デプロイグループの作成)] ページの [Deployment group name (デプロイグループ名)] に、2 番目のデプロイグループの名前 (たとえば、**CodePipelineProductionFleet**) を入力します。

5. サービスロール で、最初のデプロイに使用したものと同じ CodeDeploy サービスロールを選択します (CodePipeline サービスロールではありません) 。
6. [Deployment type] (デプロイタイプ) で、[In-place] (インプレース) を選択します。
7. [環境設定] で、[Amazon EC2 インスタンス] を選択します。[名前] を [キー] ボックスから選択し、[値] ボックスから `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。コンソールまたは を使用して CodePipeline AWS CLI 、このステージを追加できます。

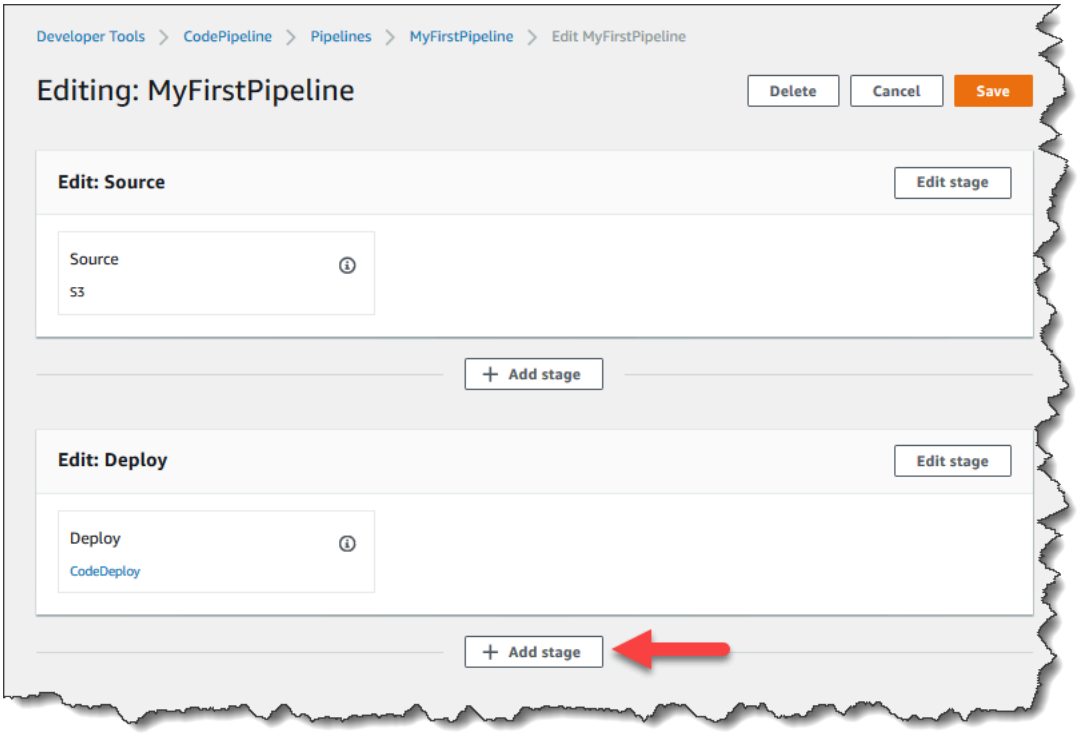
トピック

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



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

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

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

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

11. CodePipeline コンソールに戻り、パイプラインのリストで、ビューページMyFirstPipelineを開くことを選択します。

    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 (本番稼働用)] ステージでは [Deploy-Second-Deployment (2 番目のデプロイをデプロイ)] アクションが失敗したことが示されます。

12. [Deploy-Second-Deployment] アクションで、[Details] を選択します。デプロイのページ CodeDeploy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 AWS CLI にステージを追加するのは、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把握できます。

パイプラインの 3 番目のステージを作成するには

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

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

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

このコマンドは の構造を返します MyFirstPipeline。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": "codepipeline-us-east-2-250656481468",
        },
        "name": "MyFirstPipeline",
        "version": 4
    },
    "metadata": {
        "pipelineArn": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline",
        "updated": 1501626591.112,
        "created": 1501626591.112
    }
}
```

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

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

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

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

   以下の例では、[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。

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

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

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

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

8. CodePipeline コンソールを開き、パイプラインのMyFirstPipelineリストから 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。

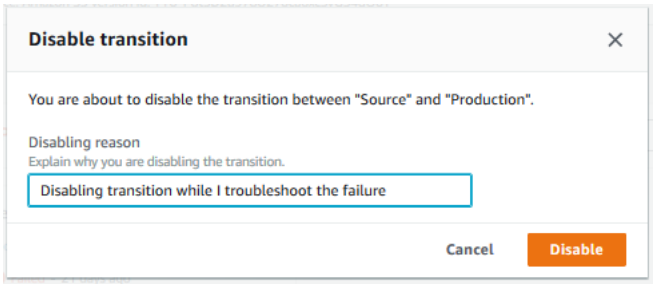
(オプション) ステップ 6: のステージ間の移行を無効または有効にする CodePipeline

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 パイプライン内のステージ間の移行を無効または有効にするには

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

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



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

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

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

ステップ 7: リソースをクリーンアップする

チュートリアル: 4 ステージのパイプラインを作成する 用にこのチュートリアルで作成したリソースの一部を使用することができます。例えば、 CodeDeploy アプリケーションとデプロイを再利用できます。クラウドのフルマネージドビルドサービス CodeBuildである などのプロバイダー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 アプリケーションとそれに関連付けられた Amazon EC2 インスタンスを削除し、最後に S3 バケットを削除します。

このチュートリアルで使用されているリソースをクリーンアップするには

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

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

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

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

パイプラインには 2 つのステージがあります。

- ソースアクションの CodeCommit ソースステージ (ソース) 。
- デプロイアクションの CodeDeploy デプロイステージ (Deploy) 。

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、 と連携するように Git クライアントが設定されていることを確認してください CodeCommit。手順については、「 のセットアップ CodeCommit」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、 にリポジトリを作成します CodeCommit。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、コードをリポジトリにプッシュする前に、コードを維持および更新するローカル CodeCommit リポジトリを作成します。

CodeCommit リポジトリを作成するには

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

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

> **Note**
>
> このチュートリアルの残りのステップでは、 CodeCommit リポジトリの名前**MyDemoRepo**にを使用します。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、リモート CodeCommitリポジトリに接続するためのローカルリポジトリを設定します。

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

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

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

ステップ 2: CodeCommit リポジトリにサンプルコードを追加する

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

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

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

      - を使用して Amazon Linux インスタンスにデプロイする場合は CodeDeploy、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。
      - を使用して Windows Server インスタンスにデプロイする場合は CodeDeploy、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、 でデプロイするための以下のファイルが含まれています CodeDeploy。

      - `appspec.yml` – アプリケーション仕様ファイル (AppSpec ファイル) は、デプロイを管理する CodeDeploy ために が使用する YAML 形式のファイルです。 AppSpec ファイルの詳細については、「 ユーザーガイド」のCodeDeploy AppSpec 「ファイルリファレンスAWS CodeDeploy 」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。複数の CodeDeploy デプロイライフサイクルイベントごとに 1 つのファイルが次のように書き込まれます。
        - (Linux サンプルのみ) `scripts` フォルダ - このフォルダに入っているのはシェルスクリプト `install_dependencies`、`start_server`、`stop_server` です。依存関係をインストールし、自動デプロイのサンプルアプリケーションを起動および停止するために使用されます。
        - (Windows サンプルのみ) `before-install.bat` - `BeforeInstall` デプロイライフサイクルイベントのバッチスクリプトです。このサンプルの前のデプロイ中に書き込ま

れた古いファイルを削除し、新しいファイルを書き込む場所をインスタンス上に作成するために実行されます。

b. 圧縮 (zip) ファイルをダウンロードします。

2. SampleApp_Linux.zip から、以前に作成したローカルディレクトリ (/tmp/MyDemoRepoや などc:\temp\MyDemoRepo) にファイルを解凍します。

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

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

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

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

      i. CodeCommit コンソールを開き、リポジトリリストから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. 次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 CodeCommit。

```
git push
```

4. これで、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がリポジトリの main ブランチ CodeCommit MyDemoRepoに追加され、パイプラインに含める準備が整いました。

ステップ 3: Amazon EC2 Linux インスタンスを作成し、 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、インスタンスに CodeDeploy エージェントをインストールおよび管理できるように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、インスタンスがアプリケーションをデプロイするために CodeDeploy エージェントが使用するファイルを取得し、SSM によるインスタンスの管理を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. [ロールを作成] 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 を選択します。[Select your use case (ユースケースを選択)] で、[EC2] を選択します。[次のステップ: アクセス許可] を選択します。
5. **AmazonEC2RoleforAWSCodeDeploy** という名前のマネージドポリシーを検索して選択します。
6. **AmazonSSMManagedInstanceCore** という名前のマネージドポリシーを検索して選択します。[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。

   [ロールを作成] 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「**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、サンプル CodeDeploy アプリケーションをこのインスタンスにデプロイするアプリケーションを作成します。 CodeDeploy はタグに基づいてデプロイするインスタンスを選択します。
4. アプリケーションイメージと OS イメージ (Amazon マシンイメージ ） で、 AWS ロゴの付いた 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。

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

ステップ 4: でアプリケーションを作成する CodeDeploy

では CodeDeploy、アプリケーションはデプロイするソフトウェアアプリケーションを含むリソースです。後で、 でこのアプリケーションを使用して CodePipeline 、Amazon EC2 インスタンスへのサンプルアプリケーションのデプロイを自動化します。

まず、 がデプロイを実行 CodeDeploy 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. [ロールを作成] を選択します。
4. [信頼されたエンティティを選択] で、[AWS のサービス] を選択します。[Use case] (ユースケース) で、[CodeDeploy] を選択します。表示されたオプションCodeDeployから選択します。[次へ] をクリックします。`AWSCodeDeployRole` マネージドポリシーはロールにアタッチ済みです。
5. [次へ] をクリックします。
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。

でアプリケーションを作成するには CodeDeploy

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。

でデプロイグループを作成するには CodeDeploy

デプロイグループは、デプロイ先のインスタンスやデプロイの速度など、デプロイ関連の設定を定義するリソースです。

1. アプリケーションが表示されるページで、[Create deployment group (デプロイグループの作成)] を選択します。
2. [Deployment group name] (デプロイグループ名) に「**`MyDemoDeploymentGroup`**」と入力します。
3. [サービスロール] で、先ほど作成したサービスロールの ARN を選択します (**`arn:aws:iam::`*`account_ID`*`:role/CodeDeployRole`** など)。
4. [Deployment type] (デプロイタイプ) で、[In-place] (インプレース) を選択します。

5. [環境設定] で、[Amazon EC2 インスタンス] を選択します。キー フィールドに **Name** と入力します。値 フィールドに、(**MyCodePipelineDemo** のような) インスタンスのタグ付けに使用した名前を入力します。
6. AWS Systems Manager を使用したエージェント設定 で、今すぐ を選択し、更新をスケジュールします。これにより、インスタンスにエージェントがインストールされます。Linux インスタンスはすでに SSM エージェントで設定されており、 CodeDeploy エージェントで更新されます。
7. [デプロイ設定] で、[CodeDeployDefault.OneAtaTime] を選択します。
8. ロードバランサー で、ロードバランシングの有効化 が選択されていないことを確認します。この例では、ロードバランサーを設定したり、ターゲットグループを選択したりする必要はありません。
9. デプロイグループの作成 を選択します。

ステップ 5: で最初のパイプラインを作成する CodePipeline

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 パイプラインを作成するには

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

   https://console.aws.amazon.com/codepipeline/ 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。
3. [ステップ 1: パイプラインの設定を選択する] の [パイプライン名] に「**MyFirstPipeline**」と入力します。
4. 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。
5. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロール を選択します。 CodePipeline
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 2: ソースステージ を追加する、ソースプロバイダー で を選択しますCodeCommit。リポジトリ名 で、 で作成した CodeCommit リポジトリの名前を選択しますス

テップ 1: CodeCommit リポジトリを作成する。[ブランチ名] で、[main] を選択し、[次のステップ] を選択します。

リポジトリ名とブランチを選択すると、このパイプライン用に作成される Amazon CloudWatch Events ルールがメッセージに表示されます。

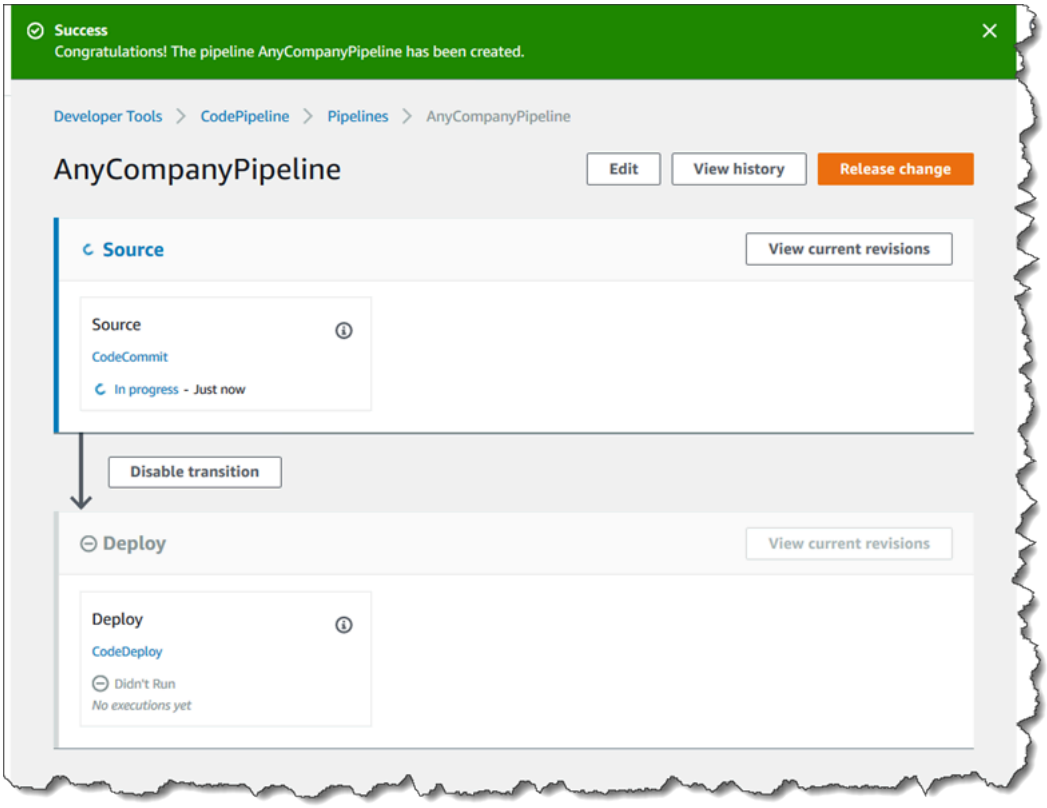
[Change detection options] で、デフォルト値のままにします。これにより、 CodePipeline は Amazon CloudWatch Events を使用してソースリポジトリの変更を検出できます。

[次へ] をクリックします。

8. [Step 3: Add build stage] (ステップ 3: ビルドステージを追加する) 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。[次へ] をクリックします。

> **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、ソースコードをインスタンスにデプロイする前に構築する必要がある場合は、このステップCodeBuildで を設定できます。

9. ステップ 4: デプロイステージ を追加、デプロイプロバイダー で を選択しますCodeDeploy。[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。
10. [ステップ 5: 確認] で情報を確認し、[パイプラインの作成] を選択します。
11. パイプラインは、作成後に実行を開始します。リポジトリから CodeCommit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 デプロイ内の Amazon EC2 インスタンスにウェブページをデプロイすると、進行状況、成功、失敗のメッセージを表示できます。



お疲れ様でした。でシンプルなパイプラインを作成しました CodePipeline。

次に、結果を確認します。

パイプラインが正常に実行されたことを確認するには

1. パイプラインの最初の進行状況を表示します。各ステージのステータスは、[まだ実行はありません] から [進行中] に変わり、その後、[Succeeded (成功)] または [Failed (失敗)] のいずれかに変わります。パイプラインの最初の実行は数分で完了します。
2. パイプラインのステータスが成功と表示されたら、デプロイステージのステータスエリアで を選択しますCodeDeploy。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、「トラブルシューティング CodePipeline」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。
4. [説明] タブの [パブリック DNS] でアドレス (例: `ec2-192-0-2-1.us-west-2.compute.amazonaws.com`) をコピーし、ウェブブラウザのアドレスバーに貼り付けます。

ダウンロードして CodeCommit リポジトリにプッシュしたサンプルアプリケーションのウェブページが表示されます。

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

ステップ 6: CodeCommit リポジトリ内のコードを変更する

パイプラインは、リポジトリにコードが変更されるたびに実行されるように CodeCommit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一部である HTML ファイルを変更します。これらの変更をプッシュすると、パイプラインが再度実行され、変更内容は先ほどアクセスしたウェブアドレスに表示されます。

1. ディレクトリをローカルリポジトリに変更する:

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

2. テキストエディタを使用して、index.html ファイルを変更します。

```
(For Linux or Unix)gedit index.html
(For OS X)open -e index.html
(For Windows)notepad index.html
```

3. index.html ファイルのコンテンツを変更して、背景色およびウェブページのテキストの一部を変更してから、ファイルを保存します。

```
<!DOCTYPE html>
<html>
<head>
  <title>Updated Sample Deployment</title>
  <style>
    body {
      color: #000000;
      background-color: #CCFFCC;
      font-family: Arial, sans-serif;
      font-size:14px;
    }

    h1 {
```

```
      font-size: 250%;
      font-weight: normal;
      margin-bottom: 0;
    }

    h2 {
      font-size: 175%;
      font-weight: normal;
      margin-bottom: 0;
    }
  </style>
</head>
<body>
  <div align="center"><h1>Updated Sample Deployment</h1></div>
  <div align="center"><h2>This application was updated using CodePipeline,
 CodeCommit, and CodeDeploy.</h2></div>
  <div align="center">
    <p>Learn more:</p>
    <p><a href="https://docs.aws.amazon.com/codepipeline/latest/
userguide/">CodePipeline User Guide</a></p>
    <p><a href="https://docs.aws.amazon.com/codecommit/latest/
userguide/">CodeCommit User Guide</a></p>
    <p><a href="https://docs.aws.amazon.com/codedeploy/latest/
userguide/">CodeDeploy User Guide</a></p>
  </div>
</body>
</html>
```

4. 次のコマンドを一度に 1 つずつ実行して、変更を CodeCommit リポジトリにコミットしてプッシュします。

```
git commit -am "Updated sample application files"
```

```
git push
```

パイプラインが正常に実行されたことを確認するには

1. パイプラインの最初の進行状況を表示します。各ステージのステータスは、[まだ実行はありません] から [進行中] に変わり、その後、[Succeeded (成功)] または [Failed (失敗)] のいずれかに変わります。パイプラインの実行は数分以内に完了します。

2. アクションステータスが [成功] と表示されたら、ブラウザで先ほどアクセスしたデモページを更新します。

   更新されたウェブページが表示されます。

ステップ 7: リソースをクリーンアップする

このガイドの他のチュートリアルでは、このチュートリアルで作成したリソースの一部を使用できます。例えば、 CodeDeploy アプリケーションとデプロイを再利用できます。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 アプリケーションとそれに関連付けられた Amazon EC2 インスタンスを削除 CodeCommitし、最後にリポジトリを削除します。

このチュートリアルで使用されているリソースをクリーンアップするには

1. CodePipeline リソースをクリーンアップするには、「」の「パイプラインの削除 AWS CodePipeline」の手順に従います。
2. CodeDeploy リソースをクリーンアップするには、「クリーンアップデプロイチュートリアルリソース」の手順に従います。
3. CodeCommit リポジトリを削除するには、「リポジトリの削除 CodeCommit」の手順に従います。

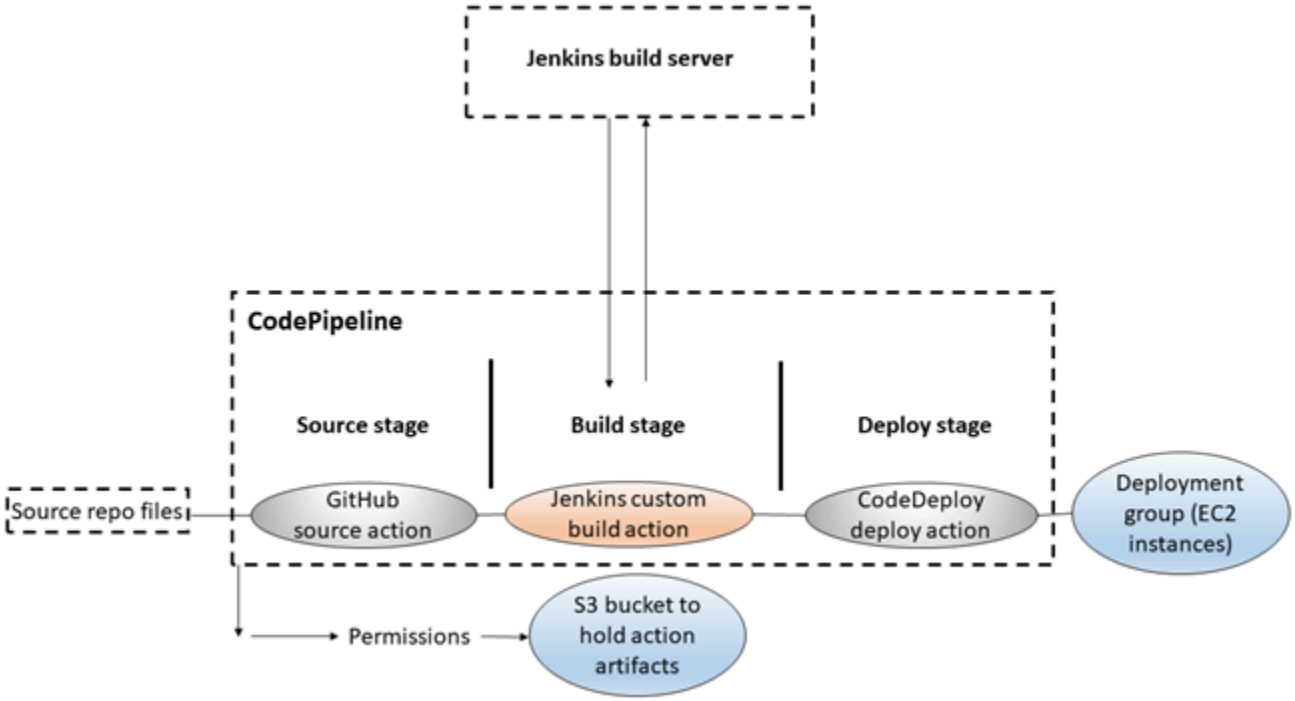
ステップ 8: 詳細情報

の CodePipeline 仕組みの詳細をご覧ください。

- ステージ、アクション、パイプラインの仕組みの詳細については、「CodePipeline の概念」を参照してください。
- を使用して実行できるアクションの詳細については、 CodePipeline「」を参照してください CodePipeline アクションタイプとの統合。
- この詳細なチュートリアル「チュートリアル: 4 ステージのパイプラインを作成する」をお試しください。デプロイ前にコードをビルドする手順を含むマルチステージパイプラインが作成されます。

チュートリアル: 4 ステージのパイプラインを作成する

これで、「チュートリアル: シンプルなパイプラインを作成する (S3 バケット)」または「チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)」に最初のパイプラインが作成されたため、より複雑なパイプラインを作成できるようになりました。このチュートリアルでは、ソースのリポジトリを使用する GitHub 4 段階のパイプライン、プロジェクトを構築するための Jenkins ビルドサーバー、構築され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 アプリケーションの作成について説明します。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、ソースアクションの pipeline. AWS resources 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。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) リージョンでパイプラインを作成する場合、 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、「でクロスリージョンアクションを追加する CodePipeline」を参照してください。

このチュートリアルを開始するには、「の開始方法 CodePipeline」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: でパイプラインを作成する CodePipeline
- 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

Jenkins と統合 AWS CodePipeline するには、 で使用する Jenkins CodePipeline のインスタンスに Plugin for Jenkins をインストールする必要があります CodePipeline。また、Jenkins プロジェクトと 間のアクセス許可に使用する専用の IAM ユーザーまたはロールを設定する必要があります CodePipeline。Jenkins を統合する最も簡単な方法は、Jenkins 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 CodePipeline することです。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。「」の手順に従って、 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できます GitHub リポジトリにサンプルをコピーまたはクローンする。.zip ファイルだけでなく、サンプル全体が GitHub リポジトリに必要です。
> 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

GitHub リポジトリにサンプルをコピーまたはクローンする

サンプルのクローンを作成して GitHub リポジトリにプッシュするには

1. GitHub リポジトリからサンプルコードをダウンロードするか、リポジトリのクローンをローカルコンピュータに作成します。2 つのサンプルパッケージがあります。
   - Amazon Linux、RHEL、または Ubuntu Server インスタンスにサンプルをデプロイする場合は、codepipeline-jenkins-aws-codedeploy_linux.zip を選択します。
   - Windows Server インスタンスにサンプルをデプロイする場合は、CodePipeline-Jenkins-AWSCodeDeploy_Windows.zip を選択します。
2. リポジトリから、[Fork] を選択してサンプルリポジトリを Github アカウントのレポジトリに複製します。詳細については、 GitHub 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、Jenkins サーバーをホストする EC2 インスタンスを起動し、IAM ロールを使用して、 を操作するため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てください CodePipeline。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで、[ロール]、[ロールを作成する] の順に選択します。
3. [Select type of trusted entity] (信頼されたエンティティの種類を選択) で、[AWS のサービス] を選択します。[Choose the service that will use this role (このロールを使用するサービスを選択)] で、[EC2] を選択します。[Select your use case (ユースケースを選択)] で、[EC2] を選択します。
4. [次のステップ: 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. レビューページのロール名 で、Jenkins 統合用に作成するロールの名前 (例: *`JenkinsAccess`*) 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 EC2 インスタンスを作成するときは、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 (など) を選択してください*`JenkinsAccess`*。

インスタンスロールおよび Amazon EC2 の詳細については、「Amazon EC2 の IAM ロール」、「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」、「AWS のサービスにアクセス許可を委任するロールの作成」を参照してください。

Jenkins と Plugin for Jenkins CodePipeline をインストールして設定する

Jenkins と Jenkins 用 CodePipeline プラグインをインストールするには

1. Jenkins をインストールする 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 (など) を必ず選択します*`JenkinsAccess`*。EC2 インスタンス作成の詳細については、「Amazon EC2 ユーザーガイド」の 「Amazon EC2 インスタンスの起動」を参照してください。

> **Note**
>
> 使用する Jenkins リソースがすでにある場合は、特別な IAM ユーザーを作成し、そのユーザーに `AWSCodePipelineCustomActionAccess` 管理ポリシーを適用してから、Jenkins リソースに対してそのユーザーのアクセス認証情報を設定して使用する必要があります。Jenkins UI を使用して認証情報を指定する場合は、HTTPS のみを許可するように Jenkins を設定します。詳細については、「トラブルシューティング CodePipeline」を参照してください。

2. EC2 インスタンスに Jenkins をインストールします。詳細については、Jenkins のドキュメントの「Jenkins のインストール」と「Jenkins の開始とアクセス」のほか、「との製品とサービスの統合 CodePipeline」の「details of integration with Jenkins」を参照してください。
3. Jenkins を起動し、ホーム ページで [Manage Jenkins] (Jenkins の管理) を選択します。
4. [Jenkins の管理] ページで、[プラグインの管理] を選択します。
5. [Available] タブを選択し、[Filter] 検索ボックスに「**`AWS CodePipeline`**」と入力します。リストからCodePipeline 「Jenkins 用プラグイン」を選択し、「今すぐダウンロード」を選択し、再起動後にインストールします。
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. 項目名 に、Jenkins プロジェクトの名前を入力します (例: *`MyDemoProject`*) 。[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 の要件を満たしていることを確認します CodePipeline。詳細については、「のクォータ AWS CodePipeline」を参照してください。

10. プロジェクトの設定ページで、[Execute concurrent builds if necessary] (必要な場合に複数のビルドを並列実行する) チェックボックスをオンにします。[ソースコードの管理] で、[AWS CodePipeline] を選択します。EC2 インスタンスに Jenkins をインストールし、 CodePipeline と Jenkins の統合用に作成した IAM ユーザーのプロファイル AWS CLI で を設定した場合は、他のすべてのフィールドを空のままにします。

11. 詳細 を選択し、プロバイダー で、アクションのプロバイダーの名前を に表示されるとおりに入力します CodePipeline （ 例: *MyJenkinsProviderName* ）。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、 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 CodePipeline。詳細については、「のクォータ AWS CodePipeline」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 CodePipeline ごとにポーリングされます。

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. ビルド後のアクションを追加 を選択し、パAWS CodePipeline ブリッシャー を選択します。[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: でパイプラインを作成する CodePipeline

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. 必要に応じて、リージョンセレクターを使用し、パイプラインリソースの配置先のリージョンに切り替えます。例えば、前のチュートリアルでリソースを `us-east-2` に作成した場合は、リージョンセレクターを必ず米国東部 (オハイオ) に設定します。

   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 CodePipeline、「AWS CodePipeline エンドポイントとクォータ」を参照してください。
3. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
4. [Step 1: Choose pipeline settings (ステップ 1: パイプラインの設定の選択)] 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。
5. このチュートリアルの目的では、[パイプラインタイプ] で、[V1] を選択します。[V2] を選択することもできますが、パイプラインタイプは特性と価格が異なることに注意してください。詳細については、「パイプラインのタイプ」を参照してください。
6. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロールを選択します。 CodePipeline
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 2: ソースステージの追加ページで、ソースプロバイダー で を選択しますGitHub。
9. 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 ソースアクションの接続を作成または管理するには、「」を参照してくださいGitHub 接続。
10. [Step 3: Add build stage (ステップ 3: ビルドステージの追加)] で、[Jenkins の追加] を選択します。プロバイダー名 に、Jenkins 用 CodePipeline プラグインで指定したアクションの名前を入力します (例: *`MyJenkinsProviderName`*)。この名前は、Jenkins 用 CodePipeline プラグインの名前と完全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 で、 など、Jenkins で作成したプロジェクトの名前を入力し*`MyDemoProject`*、次へ を選択します。
11. ステップ 4: 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利用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)。[デプロイプロバイダ] 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**
>
> 独自のリソースを使用することも、新しい CodeDeploy リソースを作成することもできますが、追加コストが発生する可能性があります。

12. [ステップ 5: 確認] で情報を確認し、[パイプラインの作成] を選択します。
13. 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 に構築し、デプロイ内の CodeDeploy各 Amazon EC2 インスタンスにウェブページをデプロイすると、進行状況、成功、失敗のメッセージを表示できます。

ステップ 3: パイプラインに別のステージを追加する

次に、テストステージ、テストアクションの順で、サンプルに含まれている Jenkins テストを使用するステージに追加し、ウェブページにコンテンツが含まれているかどうかを確認します。このテストは、デモンストレーションのみを目的としています。

> **Note**
>
> 他のステージをパイプラインに追加しない場合は、パイプラインのステージ (Staging) へテストアクションを追加します。その前後にデプロイアクションを行います。

パイプラインにテストステージを追加する

トピック

- インスタンスの IP アドレスを検索する
- デプロイのテスト用に Jenkins プロジェクトを作成する
- 4 番目のステージを作成する

インスタンスの IP アドレスを検索する

コードをデプロイしたインスタンスの IP アドレスを確認するには

1. パイプラインのステータスが "Succeeded" と表示されたら、[Staging] ステージのステータス領域で [詳細] を選択します。
2. [デプロイの詳細] (デプロイの詳細) セクションの [インスタンス ID] で、正常にデプロイされたいずれかのインスタンスの ID を選択します。
3. インスタンスの IP アドレス (*192.168.0.4* など) をコピーします。この IP アドレスは Jenkins テストで使用します。

デプロイのテスト用に Jenkins プロジェクトを作成する

Jenkins プロジェクトを作成するには

1. Jenkins をインストールしたインスタンスで、Jenkins を開き、メインページから [New Item] (新しい項目) を選択します。
2. 項目名 に、Jenkins プロジェクトの名前を入力します (例: *MyTestProject*) 。[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要件を満たしていることを確認します CodePipeline 。詳細については、「のクォータ AWS CodePipeline」を参照してください。

3. プロジェクトの設定ページで、[Execute concurrent builds if necessary] (必要な場合に複数のビルドを並列実行する) チェックボックスをオンにします。[ソースコードの管理] で、[AWS CodePipeline] を選択します。EC2 インスタンスに Jenkins をインストールし、 CodePipeline と Jenkins の統合用に作成した IAM ユーザーのプロファイル AWS CLI で を設定した場合は、他のすべてのフィールドを空のままにします。

   > **Important**
   >
   > Jenkins プロジェクトを設定していてAmazon EC2 インスタンスにインストールされていない場合、または 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設定で必要なフィール

ドに入力し、Jenkins と 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 CodePipeline。

4. [詳細設定] を選択してから、[カテゴリ] で [テスト] を選択します。
5. プロバイダー で、ビルドプロジェクトに使用したのと同じ名前 (など*MyJenkinsProviderName*) を入力します。この名前は、このチュートリアルの後半でパイプラインにテストアクションを追加するときに使用します。

> **Note**
>
> この名前は、 アクションの CodePipeline 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、「のクォータ AWS CodePipeline」を参照してください。

6. [Build Triggers] (トリガーのビルド) で、チェックボックスをすべてオフにし、[Poll SCM] (SCM のポーリング) を選択します。[Schedule] に、以下のようにスペースで区切ってアスタリスクを 5 つ入力します。

```
* * * * *
```

これは 1 分 CodePipeline ごとにポーリングされます。

7. [ビルド] で、[Add build step] (ビルドステップの追加) を選択します。Amazon Linux、RHEL、または Ubuntu Server インスタンスにデプロイする場合は、[シェルの実行] を選択します。次に、以下のように入力します。IP アドレスは、先ほどコピーした EC2 インスタンスのアドレスです。

```
TEST_IP_ADDRESS=192.168.0.4 rake test
```

Windows Server インスタンスにデプロイする場合は、[Execute batch command (バッチコマンドの実行)] を選択し、以下のように入力します。ここで、IP アドレスは、先ほどコピーした EC2 インスタンスのアドレスです。

```
set TEST_IP_ADDRESS=192.168.0.4 rake test
```

> **Note**
>
> このテストでは、デフォルトのポート 80 を想定しています。別のポートを指定する場合は、以下のように test port ステートメントを追加します。

```
TEST_IP_ADDRESS=192.168.0.4 TEST_PORT=8000 rake test
```

8. ビルド後のアクションを追加 を選択し、パAWS CodePipeline ブリッシャー を選択します。[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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. 名前 で、作成したパイプラインの名前 を選択します MySecondPipeline。
3. パイプライン詳細ページで、[編集] を選択します。
4. [編集] ページで [+ Stage (+ ステージの追加)] を選択して、ビルドステージの直後にステージを追加します。
5. 新しいステージの名前フィールドに名前 (**Testing** など) を入力し、[+ Add action group (+ アクショングループの追加)] を選択します。
6. アクション名 で、*MyJenkinsTest-Action* #入力します。テストプロバイダー で、Jenkins で指定したプロバイダー名を選択します (例: *MyJenkinsProviderName*)。プロジェクト名 に、Jenkins で作成したプロジェクトの名前を入力します (例: *MyTestProject*)。「入力アーティファクト」で、デフォルト名が の Jenkins ビルドからアーティファクトを選択し*BuildArtifact*、「完了」を選択します。

> **Note**
>
> Jenkins テストアクションは Jenkins ビルドステップで構築されたアプリケーションで動作するため、テストアクションへの入力アーティファクトのビルドアーティファクトを使用します。

入力アーティファクトと出力アーティファクト、およびパイプラインの構造の詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

7. [編集] ページで、[パイプラインの変更を保存] を選択します。[パイプラインの変更を保存] ダイアログボックスで、[保存して続行] を選択します。
8. パイプラインに新しいステージが追加されましたが、パイプラインの別の実行をトリガーした変更がないため、そのステージのステータスは [まだ実行はありません] と表示されます。修正したパイプラインによりサンプルを実行するには、パイプラインの詳細ページで [リリースの変更] を選択します。

   パイプラインビューには、パイプラインのステージとアクション、それらの 4 つのステージを実行しているリビジョンの状態が表示されます。パイプラインがすべてのステージを実行するのにかかる時間は、アーティファクトのサイズ、ビルドとテストのアクションの複雑さ、その他の要因によって異なります。

ステップ 4: リソースをクリーンアップする

これらのチュートリアルが完了したら、使用したパイプラインおよびリソースを削除する必要があるため、このリソースに対する継続利用料金がかかることはありません。を使用し続ける予定がない場合は CodePipeline、パイプラインを削除してから、 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、最後にアーティファクトの保存に使用された Amazon S3 バケットを削除します。また、 GitHub リポジトリなど、他のリソースを引き続き使用する場合は、削除するかどうかも検討する必要があります。

このチュートリアルで使用されているリソースをクリーンアップするには

1. 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、「 のクリーンアップ」の手順に従います。
3. インスタンスリソースをクリーンアップするには、Jenkins をインストールした EC2 インスタンスを削除します。詳細については、「インスタンスのクリーンアップ」を参照してください。
4. パイプラインをさらに作成したり、 CodePipeline 再度使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」の手順に従います。

チュートリアル: パイプラインの状態変更に関する E メール通知を受信するように CloudWatch イベントルールを設定する

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 CloudWatch イベントルールを設定できます。 CloudWatch イベントを使用してパイプラインの状態変更の通知を設定する方法の詳細については、「」を参照してください CodePipeline イベントのモニタリング。

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 イベントルールの作成時に入力トランスフォーマーメソッドを使用します。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 イベントルールなど、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

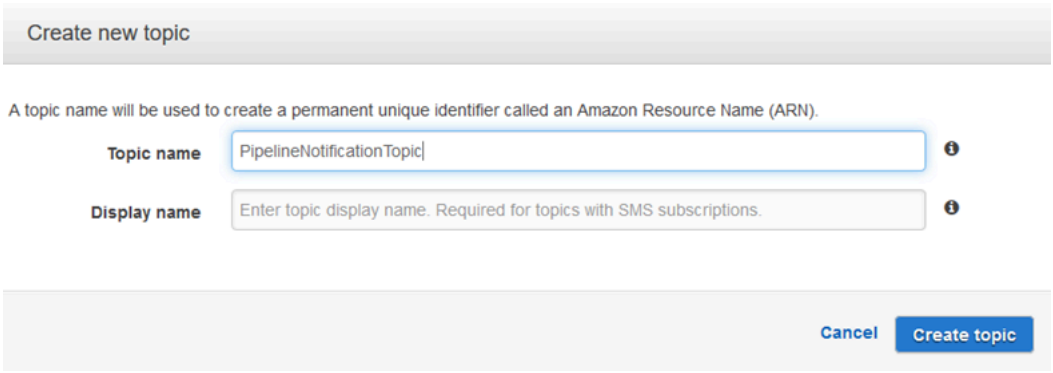
ステップ 1 : Amazon SNS を使用して E メール通知をセットアップします。

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 トピックは、 CloudWatch

イベントルールのターゲットとして追加されます。詳細については、「Amazon Simple Notification Service デベロッパーガイド」を参照してください。

Amazon SNS でトピックを作成または識別します。 CodePipeline は CloudWatch 、イベントを使用してAmazon SNSを介してこのトピックに通知を送信します。トピックを作成するには:

1. Amazon SNS コンソール (https://console.aws.amazon.com/sns) を開きます。
2. [トピックの作成] を選択します。
3. [Create new topic (新しいトピックの作成)] ダイアログボックスの [Topic name (トピック名)] で、トピックの名前 (例: **PipelineNotificationTopic**) を入力します。



4. [Create topic] (トピックの作成) を選択します。

   詳細については、Amazon SNS デベロッパーガイド の トピックの作成 を参照してください。

1 つかそれ以上の受信者にトピックをサブスクライブさせ、E メール通知を受け取ります。受信者にトピックをサブスクライブさせるには:

1. Amazon SNS コンソールで、トピック リストから、新しいトピックの横にあるチェックボックスを選択します。[Actions, Subscribe to topic] を選択します。
2. [Create subscription] ダイアログボックスで、ARN が [Topic ARN] に表示されていることを確認します。
3. [Protocol (プロトコル)] として [Email (E メール)] を選択します。
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 トピックのサブスクライブ を参照してください。

ステップ 2: ルールを作成し SNS トピックをターゲットとして追加する

イベントソース CodePipeline として を使用してイベント CloudWatch 通知ルールを作成します。

1. https://console.aws.amazon.com/cloudwatch/ で CloudWatch コンソールを開きます。
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"
       ]
     }
   }
   ```

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 がビルド通知を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、アクション 、編集 を選択します。ルールを編集し、[詳細設定] を選択し、[Update] を選択します。

    ルールを使用して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、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、アクション 、削除 を選択します。

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、Amazon Events API リファレンスの「クリーンアップ (Amazon SNS トピックからのサブスクリプション解除) 」および「 リファレンスDeleteRule」を参照してください。 CloudWatch

チュートリアル: で Android アプリを構築して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 リポジトリ内のソースコードを使用して Android アプリを構築およびテストするためのパイプラインを作成および設定する方法を示します。パイプラインは新しい GitHub コミットの到着を検出し、 CodeBuildを使用してアプリケーションを構築し、Device Farm を使用してテストします。

> **Important**
>
> 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、ソースアクションの pipeline. AWS resources 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。パイプラインを作成するのと同じ AWS リージョンに常に作成する必要があります。例えば、米国東部 (オハイオ) リージョンでパイプラインを作成する場合、 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、「でクロスリージョンアクションを追加する CodePipeline」を参照してください。

既存の Android アプリとテスト定義を使用してこれを試すか、 Device Farmが提供したサンプルアプリケーションとテスト定義 を使用できます。

> **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。
2. プロジェクトを選択します。ブラウザで、新しいプロジェクトの URL をコピーします。URL には、プロジェクト ID が含まれます。
3. プロジェクト ID をコピーしてメモしておきます。これは、 でパイプラインを作成するときに使用します CodePipeline。

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 CodePipeline ように を設定する

1. アプリコードのルートbuildspec.ymlに という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。 はこのファイル CodeBuild を使用して、アプリの構築に必要なコマンドを実行し、アーティファクトにアクセスします。

   ```
   version: 0.2

   phases:
     build:
       commands:
         - chmod +x ./gradlew
         - ./gradlew assembleDebug
   artifacts:
     files:
        - './android/app/build/outputs/**/*.apk'
     discard-paths: yes
   ```

2. (オプション) Calabash または Appium を使用してアプリケーションをテストする場合は、テスト定義ファイルをリポジトリに追加します。後のステップで、定義を使用してテストスイートを実行するように Device Farm を設定できます。

   Device Farm の組み込みのテストを使用する場合は、このステップを省略できます。

3. パイプラインを作成してソースステージを追加するには、以下の手順を実行します。

   a. にサインイン AWS Management Console し、https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

   b. パイプラインの作成 を選択します。[Step 1: Choose pipeline settings (ステップ 1: パイプラインの設定の選択)] 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。

c. 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。

d. [サービスロール] で、[New service role (新しいサービスロール)] は選択したままにして、[Role name (ロール名)] は変更しません。既存のサービスロール (ある場合) を使用することもできます。

> **Note**
>
> 2018 年 7 月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、Device Farm のアクセス許可を追加する必要があります。これを行うには、IAM コンソールを開き、ロールを見つけて、ロールのポリシーに次の許可を追加します。詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。
>
> ```
> {
>      "Effect": "Allow",
>      "Action": [
>         "devicefarm:ListProjects",
>         "devicefarm:ListDevicePools",
>         "devicefarm:GetRun",
>         "devicefarm:GetUpload",
>         "devicefarm:CreateUpload",
>         "devicefarm:ScheduleRun"
>      ],
>      "Resource": "*"
> }
> ```

e. [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。

f. ステップ 2: ソースステージの追加ページで、ソースプロバイダー で を選択しますGitHub。

g. 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 ソースアクションの接続を作成または管理するには、「」を参照してくださいGitHub 接続。

h. [リポジトリ] で、ソースリポジトリを選択します。

i. [ブランチ] で、使用するブランチを選択します。

j. 出典アクションについては、残りのデフォルトのままにしておきます。[次へ] をクリックします。

4. [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] がデフォルトでパイプラインリージョンにな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. 「 に進む」を選択します CodePipeline。これによりコンソール CodePipelineに戻り、リポジトリ`buildspec.yml`内の を使用して設定を行う CodeBuild プロジェクトが作成されます。ビルドプロジェクトでは、サービスロールを使用して AWS のサービス のアクセス許可を管理します。このステップには数分かかる場合があります。

   i. [次へ] をクリックします。

5. [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] ページで、[Skip deploy stage (デプロイステージのスキップ)] を選択し、[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] をクリックします。

6. [Step 5: Review (ステップ 5: 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。

7. 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] がデフォルトでパイプラインリージョンになることを許可します。

f. [入力アーティファクト] で、テストステージに先立つステージの出力アーティファクトと一致する入力アーティファクト (`BuildArtifact` など) を選択します。

AWS CodePipeline コンソールでは、パイプライン図の情報アイコンにカーソルを合わせると、各ステージの出力アーティファクトの名前を確認できます。パイプラインがソースステージから直接アプリケーションをテストする場合は、 を選択しますSourceArtifact。パイプラインにビルドステージが含まれている場合は、 を選択しますBuildArtifact。

g. でProjectId、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. でDevicePoolArn、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できるデバイスプール ARN を取得するには、 AWS CLI を使用して次のコマンドを入力します。

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

i. でAppType、Android と入力します。

以下は、 の有効な値のリストですAppType。

- 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、次にテスト 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。

以下は、 の有効な値のリストですTestType。

- APPIUM_JAVA_JUNIT
- APPIUM_JAVA_TESTNG
- APPIUM_NODE
- APPIUM_RUBY

- APPIUM_PYTHON
- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE
- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZZ
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
> カスタム環境ノードはサポートされていません。

l. 残りのフィールドにはテストおよびアプリケーションタイプに適した構成を入力します。

m. (オプション) [アドバンスト] で、テストランの情報の設定を行います。

n. [保存] を選択します。

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

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

チュートリアル: で iOS アプリケーション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 Amazon CloudWatch Events を通じて保存された変更の到着を検出し、Device Farm を使用してビルドされたアプリケーションをテストします。

> **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、ソースアクションの pipeline. AWS resources 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。パイプラインを作成するのと同じ AWS リージョンに常に作成する必要があります。例えば、米国東部 (オハイオ) リージョンでパイプラインを作成する場合、 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、「でクロスリージョンアクションを追加する CodePipeline」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。

> **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。
2. プロジェクトを選択します。ブラウザで、新しいプロジェクトの URL をコピーします。URL には、プロジェクト ID が含まれます。
3. プロジェクト ID をコピーしてメモしておきます。これは、 でパイプラインを作成するときに使用します CodePipeline。

   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 CodePipeline ように を設定する (Amazon S3 の例)

1. バージョニングが有効になっている S3 バケットを作成するか、使用します。「ステップ 1: アプリケーションの S3 バケットを作成する」の手順に従って、S3 バケットを作成します。
2. バケットの Amazon S3 コンソールで、アップロード を選択し、指示に従って .zip ファイルをアップロードします。

   サンプルアプリケーションは、.zip ファイルにパッケージ化する必要があります。
3. パイプラインを作成してソースステージを追加するには、以下の手順を実行します。

   a. にサインイン AWS Management Console し、https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

   b. パイプラインの作成 を選択します。[Step 1: Choose pipeline settings (ステップ 1: パイプラインの設定の選択)] 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。

   c. 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。

   d. [サービスロール] で、[New service role (新しいサービスロール)] は選択したままにして、[Role name (ロール名)] は変更しません。既存のサービスロール (ある場合) を使用することもできます。

   > **Note**
   >
   > 2018 年 7 月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、Device Farm のアクセス許可を追加する必要があります。これを行うには、IAM コンソールを開き、ロールを見つけて、ロールのポリシーに次の許可を追加します。詳細については、「 CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

   ```
   {
        "Effect": "Allow",
        "Action": [
           "devicefarm:ListProjects",
           "devicefarm:ListDevicePools",
           "devicefarm:GetRun",
           "devicefarm:GetUpload",
           "devicefarm:CreateUpload",
   ```

```
            "devicefarm:ScheduleRun"
        ],
        "Resource": "*"
    }
```

e. [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。

f. [Step 2: Add source stage] (ステップ 2: 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。

g. [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。

h. [次へ] をクリックします。

4. [ビルド] で、パイプラインのプレースホルダービルドステージを作成します。これにより、ウィザードでパイプラインを作成することができます。ウィザードを使用して 2 ステージパイプラインを作成した後は、このプレースホルダービルドステージは不要になります。パイプラインが完了した後、この第 2 ステージが削除され、ステップ 5 で新しいテストステージが追加されます。

   a. [ビルドプロバイダ] で、[Jenkins の追加] を選択します。このビルド選択はプレースホルダーです。それは使用されていません。

   b. [プロバイダ名] に名前を入力します。名前はプレースホルダーです。それは使用されていません。

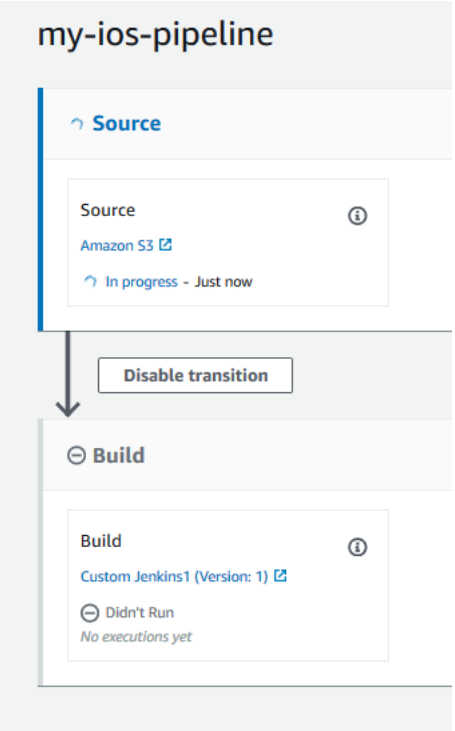
   c. [サーバー URL] にテキストを入力します。テキストはプレースホルダーです。それは使用されていません。

   d. [プロジェクト名] に名前を入力します。名前はプレースホルダーです。それは使用されていません。

   e. [次へ] をクリックします。

   f. [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] ページで、[Skip deploy stage (デプロイステージのスキップ)] を選択し、[スキップ] を選択して警告メッセージを受け入れます。

   g. [Step 5: Review (ステップ 5: 確認)] で、[パイプラインの作成] を選択します。ソースとビルドステージを示す図が表示されます。



5. 次のようにして、Device Farm テストアクションをパイプラインに追加します。

   a. 右上の [編集] を選択します。

   b. [Edit stage (ステージの編集)] を選択します。[削除] を選択します。これにより、パイプライン作成のためには使用しないプレースホルダーステージが削除されます。

   c. 図の最下部で [+ Add stage] (+ ステージの追加) を選択します。

   d. [ステージ名] にステージ名 (Test など) を入力し、[Add stage (ステージの追加)] を選択します。

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

   f. アクション名 に、 などの名前を入力します DeviceFarmTest。

   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] がデフォルトでパイプラインリージョンになることを許可します。

   h. [入力アーティファクト] で、テストステージに先立つステージの出力アーティファクトと一致する入力アーティファクト (`SourceArtifact` など) を選択します。

      AWS CodePipeline コンソールでは、パイプライン図の情報アイコンにカーソルを合わせると、各ステージの出力アーティファクトの名前を確認できます。パイプラインがソースステージから直接アプリケーションをテストする場合は、 を選択しますSourceArtifact。パイプラインにビルドステージが含まれている場合は、 を選択しますBuildArtifact。

i. でProjectId、Device Farm 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. でDevicePoolArn、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できるデバイスプール ARN を取得するには、 AWS CLI を使用して次のコマンドを入力します。

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

k. でAppType、iOS と入力します。

以下は、 の有効な値のリストですAppType。

- iOS
- Android
- Web

l. [デプロイ] に、コンパイルされたアプリケーションパッケージのパスを入力します。パスは、テストステージの入力アーティファクトのルートを基準とする相対パスです。このパスは `ios-test.ipa` に似ています。

m. TestTypeにテストのタイプを入力し、次にテスト 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。

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。でFuzzEventCount、6000 など、時間をミリ秒単位で入力します。でFuzzEventThrottle、50 などの時間をミリ秒単位で入力します。

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。

以下は、 の有効な値のリストですTestType。

- APPIUM_JAVA_JUNIT
- APPIUM_JAVA_TESTNG
- APPIUM_NODE
- APPIUM_RUBY

- APPIUM_PYTHON
- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE
- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZZ
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
> カスタム環境ノードはサポートされていません。

n. 残りのフィールドにはテストおよびアプリケーションタイプに適した構成を入力します。

o. (オプション) [アドバンスト] で、テストランの情報の設定を行います。

p. [保存] を選択します。

q. 編集中のステージで、[完了] を選択します。 AWS CodePipeline 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。このチュートリアルでは、製品テンプレートを Service Catalog にデプロイし、ソースリポジトリで行った変更 (、 GitHub、 CodeCommitまたは Amazon S3 で作成済み) を配信するパイプラインを作成して設定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である場合、すべての出典ファイルを 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。 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 (個別の設定ファイルなし)。オプション 1: 設定ファイルを使用しないで Service Catalog にデプロイする を参照してください。

  > **Note**
  >
  > テンプレートファイルは YAML または JSON 形式で作成できます。

- Service Catalog で製品を作成し、テンプレートファイルをソースリポジトリにアップロードします。個別の設定ファイルを使用して製品バージョンとデプロイ設定を指定します。オプション 2: 設定ファイルを使用して Service Catalog にデプロイする を参照してください。

オプション 1: 設定ファイルを使用しないで Service Catalog にデプロイする

この例では、S3 バケットのサンプル AWS CloudFormation テンプレートファイルをアップロードし、Service Catalog で製品を作成します。次に、パイプラインを作成し、 CodePipeline コンソールでデプロイ設定を指定します。

ステップ 1: サンプルテンプレートファイルをソースリポジトリにアップロードする

1. テキストエディタを開きます。以下のコードをファイルに貼り付けて、サンプルテンプレートを作成します。`S3_template.json` という名前でファイルを保存します。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
```

```
  "Description": "CloudFormation Sample Template S3_Bucket: Sample template showing
 how to create a privately accessible S3 bucket. **WARNING** This template creates
 an S3 bucket. You will be billed for the resources used if you create a stack from
 this template.",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {}
    }
  },
  "Outputs": {
    "BucketName": {
      "Value": {
        "Ref": "S3Bucket"
      },
      "Description": "Name of Amazon S3 bucket to hold website content"
    }
  }
}
```

このテンプレートにより AWS CloudFormation 、 は Service Catalog で使用できる S3 バケットを作成できます。

2. `S3_template.json` ファイルを AWS CodeCommit リポジトリにアップロードします。

ステップ 2: Service Catalog で製品を作成する

1. IT 管理者として、Service Catalog コンソールにサインインし、[製品] ページに移動して、[新しい製品のアップロード] を選択します。
2. [新しい製品のアップロード] ページで、以下の手順を実行します。
   a. [製品名] に、新しい製品に使用する名前を入力します。
   b. [Description (説明)] に製品カタログの説明を入力します。この説明は、製品リストでユーザーが正しい製品を選択できるように表示されます。
   c. [提供元] に IT 部門または管理者の名前を入力します。
   d. [次へ] をクリックします。
3. (オプション) [サポート詳細の入力] に製品サポートの連絡先情報を入力し、[次へ] を選択します。
4. [バージョンの詳細] に以下の情報を入力します。

a. [Upload a template file (テンプレートファイルをアップロード)] を選択します。S3_template.json ファイルを見つけ、アップロードします。

b. [バージョンタイトル] に、製品バージョンの名前 (**devops S3 v2** など) を入力します。

c. [Description (説明)] に、このバージョンと他のバージョンを区別するための詳細を入力します。

d. [次へ] をクリックします。

5. [確認] ページで、情報が正しいことを確認し、[作成] を選択します。

6. ブラウザの [製品] ページで、新しい製品の URL をコピーします。これには製品 ID が含まれています。この製品 ID をコピーして保持します。これは、 でパイプラインを作成するときに使用します CodePipeline。

   以下に示しているのは、my-product という製品の URL です。製品 ID を抽出するには、等号 (=) とアンパサンド (&) との間の値をコピーします。この例では、製品 ID は prod-example123456 です。

   ```
   https://<region-URL>/servicecatalog/home?region=<region>#/admin-products?
   productCreated=prod-example123456&createdProductTitle=my-product
   ```

   > **Note**
   >
   > ページから移動する前に、製品の URL をコピーします。このページから移動したら、CLI を使用して製品 ID を取得する必要があります。

   数秒後、製品が [製品] ページに表示されます。製品をリストに表示するには、ブラウザの更新が必要になる場合があります。

ステップ 3: パイプラインを作成する

1. パイプラインに名前を付け、パイプラインのパラメータを選択するには、以下の手順を実行します。

   a. にサインイン AWS Management Console し、https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

b. [開始方法] を選択します。[パイプラインの作成] を選択し、パイプラインの名前を入力します。

c. 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。

d. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロール を選択します。 CodePipeline

e. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。

2. ソースステージを追加するには、以下の手順を実行します。

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

b. [リポジトリ名] と [ブランチ名] に、ソースアクションに使用するリポジトリとブランチを入力します。

c. [次へ] をクリックします。

3. [Add build stage (ビルドステージの追加)] で [Skip build stage (ビルドステージのスキップ)] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

4. [Add deploy stage (デプロイステージの追加)] で、以下の手順を実行します。

a. [デプロイプロバイダ] で、[AWS Service Catalog] を選択します。

b. 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。

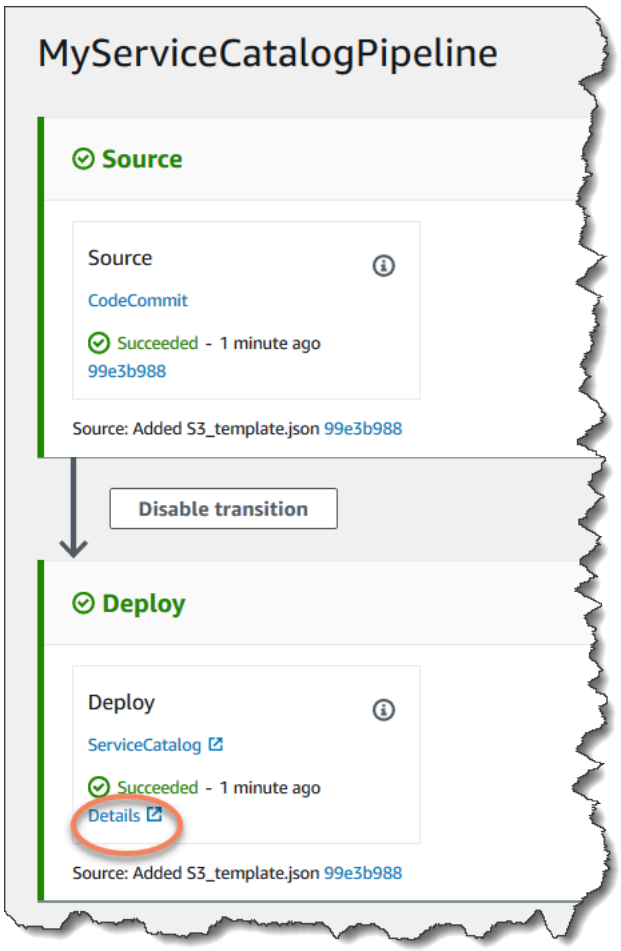
f. [製品バージョン名] に、Service Catalog で指定した製品バージョンの名前を入力します。テンプレートの変更を新しい製品バージョンにデプロイする場合は、同じ製品の以前の製品バージョンで使用されていない製品バージョン名を入力します。

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. [次へ] をクリックします。

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 [Details (詳細)] を選択します。これにより、Service Catalog で製品が開きます。



7. 製品情報で、バージョン名を選択して製品テンプレートを開きます。テンプレートのデプロイを表示します。

ステップ 4: 変更をプッシュして Service Catalog で製品を確認する

1. CodePipeline コンソールでパイプラインを表示し、ソースステージで詳細 を選択します。コンソールでソース AWS CodeCommit リポジトリが開きます。[Edit (編集)] を選択し、ファイルの内容 (説明など) を変更します。

```
"Description": "Name of Amazon S3 bucket to hold and version website content"
```

2. 変更をコミットし、プッシュします。変更をプッシュした後、パイプラインが開始されます。パイプラインの実行が完了したら、デプロイステージで [詳細] を選択して、製品を Service Catalog で開きます。
3. 製品情報で、新しいバージョン名を選択して製品テンプレートを開きます。デプロイされたテンプレートの変更を表示します。

オプション 2: 設定ファイルを使用して Service Catalog にデプロイする

この例では、S3 バケットのサンプル AWS CloudFormation テンプレートファイルをアップロードし、Service Catalog で製品を作成します。デプロイ設定を指定する個別の設定ファイルもアップロードします。次に、パイプラインを作成し、設定ファイルの場所を指定します。

ステップ 1: サンプルテンプレートファイルをソースリポジトリにアップロードする

1. テキストエディタを開きます。以下のコードをファイルに貼り付けて、サンプルテンプレートを作成します。S3_template.json という名前でファイルを保存します。

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Description": "CloudFormation Sample Template S3_Bucket: Sample template showing
 how to create a privately accessible S3 bucket. **WARNING** This template creates
 an S3 bucket. You will be billed for the resources used if you create a stack from
 this template.",
  "Resources": {
    "S3Bucket": {
      "Type": "AWS::S3::Bucket",
      "Properties": {}
    }
  },
  "Outputs": {
    "BucketName": {
      "Value": {
        "Ref": "S3Bucket"
      },
      "Description": "Name of Amazon S3 bucket to hold website content"
    }
  }
}
```

   このテンプレートにより AWS CloudFormation 、 は Service Catalog で使用できる S3 バケットを作成できます。

2. S3_template.json ファイルを AWS CodeCommit リポジトリにアップロードします。

ステップ 2: 製品デプロイ設定ファイルを作成する

1. テキストエディタを開きます。製品の設定ファイルを作成します。設定ファイルは、Service Catalog デプロイパラメータ/設定を定義するために使用されます。パイプラインを作成するときに、このファイルを使用します。

   このサンプルでは、`ProductVersionName` を「devops S3 v2」、`ProductVersionDescription` を `MyProductVersionDescription` としています。テンプレートの変更を新しい製品バージョンにデプロイする場合は、同じ製品の以前の製品バージョンで使用されていない製品バージョン名を入力するだけです。

   `sample_config.json` という名前でファイルを保存します。

   ```
   {
       "SchemaVersion": "1.0",
       "ProductVersionName": "devops S3 v2",
       "ProductVersionDescription": "MyProductVersionDescription",
       "ProductType": "CLOUD_FORMATION_TEMPLATE",
       "Properties": {
           "TemplateFilePath": "/S3_template.json"
       }
   }
   ```

   このファイルにより、パイプラインが実行されるたびに製品バージョン情報が作成されます。

2. `sample_config.json` ファイルを AWS CodeCommit リポジトリにアップロードします。必ずこのファイルはソースリポジトリにアップロードしてください。

ステップ 3: Service Catalog で製品を作成する

1. IT 管理者として、Service Catalog コンソールにサインインし、[製品] ページに移動して、[新しい製品のアップロード] を選択します。
2. [新しい製品のアップロード] ページで、以下の手順を実行します。
   a. [製品名] に、新しい製品に使用する名前を入力します。
   b. [Description (説明)] に製品カタログの説明を入力します。この説明は製品リストに表示されて、ユーザーが正しい製品を選択するのに役立ちます。
   c. [提供元] に IT 部門または管理者の名前を入力します。
   d. [次へ] をクリックします。

3. (オプション) [サポート詳細の入力] に、製品サポートの連絡先情報を入力し、[次へ] を選択します。
4. [バージョンの詳細] に以下の情報を入力します。

   a. [Upload a template file (テンプレートファイルをアップロード)] を選択します。`S3_template.json` ファイルを見つけ、アップロードします。

   b. [バージョンタイトル] に製品バージョンの名前 (devops S3 v2 など) を入力します。

   c. [Description (説明)] に、このバージョンと他のバージョンを区別するための詳細を入力します。

   d. [次へ] をクリックします。
5. [Review (確認)] ページで、情報が正しいことを確認し、[Confirm and upload (確認してアップロード)] を選択します。
6. ブラウザの [製品] ページで、新しい製品の URL をコピーします。これには製品 ID が含まれています。この製品 ID をコピーして保持します。でパイプラインを作成するときに を使用します CodePipeline。

   以下に示しているのは、`my-product` という製品の URL です。製品 ID を抽出するには、等号 (=) とアンパサンド (&) との間の値をコピーします。この例では、製品 ID は `prod-example123456` です。

   ```
   https://<region-URL>/servicecatalog/home?region=<region>#/admin-products?
   productCreated=prod-example123456&createdProductTitle=my-product
   ```

   > **Note**
   >
   > ページから移動する前に、製品の URL をコピーします。このページから移動したら、CLI を使用して製品 ID を取得する必要があります。

   数秒後、製品が [製品] ページに表示されます。製品をリストに表示するには、ブラウザの更新が必要になる場合があります。

ステップ 4: パイプラインを作成する

1. パイプラインに名前を付け、パイプラインのパラメータを選択するには、以下の手順を実行します。

a. にサインイン AWS Management Console し、https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

b. [開始方法] を選択します。[パイプラインの作成] を選択し、パイプラインの名前を入力します。

c. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロールを選択します。 CodePipeline

d. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。

2. ソースステージを追加するには、以下の手順を実行します。

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

b. [リポジトリ名] と [ブランチ名] に、ソースアクションに使用するリポジトリとブランチを入力します。

c. [次へ] をクリックします。

3. [Add build stage (ビルドステージの追加)] で [Skip build stage (ビルドステージのスキップ)] を選択し、もう一度 [スキップ] を選択して警告メッセージを受け入れます。

4. [Add deploy stage (デプロイステージの追加)] で、以下の手順を実行します。

a. [デプロイプロバイダ] で、[AWS Service Catalog] を選択します。

b. [設定ファイルの使用] を選択します。

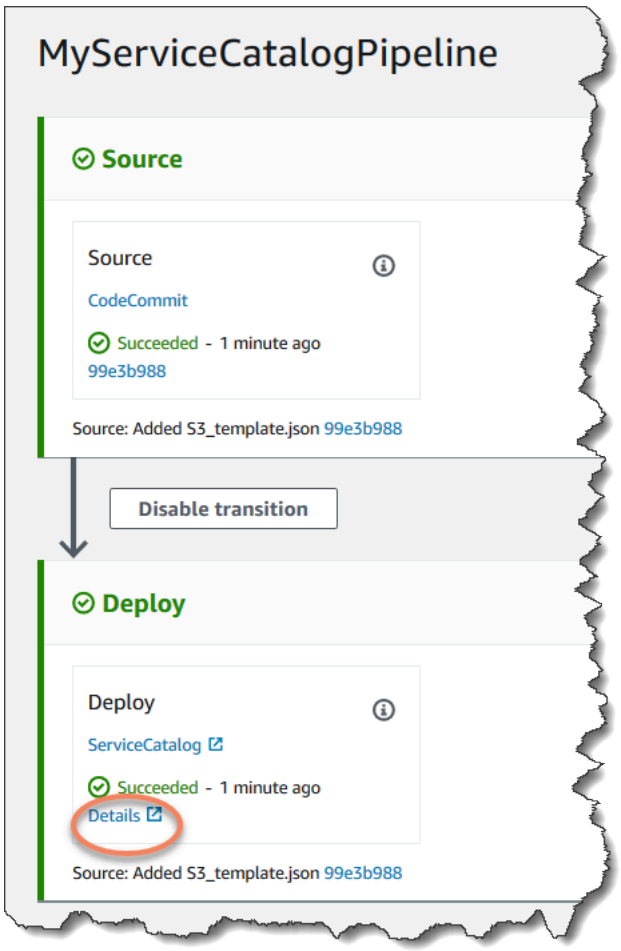
c. [プロダクト ID] に、Service Catalog コンソールからコピーしたプロダクト ID を貼り付けます。

d. [Configuration file path (設定ファイルのパス)] に、リポジトリ内の設定ファイルのファイルパスを入力します。

e. [次へ] をクリックします。

5. [確認] で、パイプライン設定を確認し、[作成] を選択します。

6. パイプラインが正常に実行されたら、デプロイステージで [詳細] を選択して、製品を Service Catalog で開きます。



7. 製品情報で、バージョン名を選択して製品テンプレートを開きます。テンプレートのデプロイを表示します。

ステップ 5: 変更をプッシュして Service Catalog で製品を確認する

1. CodePipeline コンソールでパイプラインを表示し、ソースステージで詳細 を選択します。コンソールでソース AWS CodeCommit リポジトリが開きます。[Edit (編集)] を選択して、ファイルの内容 (説明など) を変更します。

```
"Description": "Name of Amazon S3 bucket to hold and version website content"
```

2. 変更をコミットし、プッシュします。変更をプッシュした後、パイプラインが開始されます。パイプラインの実行が完了したら、デプロイステージで [詳細] を選択して、製品を Service Catalog で開きます。
3. 製品情報で、新しいバージョン名を選択して製品テンプレートを開きます。デプロイされたテンプレートの変更を表示します。

チュートリアル: でパイプラインを作成する AWS CloudFormation

この例では、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 AWS CloudFormation するために を使用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 を通じて保存された変更の到着を検出します。

トピック

例 1: で 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 ソースステージと CodeDeploy デプロイステージを持つ 2 ステージのパイプラインです。

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。で作成したリソースを使用できます CodeDeploy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)。
- パイプラインを作成するためのサンプル AWS CloudFormation テンプレートファイルをダウンロードするには、次のいずれかのリンクを選択します。YAML | JSON

  ファイルを解凍し、ローカルコンピュータに配置します。
- SampleApp_Linux.zip サンプルアプリケーションファイルをダウンロードします。

でパイプラインを作成する AWS CloudFormation

1. ファイルを SampleApp_Linux.zip から解凍し、リポジトリにアップロードします AWS CodeCommit 。解凍したファイルをリポジトリのルートディレクトリにアップロードする必要があります。「ステップ 2: CodeCommit リポジトリにサンプルコードを追加する」の指示に従って、ファイルをリポジトリにプッシュできます。
2. AWS CloudFormation コンソールを開き、スタックの作成 を選択します。[With new resources (standard)] (新しいリソースの使用 (標準)) を選択します。
3. [テンプレートの指定] で、[テンプレートのアップロード] を選択します。[ファイルを選択] を選択し、ローカルコンピュータからテンプレートファイルを選択します。[次へ] をクリックします。
4. [スタック名] に、パイプラインの名前を入力します。サンプルテンプレートで指定されたパラメータが表示されます。以下のパラメータを入力します。

   a. でApplicationName、アプリケーションの名前 CodeDeployを入力します。

   b. でBetaFleet、 CodeDeploy デプロイグループの名前を入力します。

   c. でBranchName、使用するリポジトリブランチを入力します。

   d. でRepositoryName、 CodeCommit ソースリポジトリの名前を入力します。
5. [次へ] をクリックします。以下のページのデフォルト値を受け入れ、[次へ] を選択します。
6. 機能 で、 が IAM リソース を作成する AWS CloudFormation 可能性があることを承認し、スタックの作成 を選択します。
7. スタックの作成が完了したら、イベントリストを表示して、エラーがないか確認します。

トラブルシューティング

でパイプラインを作成する IAM ユーザーには、パイプラインのリソースを作成するための追加のアクセス許可が必要になる AWS CloudFormation 場合があります。がパイプラインに必要な CodeCommit Amazon CloudWatch Events リソースを作成 AWS CloudFormation できるようにするには、ポリシーで次のアクセス許可が必要です。

```
{
     "Effect": "Allow",
     "Action": [
        "events:PutRule",
        "events:PutEvents",
        "events:PutTargets",
        "events:DeleteRule",
```

```
            "events:RemoveTargets",
            "events:DescribeRule"
        ],
        "Resource": "resource_ARN"
}
```

8. にサインイン AWS Management Console し、https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

   > **Note**
   >
   > 作成されたパイプラインを表示するには、 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。では CodePipeline、スタックを作成したリージョンで同じ物理 ID (パイプライン名) を持つ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、ソースバケット、アーティファクトストア、パイプライン、Amazon CloudWatch Events ルール CloudTrailや証跡などの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 ソースステージとデプロイステージを持つ 2 ステージの CodeDeployパイプラインです。

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。

- インスタンスに CodeDeploy エージェントをインストールした Amazon EC2 インスタンスを作成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。で作成した Amazon EC2 と CodeDeploy リソースを使用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)。
- 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 バケットと証跡のサンプルテンプレートをダウンロードします: 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 からサンプルアプリケーションをダウンロードします。

  .zip ファイルをローカルコンピューターに保存します。スタックの作成後、.zip ファイルをアップロードします。

でパイプラインを作成する AWS CloudFormation

1. AWS CloudFormation コンソールを開き、スタックの作成 を選択します。[With new resources (standard)] (新しいリソースの使用 (標準)) を選択します。
2. [テンプレートの選択] で、[テンプレートのアップロード] を選択します。[ファイルの選択] を選択し、ローカルコンピュータからテンプレートファイルを選択します。[次へ] をクリックします。
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. でApplicationName、 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. にBetaFleet、 CodeDeploy デプロイグループの名前を入力します。`DemoFleet` デフォルト名は置き換えることができます。

   c. [SourceObjectKey] に `SampleApp_Linux.zip` と入力します。このファイルは、テンプレートによってバケットとパイプラインが作成された後に、バケットにアップロードします。
4. [次へ] をクリックします。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 AWS CloudFormation できるようにするには、ポリシーで次のアクセス許可が必要です。

```
{
     "Effect": "Allow",
     "Action": [
        "events:PutRule",
        "events:PutEvents",
        "events:PutTargets",
        "events:DeleteRule",
        "events:RemoveTargets",
        "events:DescribeRule"
     ],
     "Resource": "resource_ARN"
}
```

7. で AWS CloudFormation、スタックのリソース 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。では CodePipeline、スタックを作成したリージョンで同じ物理 ID (パイプライン名) を持つパイプラインを表示できます。

名前に `sourcebucket` ラベルが付いた S3 バケットを選択します (`s3-cfn-codepipeline-sourcebucket-y04EXAMPLE.` など)。パイプラインアーティファクトバケットは選択しないでください。

リソースは 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。

> **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://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。
9. AWS CloudTrail リソースを作成するには、以下の手順を実行します。

で AWS CloudTrail リソースを作成する AWS CloudFormation

1. AWS CloudFormation コンソールを開き、スタックの作成 を選択します。
2. [テンプレートの選択] で、[テンプレートを Amazon S3 にアップロード] を選択します。参照 を選択し、ローカルコンピュータから AWS CloudTrail リソースのテンプレートファイルを選択します。[次へ] をクリックします。
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。

   - でSourceObjectKey、サンプルアプリケーションの zip ファイルのデフォルトを受け入れます。
4. [次へ] をクリックします。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、IAM リソース を作成する AWS CloudFormation 可能性があることを確認してから、「 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。

   が Amazon S3 パイプラインに必要な CloudTrail リソースを作成 AWS CloudFormation できるようにするには、ポリシーで次のアクセス許可が必要です。

```
{
     "Effect": "Allow",
     "Action": [
        "cloudtrail:CreateTrail",
        "cloudtrail:DeleteTrail",
```

```
            "cloudtrail:StartLogging",
            "cloudtrail:StopLogging",
            "cloudtrail:PutEventSelectors"
        ],
        "Resource": "resource_ARN"
}
```

7. にサインイン AWS Management Console し、https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

8. ソースバケットで、変更をコミットしてプッシュします。変更検出リソースが変更を受け取り、パイプラインが開始されます。

チュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する

このチュートリアルでは、 AWS CodePipeline コンソールを使用してデプロイアクションを含むパイプラインを作成します。パイプラインが実行されると、テンプレートはスタックを作成し、さらに `outputs` ファイルを作成します。スタックテンプレートによって生成される出力は、 の AWS CloudFormation アクションによって生成される変数です CodePipeline。

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。

トピック

- 前提条件: AWS CloudFormation サービスロールと CodeCommit リポジトリを作成する
- ステップ 1: サンプル AWS CloudFormation テンプレートをダウンロード、編集、アップロードする
- ステップ 2: パイプラインを作成する
- ステップ 3: 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デプロイアクションを追加する CloudFormation
- ステップ 6: CloudFormationデプロイアクションを追加してスタックを削除する

前提条件: AWS CloudFormation サービスロールと CodeCommit リポジトリを作成する

以下のものを用意しておく必要があります。

- CodeCommit リポジトリ。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)。
- 次の例では、テンプレートから Amazon DocumentDB スタックを作成します。 AWS Identity and Access Management (IAM) を使用して、Amazon DocumentDB の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。

1. リージョンのサンプルテンプレートページに移動します。例えば、us-west-2 のページは https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/sample-templates-services-us-west-2.html にあります。Amazon DocumentDB で、Amazon DocumentDB クラスターのテンプレートをダウンロードします。ファイル名は `documentdb_full_stack.yaml` です。
2. `documentdb_full_stack.yaml` ファイルを解凍し、テキストエディタで開きます。以下の変更を加えます。

   a. 次の例では、テンプレートの `Parameters` セクションに次の `Purpose:` パラメータを追加します。

```
  Purpose:
```

```
    Type: String
    Default: testing
    AllowedValues:
      - testing
      - production
    Description: The purpose of this instance.
```

b. 次の例では、テンプレートの Outputs: セクションに次の StackName 出力を追加します。

```
  StackName:
    Value: !Ref AWS::StackName
```

3. テンプレートファイルを AWS CodeCommit リポジトリにアップロードします。解凍および編集したテンプレートファイルを、リポジトリのルートディレクトリにアップロードする必要があります。

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

   a. CodeCommit コンソールを開き、リポジトリリストから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」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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: パイプラインの設定を選択する] の [パイプライン名] に「**MyCFNDeployPipeline**」と入力します。
4. 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。
5. [Service role (サービスロール)] で、次のいずれかの操作を行います。
   - IAM でサービスロールを作成することを許可する新しいサービスロールを選択します。CodePipeline
   - [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。
6. アーティファクトストア:
   a. パイプライン用に選択したリージョンのパイプラインに、デフォルトとして指定された Amazon S3 アーティファクトバケットなどのデフォルトのアーティファクトストアを使用するには、デフォルトの場所 を選択します。
   b. Amazon S3 アーティファクトバケットなどのアーティファクトストアがパイプラインと同じリージョンに既に存在する場合は、カスタムの場所 を選択します。

> **Note**
>
> これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。パイプラインを作成または編集するときは、パイプラインリージョンにアーティファクトバケットと、アクションを実行している AWS リージョンごとに 1 つのアーティファクトバケットが必要です。

詳細については、「入力および出力アーティファクト」および「CodePipeline パイプライン構造リファレンス」を参照してください。

[次へ] を選択します。

7. [ステップ 2: 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、 で作成した CodeCommit リポジトリの名前を選択しますステップ 1: CodeCommit リポジトリを作成する。

   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 ルールが表示されます。

   [次へ] をクリックします。

8. [Step 3: Add build stage] (ステップ 3: ビルドステージを追加する) 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。

   [次へ] をクリックします。

9. ステップ 4: 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] で、[CloudFormation] を選択します。

   b. [アクションモード] で、[スタックを作成または更新する] をクリックします。

   c. [スタック名] に、スタックの名前を入力します。これは、テンプレートが作成するスタックの名前です。

   d. [出力ファイル名] に、出力ファイルの名前 (**outputs** など) を入力します。これは、スタックの作成後にアクションによって作成されるファイルの名前です。

   e. [Advanced] を展開します。[パラメータの上書き] で、テンプレートの上書きをキーと値のペアとして入力します。例えば、このテンプレートには、次の上書きが必要です。

```
{
"DBClusterName": "MyDBCluster",
"DBInstanceName": "MyDBInstance",
```

```
"MasterUser": "UserName",
"MasterPassword": "Password",
"DBInstanceClass": "db.r4.large",
"Purpose": "testing"}
```

上書きを入力しない場合、テンプレートはスタックをデフォルト値で作成します。

f. [次へ] をクリックします。

g. [パイプラインの作成] を選択します。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 が変更セットを作成 AWS CloudFormation 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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

2. [編集] モードで、パイプラインを編集するか、引き続きパイプラインを表示するかを選択します。

3. デプロイステージを編集することを選択します。

4. 前のアクションで作成したスタックに対する変更セットを作成するデプロイアクションを追加します。このアクションは、ステージ内の既存のアクションの後に追加します。

   a. [アクション名] に、「Change_Set」と入力します。[アクションプロバイダー] で、[AWS CloudFormation] を選択します。

   b. 入力アーティファクト で、 を選択しますSourceArtifact。

   c. [Action mode] (アクションモード) で [Create or replace a change set] (変更セットの作成または置換) を選択します。

   d. [スタック名] に、次のように変数の構文を入力します。これは、変更セットを作成する対象のスタックの名前です。アクションには、デフォルトの名前空間 `DeployVariables` が割り当てられます。

```
#{DeployVariables.StackName}
```

e. [変更セット名] に、変更セットの名前を入力します。

```
my-changeset
```

f. [パラメータの上書き] で、Purpose パラメータを testing から production に変更します。

```
{
"DBClusterName": "MyDBCluster",
"DBInstanceName": "MyDBInstance",
"MasterUser": "UserName",
"MasterPassword": "Password",
"DBInstanceClass": "db.r4.large",
"Purpose": "production"}
```

g. [完了] をクリックしてアクションを保存します。

ステップ 4: 手動承認アクションを追加する

パイプラインで手動承認アクションを作成します。

1. [編集] モードで、パイプラインを編集するか、引き続きパイプラインを表示するかを選択します。
2. デプロイステージを編集することを選択します。
3. 変更セットを作成するデプロイアクションの後に、手動承認アクションを追加します。このアクションにより、パイプラインが変更セットを実行する 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デプロイアクションを追加する CloudFormation

手動承認アクションの後に が変更セットを実行 AWS CloudFormation 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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

2. [編集] モードで、パイプラインを編集するか、引き続きパイプラインを表示するかを選択します。

3. デプロイステージを編集することを選択します。

4. 前の手動アクションで承認した変更セットを実行するデプロイアクションを追加します。

   a. [アクション名] に、「Execute_Change_Set」と入力します。[アクションプロバイダー] で、[AWS CloudFormation] を選択します。

   b. 入力アーティファクト で、 を選択しますSourceArtifact。

   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: CloudFormationデプロイアクションを追加してスタックを削除する

が出力ファイルの 変数からスタック名を取得 AWS CloudFormation してスタックを削除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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。

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、 を選択しますDeleteStack。[デプロイプロバイダ] 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: を使用した Amazon ECS 標準デプロイ CodePipeline

このチュートリアルは、 を使用して Amazon ECS で完全で end-to-end 継続的なデプロイ (CD) パイプラインを作成するのに役立ちます CodePipeline。

> **Note**
>
> このチュートリアルでは、 の Amazon ECS 標準デプロイアクションについて説明します CodePipeline。Amazon ECS を使用して で CodeDeploy ブルー/グリーン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 CodePipeline、「」を参照してくださいチュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する。

前提条件

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

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

- Dockerfile とアプリケーションソースを含む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 を使用します CodeCommit)。詳細については、「 ユーザーガイド」の CodeCommit「リポジトリの作成AWS CodeCommit 」を参照してください。
- Dockerfile およびアプリケーションソースから作成したイメージを含む Docker イメージリポジトリ (このチュートリアルでは Amazon ECR を使用します)。詳細については、Amazon Elastic Container Registry ユーザーガイド の「リポジトリの作成」と「イメージをプッシュする」を参照してください。
- イメージリポジトリでホストされた Docker イメージを参照する Amazon ECS タスク定義。詳細については、Amazon Elastic Container Service デベロッパーガイド の「タスク定義の作成」を参照してください。

> **Important**
>
> 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 CodePipeline を作成します。Amazon ECS サービスを更新せずにタスク定義の新しいリビジョンを作成した場合、デプロイアクションはそれらのリビジョンを無視します。

このチュートリアルで使用するタスク定義の例を以下に示します。`name` と `family` に使用する値は、ビルド仕様ファイルのために次のステップで使用します。

```
{
  "ipcMode": null,
  "executionRoleArn": "role_ARN",
  "containerDefinitions": [
    {
      "dnsSearchDomains": null,
      "environmentFiles": null,
      "logConfiguration": {
        "logDriver": "awslogs",
        "secretOptions": null,
        "options": {
          "awslogs-group": "/ecs/hello-world",
```

```
        "awslogs-region": "us-west-2",
        "awslogs-stream-prefix": "ecs"
      }
    },
    "entryPoint": null,
    "portMappings": [
      {
        "hostPort": 80,
        "protocol": "tcp",
        "containerPort": 80
      }
    ],
    "command": null,
    "linuxParameters": null,
    "cpu": 0,
    "environment": [],
    "resourceRequirements": null,
    "ulimits": null,
    "dnsServers": null,
    "mountPoints": [],
    "workingDirectory": null,
    "secrets": null,
    "dockerSecurityOptions": null,
    "memory": null,
    "memoryReservation": 128,
    "volumesFrom": [],
    "stopTimeout": null,
    "image": "image_name",
    "startTimeout": null,
    "firelensConfiguration": null,
    "dependsOn": null,
    "disableNetworking": null,
    "interactive": null,
    "healthCheck": null,
    "essential": true,
    "links": null,
    "hostname": null,
    "extraHosts": null,
    "pseudoTerminal": null,
    "user": null,
    "readonlyRootFilesystem": null,
    "dockerLabels": null,
    "systemControls": null,
    "privileged": null,
```

```
      "name": "hello-world"
    }
  ],
  "placementConstraints": [],
  "memory": "2048",
  "taskRoleArn": null,
  "compatibilities": [
    "EC2",
    "FARGATE"
  ],
  "taskDefinitionArn": "ARN",
  "family": "hello-world",
  "requiresAttributes": [],
  "pidMode": null,
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "1024",
  "revision": 1,
  "status": "ACTIVE",
  "inferenceAccelerators": null,
  "proxyConfiguration": null,
  "volumes": []
}
```

- 前に説明したタスク定義を使用するサービスを実行する Amazon ECS クラスター。詳細については、Amazon Simple Queue Serviceデベロッパーガイド の クラスターの作成 と サービスの作成 を参照してください。

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

ステップ 1: ビルド仕様ファイルをソースリポジトリに追加する

このチュートリアルでは CodeBuild 、 を使用して Docker イメージを構築し、そのイメージを Amazon ECR にプッシュします。ソースコードリポジトリに `buildspec.yml` ファイルを追加して、その方法を説明します CodeBuild。ビルド仕様の以下の例では、次のように動作します。

- プレビルドステージ:
  - Amazon ECR にログインします。

  - リポジトリ URI を ECR イメージに設定して、ソースの Git コミット ID の最初の 7 文字を使用するイメージタグを追加します。
- ビルドステージ
  - Docker イメージを作成し、イメージに `latest` と Git コミット ID の両方をタグ付けします。
- ポストビルドステージ:
  - 両方のタグを持った ECR リポジトリにイメージをプッシュします。
  - Amazon ECS サービスのコンテナ名およびイメージとタグがあるビルドのルートに `imagedefinitions.json` という名前のファイルを作成します。CD パイプラインのデプロイステージでこの情報を使用してサービスのタスク定義の新しいリビジョンを作成し、新しいタスク定義を使用してサービスを更新します。`imagedefinitions.json` ファイルは ECS ジョブワーカーに必須です。

このサンプルテキストを貼り付けて、`buildspec.yml` ファイルを使用して、イメージとタスク定義の値を置き換えます。このテキストでは、例としてアカウント ID 111122223333 を使用しています。

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
      - REPOSITORY_URI=012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
```

```
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"hello-world","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG >
 imagedefinitions.json
artifacts:
    files: imagedefinitions.json
```

このチュートリアルで使用する Amazon ECS サービスで、前提条件 で提供されているサンプルタスクの定義に合わせてビルド仕様が書き込まれています。REPOSITORY_URI 値は image リポジトリ (イメージタグなし) に対応し、ファイルの末尾近くの *hello-world* 値はサービスのタスク定義のコンテナ名に対応します。

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

1. テキストエディタを開き、上記のビルド仕様をコピーして新しいファイルに貼り付けます。
2. REPOSITORY_URI の値 (*012345678910.dkr.ecr.us-west-2.amazonaws.com/hello-world*) を、Docker イメージの自分の Amazon ECR リポジトリ URI (イメージタグなし) に置き換えます。*hello-world* を、Docker イメージを参照するサービスのタスク定義のコンテナ名に置き換えます。
3. ソースリポジトリに buildspec.yml ファイルをコミットし、プッシュします。

   a. ファイルを追加します。

   ```
   git add .
   ```

   b. 変更をコミットします。

   ```
   git commit -m "Adding build specification."
   ```

   c. コミットをプッシュします。

   ```
   git push
   ```

ステップ 2: 継続的デプロイパイプラインを作成する

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

パイプラインを作成するには

1. https://console.aws.amazon.com/codepipeline/ 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。

   を初めて使用する場合は CodePipeline、ようこそ の代わりに概要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: 名前 ページで、パイプライン名 にパイプラインの名前を入力します。このチュートリアルでは、パイプライン名は hello-world です。
4. 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。[Next] (次へ) を選択します。
5. [ステップ 2: ソースステージの追加] ページの [ソースプロバイダー] で、[AWS CodeCommit] を選択します。

   a. リポジトリ名 で、パイプラインの送信 CodeCommit元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。
6. [ステップ 3: ビルドステージの追加] ページの [ビルドプロバイダー] で [AWS CodeBuild] を選択し、[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 ログ 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. 続行 を選択します CodePipeline。

j. [次へ] をクリックします。

> **Note**
>
> ウィザードは、codebuild--service-role ***build-project-name***と呼ばれるビルドプロジェクトの CodeBuild サービスロールを作成します。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。

7. [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] の [デプロイプロバイダ] に、[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。

8. [ステップ 5: レビュー] ページで、パイプラインの設定を確認し、[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。

> **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すべてのコマンドを実行するアクセス許可がないため、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: ロールに CodeBuild Amazon ECR アクセス許可を追加する

CodePipeline ウィザードは、codebuild--service-role と呼ばれる CodeBuild ビルドプロジェクトの IAM ロールを作成しました。***build-project-name***このチュートリアルでは、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。

Amazon ECR アクセス許可を CodeBuild ロールに追加するには

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] を選択します。
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 ポリシーの左側にあるボックスを選択し、ポリシーのアタッチ 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 ネイティブの 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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
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 ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する

このチュートリアルでは、Docker イメージをサポートするブルー/グリーン 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。

> **Note**
>
> このチュートリアルでは、 の Amazon ECS から CodeDeploy Blue/Green へのデプロイアクションについて説明します CodePipeline。で Amazon ECS 標準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 CodePipeline、「」を参照してくださいチュートリアル: を使用した Amazon ECS 標準デプロイ CodePipeline。

完成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、 を使用してトラフィックを Amazon ECS クラスター CodeDeploy にルーティングおよびデプロイし、ロードバランサー CodeDeploy はリスナーを使用して、 AppSpec ファイルで指定された更新されたコンテナのポートにトラフィックを再ルーティングします。ロードバランサー、製造リスナー、ターゲットグループ、Amazon ECS アプリケーションが青/緑のデプロイでどのように使用されるかについては、チュートリアル：Amazon ECS サービスのデプロイ を参照してください。

パイプラインは、Amazon ECS タスク定義が保存されている CodeCommitなどのソースロケーションを使用するようにも設定されています。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 が CodeDeploy本番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 CodeDeploy AppSpec ファイル。デプロイライフサイクルイベントフック中に実行できるオプションのネットワーク設定と Lambda 関数も指定できます。

> **Note**
>
> Amazon ECR イメージリポジトリへの変更をコミットすると、パイプラインソースアクションはそのコミット用に `imageDetail.json` ファイルを作成します。`imageDetail.json` ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

パイプラインを作成または編集し、デプロイステージのソースアーティファクトを更新または指定するときは、使用する最新の名前とバージョンのソースアーティファクトを必ず指してください。パイプラインを設定した後、イメージまたはタスク定義を変更したら、リポジトリ内のソースアーティファクトファイルを更新してから、パイプラインのデプロイステージを編集する必要があります。

トピック

前提条件

以下のリソースがすでに作成されている必要があります。

- CodeCommit リポジトリ。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)。
- このチュートリアルに示すように、Amazon EC2 Linux インスタンスを起動し、Docker をインストールしてイメージを作成します。使用するイメージがすでにある場合、この前提条件は省略できます。

ステップ 1: イメージを作成して Amazon ECR リポジトリにプッシュする

このセクションでは、Docker を使用してイメージを作成し、 を使用して Amazon ECR リポジトリ AWS CLI を作成し、そのイメージをリポジトリにプッシュします。

> **Note**
>
> 使用するイメージがすでにある場合、このスキップは省略できます。

イメージを作成するには

1. Docker がインストールされている Linux インスタンスにサインインします。

nginx のイメージをプルダウンから選択します。このコマンドは nginx:latest イメージを返します。

```
docker pull nginx
```

2. docker images を実行します。リストにイメージが表示されます.。

```
docker images
```

Amazon ECR リポジトリを作成してイメージをプッシュするには

1. イメージを保存する Amazon ECR リポジトリを作成します。出力の repositoryUri を書き留めます。

```
aws ecr create-repository --repository-name nginx
```

出力:

```
{
    "repository": {
        "registryId": "aws_account_id",
        "repositoryName": "nginx",
        "repositoryArn": "arn:aws:ecr:us-east-1:aws_account_id:repository/nginx",
        "createdAt": 1505337806.0,
        "repositoryUri": "aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx"
    }
}
```

2. 前のステップの repositoryUri でイメージをタグ付けします。

```
docker tag nginx:latest aws_account_id.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

3. この例に示すように、us-west-2 リージョンと 111122223333 アカウント ID を指定して aws ecr get-login-password コマンドを実行します。

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com/nginx
```

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

```
docker push 111122223333.dkr.ecr.us-east-1.amazonaws.com/nginx:latest
```

ステップ 2: タスク定義と 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、 用の AppSpec ファイルを作成し CodeDeploy 、Git クライアントを使用してファイルを CodeCommit リポジトリにプッシュします。

イメージのタスク定義を作成するには

1. 次の内容で、taskdef.json という名前のファイルを作成します。image に、イメージの名前 (例: nginx) を入力します。この値は、パイプラインの実行時に更新されます。

> **Note**
>
> タスク定義に指定されている実行ロールに AmazonECSTaskExecutionRolePolicy が含まれていることを確認します。詳細については、Amazon ECS デベロッパーガイドの Amazon ECS タスク実行 IAM ロール を参照してください。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "nginx",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
```

```
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

2. タスク定義を taskdef.json ファイルに登録します。

```
aws ecs register-task-definition --cli-input-json file://taskdef.json
```

3. タスク定義が登録されたら、イメージ名を削除して、イメージフィールド名に <IMAGE1_NAME> プレースホルダーが含まれるようにファイルを編集します。

```
{
    "executionRoleArn": "arn:aws:iam::account_ID:role/ecsTaskExecutionRole",
    "containerDefinitions": [
        {
            "name": "sample-website",
            "image": "<IMAGE1_NAME>",
            "essential": true,
            "portMappings": [
                {
                    "hostPort": 80,
                    "protocol": "tcp",
                    "containerPort": 80
                }
            ]
        }
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "cpu": "256",
    "memory": "512",
    "family": "ecs-demo"
}
```

AppSpec ファイルを作成するには

- AppSpec ファイルは CodeDeploy デプロイに使用されます。オプションのフィールドが含まれるファイルでは、この形式を使用します。

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "task-definition-ARN"
        LoadBalancerInfo:
          ContainerName: "container-name"
          ContainerPort: container-port-number
# Optional properties
        PlatformVersion: "LATEST"
        NetworkConfiguration:
            AwsvpcConfiguration:
              Subnets: ["subnet-name-1", "subnet-name-2"]
              SecurityGroups: ["security-group"]
              AssignPublicIp: "ENABLED"
Hooks:
- BeforeInstall: "BeforeInstallHookFunctionName"
- AfterInstall: "AfterInstallHookFunctionName"
- AfterAllowTestTraffic: "AfterAllowTestTrafficHookFunctionName"
- BeforeAllowTraffic: "BeforeAllowTrafficHookFunctionName"
- AfterAllowTraffic: "AfterAllowTrafficHookFunctionName"
```

例を含む AppSpec ファイルの詳細については、「ファイルCodeDeploy AppSpec リファレンス」を参照してください。

次の内容で、appspec.yaml という名前のファイルを作成します。TaskDefinition の <TASK_DEFINITION> プレースホルダーテキストは変更しないでください。この値は、パイプラインの実行時に更新されます。

```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: <TASK_DEFINITION>
        LoadBalancerInfo:
```

```
          ContainerName: "sample-website"
          ContainerPort: 80
```

CodeCommit リポジトリにファイルをプッシュするには

1. ファイルを CodeCommit リポジトリにプッシュまたはアップロードします。これらのファイルは、 でデプロイアクション用にパイプラインの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 CodePipeline。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. 次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リポジトリを選択します。

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

iii. [Choose file] を選択し、ファイルを参照します。ユーザー名とメールアドレスを入力して、変更をコミットします。[Commit changes] (変更のコミット) を選択します。

iv. アップロードするファイルごとにこのステップを繰り返します。

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

このセクションでは、Amazon EC2 Application Load Balancer を作成します。後で Amazon ECS サービスを作成するときに、ロードバランサーで作成したサブネット名とターゲットグループ値を使用します。Application Load Balancer または Network Load Balancer を作成できます。ロードバランサーでは、2 つのパブリックサブネットを別々のアベイラビリティーゾーンに持つ VPC を使用する必要があります。以下のステップでは、デフォルト VPC を確認して、ロードバランサーを作成してから、ロードバランサーの 2 つのターゲットグループを作成します。詳細については、「Network Load Balancer のターゲットグループ」を参照してください。

デフォルト VPC とパブリックサブネットを確認するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/vpc/ で Amazon VPC コンソールを開きます。
2. 使用するデフォルト VPC を確認します。ナビゲーションペインで、[Your VPCs (お使いの VPC)] を選択します。デフォルトのVPC 列に はい と表示されている VPC に注意してください。これがデフォルト VPC です。選択するデフォルトのサブネットが含まれています。
3. [サブネット] を選択します。デフォルトのサブネット 列に はい と表示されている 2 つのサブネットを選択します。

   > **Note**
   >
   > サブネット ID を書き留めます。これらは、このチュートリアルで後ほど必要になります。

4. サブネットを選択後、[説明] タブを選択します。使用するサブネットが、異なるアベイラビリティーゾーンにあることを確認します。
5. サブネットを選択後、[Route Table] タブを選択します。使用する各サブネットがパブリックサブネットであることを確認するには、ルートテーブルにゲートウェイ行が含まれていることを確認します。

Amazon EC2 Application Load Balancer を作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/ec2/ で Amazon EC2 コンソールを開きます。
2. ナビゲーションペインで、[ロードバランサー] を選択します。
3. [Create Load Balancer] を選択します。
4. [Application Load Balancer] を選択し、[Create] を選択します。
5. [Name] に、ロードバランサーの名前を入力します。
6. [Scheme] で、[インターネット向け] を選択します。
7. [IP address type] で、[ipv4] を選択します。
8. ロードバランサー用に 2 つのリスナーポートを設定するには:
   a. [Load Balancer Protocol (ロードバランサーのプロトコル)] で、[HTTP] を選択します。[Load Balancer Port (ロードバランサーポート)] に [**80**] を入力します。
   b. [リスナーの追加] を選択します。
   c. 2 番目のリスナーの [Load Balancer Protocol] で、[HTTP] を選択します。[Load Balancer Port (ロードバランサーポート)] に [**8080**] を入力します。
9. [アベイラビリティーゾーン] の [VPC] で、デフォルトの VPC を選択します。次に、使用する 2 つのデフォルトサブネットを選択します。
10. [Next: Configure Security Settings] を選択します。
11. [Next: Configure Security Groups] を選択します。
12. [Select an existing security group] を選択し、セキュリティグループ ID を書き留めます。
13. [Next: Configure Routing] を選択します。
14. [Target group] で、[New target group] を選択し、最初のターゲットグループを設定します。
    a. [Name] に、ターゲットグループの名前 (例: **target-group-1**) を入力します。
    b. [Target type] で、[IP] を選択します。
    c. [Protocol] で、[HTTP] を選択します。[Port] に「**80**」と入力します。
    d. [Next: Register Targets] を選択します。
15. [Next: Review]、[Create] の順に選択します。

ロードバランサーの 2 番目のターゲットグループを作成するには

1. ロードバランサーがプロビジョニングされたら、Amazon EC2 コンソールを開きます。ナビゲーションペインで、[ターゲットグループ] を選択します。
2. [ターゲットグループの作成] を選択します。
3. [Name] に、ターゲットグループの名前 (例: **`target-group-2`**) を入力します。
4. [Target type] で、[IP] を選択します。
5. [Protocol] で、[HTTP] を選択します。[Port] に「**`8080`**」と入力します。
6. [VPC] で、デフォルトの VPC を選択します。
7. [作成] を選択します。

> **Note**
>
> デプロイを実行するには、ロードバランサー用に 2 つのターゲットグループを作成する必要があります。最初のターゲットグループの ARN を書き留める必要があります。この ARN は、次のステップの `create-service` JSON ファイルで使用されます。

2 番目のターゲットグループを含めるようにロードバランサーを更新するには

1. Amazon EC2 コンソールを開きます。ナビゲーションペインで、[ロードバランサー] を選択します。
2. ロードバランサーを選択後、[Listeners] を選択します。ポート 8080 のリスナーを選択後、[編集] を選択します。
3. [Forward to] の横の鉛筆アイコンを選択します。2 番目のターゲットグループを選択してから、チェックマークを選択します。[更新] を選択して、更新を保存します。

ステップ 4: Amazon ECS クラスターとサービスを作成する

このセクションでは、 がデプロイ中に CodeDeploy 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 (EC2 インスタンスではなく Amazon ECS クラスター) 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 [クラスター] を選択します。
3. [クラスターを作成] を選択します。
4. AWS Fargate を使用する ネットワークのみ クラスターテンプレートを選択後、次のステップを選択します。
5. [Configure cluster (クラスターの設定)] ページで、クラスター名を入力します。リソースに任意のタグを追加することができます。[作成] を選択します。

Amazon ECS サービスを作成する

を使用して AWS CLI 、Amazon ECS でサービスを作成します。

1. JSON ファイルを作成し、`create-service.json` と名付けます。次の内容を JSON ファイルに貼り付けます。

   `taskDefinition`フィールドでは、Amazon ECS にタスク定義を登録するときに、ファミリーを指定します。これは、リビジョン番号で指定された、複数バージョンのタスク定義の名前に似ています。この例では、ファイル内のファミリーとリビジョン番号に「`ecs-demo:1`」を使用します。ステップ 3: Application Load Balancer とターゲットグループを作成する でロードバランサーを使用して作成したサブネット名、セキュリティグループ、ターゲットグループの値を使用します。

   > **Note**
   >
   > ターゲットグループ ARN をこのファイルに含める必要があります。Amazon EC2 コンソールを開き、ナビゲーションペインの ロードバランシング で ターゲットグループ を

選択します。最初のターゲットグループを選択します。[説明] タブから ARN をコピーします。

```
{
    "taskDefinition": "family:revision-number",
    "cluster": "my-cluster",
    "loadBalancers": [
        {
            "targetGroupArn": "target-group-arn",
            "containerName": "sample-website",
            "containerPort": 80
        }
    ],
    "desiredCount": 1,
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                "subnet-1",
                "subnet-2"
            ],
            "securityGroups": [
                "security-group"
            ],
            "assignPublicIp": "ENABLED"
        }
    }
}
```

2. JSON ファイルを指定して、create-service コマンドを実行します。

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

この例では、my-service という名前のサービスが作成されます。

> **Note**
>
> このコマンド例では、my-service という名前のサービスを作成します。この名前のサービスがすでにある場合、このコマンドはエラーを返します。

```
aws ecs create-service --service-name my-service --cli-input-json file://create-service.json
```

出力はサービスの説明フィールドが返ります。

3. describe-services コマンドを実行して、サービスが作成されたことを確認します。

```
aws ecs describe-services --cluster cluster-name --services service-name
```

ステップ 5: アプリケーションとデプロイグループを作成する CodeDeploy (ECS コンピューティングプラットフォーム)

Amazon ECS コンピューティングプラットフォームの 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正しいデプロイグループ、ターゲットグループ、リスナー、トラフィックの再ルーティング動作を参照するために使用されます。

CodeDeploy アプリケーションを作成するには

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。

CodeDeploy デプロイグループを作成するには

1. アプリケーションページの [デプロイグループ] タブで [デプロイグループの作成] を選択します。

2. [デプロイグループ名] に、デプロイグループを表す名前を入力します。
3. サービスロール で、Amazon ECS へのアクセスを許可する CodeDeploy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. [ロールを作成] 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、 を選択しますCodeDeploy。ユースケースの選択で、 CodeDeploy - ECS を選択します。[次のステップ: 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。
6. [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。
7. [Target group 1 name] および [Target group 2 name] から、デプロイ時にトラフィックをルーティングするターゲットグループを選択します。これらが、ロードバランサー用に作成したターゲットグループであることを確認します。
8. すぐにトラフィックを再ルーティング を選択して、デプロイが成功してから更新された Amazon ECS タスクにトラフィックを再ルーティングするまでの時間を決定します。
9. デプロイグループの作成 を選択します。

ステップ 6: パイプラインを作成する

このセクションでは、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含むデプロイステージ。

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: パイプラインの設定を選択する] の [パイプライン名] に「**MyImagePipeline**」と入力します。
4. 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。
5. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロールを選択します。 CodePipeline
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. [Step 2: Add source stage (ステップ 2: ソースステージの追加)] の [ソースプロバイダ] で、[AWS CodeCommit] を選択します。リポジトリ名 で、 で作成したリポジトリの名前 CodeCommitを選択しますステップ 1: CodeCommit リポジトリを作成する。[Branch name] で、最新のコード更新を含むブランチの名前を選択します。

   [次へ] をクリックします。
8. [Step 3: Add build stage] (ステップ 3: ビルドステージを追加する) 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。[次へ] をクリックします。
9. ステップ 4: 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、 を選択しますSourceArtifact。フィールドに [**taskdef.json**] と入力します。

c. AWS CodeDeploy AppSpec ファイル で、 を選択しますSourceArtifact。フィールドに [**appspec.yaml**] と入力します。

> **Note**
>
> この時点では、[Dynamically update task definition image] に情報は入力しないでください。

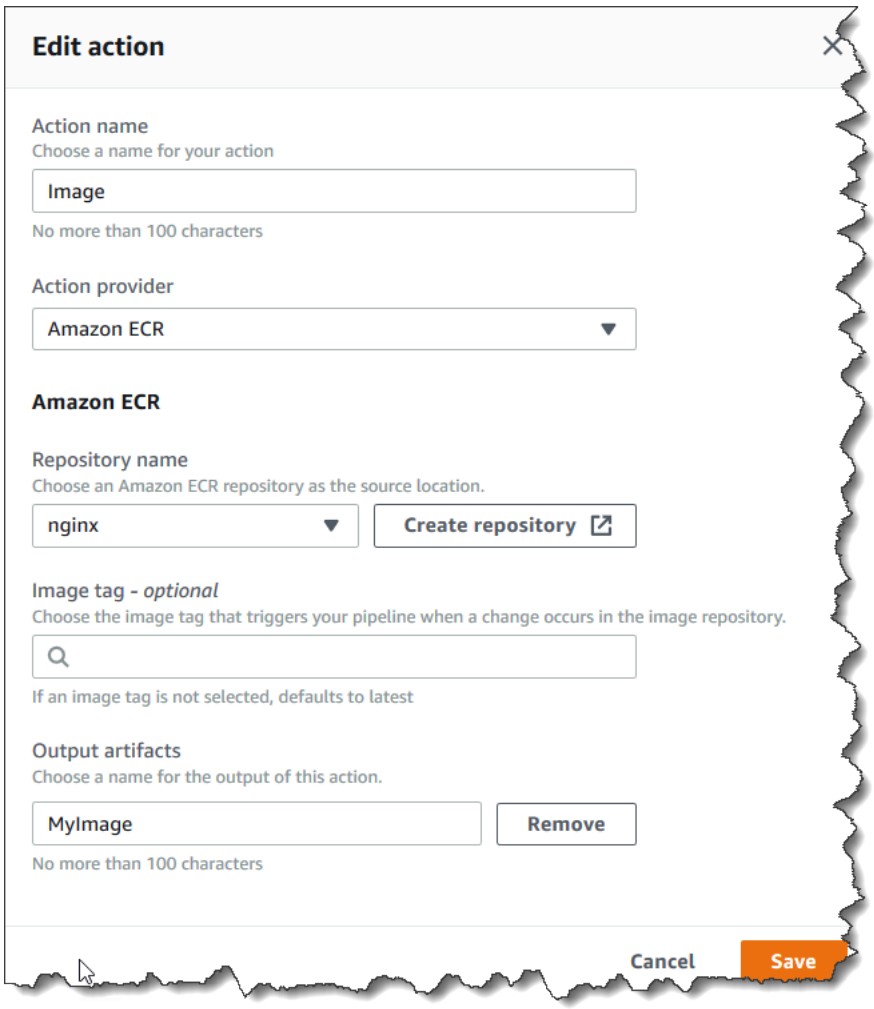
d. [次へ] をクリックします。

10. [ステップ 5: 確認] で情報を確認し、[パイプラインの作成] を選択します。

Amazon ECR ソースアクションをパイプラインに追加するには

パイプラインを表示し、Amazon ECR ソースアクションをパイプラインに追加します。

1. パイプラインを選択します。左上の [Edit] (編集) を選択します。
2. ソースステージで、[ステージを編集] を選択します。
3. CodeCommit ソースアクションの横にある + アクションを追加 を選択して、並列アクションを追加します。
4. [アクション名] に、名前を入力します (例えば、**Image**)。
5. [アクションプロバイダ] で [Amazon ECR] を選択します。



6. リポジトリ名 で、Amazon ECR リポジトリの名前を選択します。
7. [Image tag] で、イメージの名前とバージョンを指定します (最新でない場合)。
8. 出力アーティファクトで、次のステージで使用するイメージ名およびリポジトリ URI 情報が含まれている出力アーティファクトのデフォルト(例: `MyImage`)を選択します。
9. アクション画面で、[Save] を選択します。ステージ画面で、[Done] を選択します。パイプラインで、[Save] を選択します。Amazon ECR ソースアクション用に作成される Amazon CloudWatch Events ルールを示すメッセージが表示されます。

ソースアーティファクトをデプロイアクションに関連付けるには

1. デプロイステージで 編集 を選択後、アイコンを選択して、Amazon ECS (Blue/Green) アクションを編集します。
2. ペインの下部までスクロールします。[入力アーティファクト] で [Add] を選択します。新しい Amazon ECR リポジトリ (例: `MyImage` など) からソースアーティファクトを追加します。

3. タスク定義 で を選択しSourceArtifact、 **taskdef.json**が入力されていることを確認します。
4. AWS CodeDeploy AppSpec ファイル で を選択しSourceArtifact、 **appspec.yaml**が入力されていることを確認します。
5. 動的に更新するタスク定義イメージ で、Input Artifact with Image URI で を選択しMyImage、 `taskdef.json` ファイルで使用されるプレースホルダーテキストを入力します **IMAGE1_NAME**。[保存] を選択します。
6. AWS CodePipeline ペインで、パイプライン変更の保存 を選択し、変更の保存 を選択します。更新されたパイプラインを表示します。

   このサンプルパイプラインが作成されると、コンソールエントリのアクション設定が以下のようにパイプライン構造に表示されます。

```
"configuration": {
  "AppSpecTemplateArtifact": "SourceArtifact",
  "AppSpecTemplatePath": "appspec.yaml",
  "TaskDefinitionTemplateArtifact": "SourceArtifact",
  "TaskDefinitionTemplatePath": "taskdef.json",
  "ApplicationName": "codedeployapp",
  "DeploymentGroupName": "codedeploydeplgroup",
  "Image1ArtifactName": "MyImage",
  "Image1ContainerName": "IMAGE1_NAME"
},
```

7. 変更を送信してパイプラインのビルドを開始するには、[変更をリリース]、[リリース] の順に選択します。
8. デプロイアクションを選択して表示します。 CodeDeploy また、トラフィックシフトの進行状況を確認します。

> **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。デフォルトでは、 はデプロイが成功してから 1 時間後に CodeDeploy 待機してから、元のタスクセットを終了します。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。

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

Lambda 関数としてカスタムスキルを作成するには、「カスタムスキルを AWS Lambda 関数としてホストする」を参照してください。Lambda ソースファイルを使用するパイプラインと、スキルの変更を Lambda にデプロイする CodeBuild プロジェクトを作成することもできます。たとえば、新しいスキルおよび関連する Lambda 関数を作成するには、 AWS CodeStar プロジェクトを作成します。「AWS CodeStarで Alexa スキルプロジェクトを作成する」を参照してください。このオプションの場合、パイプラインには のデプロイステージに CodeBuildアクションと アクションを含む 3 番目のビルドステージが含まれます AWS CloudFormation。

前提条件

以下のものを用意しておく必要があります。

- CodeCommit リポジトリ。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) 。
- Amazon 開発者アカウント。これは Alexa スキルを所有するアカウントです。Alexa Skills Kit でアカウントを無料で作成できます。

- Alexa スキル。「カスタムスキルサンプルコードを取得する」チュートリアルを使用してサンプルスキルを作成できます。
- ASK CLI をインストールし、`ask init` 認証情報で AWS を使用して設定します。「ASK CLI のインストールと初期化」を参照してください。

ステップ 1: Alexa デベロッパーサービス LWA セキュリティプロファイルを作成する

このセクションでは、Login with Amazon (LWA) で使用するセキュリティプロファイルを作成します。プロファイルがすでにある場合、このステップは省略できます。

- のステップを使用してgenerate-lwa-tokens、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、Alexa スキルのソースファイルを作成し、パイプラインによってソースステージに使用されるリポジトリにプッシュします。Amazon 開発者コンソールで作成したスキル用に、以下のものを作成してプッシュします。

- `skill.json` ファイル。
- `interactionModel/custom` フォルダ。

> **Note**
>
> このディレクトリ構造は、「スキルパッケージ形式」で説明されているように、Alexa Skills Kit スキルパッケージ形式の要件に準拠しています。ディレクトリ構造で正しいスキルパッケージ形式が使用されていない場合、変更は Alexa Skills Kit コンソールに正常にデプロイされません。

スキルのソースファイルを作成するには

1. Alexa Skills Kit 開発者コンソールからスキル ID を取得します。以下のコマンドを使用します。

   ```
   ask api list-skills
   ```

   名前に基づいてスキルを見つけ、skillId フィールドで、関連付けられた ID をコピーします。

2. スキルの詳細を含む skill.json ファイルを生成します。以下のコマンドを使用します。

   ```
   ask api get-skill -s skill-ID > skill.json
   ```

3. (オプション) interactionModel/custom フォルダを作成します。

   以下のコマンドを使用して、フォルダ内にインタラクションモデルファイルを生成します。locale について、このチュートリアルではファイル名のロケールとして en-US を使用します。

   ```
   ask api get-model --skill-id skill-ID --locale locale >
       ./interactionModel/custom/locale.json
   ```

CodeCommit リポジトリにファイルをプッシュするには

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

   ```
   skill.json
   /interactionModel
     /custom
       |en-US.json
   ```

2. ファイルをアップロードする方法を選択します。

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

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

```
git add -A
```

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

```
git commit -m "Added Alexa skill files"
```

iii. 次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いて更新トークンを使用して、ユーザーに代わって実行するアクションを承認します。このセクションでは、ASK CLI を使用してトークンを作成します。[パイプラインを作成する] ウィザードでこれらの認証情報を使用します。

Amazon 開発者アカウントの認証情報を使用して更新トークンを作成するには

1. 以下のコマンドを使用します。

```
ask util generate-lwa-tokens
```

2. プロンプトが表示されたら、以下の例に示すようにクライアント ID とシークレットを入力します。

```
? Please type in the client ID:
amzn1.application-client.example112233445566
? Please type in the client secret:
```

```
example112233445566
```

3. サインインのブラウザページが表示されます。Amazon アカウント認証情報を使用してサインインします。
4. コマンドライン画面に戻ります。アクセストークンと更新トークンが出力に生成されます。出力に返された更新トークンをコピーします。

ステップ 4: パイプラインを作成する

このセクションでは、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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. プロジェクトとそのリソースを作成する AWS リージョンを選択します。Alexa スキルランタイムは、以下のリージョンでのみ利用できます。
   - アジアパシフィック (東京)
   - 欧州 (アイルランド)
   - 米国東部 (バージニア北部)
   - 米国西部 (オレゴン)
3. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
4. [ステップ 1: パイプラインの設定を選択する] の [パイプライン名] に「**MyAlexaPipeline**」と入力します。
5. このチュートリアルの目的では、[パイプラインタイプ] で、[V1] を選択します。[V2] を選択することもできますが、パイプラインタイプは特性と価格が異なることに注意してください。詳細については、「パイプラインのタイプ」を参照してください。
6. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロールを選択します。 CodePipeline

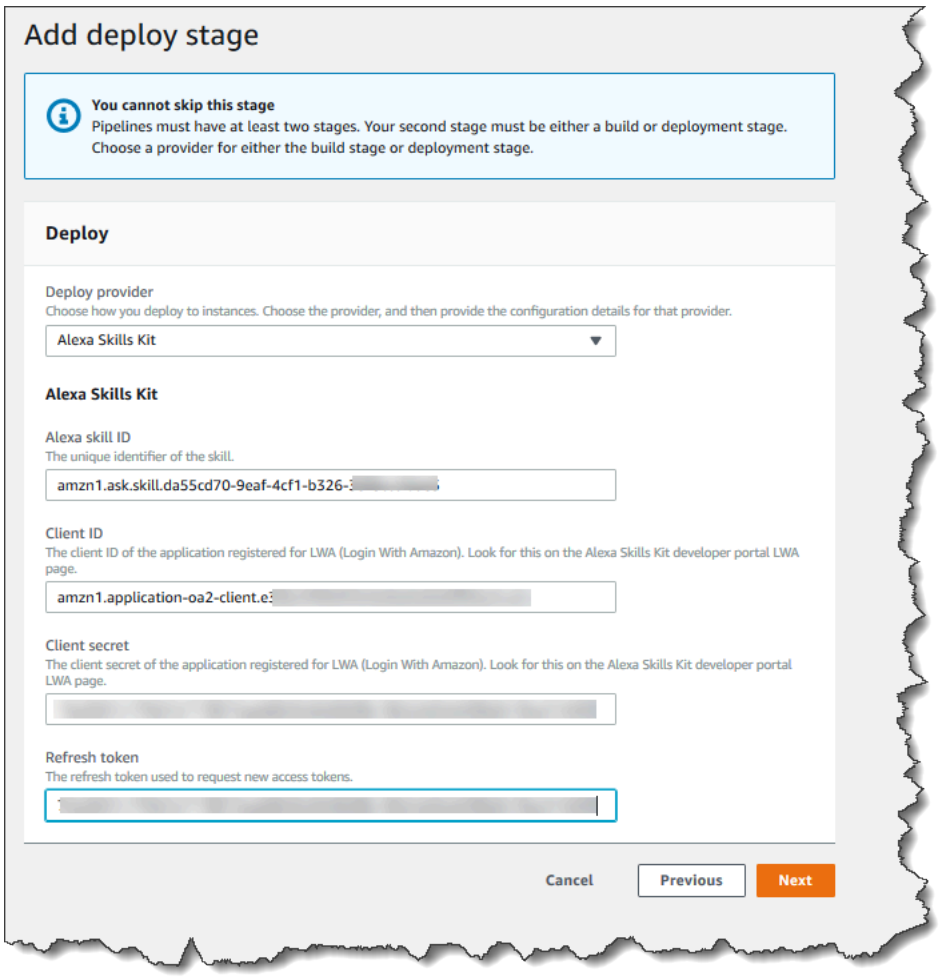
7. [詳細設定] をデフォルト設定のままにし、[次へ] を選択します。
8. [Step 2: Add source stage (ステップ 2: ソースステージの追加)] の [ソースプロバイダ] で、[AWS CodeCommit] を選択します。リポジトリ名 で、 で作成した CodeCommit リポジトリの名前を選択しますステップ 1: CodeCommit リポジトリを作成する。[Branch name] で、最新のコード更新を含むブランチの名前を選択します。

   リポジトリ名とブランチを選択すると、このパイプライン用に作成される Amazon CloudWatch Events ルールを示すメッセージが表示されます。

   [次へ] をクリックします。
9. [Step 3: Add build stage] (ステップ 3: ビルドステージを追加する) 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。

   [次へ] をクリックします。
10. ステップ 4: 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。



f. [次へ] をクリックします。

11. [ステップ 5: 確認] で情報を確認し、[パイプラインの作成] を選択します。

ステップ 5: 任意のソースファイルに変更を加えてデプロイを確認する

スキルに変更を加え、その変更をリポジトリにプッシュします。これにより、パイプラインの実行がトリガーされます。スキルが Alexa Skills Kit 開発者コンソールで更新されていることを確認します。

チュートリアル: Amazon S3 をデプロイプロバイダとして使用するパイプラインを作成する

このチュートリアルでは、デプロイステージでデプロイアクションプロバイダーとして Amazon S3 を使用して、ファイルを継続的に配信するパイプラインを設定します。ソースリポジトリ内のソー

スファイルに変更を加えると、完成したパイプラインはその変更を検出します。パイプラインは Amazon S3 を使用してそれらのファイルをバケットにデプロイします。ソースの場所で Web サイトのファイルを変更または追加するたびに、デプロイで最新のファイルを使用して Web サイトが作成されます。

> **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 にコンパイル JavaScript し、 CodeBuild 出力アーティファクトをアーカイブ用に 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 ビルドアクション、および 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 2 : 構築されたアーカイブファイルを S3 ソースバケットから Amazon S3 にデプロイする を参照してください。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、ソースアクションの pipeline. AWS resources 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。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) リージョンでパイプラインを作成する場合、 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、「でクロスリージョンアクションを追加する CodePipeline」を参照してください。

オプション 1: 静的ウェブサイトファイルを Amazon S3 にデプロイする

この例では、サンプルの静的ウェブサイトテンプレートファイルをダウンロードし、 AWS CodeCommit リポジトリにファイルをアップロードして、バケットを作成し、ホスティング用に設定します。次に、 AWS CodePipeline コンソールを使用してパイプラインを作成し、Amazon S3 デプロイ設定を指定します。

前提条件

以下のものを用意しておく必要があります。

- CodeCommit リポジトリ。で作成した AWS CodeCommit リポジトリを使用できますチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)。
- 静的ウェブサイトのソースファイル。このリンクを使用して サンプル静的ウェブサイトをダウンロードします。sample-website.zip をダウンロードすると、以下のファイルが生成されます。
  - `index.html` ファイル
  - `main.css` ファイル
  - `graphic.jpg` ファイル
- ウェブサイトホスティング用に設定された S3 バケット。「静的ウェブサイトを Amazon S3 でホスティングする」を参照してください。必ずパイプラインと同じリージョンにバケットを作成します。

> **Note**
>
> ウェブサイトのホスティングには、バケットへのパブリック読み取りアクセスを許可するアクセス設定が必要です。ウェブサイトのホスティングを除き、S3 バケットへのパブリックアクセスをブロックするデフォルトのアクセス設定を維持してください。

ステップ 1: ソースファイルを CodeCommit リポジトリにプッシュする

このセクションでは、パイプラインによってソースステージに使用されるリポジトリに、ソースファイルをプッシュします。

ファイルを CodeCommit リポジトリにプッシュするには

1. ダウンロードしたサンプルファイルを解凍します。ZIP ファイルをリポジトリにアップロードしないでください。

2. ファイルを CodeCommit リポジトリにプッシュまたはアップロードします。これらのファイルは、 でのデプロイアクション用にパイプラインの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 CodePipeline。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. 次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ウェブサイトのファイルである CodeCommit 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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: パイプラインの設定を選択する] の [パイプライン名] に「**MyS3DeployPipeline**」と入力します。
4. 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。
5. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロール を選択します。 CodePipeline
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. [Step 2: Add source stage (ステップ 2: ソースステージの追加)] の [ソースプロバイダ] で、[AWS CodeCommit] を選択します。リポジトリ名 で、 で作成したリポジトリの名前 CodeCommitを選択しますステップ 1: CodeCommit リポジトリを作成する。[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 ルールが表示されます。

   [次へ] をクリックします。
8. [Step 3: Add build stage] (ステップ 3: ビルドステージを追加する) 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。

   [次へ] をクリックします。
9. ステップ 4: デプロイステージを追加する:

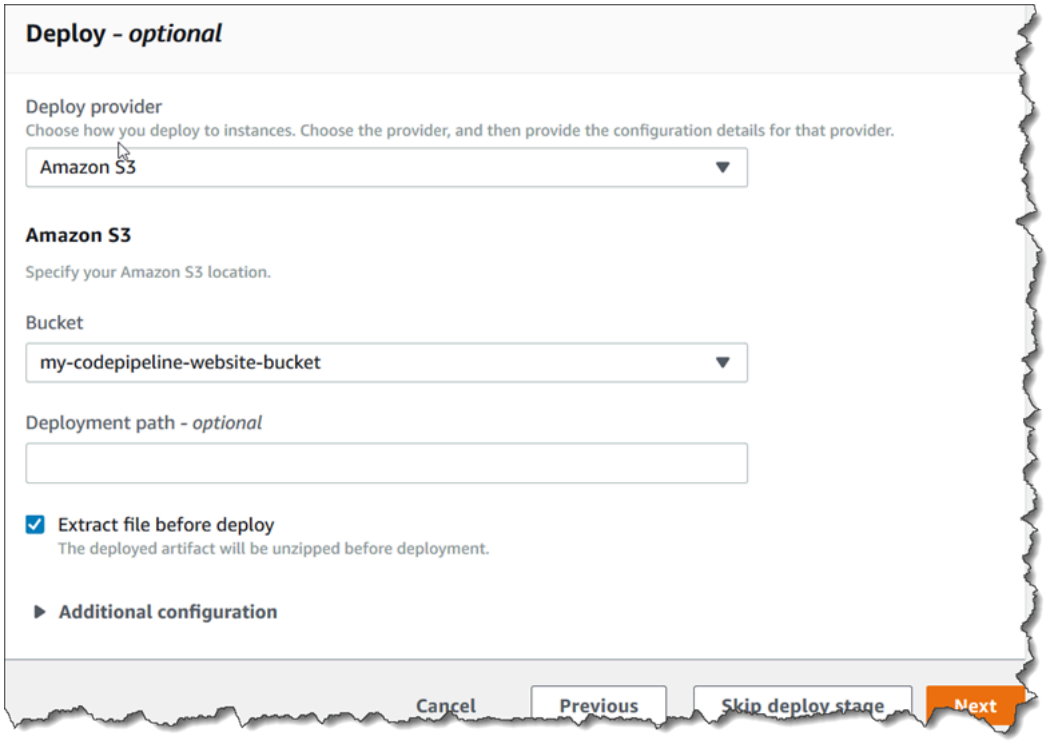
   a. [デプロイプロバイダ] で、[Amazon S3] を選択します。

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。



d. (オプション) [既定 ACL] で、既定 ACL と呼ばれる、あらかじめ定義された一連の許可を、アップロードされたアーティファクトに適用できます。

e. (オプション) [キャッシュコントロール] で、キャッシュパラメータを入力します。これを設定して、リクエスト/レスポンスのキャッシュ動作を制御できます。有効な値については、HTTP オペレーションの `Cache-Control` ヘッダーフィールドを参照してください。

f. [次へ] をクリックします。

10. [ステップ 5: 確認] で情報を確認し、[パイプラインの作成] を選択します。
11. パイプラインが正常に実行されたら、Amazon S3 コンソールを開き、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
```

12. 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 コードにコンパイルし、タイムスタンプが付けられた別のフォルダの S3 ターゲットバケットに出力をデプロイします。まず、 TypeScript コードと buildspec.yml ファイルを作成します。ZIP ファイルにソースファイルを組み合わせたら、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ビルドされたアプリケーション ZIP ファイルを S3 ターゲットバケットにデプロイします。コンパイルされたコードはアーカイブとしてターゲットバケットに保存されます。

前提条件

以下のものを用意しておく必要があります。

- S3 ソースバケット。「チュートリアル: シンプルなパイプラインを作成する (S3 バケット)」で作成したバケットを使用できます。
- S3 ターゲットバケット。「静的ウェブサイトを Amazon S3 でホスティングする」を参照してください。バケットは、作成するパイプライン AWS リージョン と同じ に作成してください。

> **Note**
>
> この例では、ファイルをプライベートバケットにデプロイする方法を示します。ウェブサイトのホスティング用にターゲットバケットを有効にしたり、バケットを公開するポリシーをアタッチしたりしないでください。

ステップ 1: ソースファイルを作成して S3 ソースバケットにアップロードする

このセクションでは、ソースファイルを作成し、パイプラインによってソースステージに使用されるバケットにプッシュします。このセクションでは、以下のソースファイルを作成する手順について説明します。

- CodeBuild ビルド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 TypeScript、 コンパイラをインストールして使用し、 のコードをコードに書き換えindex.tsます JavaScript。

```
version: 0.2

phases:
  install:
    commands:
      - npm install -g typescript
  build:
    commands:
      - tsc index.ts
artifacts:
  files:
    - index.js
```

index.ts ファイルを作成するには

- 次の内容で、`index.ts` という名前のファイルを作成します。

```
interface Greeting {
    message: string;
}

class HelloGreeting implements Greeting {
    message = "Hello!";
}

function greet(greeting: Greeting) {
    console.log(greeting.message);
}

let greeting = new HelloGreeting();

greet(greeting);
```

ファイルを S3 ソースバケットにアップロードするには

1. ファイルは、ローカルディレクトリに次のように表示されます。

   ```
   buildspec.yml
   index.ts
   ```

   ファイルを圧縮して、ファイルに source.zip という名前を付けます。

2. Amazon S3 コンソールで、ソースバケット用に アップロード を選択します。[ファイルを追加] を選択し、作成した ZIP ファイルを参照します。

3. [アップロード] を選択します。これらのファイルは、 のデプロイアクション用にパイプラインの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 CodePipeline。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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: パイプラインの設定を選択する] の [パイプライン名] に「**MyS3DeployPipeline**」と入力します。
4. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロールを選択します。 CodePipeline
5. [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。
6. [Step 2: Add source stage (ステップ 2: ソースステージの追加)] ページの [ソースプロバイダ] で、[Amazon S3] を選択します。[バケット] で、ソースバケットの名前を選択します。[S3 object key (S3 オブジェクトキー)] に、ソース ZIP ファイルの名前を入力します。必ず、ファイル拡張子.zip を含めてください。

   [次へ] をクリックします。
7. [Step 3: Add build stage (ステップ 3: ビルドステージの追加] で:

   a. [ビルドプロバイダ] 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. サービスロール で、 CodeBuild サービスロールを選択するか、作成します。

h. [ビルド仕様] で、[Use a buildspec file (ビルド仕様ファイルの使用)] を選択します。

i. 「続行」を選択します CodePipeline。プロジェクトが正常に作成された場合はメッセージが表示されます。

j. [次へ] をクリックします。

8. ステップ 4: デプロイステージを追加する:

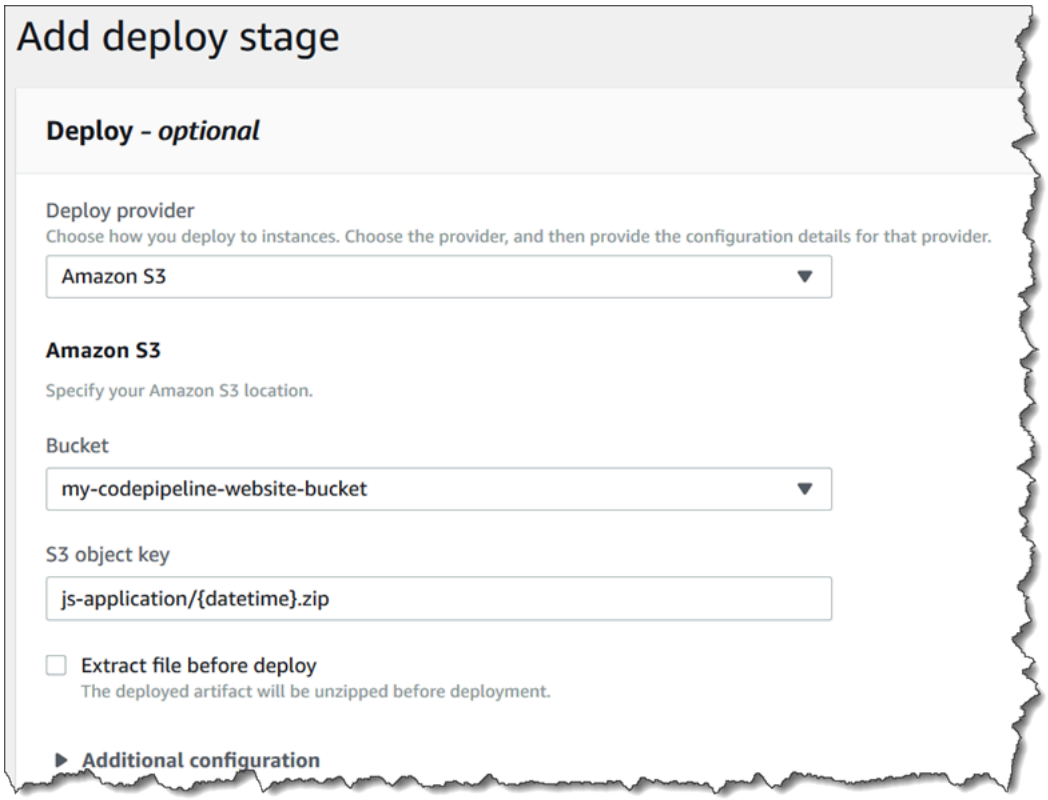
a. [デプロイプロバイダ] で、[Amazon S3] を選択します。

b. [バケット] に、S3 ターゲットバケットの名前を入力します。

c. [Extract file before deploy (デプロイ前にファイルを展開)] がオフになっていることを確認します。

[Extract file before deploy (デプロイ前にファイルを展開)] がオフになっていると、[S3 object key (S3 オブジェクトキー)] が表示されています。使用するパスの名前を入力します: `js-application/{datetime}.zip`。

これにより、ファイルが展開される `js-application` フォルダが Amazon S3 に作成されます。このフォルダでは、パイプラインの実行時に `{datetime}` 変数によって各出力ファイルにタイムスタンプが付けられます。



d. (オプション) [既定 ACL] で、既定 ACL と呼ばれる、あらかじめ定義された一連の許可を、アップロードされたアーティファクトに適用できます。

e. (オプション) [キャッシュコントロール] で、キャッシュパラメータを入力します。これを設定して、リクエスト/レスポンスのキャッシュ動作を制御できます。有効な値については、HTTP オペレーションの Cache-Control ヘッダーフィールドを参照してください。

f. [次へ] をクリックします。

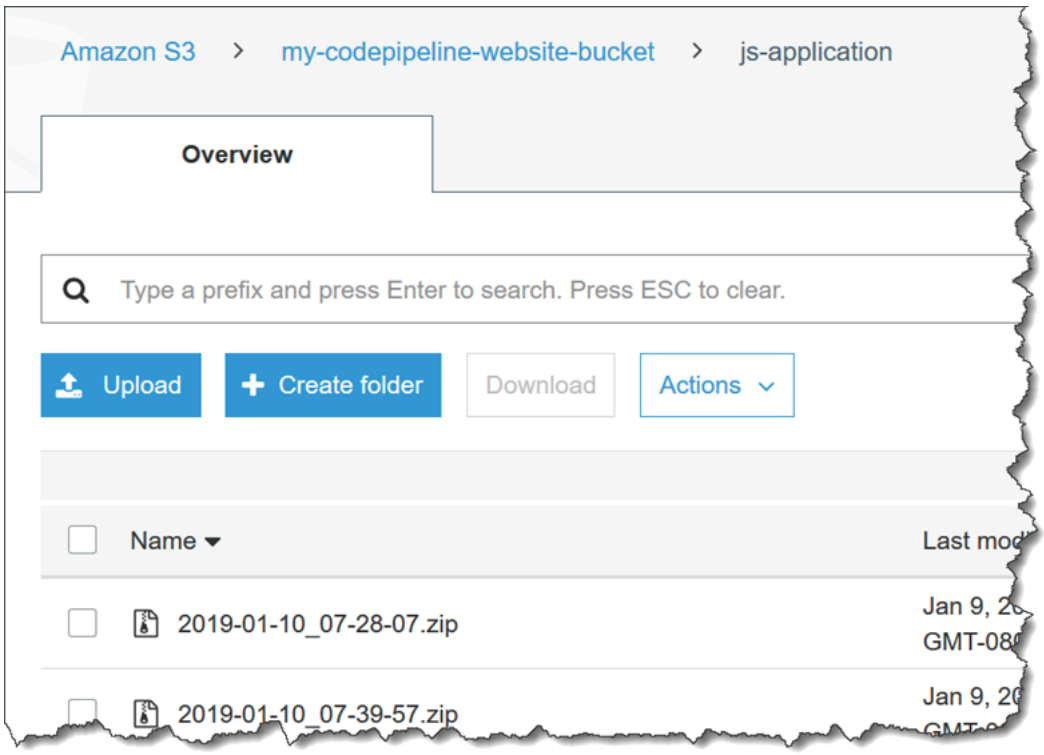
9. [ステップ 5: 確認] で情報を確認し、[パイプラインの作成] を選択します。

10. 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 JavaScript ファイルに含まれるファイルは である必要がありますindex.js。`index.js` ファイルには、以下の出力が含まれています。

```
var HelloGreeting = /** @class */ (function () {
    function HelloGreeting() {
        this.message = "Hello!";
    }
    return HelloGreeting;
}());
function greet(greeting) {
    console.log(greeting.message);
}
var greeting = new HelloGreeting();
greet(greeting);
```

ステップ 3: 任意のソースファイルに変更を加えてデプロイを確認する

ソースファイルに変更を加え、それらのファイルをソースバケットにアップロードします。これにより、パイプラインの実行がトリガーされます。ターゲットのバケットを表示し、デプロイされた出力ファイルが以下のように `js-application` フォルダにあることを確認します。



チュートリアル: サーバーレスアプリケーションを に発行するパイプラインを作成する AWS Serverless Application Repository

を使用して AWS CodePipeline 、 AWS SAM サーバーレスアプリケーションを に継続的に配信できます AWS Serverless Application Repository。

このチュートリアルでは、 でホストされているサーバーレスアプリケーションを構築して AWS Serverless Application Repository に自動的に GitHub 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。パイプラインは、ソースプロバイダー GitHub として を使用し、ビルドプロバイダー CodeBuild として 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、コードを記述せずに AWS Serverless Application Repository、アプリケーションの更新を に継続的に配信できます。

> **Important**
>
> この手順でパイプラインに追加するアクションの多くには、ソースアクションの pipeline. AWS resources 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。パイプラインを作成するのと同じ AWS リージョンに常に作成する必要があります。例えば、米国東

部 (オハイオ) リージョンでパイプラインを作成する場合、 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、「でクロスリージョンアクションを追加する CodePipeline」を参照してください。

開始する前に

このチュートリアルでは、以下のことを前提としています。

- AWS Serverless Application Model (AWS SAM) と AWS Serverless Application Repository をよく理解していること。
- AWS SAM CLI AWS Serverless Application Repository を使用して に発行 GitHub したサーバーレスアプリケーションが でホストされている。サンプルアプリケーションを に公開するには AWS Serverless Application Repository、「 AWS Serverless Application Repository デベロッパーガイド」の「クイックスタート: アプリケーションの公開」を参照してください。独自のアプリケーションを に公開するには AWS Serverless Application Repository、「 AWS Serverless Application Model デベロッパーガイド」の「 AWS SAM 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

次の内容のbuildspec.ymlファイルを作成し、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 でパイプラインを作成するには AWS リージョン 、次の手順に従います。

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
2. 必要に応じて、サーバーレスアプリケーションを公開 AWS リージョン する に切り替えます。
3. パイプラインの作成 を選択します。[Choose pipeline settings (パイプラインの設定の選択)] 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。
4. 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。
5. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロールを選択します。 CodePipeline
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. 「ソースステージの追加」ページの「ソースプロバイダー」で、「」を選択しますGitHub。
8. 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 ソースアクションの接続を作成または管理する方法については、「」を参照してくださいGitHub 接続。
9. リポジトリ で、 GitHub ソースリポジトリを選択します。
10. ブランチ で、 GitHub ブランチを選択します。
11. 出典アクションについては、残りのデフォルトのままにしておきます。[次へ] をクリックします。
12. [Add build stage (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. 「続行」を選択します CodePipeline。これにより、 CodePipeline コンソールが開き、リポジトリ`buildspec.yml`内の を使用して設定を行う 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] をクリックします。

13. [Add deploy stage (デプロイステージの追加)] ページで、[Skip deploy stage (デプロイステージのスキップ)] を選択し、[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] をクリックします。

14. [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。

15. 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 バケットにアクセスするアクセス許可を 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構築ステージで、 を選択しますCodeBuild。

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。ステートメントでは CodeBuild 、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. [変更を保存] を選択します。

ステップ 3: 発行アプリケーションをデプロイする

以下のステップに従って、 AWS Serverless Application Repositoryの公開を実行する Lambda 関数を含むアプリケーションをデプロイします。このアプリケーションは aws-serverless-codepipeline-serverlessrepo-publish です。

> **Note**
> パイプライン AWS リージョン と同じ にアプリケーションをデプロイする必要があります。

1. アプリケーションのページに移動し、[デプロイ] を選択します。
2. [I acknowledge that this app creates custom IAM roles (このアプリケーションがカスタム IAM ロールを作成することを承認します)] を選択します。
3. [デプロイ] を選択します。
4. AWS CloudFormation スタックを表示 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。タイプ ServerlessRepoPublishの が表示されますAWS::Lambda::Function。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、 で新しい発行アクションを作成するときに使用します CodePipeline。

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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
2. 左のナビゲーションセクションで、編集するパイプラインを選択します。
3. [編集] を選択します。
4. 現在のパイプラインの最後のステージが終わったら、[+ Add stage (+ ステージの追加)] を選択します。[Stage name (ステージ名)] に名前 (**`Publish`** など) を入力し、[Add stage (ステージの追加)] を選択します。
5. 新しいステージで、[+ Add action group (+ アクショングループの追加)] を選択します。

6. アクション名を入力します。[アクションプロバイダ] の [呼び出し] で、[AWS Lambda] を選択します。
7. 入力アーティファクト から、 を選択しますBuildArtifact。
8. [関数名] で、前のステップで書き留めた Lambda 関数の物理 ID を選択します。
9. アクションの [保存] を選択します。
10. ステージの [完了] を選択します。
11. 右上の [保存] を選択します。
12. パイプラインを確認するには、 でアプリケーションを変更します GitHub。例えば、 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、 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。のアクションの変数については CodePipeline、「」を参照してください変数。

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`CommitId`変数を消費します。
  - 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。

- パイプラインは、 の CodeCommit ソースで作成または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)。
- CodeCommit ソースアクションに名前空間が含まれるように、既存のパイプラインを編集します。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、https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
2. [関数を作成] 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。
3. [関数名] に、関数の名前 (**`myInvokeFunction`** など) を入力します。[ランタイム] は、デフォルトのオプションを選択したままにします。
4. [実行ロールの選択または作成] を選択します。[基本的な Lambda アクセス権限で新しいロールを作成] を選択します。
5. [関数を作成] を選択します。
6. 別のアクションからの変数を使用するには、Lambda 呼び出しアクションの設定で `UserParameters` にその変数を渡す必要があります。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。

   ```
   const commitId =
   event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;
   ```

   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 successInput = {
     jobId: jobId,
   ```

```
  outputVariables: {
    testRunId: Math.floor(Math.random() * 1000).toString(),
    dateTime: Date(Date.now()).toString(),
    region: lambdaRegion
  }
};
```

新しい関数で、[コードをインラインで編集] を選択したまま、次のコード例を index.js の下に貼り付けます。

```
var AWS = require('aws-sdk');

exports.handler = function(event, context) {
    var codepipeline = new AWS.CodePipeline();

    // Retrieve the Job ID from the Lambda action
    var jobId = event["CodePipeline.job"].id;

    // Retrieve the value of UserParameters from the Lambda action configuration in
 CodePipeline,
    // in this case it is the Commit ID of the latest change of the pipeline.
    var params =
 event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // The region from where the lambda function is being executed.
    var lambdaRegion = process.env.AWS_REGION;

    // Notify CodePipeline of a successful job
    var putJobSuccess = function(message) {
        var params = {
            jobId: jobId,
            outputVariables: {
                testRunId: Math.floor(Math.random() * 1000).toString(),
                dateTime: Date(Date.now()).toString(),
                region: lambdaRegion
            }
        };
        codepipeline.putJobSuccessResult(params, function(err, data) {
            if(err) {
                context.fail(err);
            } else {
                context.succeed(message);
            }
```

```
        });
    };

    // Notify CodePipeline of a failed job
    var putJobFailure = function(message) {
        var params = {
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.invokeid
            }
        };
        codepipeline.putJobFailureResult(params, function(err, data) {
            context.fail(message);
        });
    };

    var sendResult = function() {
        try {
            console.log("Testing commit - " + params);

            // Your tests here

            // Succeed the job
            putJobSuccess("Tests passed.");
        } catch (ex) {
            // If any of the assertions failed then fail the job
            putJobFailure(ex);
        }
    };

    sendResult();
};
```

7. [保存] を選択します。
8. 画面上部の Amazon リソースネーム (ARN) をコピーします。
9. 最後のステップとして、https://console.aws.amazon.com/iam/ で AWS Identity and Access Management (IAM) コンソールを開きます。Lambda 実行ロールを変更して、次のポリシーを追加します: AWSCodePipelineCustomActionAccess。Lambda 実行ロールを作成したり、ロールポリシーを変更したりする手順については、「ステップ 2 : Lambda 関数を作成する」を参照してください。

ステップ 2: Lambda 呼び出しアクションと手動承認アクションをパイプラインに追加する

このステップでは、パイプラインに Lambda 呼び出しアクションを追加します。Test という名前のステージの一部としてアクションを追加します。アクションタイプは、呼び出しアクションです。次に、呼び出しアクションの後に、手動承認アクションを追加します。

パイプラインに Lambda アクションと手動承認アクションを追加するには

1. https://console.aws.amazon.com/codepipeline/ 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、アクションを追加するアイコンを選択します。「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 関数の名前を選択します。
   f. ユーザーパラメータ に、 CodeCommit コミット ID の変数構文を入力します。これにより、パイプラインが実行されるたびに確認および承認されるコミットに解決される出力変数が作成されます。

      ```
      #{SourceVariables.CommitId}
      ```
   g. [変数名前空間] に名前空間名 (**TestVariables** 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がありますSourceVariables。Lambda リージョン出力変数には、TestVariables 名前空間があります。URL は次のようになります。

```
https://#{TestVariables.region}.console.aws.amazon.com/codesuite/codecommit/
repositories/MyDemoRepo/commit/#{SourceVariables.CommitId}
```

[コメント] で、testRunId 変数の変数構文を追加して、承認メッセージテキストを作成します。この例では、Lambda testRunId 出力変数の変数構文を持つ URL には TestVariables 名前空間があります。以下のメッセージを入力します。

```
Make sure to review the code before approving this action. Test Run ID:
 #{TestVariables.testRunId}
```

4. [完了] を選択してアクションの編集画面を閉じ、[完了] を選択してステージの編集画面を閉じます。パイプラインを保存するには、[完了] を選択します。完成したパイプラインには、ソース、テスト、承認、デプロイの各ステージがある構造が含まれています。

[変更のリリース] を選択して、パイプライン構造で最新の変更を実行します。

5. パイプラインが手動承認ステージに達したら、[確認] を選択します。解決された変数は、コミット ID の URL として表示されます。承認者は、コミットを表示する URL を選択できます。

6. パイプラインが正常に実行されたら、アクションの実行履歴ページで変数の値を表示することもできます。

チュートリアル: パイプライン AWS Step Functions で呼び出しアクションを使用する

を使用して AWS Step Functions 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。

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

このチュートリアルでは、既存のパイプラインに呼び出しアクションを追加します。チュートリアル: シンプルなパイプラインを作成する (S3 バケット) または チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) で作成したパイプラインを使用できます。

ソースアクションと少なくとも 2 ステージ構造を持つ既存のパイプラインを使用しますが、この例ではソースアーティファクトを使用しません。

> **Note**
>
> このアクションを実行するために必要な追加のアクセス許可を使用して、パイプラインで使用されるサービスロールを更新する必要がある場合があります。これを行うには、 AWS Identity and Access Management (IAM) コンソールを開き、ロールを検索してから、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、「 CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

ステップ 1: サンプルステートマシンを作成する

ステップ関数コンソールで、`HelloWorld` サンプルテンプレートを使用してステートマシンを作成します。手順については、AWS Step Functions デベロッパーガイド の ステートマシンの作成 を参照してください。

ステップ 2: パイプラインにステップ関数呼び出しアクションを追加する

次のように、ステップ関数呼び出しアクションをパイプラインに追加します。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、編集するパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインの各ステージの各アクションの状態などがわかります。
3. パイプライン詳細ページで、[編集] を選択します。
4. シンプルなパイプラインの 2 番目のステージで、[Edit stage (ステージの編集)] を選択します。[削除] を選択します。これで、不要になった 2 番目のステージが削除されました。
5. 図の最下部で [+ Add stage] (+ ステージの追加) を選択します。
6. [ステージ名] にステージ名 (**`Invoke`** など) を入力し、[Add stage (ステージの追加)] を選択します。
7. [+ Add action group (+ アクションの追加)] を選択します。
8. [アクション名] に名前 (**`Invoke`** など) を入力します。
9. [アクションプロバイダー] で、[AWS ステップ関数] を選択します。[リージョン] がデフォルトでパイプラインリージョンにな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、アクションの入力アーティファクトを記述 CodePipeline するために で使用される用語とは異なります。

    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。

16. 編集中のステージで、[完了] を選択します。 AWS CodePipeline のペインで [保存] を選択し、警告メッセージで [保存] を選択します。
17. 変更を送信してパイプラインの実行を開始するには、[変更のリリース]、[リリース] の順に選択します。
18. 完了したパイプライン内の呼び出しアクションで、[AWS ステップ関数] を選択します。 AWS Step Functions コンソールで、ステートマシンの実行 ID を表示します。ID には、ステートマシン名 `HelloWorld` と、ステートマシンの実行 ID とプレフィックス `my-prefix` が表示されます。

```
arn:aws:states:us-west-2:account-ID:execution:HelloWorld:my-
prefix-0d9a0900-3609-4ebc-925e-83d9618fcca1
```

チュートリアル: をデプロイプロバイダー AWS AppConfig として使用するパイプラインを作成する

このチュートリアルでは、デプロイステージのデプロイアクションプロバイダー AWS AppConfig として 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。

トピック

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: リソースを作成する AWS AppConfig

このセクションでは、次のリソースを作成します。

- のアプリケーション AWS AppConfig は、顧客に機能を提供するコードの論理単位です。
- の環境 AWS AppConfig は、ベータ環境や本番環境のアプリケーションなど、 AppConfig ターゲットの論理的な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) の AWS デプロイ戦略は、デプロイ中に任意の時点で新しいデプロイされた設定を受け取るクライアントの割合など、設定デプロイの動作 AppConfig 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. 以下のトピックのステップを使用して、 で AWS リソースを作成します AppConfig。
   - アプリケーションを作成します。
   - 環境を作成します。
   - AWS CodePipeline 設定プロファイル を作成します。
   - (オプション) 定義済みのデプロイ戦略を選択するか、独自のデプロイメント戦略を作成します。

ステップ 2 : ファイルを S3 ソースバケットにアップロードします。

このセクションでは、設定ファイルを作成します。次に、パイプラインがソースステージに使用するバケットにソースファイルを zip してプッシュします。

設定ファイルを作成します。

1. 各リージョンの設定ごとに `configuration.json` ファイルを作成します。次の内容を含めます。:

   ```
   Hello World!
   ```

2. 次のステップを使用して、設定ファイルを zip してアップロードします。

ソースファイルを zip してアップロードします。

1. ファイルで .zip ファイルを作成し、.zip ファイル `configuration-files.zip` に名前を付けます。たとえば、.zip ファイルは次の構造を使用できます。:

```
.
### appconfig-configurations
    ### MyConfigurations
        ### us-east-1
        #   ### configuration.json
        ### us-west-2
            ### configuration.json
```

2. バケットの Amazon S3 コンソールで、アップロード を選択し、指示に従って、zip ファイルをアップロードします。

ステップ 3: パイプラインを作成する

このセクションでは、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。
- デプロイアクションを含む AppConfig 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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: パイプラインの設定を選択する] の [パイプライン名] に「**MyAppConfigPipeline**」と入力します。
4. 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。
5. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロールを選択します。 CodePipeline
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. [Step 2: Add source stage (ステップ 2: ソースステージの追加)] ページの [ソースプロバイダ] で、[Amazon S3] を選択します。バケット で、S3 ソースバケットの名前を選択します。

   S3 オブジェクトキー に、ZIP ファイル名に `configuration-files.zip` を入力します。

   [次へ] をクリックします。

8. [Step 3: Add build stage] (ステップ 3: ビルドステージを追加する) 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。

   [次へ] をクリックします。

9. ステップ 4: デプロイステージを追加する:

   a. [デプロイプロバイダ] で、[AWS AppConfig] を選択します。

   b. アプリケーション で、 で作成したアプリケーションの名前を選択します AWS AppConfig。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. 環境 で、 で作成した環境の名前を選択します AWS AppConfig。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、 で作成した設定プロファイルの名前を選択します AWS AppConfig。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、 で作成したデプロイ戦略 AppConfig でも、 で事前定義されたデプロイ戦略から選択したものでもかまいません AppConfig。このフィールドには、デプロイ戦略の ID が表示されます。

   f. アーティファクト設定パスを入力 に、ファイルパスを入力します。入力アーティファクト設定パスが S3 バケット.zip ファイルのディレクトリ構造と一致していることを確認します。この例では、次のファイルパス : `appconfig-configurations/MyConfigurations/us-west-2/configuration.json` を入力します。

   g. [次へ] をクリックします。

10. [ステップ 5: 確認] で情報を確認し、[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: GitHub パイプラインソースでフルクローンを使用する

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 CodePipeline。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。

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、ソースデータのフルクローンオプションを使用し、リポジトリをクローンしてリポジトリの Git コマンドを実行する CodeBuild ビルドを実行します。

> **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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

トピック

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 認証情報を用意します。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。

- GitHub リポジトリとアクションへの接続があるソースステージ。
- 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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソール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: パイプラインの設定を選択する] の [パイプライン名] に「**MyGitHubPipeline**」と入力します。
4. 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。
5. [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**
>
> 代わりに既存の CodePipeline サービスロールを使用する場合は、サービスロールポリシーに `codeconnections:UseConnection` IAM アクセス許可を追加していることを確認してください。 CodePipeline サービスロールの手順については、 CodePipeline「サービスロール にアクセス許可を追加する」を参照してください。

6. [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] をクリックします。

7. ステップ 2: [Add source stage] (ソースステージの追加) ページで、ソースステージを追加します。

   a. ソースプロバイダー で、 GitHub （ バージョン 2) 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 ソースアクションの接続を作成または管理するには、「」を参照してくださいGitHub 接続。

   c. リポジトリ名 で、リポジトリの名前 GitHub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 CodeBuild できるのは、 によって提供されるアクションのみです。このチュートリアルステップ 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では、 を使用して CodeBuild 、プロジェクトサービスロールのアクセス許可を更新し、このオプションを使用します。

[次へ] をクリックします。

8. [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] がデフォルトでパイプラインリージョンになることを許可します。

   b. [プロジェクトを作成] を選択します。

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

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

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

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

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

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

   > **Note**
   >
   > ビルド仕様の env セクションで、この例に示すように、git コマンドの認証情報ヘルパーが有効になっていることを確認します。

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
```

```
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git archive --format=zip HEAD > application.zip
  #post_build:
    #commands:
      # - command
      # - command
artifacts:
  files:
    - application.zip
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

h. 「続行」を選択します CodePipeline。これにより、 CodePipeline コンソールに戻り、ビルドコマンドを使用して設定を行う 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] をクリックします。

9. [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] ページで、[Skip deploy stage (デプロイステージのスキップ)] を選択し、[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] をクリックします。

10. [Step 5: Review (ステップ 5: 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 サービスロールは接続を使用するためのアクセス許可で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。`codeconnections: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 Enterprise Server GitHub、または GitLab.com への接続の CodeBuild GitClone アクセス許可を追加する 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

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 CodePipeline。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 CodeBuild が 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、ソースデータのフル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、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**
>
> CodeBuild アクションは、Git クローンオプションで使用できる Git メタデータの使用をサポートする唯一のダウンストリームアクションです。また、パイプラインにはクロスアカウ

ントアクションを含めることができますが、 CodeCommit完全なクローンオプションを成功させるには、アクションと CodeBuild アクションが同じアカウントにある必要があります。

トピック

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ソースファイルの例を提供します。

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。

- ソースアクションを含む CodeCommit 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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソール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: パイプラインの設定を選択する] の [パイプライン名] に「**MyCodeCommitPipeline**」と入力します。
4. 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。
5. [Service role (サービスロール)] で、次のいずれかの操作を行います。
   - [Existing service role (既存のサービスロール)] を選択します。
   - 既存の CodePipeline サービスロールを選択します。このロールには、サービスロールポリシーに対する `codecommit:GetRepository` IAM 許可が必要です。CodePipeline 「サービスロール にアクセス許可を追加する」を参照してください。
6. [詳細設定] では、デフォルト値のままにします。[次へ] をクリックします。
7. ステップ 2: ソースステージの追加 ページで、次を行います。

   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 CodeBuild できるのは、 によって提供されるアクションのみです。

   [次へ] をクリックします。
8. 構築ステージの追加 で、次を行います。

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

b. [プロジェクトを作成] を選択します。

c. [プロジェクト名] に、このビルドプロジェクトの名前を入力します。

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

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

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

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

g. [Buildspec] の Build specifications (ビルド仕様) で、[Insert build commands] (ビルドコマンドの挿入) を選択します。エディタに切り替え を選択し、構築コマンド の下に次のコードを貼り付けます。

```
version: 0.2

env:
  git-credential-helper: yes
phases:
  install:
    #If you use the Ubuntu standard image 2.0 or later, you must specify
 runtime-versions.
    #If you specify runtime-versions and use an image other than Ubuntu
 standard image 2.0, the build fails.
    runtime-versions:
      nodejs: 12
      # name: version
    #commands:
      # - command
      # - command
  pre_build:
    commands:
      - ls -lt
      - cat README.md
```

```
  build:
    commands:
      - git log | head -100
      - git status
      - ls
      - git describe --all
  #post_build:
    #commands:
      # - command
      # - command
#artifacts:
  #files:
    # - location
  #name: $(date +%Y-%m-%d)
  #discard-paths: yes
  #base-directory: location
#cache:
  #paths:
    # - paths
```

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

i. [次へ] をクリックします。

9. [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] ページで、[Skip deploy stage (デプロイステージのスキップ)] を選択し、[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] をクリックします。

10. [Step 5: Review (ステップ 5: 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のクローンを作成する

リポジトリからプルするアクセス許可で 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、ソースアクションの CodeBuild GitClone CodeCommitアクセス許可を追加する を参照してください。

ステップ 4: 構築出力でリポジトリコマンドを表示する

ビルド出力 を表示するには

1. サービスロールが正常に更新されたら、失敗した CodeBuild ステージで再試行を選択します。
2. パイプラインが正常に実行されたら、成功したビルドステージで [詳細を表示] を選択します。

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

   コマンドは、ファイルの内容を出力し、ディレクトリ内の `README.md` ファイルを一覧表示し、リポジトリのクローンを作成し、ログを表示して、`git describe --all` を実行します。

チュートリアル: AWS CloudFormation StackSets デプロイアクションを使用してパイプラインを作成する

このチュートリアルでは、 AWS CodePipeline コンソールを使用して、スタックセットとスタックインスタンスを作成するためのデプロイアクションを含むパイプラインを作成します。パイプラインが実行されると、テンプレートはスタックセットを作成し、スタックセットをデプロイするインスタンスを作成および更新します。

スタックセットのアクセス許可を管理するには、セルフマネージド型と マネージド AWS型の IAM ロールの 2 つの方法があります。このチュートリアルでは、セルフマネージド型の許可の例を示します。

で スタックセットを最も効果的に使用するには CodePipeline、背後にある概念 AWS CloudFormation StackSets とその仕組みを明確に理解しておく必要があります。「 ユーザーガイド」の「 のStackSets 概念」を参照してください。 AWS CloudFormation

トピック

- ステップ 5: デプロイのスタックセットリソースを表示する
- ステップ 6: スタックセットを更新する

前提条件

スタックセットオペレーションでは、管理者アカウントとターゲットアカウントの 2 つの異なるアカウントを使用します。管理者アカウントでは、スタックセットを作成します。ターゲットアカウントでは、スタックセットに属する個別のスタックを作成します。

管理者アカウントで管理者ロールを作成するには

- 「スタックセットオペレーションの基本アクセス許可の設定」の手順に従います。ロールは **AWSCloudFormationStackSetAdministrationRole** という名前にする必要があります。

ターゲットアカウントにサービスロールを作成するには

- 管理者アカウントを信頼するターゲットアカウントにサービスロールを作成します。「スタックセットオペレーションの基本アクセス許可の設定」の手順に従います。ロールは **AWSCloudFormationStackSetExecutionRole** という名前にする必要があります。

ステップ 1: サンプル AWS CloudFormation テンプレートとパラメータファイルをアップロードする

スタックセットテンプレートとパラメータファイルのソースバケットを作成します。サンプル AWS CloudFormation テンプレートファイルをダウンロードし、パラメータファイルを設定し、ファイルを圧縮してから S3 ソースバケットにアップロードします。

> **Note**
>
> ソースファイルが唯一のテンプレートであっても、S3 ソースバケットにアップロードする前に、必ずソースファイルを圧縮してください。

S3 ソースバケットを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. [バケットを作成] を選択します。
3. [バケット名] にバケットの名前を入力します。

   [リージョン] で、パイプラインを作成するリージョンを選択します。[バケットを作成] を選択します。
4. バケットが作成されると、成功バナーが表示されます。[バケットの詳細に移動] を選択します。
5. [プロパティ] タブで、[バージョニング] を選択します。[バージョニングの有効化] を選択し、[保存] を選択します。

AWS CloudFormation テンプレートファイルを作成するには

1. スタックセット CloudTrail の設定を生成するためのサンプルテンプレートファイルをダウンロードします。 https://s3.amazonaws.com/cloudformation-stackset-sample-templates-us-east-1/EnableAWSCloudtrail.yml
2. `template.yml` という名前でファイルを保存します。

parameters.txt ファイルを作成するには

1. デプロイのパラメータでファイルを作成します。パラメータは、実行時にスタック内で更新する値です。次のサンプルファイルは、スタックセットのテンプレートパラメータを更新して、ログ記録検証とグローバルイベントを有効にします。

```
[
  {
    "ParameterKey": "EnableLogFileValidation",
    "ParameterValue": "true"
  },
  {
    "ParameterKey": "IncludeGlobalEvents",
    "ParameterValue": "true"
  }
]
```

2. `parameters.txt` という名前でファイルを保存します。

accounts.txt ファイルを作成するには

1. 次のサンプルファイルに示されているように、インスタンスを作成するアカウントでファイルを作成します。

```
[
    "111111222222","333333444444"
]
```

2. accounts.txt という名前でファイルを保存します。

ソースファイルを作成してアップロードするには

1. ファイルを単一の zip ファイルに結合します。ファイルは zip ファイルで以下のようになっています。

```
template.yml
parameters.txt
accounts.txt
```

2. zip ファイルを S3 バケットにアップロードします。このファイルは、 でのデプロイアクション用にパイプラインの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 CodePipeline。

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [ようこそ] ページ、[開始方法] ページ、または [パイプライン] ページで、[パイプラインの作成] を選択します。
3. [ステップ 1: パイプラインの設定を選択する] の [パイプライン名] に「**MyStackSetsPipeline**」と入力します。
4. 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。
5. サービスロール で、IAM でサービスロールを作成することを許可する新しいサービスロール を選択します。 CodePipeline
6. [アーティファクトストア] では、デフォルト値はそのままにしておきます。

   > **Note**
   >
   > これはソースコードのソースバケットではありません。パイプラインのアーティファクトストアです。パイプラインごとに S3 バケットなどの個別のアーティファクトストアが必要です。パイプラインを作成または編集するときは、パイプラインリージョンにアーティファクトバケットと、アクションを実行している AWS リージョンごとに 1 つのアーティファクトバケットが必要です。
   > 詳細については、「入力および出力アーティファクト」および「CodePipeline パイプライン構造リファレンス」を参照してください。

   [次へ] を選択します。
7. [Step 2: Add source stage] (ステップ 2: 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。
8. [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。
9. [次へ] をクリックします。

10. [Step 3: Add build stage] (ステップ 3: ビルドステージを追加する) 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。

    [次へ] をクリックします。

11. ステップ 4: 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。これは、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 で AWS リージョン、 など、初期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. [次へ] をクリックします。

l. [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。

m. パイプラインの実行を許可します。

ステップ 3: 初期デプロイを表示する

初期デプロイのリソースとステータスを表示します。デプロイでスタックセットが正常に作成されたことを確認したら、2 番目のアクションを [デプロイ] ステージに追加します。

リソースを表示するには

1. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
2. [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。
3. パイプライン内の AWS CloudFormation アクションに対するCloudFormationStackSetアクションを選択します。スタックセットのテンプレート、リソース、イベントが AWS CloudFormation コンソールに表示されます。
4. 左側のナビゲーションパネルで、 を選択しますStackSets。リストで、新しいスタックセットを選択します。
5. [スタックインスタンス] タブを選択します。us-east-1 リージョンでは、提供したアカウントごとに 1 つのスタックインスタンスが作成されていることを確認します。各スタックインスタンスのステータスが CURRENT になっていることを確認します。

ステップ 4: アクションを追加する CloudFormationStackInstances

パイプラインに、 が残りのスタックインスタンス AWS CloudFormation StackSets を作成できるようにする次のアクションを作成します。

パイプラインで次のアクションを作成するには

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

   [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。
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. [アーティファクト入力] で SourceArtifact 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 で AWS リージョン、 `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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
2. [パイプライン] で、パイプラインを選択してから、[表示] を選択します。この図は、パイプラインのソースとデプロイのステージを示しています。
3. パイプライン内の AWS CloudFormation アクションに対する**AWS CloudFormation Stack Instances**アクションを選択します。スタックセットのテンプレート、リソース、イベントが AWS CloudFormation コンソールに表示されます。
4. 左側のナビゲーションパネルで、 を選択しますStackSets。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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
2. [パイプライン] で、パイプラインを選択してから、[編集] を選択します。[デプロイ] ステージで、[編集] を選択します。
3. パイプラインで、[AWS CloudFormation スタックセット] アクションを選択して編集します。[説明] で、既存の説明をスタックセットの新しい説明に書き直します。
4. パイプラインで、[AWS CloudFormation スタックインスタンス] アクションを選択して編集します。デプロイターゲット AWS リージョンで、アクションの作成時に入力されたus-east-2値を削除します。

5. 変更を保存します。[変更のリリース] を選択して、パイプラインを実行します。
6. AWS CloudFormationでアクションを開きます。StackSet 情報タブを選択します。StackSet 説明 で、新しい説明が表示されていることを確認します。
7. [スタックインスタンス] タブを選択します。[ステータス] で、us-east-2 のスタックインスタンスのステータスが OUTDATED であることを確認します。

CodePipeline ベストプラクティスとユースケース

以下のセクションでは、 のベストプラクティスについて説明します CodePipeline。

トピック

- のユースケース CodePipeline

のユースケース CodePipeline

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 AWS のサービス、Amazon S3 などの でも、 などのサードパーティー製品でもかまいません GitHub。このセクションでは、 を使用して CodePipeline 、さまざまな製品統合を使用してコードリリースを自動化する例を示します。アクションタイプ別に CodePipeline 整理された との統合の完全なリストについては、「」を参照してくださいCodePipeline パイプライン構造リファレンス。

トピック

- Amazon S3 CodePipeline で を使用する AWS CodeCommit、および AWS CodeDeploy
- サードパーティーのアクションプロバイダー (GitHub および Jenkins) CodePipeline で を使用する
- CodePipeline で AWS CodeStar を使用してコードプロジェクトでパイプラインを構築する
- CodePipeline を使用して、 でコードをコンパイル、構築、テストする CodeBuild
- Amazon ECS CodePipeline で を使用して、コンテナベースのアプリケーションをクラウドに継続的に配信する
- Elastic Beanstalk CodePipeline で を使用してウェブアプリケーションをクラウドに継続的に配信する
- Lambda ベースおよびサーバーレスアプリケーションの継続的な配信 AWS Lambda に CodePipeline を使用する
- AWS CloudFormation テンプレート CodePipeline で を使用してクラウドに継続的に配信する

Amazon S3 CodePipeline で を使用する AWS CodeCommit、および AWS CodeDeploy

パイプラインを作成すると、 はパイプラインの各段階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 CodePipeline 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。 Amazon EC2
- のステップは、ウィザードを使用して、 AWS CodeCommit リポジトリをソースプロバイダーとして使用する「ソース」ステージでパイプラインを作成するチュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) のに役立ちます。このチュートリアルでは、AWS CodeDeploy を使用して AWS CodeCommit リポジトリから Amazon Linux を実行している Amazon EC2 インスタンスにサンプルアプリケーションをデプロイするパイプラインを作成します。

サードパーティーのアクションプロバイダー (GitHub および Jenkins) CodePipeline で を使用する

GitHub や Jenkins などのサードパーティー製品と統合するパイプラインを作成できます。チュートリアル: 4 ステージのパイプラインを作成する のステップは、次の操作を実行するパイプラインの作成方法を示しています。

- GitHub リポジトリからソースコードを取得します。
- Jenkins を使用してソースコードの構築とテストを実行、
- AWS CodeDeploy を使用して、Amazon Linux または Microsoft Windows Server を実行している Amazon EC2 インスタンスに、構築およびテスト済みのソースコードをデプロイします。

CodePipeline で AWS CodeStar を使用してコードプロジェクトでパイプラインを構築する

AWS CodeStar は、. AWS AWS CodeStar works でソフトウェア開発プロジェクトを管理し、 AWS リソースをプロジェクト開発ツールチェーンに結合 CodePipeline するための統合ユーザーインターフェイスを提供するクラウドベースのサービスです。 AWS CodeStar ダッシュボードを使用して、パイプライン、リポジトリ、ソースコード、ビルド仕様ファイル、デプロイ方法、完全なコードプロジェクトに必要なインスタンスまたはサーバーレスインスタンスのホスティングを自動的に作成できます。

AWS CodeStar プロジェクトを作成するには、コーディング言語とデプロイするアプリケーションのタイプを選択します。ウェブアプリケーション、ウェブサービス、Alexa スキルといったプロジェクトタイプを作成できます。

任意の IDE を AWS CodeStar ダッシュボードにいつでも統合できます。チームメンバーの追加や削除を行ったり、プロジェクトでチームメンバーのアクセス権限を管理することもできます。 AWS CodeStar を使用してサーバーレスアプリケーションのサンプル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、「チュートリアル: でのサーバーレスプロジェクトの作成と管理 AWS CodeStar」を参照してください。

CodePipeline を使用して、 でコードをコンパイル、構築、テストする CodeBuild

CodeBuild は、サーバーやシステムを使用せずにコードを構築およびテストできるクラウド内のマネージドビルドサービスです。 CodePipeline で CodeBuild を使用すると、ソースコードに変更があるたびにソフトウェアビルドを継続的に配信するために、パイプラインを介したリビジョンの実行を自動化できます。詳細については、「 CodePipeline で CodeBuild を使用してコードをテストし、ビルドを実行する」を参照してください。

Amazon ECS CodePipeline で を使用して、コンテナベースのアプリケーションをクラウドに継続的に配信する

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 CodePipeline で を使用すると、ソースイメージリポジトリに変更があるたびに、コンテナベースのアプリケーションの継続的なデプロイのためにパイプラインを介してリビジョンの実行を自動化できます。詳細については、「チュートリアル: を使用した継続的デプロイ CodePipeline」を参照してください。

Elastic Beanstalk CodePipeline で を使用してウェブアプリケーションをクラウドに継続的に配信する

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。Elastic Beanstalk CodePipeline で を使用すると、ウェブアプリケーションをアプリケーション環境に継続的にデプロイできます。 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。

Lambda ベースおよびサーバーレスアプリケーションの継続的な配信 AWS Lambda に CodePipeline を使用する

「サーバーレスアプリケーションのデプロイ」で説明されているように、 AWS Lambda を使用して AWS Lambda 関数を CodePipeline 呼び出すことができます。 https://docs.aws.amazon.com/lambda/latest/dg/automating-deployment.html AWS Lambda および を使用して、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 AWS CodeStar を作成することもできます。

AWS CloudFormation テンプレート CodePipeline で を使用してクラウドに継続的に配信する

AWS CloudFormation で を使用して、 CodePipeline 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、「 を使用した継続的デリバリー CodePipeline」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。

リソースのタグ付け

タグは、ユーザーまたは が AWS リソース AWS に割り当てるカスタム属性ラベルです。各 AWS タグには 2 つの部分があります。

- タグキー (CostCenter、Environment、Project、Secret など)。タグキーでは、大文字と小文字が区別されます。
- タグ値と呼ばれるオプションのフィールド (111122223333、Production、チーム名など)。タグ値を省略すると、空の文字列を使用した場合と同じになります。タグキーと同様に、タグ値では大文字と小文字が区別されます。

これらを合わせて、キーと値のペアと呼ばれます。

タグは、 AWS リソースの識別と整理に役立ちます。多くの はタグ付け AWS のサービス をサポートしているため、異なるサービスのリソースに同じタグを割り当てて、リソースが関連していることを示すことができます。例えば、Amazon S3 ソースバケットに割り当てたのと同じタグをパイプラインに割り当てることができます。

タグの使用方法のヒントについては、AWS の回答ブログのAWS タグ付け戦略記事を参照してください。

では、次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を付ける CodePipeline
- でカスタムアクションにタグを付ける CodePipeline

AWS CLI、 CodePipeline APIsを使用して、次のことができます。 AWS SDKs

- パイプライン、カスタムアクション、ウェブフックを作成するときに、タグを追加します。
- パイプライン、カスタムアクション、ウェブフックのタグを追加、管理、削除します。

また、コンソールを使用してパイプラインのタグを追加、管理、削除することもできます。

タグを使用してリソースを識別、整理、追跡するだけでなく、IAM ポリシーのタグを使ってリソースを表示および操作できるユーザーを制御することもできます。タグベースのアクセスポリシーの例については、「タグを使用して リソースへのアクセス CodePipelineを制御する」を参照してください。

Amazon Virtual Private Cloud CodePipeline で を使用する

AWS CodePipeline は、 を搭載した Amazon Virtual Private Cloud (Amazon VPC) エンドポイントをサポートするようになりましたAWS PrivateLink。つまり、VPC 内のプライベートエンドポイント 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保持できます。

Amazon VPC は AWS のサービス 、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、 を利用しています。これは AWS PrivateLink、Elastic Network Interface とプライベート IP アドレス AWS のサービス を使用した 間のプライベート通信を容易にする AWS テクノロジーです。VPC を 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 CodePipeline。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。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、または VPN 接続を必要と CodePipeline せずに、信頼性が高くスケーラブルな 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。

可用性

CodePipeline は現在、次の で 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 の VPC エンドポイントを作成する

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) リージョンの など CodePipeline、 で AWS リージョン サポートされている `us-east-2`のリージョン識別子です。詳細については、『Amazon VPC ユーザーガイド』の「インターフェイスエンドポイントの作成」を参照してください。

エンドポイントには、 AWSにサインインしたときに指定したリージョンが事前に設定されています。別のリージョンにサインインすると、VPC エンドポイントは新しいリージョンに更新されます。

> **Note**
>
> VPC サポートを提供し CodePipeline、 など と統合 AWS のサービス する他の は CodeCommit、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 の間のトラフィックを VPC サブネット範囲に制限 CodeCommit することはできません。

VPC 設定のトラブルシューティング

VPC の問題をトラブルシューティングする場合、インターネット接続エラーメッセージに表示される情報を、問題の特定、診断、対処のために使用します。

1. インターネットゲートウェイが VPC にアタッチされていることを確認します。
2. パブリックサブネットのルートテーブルがインターネットゲートウェイを参照していることを確認します。
3. ネットワーク ACL がトラフィックのフローを許可していることを確認します。
4. セキュリティグループがトラフィックのフローを許可していることを確認します。
5. プライベートサブネットのルートテーブルが仮想バーチャルゲートウェイを参照していることを確認します。
6. が使用するサービスロールに適切なアクセス許可 CodePipeline があることを確認してください。例えば、 CodePipeline に Amazon VPC の操作に必要な Amazon EC2 アクセス許可がない場合、「予期しない EC2 エラー: 」というエラーが表示されることがあります UnauthorizedOperation。

でのパイプラインの使用 CodePipeline

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**
>
> ビルド、デプロイ、テスト、または呼び出しの各ステージを追加すると、 で提供されるデフォルトのオプションに加えて CodePipeline、パイプラインで使用する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、「でカスタムアクションを作成して追加する CodePipeline」を参照してください。

パイプラインを作成する前に、まずはの開始方法 CodePipelineのステップを完了する必要があります。

パイプラインの詳細については、CodePipeline の概念「」、CodePipeline チュートリアル「」、および「」を参照してください。 AWS CLI を使用してパイプラインを作成する場合は、「」を参照してくださいCodePipeline パイプライン構造リファレンス。パイプラインのリストを表示するには、でパイプラインと詳細を表示する CodePipeline を参照してください。

トピック

- でパイプラインを開始する CodePipeline
- でパイプラインの実行を停止する CodePipeline
- でパイプラインを作成する 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
- でパイプラインを削除する CodePipeline
- 別の AWS アカウントのリソース CodePipeline を使用するパイプラインを に作成する
- 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
- でパイプラインにタグを付ける CodePipeline

- 通知ルールの作成

でパイプラインを開始する CodePipeline

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 です CodeCommit。このタイプは、パイプラインを開始するスケジュールにも使用されます。スケジュールに基づいたパイプラインの開始 を参照してください。
- ソース変更のポーリング: このカテゴリは実行履歴で `PollForSourceChanges` トリガータイプを表します。ポーリングによりソースリポジトリ内のソースコミットとブランチに対する変更が検出されると、パイプラインが開始されます。このトリガータイプは推奨されないため、自動変更検出を使用するように移行する必要があります。このトリガータイプを使用するソースアクションプロバイダーは S3 と です CodeCommit。
- 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、セルフ GitLab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 (Bitbucket Cloud、 GitHub Enterprise Server GitHub、.com) 用に設定された接続 GitLab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 の`RollbackStage`トリガータイプを表します。コンソールまたは を使用して AWS CLI 、ステージを手動または自動でロールバックできます。詳細については、「ステージロールバックの設定」を参照してください。

自動変更検出トリガータイプを使用するソースアクションをパイプラインに追加すると、そのアクションは追加のリソースと連携して動作します。各ソースアクションの作成については、変更検出のためのこれらの追加のリソースのため、別のセクションで詳しく説明します。自動変更検出に必要な各ソースプロバイダーと変更検出方法の詳細については、「 ソースアクションと変更検出方法」を参照してください。

トピック

- ソースアクションと変更検出方法
- 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

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、設定された変更検出リソース (EventBridge ルール) または オプションを使用して、ソースの変更についてリポジトリをポーリングする必要があります。Bitbucket GitHubまたは GitHub Enterprise Server ソースアクションを使用

する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にする必要はありません。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
Amazon S3	このソースアクションは、追加のリソースを使用します。CLI または を使用してこのアクション CloudFormation 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。	でパイプラインを作成する CodePipeline および EventBridge を使用した Amazon S3 ソースアクション AWS CloudTrail を参照してください。
Bitbucket Cloud	このソースアクションは、コネクションリソースを使用します。	「Bitbucket Cloud への接続」を参照してください。
AWS CodeCommit	Amazon EventBridge （推奨）。これは、コンソールで作成または編集された CodeCommit ソースを持つパイプラインのデフォルトです。	でパイプラインを作成する CodePipeline および CodeCommit ソースアクションと EventBridge を参照してください。
Amazon ECR	Amazon EventBridge。これは、コンソールで作成または編集した Amazon ECR ソースを含むパイプライン用にウィザードで作成されます。	「でパイプラインを作成する CodePipeline」および「 Amazon ECR ソースアクションと EventBridge リソース」を参照してください。
GitHub または GitHub エンタープライズクラウド	このソースアクションは、コネクションリソースを使用します。	「GitHub 接続」を参照してください。
GitHub エンタープライズサーバー	このソースアクションは、コネクションリソースとホストリソースを使用します。	GitHub Enterprise Server 接続 を参照してください。

ソース	その他のリソースを使用しますか？	ステップ
GitLab.com	このソースアクションは、コネクションリソースを使用します。	「GitLab.com 接続」を参照してください。
GitLab セルフマネージド型	このソースアクションは、コネクションリソースとホストリソースを使用します。	GitLab セルフマネージド型 の接続を参照してください。

ポーリングを使用するパイプラインがある場合は、推奨される検出方法を使用するように更新できます。詳細については、「ポーリングパイプラインを推奨される変更検出方法に更新する」を参照してください。

接続を使用するソースアクションの変更検出をオフにするには、CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用 を参照してください。

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。 CodePipeline コンソールまたは AWS CLI および start-pipeline-execution コマンドを使用して、パイプラインを介して最新のリビジョンを手動で再実行できます。

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Name] で、開始するパイプラインの名前を選択します。

3. パイプラインの詳細ページで、[Release change] を選択します。パイプラインがパラメータ (パイプライン変数) を渡すように設定されている場合、[変更のリリース] を選択すると、[変更のリリース] ウィンドウが開きます。[パイプライン変数] で、パイプラインレベルの変数のフィールドに、このパイプライン実行に渡す値を入力します。詳細については、「変数」を参照してください。

   これにより、ソースアクションで指定した各ソース場所における最新のリビジョンがパイプラインにより開始されます。

パイプラインを手動で開始する (CLI)

パイプラインを手動で開始し、アーティファクトの最新バージョンをパイプラインにより実行するには

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、AWS CLI を使用して start-pipeline-execution コマンドを実行し、開始するパイプラインの名前を指定します。例えば、 という名前のパイプラインを使用して最後の変更の実行を開始するには、次のようにします*MyFirstPipeline*。

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

   パイプラインレベルの変数が設定されているパイプラインを開始するには、start-pipeline-execution コマンドにオプションの --variables 引数を使用してパイプラインを開始し、実行で使用される変数を追加します。例えば、値が 1 の変数 var1 を追加するには、以下のコマンドを使用します。

   ```
   aws codepipeline start-pipeline-execution --name MyFirstPipeline --variables
    name=var1,value=1
   ```

2. 成功したかどうかを確認するには、返されたオブジェクトを表示します。このコマンドでは、以下のような 実行 ID が返ります。

   ```
   {
       "pipelineExecutionId": "c53dbd42-This-Is-An-Example"
   }
   ```

> **Note**
>
> パイプラインを開始したら、 CodePipeline コンソールで、または get-pipeline-state コマンドを実行して、パイプラインの進行状況をモニタリングできます。詳細については、「パイプラインを表示する (コンソール)」および「パイプラインの詳細と履歴を表示する (CLI)」を参照してください。

スケジュールに基づいたパイプラインの開始

でルールを設定 EventBridge して、スケジュールに従ってパイプラインを開始できます。

パイプラインの開始をスケジュールする 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. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. ナビゲーションペインで [Rules (ルール)] を選択します。
3. [ルールの作成] を選択してから、[ルールの詳細] で [スケジュール] を選択します。
4. 一定間隔または式を使用してスケジュールを設定します。詳細については、「ルールのスケジュール式」を参照してください。
5. ターゲット で、 を選択しますCodePipeline。
6. このスケジュールのパイプラインを実行するには、パイプライン ARN を入力します。

> **Note**
>
> パイプライン ARN は、コンソールの [設定] に表示されます。パイプラインの ARN とサービスロール ARN (コンソール) を表示します。 を参照してください。

7. 次のいずれかを選択して、 EventBridge ルールに関連付けられたターゲットを呼び出す EventBridge アクセス許可を付与する 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 です CodePipeline)。

   - この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始するアクセス許可を付与 EventBridgeするサービスロールを作成します。

- 「既存のロールを使用」を選択して、パイプラインの実行を開始する EventBridge アクセス許可を付与するサービスロールを入力します。

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。

パイプラインの開始をスケジュールする 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 CodePipeline に関連付けられた で作成するすべてのパイプラインで一意である必要があります。
- ルールのためのスケジュール式。

スケジュールをイベントソースとする EventBridge ルールを作成するには

1. put-rule コマンドを呼び出し、`--name` と `--schedule-expression` パラメータを含めます。

   例:

   次のサンプルコマンドでは--schedule-expression、 を使用して、スケジュール EventBridge に基づいてフィルタリング`MyRule2`する というルールを作成します。

   ```
   aws events put-rule --schedule-expression 'cron(15 10 ? * 6L 2002-2005)' --name
    MyRule2
   ```

2. ルールの呼び出し EventBridge に使用するアクセス許可 CodePipeline を に付与します。詳細については、「Amazon のリソースベースのポリシーの使用 EventBridge」を参照してください。

   a. 次のサンプルを使用して、 がサービスロールを引き受けることを許可する EventBridge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は、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。

のソースリビジョンには 4 つのタイプがありますrevisionType。

- COMMIT_ID
- IMAGE_DIGEST
- S3_OBJECT_VERSION_ID
- S3_OBJECT_OBJECT_KEY

> **Note**
>
> COMMIT_ID および 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、すべてのブランチのリポジトリ内のすべてのコンテンツに適用されます。

> **Note**
>
> S3_OBJECT_VERSION_ID および S3_OBJECT_KEYタイプのソースリビジョンでは、いずれかのタイプを個別に使用することも、一緒に使用してソースを特定の ObjectKey および VersionID で上書きすることもできます。

トピック

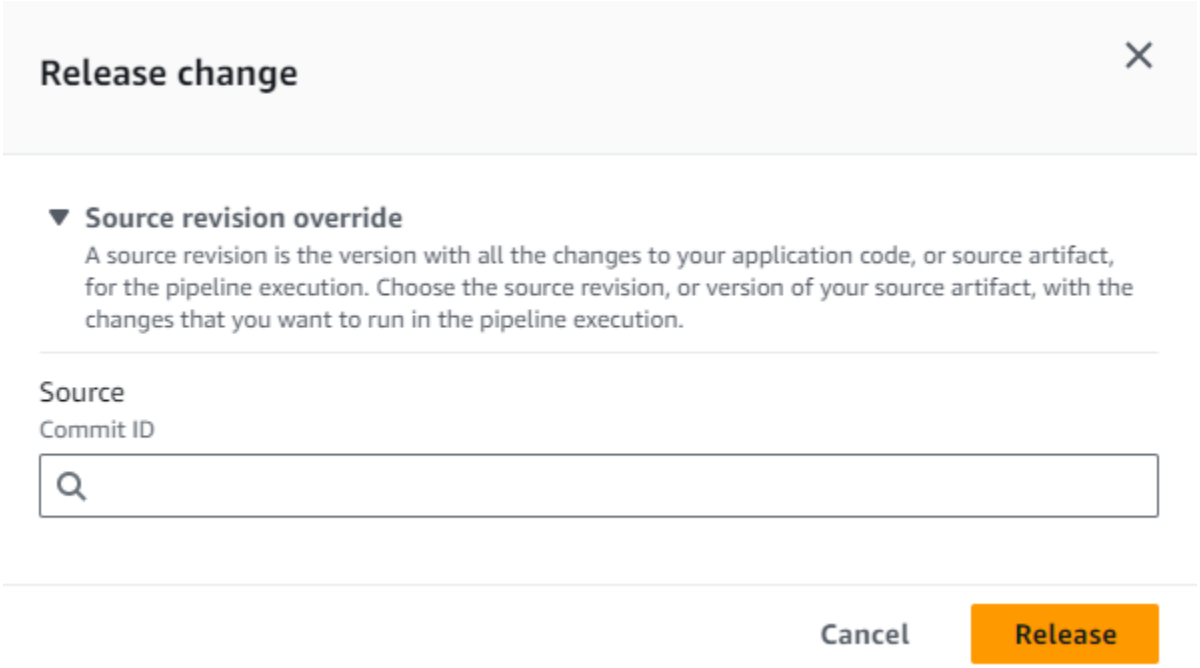
- ソースリビジョンオーバーライドでパイプラインを開始する (コンソール)
- ソースリビジョンオーバーライドでパイプラインを開始する (CLI)

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

2. [Name] で、開始するパイプラインの名前を選択します。
3. パイプラインの詳細ページで、[Release change] を選択します。[リリース変更: を選択すると、[リリース変更] ウィンドウが開きます。[ソースリビジョンオーバーライド: では、矢印を選択してフィールドを展開します。[ソース] に、ソースリビジョン ID を入力します。例えば、パイプラインに CodeCommit ソースがある場合は、使用するフィールドからコミット ID を選択します。



ソースリビジョンオーバーライドでパイプラインを開始する (CLI)

パイプラインを手動で開始し、パイプラインを通じてアーティファクトの指定したソースリビジョン ID を実行するには

1. ターミナル (Linux、macOS、または Unix) またはコマンドプロンプト (Windows) を開き、AWS CLI を使用して start-pipeline-execution コマンドを実行し、開始するパイプラインの名前を指定します。また、--source-revisions 引数を使用して、ソースリビジョン ID を指定することもできます。ソースリビジョンは、actionName、revisionType、oおよび revisionValue で構成されます。有効な れヴぃしお n Type の値は `COMMIT_ID | IMAGE_DIGEST | S3_OBJECT_VERSION_ID | S3_OBJECT_KEY` です。

   次の例では、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、パイプラインの進行状況をモニタリングできます。詳細については、「パイプラインを表示する (コンソール)」および「パイプラインの詳細と履歴を表示する (CLI)」を参照してください。

でパイプラインの実行を停止する CodePipeline

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。

- すべての進行中のアクションが完了するまで (つまり、アクションのステータスが Succeededまたは になるまで) 、実行を停止 AWS CodePipeline して待機しますFailed。このオプションは、進行中のアクションを保持します。進行中のアクションが完了するまで、実行は Stopping 状態になります。その後、実行は Stopped 状態になります。アクションが完了すると、ステージがロック解除します。

  [Stop and wait (停止して待機)] を選択し、実行がまだ Stopping 状態にある間に待機するのを止める場合は、[中止] を選択できます。

- Stop and abandon : 進行中のアクションが完了するのを待たずに実行を AWS CodePipeline 停止します。進行中のアクションが中止される間、実行は非常に短い時間 Stopping 状態になります。実行が停止すると、アクションの実行が Abandoned 状態である間、パイプライン実行は Stopped 状態になります。ステージのロックが解除されます。

  Stopped 状態のパイプライン実行の場合、実行が停止されたステージ内のアクションを再試行できます。

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

トピック

パイプライン実行を停止する (コンソール)

コンソールを使用してパイプラインの実行を停止できます。実行を選択し、パイプラインの実行を停止する方法を選択します。

> ⓘ Note
>
> インバウンド実行であるパイプラインの実行を停止することもできます。インバウンド実行を停止する方法については、インバウンド実行を停止します (コンソール)。 を参照してください。

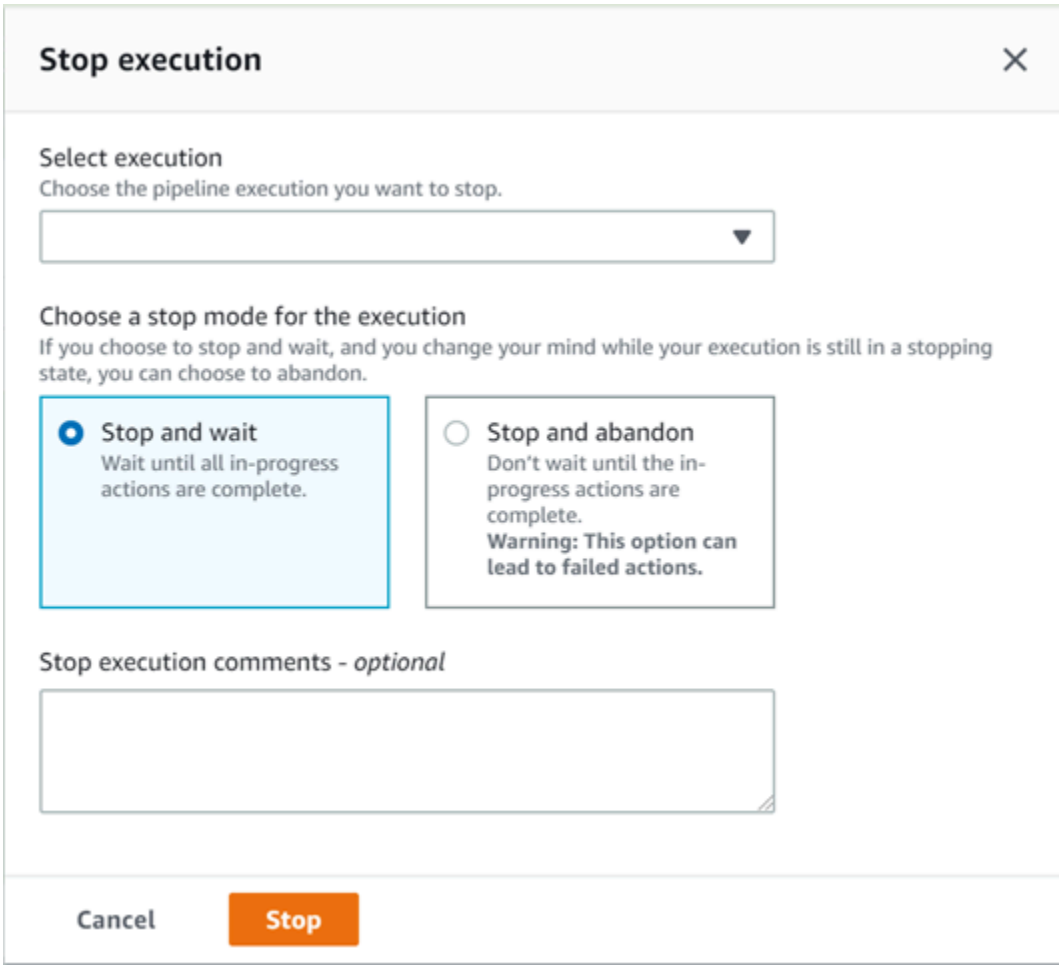
1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. 次のいずれかを行います。

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

- [名前] で、停止する実行を含むパイプラインの名前を選択します。パイプラインの詳細ページで、[Stop execution (実行の停止)] を選択します。
- [View history (履歴の表示)] を選択します。履歴ページで、[Stop execution (実行を停止)] を選択します。

3. [Stop execution (実行を停止)] ページの [Select execution (実行の選択)] で、停止する実行を選択します。

> **Note**
>
> 実行は、進行中の場合にのみ表示されます。すでに完了している実行は表示されません。



4. [Select an action to apply to execution (実行に適用するアクションの選択)] で、次のいずれかを選択します。

   - 進行中のすべてのアクションが完了するまで実行が停止しないようにするには、[Stop and wait (停止して待機)] を選択します。

     > **Note**
     >
     > 実行がすでに [停止] 状態になっている場合、[Stop and wait (停止して待機)] を選択することはできませんが、[Stop and abandon (停止して中止)] は選択できます。

   - 進行中のアクションが完了するのを待たずに停止するには、[Stop and abandon (停止して中止)] を選択します。

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

5. （オプション）コメントを入力します。これらのコメントは、実行ステータスとともに、実行の履歴ページに表示されます。
6. [Stop] (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. インバウンド実行を停止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、以下のパラメータを指定して stop-pipeline-execution コマンドを使用します。

- 実行 ID (必須)
- コメント (オプション)
- パイプライン名 (必須)
- 中止フラグ (オプション、デフォルトは false)

コマンド形式:

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --pipeline-
execution-id Execution_ID [--abandon | --no-abandon] [--reason STOP_EXECUTION_REASON]
```

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。
2. パイプライン実行を停止するには、次のいずれかを選択します。

   - 進行中のすべてのアクションが完了するまで実行が停止しないようにするには、[Stop and wait (停止して待機)] を選択します。これを行うには、no-abandon パラメータを含めます。パラメータを指定しない場合、コマンドのデフォルトは [Stop and wait (停止して待機)] になります。を使用して stop-pipeline-execution コマンド AWS CLI を実行し、パイプラインの名前と実行 ID を指定します。例えば、stop and wait オプションを指定*MyFirstPipeline*して という名前のパイプラインを停止するには、次のようにします。

     ```
     aws codepipeline stop-pipeline-execution --pipeline-name MyFirstPipeline --
     pipeline-execution-id d-EXAMPLE --no-abandon
     ```

     例えば、 という名前のパイプラインを停止するには*MyFirstPipeline*、デフォルトで stop and wait オプションに、コメントを含めることを選択します。

     ```
     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。を使用して stop-pipeline-execution コマンド AWS CLI を実行し、パイプラインの名前と実行 ID を指定します。

  例えば、 という名前のパイプラインを停止し*MyFirstPipeline*、中止オプションを指定して、コメントを含めることを選択するには、次のようにします。

```
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 stop-pipeline-execution コマンドを使用します。

- インバウンドの実行 ID (必須)
- コメント (オプション)
- パイプライン名 (必須)
- 中止フラグ (オプション、デフォルトは false)

コマンド形式:

```
aws codepipeline stop-pipeline-execution --pipeline-name Pipeline_Name --
pipeline-execution-id Inbound_Execution_ID [--abandon | --no-abandon] [--
reason STOP_EXECUTION_REASON]
```

上記の手順に従って、コマンドを入力し、停止方法を指定します。

get-pipeline-state コマンドを使用して、インバウンド実行のステータスを表示します。

でパイプラインを作成する CodePipeline

AWS CodePipeline コンソールまたは を使用してパイプライン AWS CLI 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。

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。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、「でクロスリージョンアクションを追加する CodePipeline」を参照してください。

Amazon ECS をデプロイプロバイダとして使用して、コンテナベースのアプリケーションを構築およびデプロイするパイプラインを作成することもできます。Amazon ECS でコンテナベースのアプリケーションをデプロイするパイプラインを作成する前に、イメージ定義ファイルのリファレンスの説明に従ってイメージ定義ファイルを作成する必要があります。

CodePipeline は、ソースコードの変更がプッシュされたときに、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 を使用して、 CodeCommit ソースリポジトリとブランチ、または S3 ソースバケットの変更を検出します。

> **Note**
>
> コンソールを使用してパイプラインを作成または編集すると、変更検出リソースが作成されます。 AWS CLI を使用してパイプラインを作成する場合は、追加のリソースを自分で作成する必要があります。詳細については、「 CodeCommit ソースアクションと EventBridge」を参照してください。

トピック

パイプラインを作成する (コンソール)

コンソールでパイプラインを作成するには、ソースファイルの場所と、アクションに使用するプロバイダに関する情報を提供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と、 はステージの名前 (ソース、ビルド、ステージング) CodePipeline を作成します。これらの名前は変更できません。後で追加するステージには、より具体的な名前 (BuildToGamma や など DeployToProd) を使用できます。

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Welcome (ようこそ)] ページで、[Create pipeline (パイプラインの作成)] を選択します。

   を初めて使用する場合は CodePipeline、「 の開始方法」を選択します。

3. [Step 1: Choose pipeline settings (ステップ 1: パイプラインの設定の選択)] 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、「のクォータ AWS CodePipeline」を参照してください。

4. [パイプラインのタイプ] で、次のいずれかのオプションを選択します。パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。

   - V1 タイプのパイプラインは、標準のパイプライン、ステージ、アクションレベルのパラメータを含む JSON 構造になっています。
   - V2 タイプのパイプラインは、V1 タイプと同じ構造で、Git タグやパイプラインレベルの変数に対するトリガーなど、追加のパラメータがサポートされています。

5. [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 が IAM で新しいサービスロールを作成できるようにします。
   - 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時期によっては、追加の をサポートするようにアクセス許可を更新する必要がある場合があります AWS のサービス。詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

   サービスロールとそのポリシーステートメントの詳細については、「CodePipeline サービスロールを管理する」を参照してください。

6. (オプション) [変数] で [変数を追加] を選択し、パイプラインレベルの変数を追加します。

パイプラインレベルの変数の詳細については、「変数」を参照してください。パイプラインの実行時に渡されるパイプラインレベルの変数のチュートリアルについては、「チュートリアル: パイプラインレベルの変数を使用する」を参照してください。

> **Note**
>
> パイプラインレベルの変数の追加はオプションですが、パイプラインレベルの変数に値が設定されていない場合、パイプラインの実行は失敗します。

7. (オプション) [詳細設定] を展開します。
8. [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、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があり、アクションを実行している AWS リージョンごとに 1 つのアーティファクトバケットが必要です。
> 詳細については、「入力および出力アーティファクト」および「CodePipeline パイプライン構造リファレンス」を参照してください。

9. [暗号化キー] で、次のいずれかの操作を行います。

   a. CodePipeline 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。

10. [次へ] をクリックします。

ステップ 2: ソースステージを作成する

- [Step 2: Add source stage (ステップ 2: ソースステージの追加)] ページの [ソースプロバイダ] 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てから [次のステップ] を選択します。

  - Bitbucket Cloud、 GitHub (バージョン 2)、 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 ソースアクションの接続を作成または管理する方法については、「」を参照してくださいGitHub 接続。
    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してパイプラインを開始するかを選択します。トリガーの操作の詳細については、「」を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。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デフォルトの を選択します。アクションは GitHub リポジトリからファイルにアクセスし、アーティファクトをパイプラインアーティファクトストアの ZIP ファイルに保存します。
       - 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、「」に示すように、 CodeBuild プロジェクトサービスロールのアクセス許可を更新する必要がありますトラブルシューティング CodePipeline。Full clone オプションを使用する方法を示すチュートリアルについては、チュートリアル: GitHub パイプラインソースでフルクローンを使用する を参照してください。

- Amazon S3 については：

  1. [Amazon S3 の場所] で、S3 バケットの名前と、バージョニングが有効になっているバケット内のオブジェクトへのパスを指定します。バケット名とパスの形式は以下のようになります。

     ```
     s3://bucketName/folderName/objectName
     ```

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

  2. S3 ソースバケットを選択すると、 はこのパイプライン用に作成される Amazon CloudWatch Events ルールと AWS CloudTrail 証跡 CodePipeline 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 を使用して AWS CloudTrail 、新しいパイプラインの変更を検出できます。[次へ] をクリックします。

- AWS CodeCommit の場合:

  - リポジトリ名 で、パイプラインの送信元として使用する 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デフォルトの 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、アーティファクトをパイプラインアーティファクトストアの ZIP ファイルに保存します。
    - 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、「」に示すように、 CodeBuild サービスロールにアクセスcodecommit:GitPull許可を追加する必要がありますソースアクションの

CodeBuild GitClone CodeCommitアクセス許可を追加する。また、「」に示すように、CodePipeline サービスロールにアクセスcodecommit:GetRepository許可を追加する必要がありますCodePipeline サービスロールにアクセス許可を追加する。[完全クローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: GitHub パイプラインソースでフルクローンを使用する を参照してください。

- CodeCommit リポジトリ名とブランチを選択すると、変更検出オプションに、このパイプライン用に作成される Amazon CloudWatch Events ルールを示すメッセージが表示されます。[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、次のステージで使用するイメージ名とリポジトリ URI 情報を含む MyAppなどの出力アーティファクトのデフォルトを選択します。

    Amazon ECR ソースステージを含むブルー/グリーンデプロイで CodeDeploy Amazon ECS の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、「」を参照してくださいチュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する。

  パイプラインに Amazon ECR のソースステージを含めると、変更をコミットしたときにソースアクションによって、出力アーティファクトとして imageDetail.json のファイルが生成されます。imageDetail.json ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

> **Note**
>
> オブジェクトとファイルタイプは、使用するデプロイシステムと互換性がある必要があります (Elastic Beanstalk や など CodeDeploy) 。サポートされているファイルのタイプは .zip、.tar、.tgz ファイルなどです。Elastic Beanstalk でサポートされているコンテナのタイプの詳細については、Customizing and Configuring Elastic Beanstalk Environments と Supported Platforms を参照してください。を使用してリビジョンをデプロイする方法の詳細については CodeDeploy、「アプリケーションリビジョンのアップロード」および「リビジョンの準備」を参照してください。

ステップ 3: 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。

- [Step 3: Add build stage (ステップ 3: ビルドステージの追加)] ページで、以下のいずれかの操作を行い、[次へ] を選択します。
  - デプロイステージを作成する予定の場合は、[Skip build stage (ビルドステージのスキップ)] を選択します。
  - [ビルドプロバイダー] から、ビルドサービスのカスタムアクションプロバイダーを選択し、そのプロバイダーの設定詳細を入力します。Jenkins をビルドプロバイダとして追加する方法の例については、「チュートリアル: 4 ステージのパイプラインを作成する」を参照してください。
  - [ビルドプロバイダ] から、[AWS CodeBuild] を選択します。

    リージョン で、リソースが存在する AWS リージョンを選択します。リージョンフィールドは、このアクションタイプとプロバイダータイプに対して AWS リソースが作成される場所を指定します。このフィールドは、アクションプロバイダーが AWS のサービスである場合にのみ表示されます。リージョンフィールドのデフォルトは、パイプラインと同じ AWS リージョンです。

    [プロジェクト名] で、ビルドプロジェクトを選択します。でビルドプロジェクトをすでに作成している場合は CodeBuild、それを選択します。または、 でビルドプロジェクトを作成し CodeBuild 、このタスクに戻ることもできます。「 CodeBuildユーザーガイド」の「 が使用するパイプラインを作成する CodeBuild」の手順に従います。

    環境変数 で、ビルドアクションに環境変数を追加するには CodeBuild、環境変数 を追加 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。変数タイプに Parameter を選択した場合、この値が AWS Systems Manager Parameter Store に既に保存されているパラメータの名前であることを確認します。

      > **Note**
      >
      > 機密値、特に AWS 認証情報を保存するために環境変数を使用することを強くお勧めします。 CodeBuild コンソールまたは AWS CLI を使用すると、環境変数がプ

レーンテキストで表示されます。機密の値の場合は、代わりに [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。

ステップ 4: 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。

- [Step 4: Add deploy stage (ステップ 4: デプロイステージの追加)] ページで、以下のいずれかの操作を行い、[次へ] を選択します。
  - 前のステップでビルドステージを作成した場合は、[Skip deploy stage (デプロイステージのスキップ)] を選択します。

    > **Note**
    > すでにビルドステージをスキップしている場合、このオプションは表示されません。

  - [デプロイプロバイダ] で、デプロイプロバイダ用に作成したカスタムアクションを選択します。

    リージョン では、クロスリージョンアクションでのみ、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のデフォルトは、パイプラインと同じ AWS リージョンです。
  - [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

アプリケーション名 で、既存の CodeDeploy アプリケーションの名前を入力または選択します。[デプロイグループ] に、アプリケーションのデプロイグループの名前を入力します。[次へ] をクリックします。コンソールでアプリケーション、デプロイグループ、またはその両方 CodeDeployを作成することもできます。

- AWS Elastic Beanstalk

  Application name で、既存の Elastic Beanstalk アプリケーションの名前を入力または選択します。[環境名] に、アプリケーションの環境を入力します。[次へ] をクリックします。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。

  の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。

  AWS OpsWorks レイヤーで実行するコードのソース CodePipeline として でシンプルなパイプラインを使用する方法 end-to-end の例については、「 CodePipeline で AWS OpsWorks Stacksを使用する」を参照してください。

- 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 を作成または更新を選択し、スタック名とテンプレートファイル名を入力し、 が引き受け AWS CloudFormation るロールの名前を選択します。必要に応じて、設定ファイルの名前を入力し、IAM 機能オプションを選択します。
  - アクションモードで、変更セット を作成または置換を選択し、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。

  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 CodePipeline、「 ユーザーガイド」の「 による継続的デリバリー CodePipelineAWS CloudFormation 」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、「Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル」を参照してください。

[Next] (次へ) を選択します。

> **Note**
>
> Amazon ECS クラスターが 2 つ以上のインスタンスで設定されていることを確認してください。Amazon ECS クラスターには、少なくとも 2 つのインスタンスが必要です。1 つはプライマリインスタンスとして維持し、もう 1 つは新しいデプロイに対応するために使用します。

パイプラインを使用したコンテナベースのアプリケーションのデプロイに関するチュートリアルについては、「チュートリアル: を使用した継続的デプロイ CodePipeline」を参照してください。

- Amazon ECS (Blue/Green)

  CodeDeploy アプリケーションとデプロイグループ、Amazon ECS タスク定義、 AppSpec ファイル情報を入力し、次へ を選択します。

  > **Note**
  >
  > Amazon ECS (Blue/Green) アクションには、デプロイアクションの入力アーティファクトとして imageDetail.json ファイルが必要です。Amazon ECR ソースアク

ションがこのファイルを作成するので、Amazon ECR ソースアクションを持つパイプラインは、`imageDetail.json` のファイルを提供する必要はありません。詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

を使用して Amazon ECS クラスターへのブルー/グリーンデプロイ用のパイプラインを作成するチュートリアルについては CodeDeploy、「」を参照してくださいチュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する。

- AWS Service Catalog

  コンソールのフィールドを使用して設定を指定する場合は [Enter deployment configuration (デプロイ設定の入力)] を選択します。あるいは、個別の設定ファイルがある場合は [設定ファイル] を選択します。製品と設定の情報を入力し、[次へ] を選択します。

  パイプラインを使用して製品の変更を Service Catalog にデプロイする方法のチュートリアルについては、「チュートリアル: Service Catalog にデプロイするパイプラインを作成する」を参照してください。

- Alexa Skills Kit

  [Alexa Skill ID (Alexa スキル ID)] に Alexa スキルのスキル ID を入力します。[クライアント ID] と [クライアントシークレット] に、Login with Amazon (LWA) セキュリティプロファイルを使用して生成された認証情報を入力します。[Refresh token (更新トークン)] に、ASK CLI コマンドを使用して生成した更新トークンを入力します。[次へ] をクリックします。

  パイプラインにより Alexa スキルをデプロイする方法、LWA 認証情報を生成する方法のチュートリアルについては、「チュートリアル: Amazon Alexa Skill をデプロイするパイプラインを作成する」を参照してください。

- Amazon S3

  [バケット] に、使用する S3 バケットの名前を入力します。デプロイステージへの入力アーティファクトが ZIP ファイルの場合は、[Extract file before deploy (デプロイ前にファイルを展開)] を選択します。[Extract file before deploy (デプロイ前にファイルを展開)] を選択した場合は、オプションで [Deployment path (デプロイパス)] に ZIP ファイルの解凍先を入力できます。選択しなかった場合は、[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。

> **Note**
>
> ほとんどのソースステージおよびビルドステージの出力アーティファクトは圧縮されます。Amazon S3 zip を除くすべてのパイプラインソースプロバイダーは、ソースファイルを Zip してから、次のアクションに入力アーティファクトとして提供します。

(オプション) Canned ACL で、canned ACL を入力し、Amazon S3 にデプロイされたオブジェクトに適用します。

> **Note**
>
> 既定 ACL を適用すると、オブジェクトに適用された既存の ACL が上書きされます。

(オプション) [キャッシュコントロール] で、バケットからオブジェクトをダウンロードするリクエストのキャッシュコントロールパラメータを指定します。有効な値のリストについては、HTTP オペレーションの `Cache-Control` ヘッダーフィールドを参照してください。[Cache control (キャッシュコントロール)] に複数の値を入力するには、各値の間にカンマを使用します。この例に示すように、各カンマの後にスペースを追加できます (オプション)。

上記のエントリ例は、CLI に次のように表示されます。

```
"CacheControl": "public, max-age=0, no-transform"
```

[次へ] をクリックします。

Amazon S3 デプロイアクションプロバイダとして を使用するパイプラインを作成する方法のチュートリアルについては、チュートリアル: Amazon S3 をデプロイプロバイダとして使用するパイプラインを作成する を参照してください。

ステップ 5: パイプラインの確認

- [ステップ 5: 確認] ページで、パイプラインの設定を確認したら、[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、[戻る] を選択してオプションを編集します。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、「でパイプラインと詳細を表示する CodePipeline」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、「でパイプラインを編集する CodePipeline」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`パラメータを指定して create-pipeline コマンドを実行します。

> **Important**
>
> AWS CLI を使用して、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。

パイプライン構造の詳細については、 CodePipeline API リファレンスのCodePipeline パイプライン構造リファレンス「」および「create-pipeline」を参照してください。

JSON ファイルを作成するには、同じパイプラインの JSON ファイルを使用して編集し、create-pipeline コマンドを実行するときにこのファイルを呼び出します。

前提条件:

CodePipeline で 用に作成したサービスロールの ARN が必要ですの開始方法 CodePipeline。create-pipeline コマンドを実行するときは、パイプライン JSON ファイルで CodePipeline サービスロール ARN を使用します。サービスロールの作成の詳細については、「 CodePipeline サービスロールを作成する」を参照してください。コンソールとは異なり、 で create-pipeline コマンドを実行すると、CodePipeline サービスロールを作成するオプション AWS CLI はありません。サービスロールがすでに存在している必要があります。

パイプラインのアーティファクトの保存先である S3 バケットの名前が必要です。このバケットはパイプラインと同じリージョンに存在する必要があります。バケット名は、create-pipeline コマンドの実行時にパイプライン JSON ファイルで使用します。コンソールとは異なり、 で create-pipeline

コマンドを実行して AWS CLI も、アーティファクトを保存するための S3 バケットは作成されません。バケットが存在している必要があります。

> **Note**
>
> get-pipeline コマンドを使用して、パイプラインの JSON 構造のコピーを取得し、プレーンテキストエディタで構造を変更することもできます。

JSON ファイルを作成するには

1. ターミナル (Linux、 macOS、あるいは Unix)またはコマンドプロンプト(Windows)で、ローカルディレクトリに新規のテキストファイルを作成します。
2. (オプション) パイプラインレベルでは 1 つ以上の変数を追加できます。この値は、CodePipeline アクションの設定で参照できます。パイプラインを作成するときに変数名と値を追加できます。また、コンソールでパイプラインを開始するときに値を割り当てることもできます。

   > **Note**
   >
   > パイプラインレベルの変数の追加はオプションですが、パイプラインレベルの変数に値が設定されていない場合、パイプラインの実行は失敗します。

   パイプラインレベルの変数は、パイプラインの実行時に解決されます。すべての変数は不変であり、値が割り当てられた後は更新できません。パイプラインレベルの変数のうち値が解決されたものは、実行ごとの履歴に表示されます。

   パイプライン構造の変数属性を使用して、パイプラインレベルの変数を指定します。以下の例では、変数 `Variable1` の値は `Value1` です。

   ```
   "variables": [
        {
            "name": "Timeout",
            "defaultValue": "1000",
            "description": "description"
        }
    ]
   ```

この構造をパイプラインの JSON に追加するか、次のステップのサンプルの JSON に追加します。名前空間の情報など変数の詳細については、「変数」を参照してください。

3. プレーンテキストエディタでファイルを開き、作成する構造を反映するように値を編集します。少なくとも、パイプラインの名前を変更する必要があります。また、変更するかを考慮する必要もあります。

   - このパイプラインのアーティファクトの保存先の S3 バケット。
   - コードのソースの場所。
   - デプロイのプロバイダ。
   - コードをデプロイする方法。
   - パイプラインのタグ。

   以下の 2 ステージのサンプルパイプライン構造では、変更を検討する必要があるパイプラインの値を強調表示しています。パイプラインには多くの場合、2 つ以上のステージが含まれます。

```
{
    "pipeline": {
        "roleArn": "arn:aws:iam::80398EXAMPLE::role/AWS-CodePipeline-Service",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "S3"
                        },
                        "outputArtifacts": [
                            {
                                "name": "MyApp"
                            }
                        ],
                        "configuration": {
                            "S3Bucket": "awscodepipeline-demobucket-example-date",
                            "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。でのリソースのタグ付けの詳細については CodePipeline、「」を参照してくださいリソースのタグ付け。

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、または S3 ソースバケットの変更を検出します。次のステップには、パイプラインにこれらのリソースを手動で作成する手順が含まれています。フラグを `false` に設定すると、定期的なチェックが無効になります。これは、推奨される変更検出メソッドを使用している場合には、必要ではありません。

4. パイプラインとは異なるリージョンでビルド、テスト、またはデプロイアクションを作成するには、パイプライン構造に以下を追加する必要があります。手順については、「でクロスリージョンアクションを追加する CodePipeline」を参照してください。

   - `Region` パラメータをアクションのパイプライン構造に追加します。
   - `artifactStores` パラメータを使用して、アクションがある各 AWS リージョンのアーティファクトバケットを指定します。

5. その構造で問題がなければ、**`pipeline.json`** のような名前でファイルを保存します。

パイプラインを作成するには

1. 先ほど作成した JSON ファイルを `--cli-input-json` パラメータで指定して、create-pipeline コマンドを実行します。

   JSON の の値*MySecondPipeline*として「」という名前を含む pipeline.json という名前の JSON ファイル*MySecondPipeline*を使用して という名前nameのパイプラインを作成するには、コマンドは次のようになります。

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

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

   このコマンドは、作成したパイプライン全体の構造を返します。

2. パイプラインを表示するには、 CodePipeline コンソールを開いてパイプラインのリストから選択するか、 get-pipeline-state コマンドを使用します。詳細については、「でパイプラインと詳細を表示する CodePipeline」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 リポジトリを持つパイプラインの場合、「」の説明に従って、 CloudWatch イベントルールを手動で作成する必要があります CodeCommit ソースの EventBridge ルールを作成する (CLI)。
   - 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 イベントルールと AWS CloudTrail 証跡を手動で作成する必要があります EventBridge を使用した Amazon S3 ソースアクション AWS CloudTrail。

Amazon ECR ソースアクションと EventBridge リソース

で Amazon ECR ソースアクションを追加するには CodePipeline、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 (パイプラインを作成する (コンソール)) またはアクションの編集ページを使用して、Amazon ECR 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 EventBridge ルールを作成します。
- CLI を使用して、ECR アクションのアクション設定を追加し、次のように追加のリソースを作成します。
  - ECR でのアクション設定の例を Amazon ECR で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを作成します。
  - 変更検出方法のデフォルトは、ソースをポーリングすることによってパイプラインを開始します。定期的なチェックを無効にして、手動で変更検出ルールを作成することをお勧めします。Amazon ECR ソースの EventBridge ルールを作成する (コンソール) 、Amazon ECR ソースの 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) 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 ソースの EventBridge ルールを作成する (コンソール)
- Amazon ECR ソースの 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 ソースの 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. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. ナビゲーションペインの [Events] (イベント) を選択します。
3. [ルールの作成] を選択し、[イベントソース] の [サービス名] から [Elastic Container Registry (ECR)] を選択します。
4. [イベントソース] で、[イベントパターン] を選択します。

   [編集] をクリックし、次のイベントパターン例を [イベントソース] のウィンドウに貼り付ける事で、 `eb-test` のリポジトリに `cli-testing` イメージタグが追加されます。

```
{
    "detail-type": [
        "ECR Image Action"
    ],
    "source": [
        "aws.ecr"
    ],
    "detail": {
        "action-type": [
            "PUSH"
        ],
        "image-tag": [
            "latest"
        ],
        "repository-name": [
            "eb-test"
        ],
        "result": [
            "SUCCESS"
        ]
    }
}
```

> **Note**
>
> Amazon ECR イベントでサポートされている完全なイベントパターンを確認するには、「Amazon ECR Events」および EventBridge「」または「Amazon Elastic Container Registry Events」を参照してください。

5. [保存] を選択します。

   [イベントパターンのプレビュー] ペインで、ルールを表示します。

6. ターゲット で、 を選択しますCodePipeline。

7. このルールによって開始するパイプラインの、パイプライン ARN を入力します。

> **Note**
>
> get-pipeline コマンドを実行した後、メタデータ出力でパイプライン ARN を見つけることができます。パイプライン ARN はこの形式で作成されます。
> arn:aws:codepipeline:*region*:*account*:pipeline-name
> パイプライン ARN の例:
> `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

8. EventBridge ルールに関連付けられたターゲットを呼び出す EventBridge アクセス許可を付与する 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 です CodePipeline) 。

   - この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始するアクセス許可を付与 EventBridgeするサービスロールを作成します。
   - 「既存のロールを使用」を選択して、パイプラインの実行を開始するアクセス EventBridge 許可を付与するサービスロールを入力します。

9. 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。

10. [詳細の設定] を選択します。

11. [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。

12. 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 ソースの 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 CodePipeline に関連付けられた で作成するすべてのパイプラインで一意である必要があります。
- ルールで使用するソースと詳細フィールドのイベントパターン。詳細については、「Amazon EventBridge とイベントパターン」を参照してください。

Amazon ECR をイベントソースとして、ターゲット CodePipeline として EventBridge ルールを作成するには

1. ルール EventBridge の呼び出しに使用する CodePipeline のアクセス許可を追加します。詳細については、「Amazon でのリソースベースのポリシーの使用 EventBridge」を参照してください。

   a. 次のサンプルを使用して、 がサービスロールを引き受け EventBridge ること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 このポリシーをロールに追加すると、 のアクセス許可が作成されます EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

この変更を行う理由 イメージプッシュを行う方法を指定するルールと、そのイベントによって開始されるパイプラインを指定するターゲットを持つイベントを作成する必要があります。

次のサンプルコマンドは、MyECRRepoRule というルールを作成します。

```
aws events put-rule --name "MyECRRepoRule" --event-pattern "{\"detail-type\":[\"ECR
 Image Action\"],\"source\":[\"aws.ecr\"],\"detail\":{\"action-type\":[\"PUSH\"],
\"image-tag\":[\"latest\"],\"repository-name\":[\"eb-test\"],\"result\":[\"SUCCESS
\"]}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

> **Note**
>
> Amazon ECR イベントでサポートされている完全なイベントパターンを確認するには、「Amazon ECR Events」および EventBridge「」または「Amazon Elastic Container Registry Events」を参照してください。

3. をターゲット CodePipeline として追加するには、 put-targets コマンドを呼び出し、次のパラメータを含めます。

   - `--rule` パラメータは、put-rule を使用して作成した `rule_name` で使用されます。
   - `--targets` パラメータは、ターゲットリストのリスト `Id` とターゲットパイプラインの `ARN` で使用されます。

   次のサンプルコマンドでは、`MyECRRepoRule` と呼ばれるルールに対して指定し、ターゲット `Id` は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。サンプルコマンドでは、パイプラインの例 `Arn` とルールの例 `RoleArn` も指定します。パイプラインは、リポジトリ内に変更が加えられると開始します。

```
aws events put-targets --rule MyECRRepoRule --targets
 Id=1,Arn=arn:aws:codepipeline:us-
west-2:80398EXAMPLE:TestPipeline,RoleArn=arn:aws:iam::80398EXAMPLE:role/Role-for-
MyRule
```

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

1. 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 EventBridge を開始するには、 が引き受けることができるロールを作成する必要があります。

YAML

```
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Sub arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}
```

JSON

```
{
    "EventRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
```

```
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "events.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "eb-pipeline-execution",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": "codepipeline:StartPipelineExecution",
                                "Resource": {
                                    "Fn::Sub": "arn:aws:codepipeline:
${AWS::Region}:${AWS::AccountId}:${AppPipeline}"
                                }
                            }
                        ]
                    }
                }
            ]
        }
    }
}
...
```

2. テンプレートの でResources、 AWS::Events::Rule 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。がリポジトリの状態の変更 EventBridge を検出すると、ルールはターゲットパイプラインStartPipelineExecutionで を呼び出します。

   この変更を行う理由 イメージプッシュを行う方法を指定するルールと、そのイベントによって開始されるパイプラインを指定するターゲットを持つイベントを作成する必要があります。

このスニペットでは、latest のタグが付いた eb-test というイメージを使用しています。

YAML

```
EventRule:
  Type: 'AWS::Events::Rule'
  Properties:
    EventPattern:
      detail:
        action-type: [PUSH]
        image-tag: [latest]
        repository-name: [eb-test]
        result: [SUCCESS]
      detail-type: [ECR Image Action]
      source: [aws.ecr]
    Targets:
      - Arn: !Sub arn:aws:codepipeline:${AWS::Region}:${AWS::AccountId}:
${AppPipeline}
        RoleArn: !GetAtt
          - EventRole
          - Arn
        Id: codepipeline-AppPipeline
```

JSON

```
{
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "detail": {
                    "action-type": [
                        "PUSH"
                    ],
                    "image-tag": [
                        "latest"
                    ],
                    "repository-name": [
                        "eb-test"
                    ],
                    "result": [
```

```
                        "SUCCESS"
                    ]
                },
                "detail-type": [
                    "ECR Image Action"
                ],
                "source": [
                    "aws.ecr"
                ]
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Sub": "arn:aws:codepipeline:${AWS::Region}:
${AWS::AccountId}:${AppPipeline}"
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
},
```

> **Note**
>
> Amazon ECR イベントでサポートされている完全なイベントパターンを確認するには、「Amazon ECR Events」および EventBridge「」または「Amazon Elastic Container Registry Events」を参照してください。

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. 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] を選択します。

EventBridge を使用した Amazon S3 ソースアクション AWS CloudTrail

で Amazon S3 ソースアクションを追加するには CodePipeline、次のいずれかを選択します。

- CodePipeline コンソールのパイプライン作成ウィザード (パイプラインを作成する (コンソール)) またはアクションの編集ページを使用して、S3 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 EventBridge ルールと CloudTrail 証跡を作成します。
- を使用して AWS CLI 、アクションのS3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。
  - S3 でのアクション設定の例を Amazon S3 ソースアクション で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを作成します。
  - 変更検出方法のデフォルトは、ソースをポーリングすることによってパイプラインを開始します。定期的なチェックを無効にして、手動で変更検出ルールと証跡を作成する必要があります。Amazon S3 ソースの EventBridge ルールを作成する (コンソール) 、Amazon S3 ソースの 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) 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口グに記録してフィルタリングするサービスです。証跡は、フィルタリングされたソースの変更を EventBridge ルールに送信します。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 ソースバケットにイベントを記録し、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。コンソールで証跡の作成または更新の一環として 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、「 の Amazon S3 バケットポリシー CloudTrail」を参照してください。

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

でルールを設定する前に EventBridge、証 AWS CloudTrail 跡を作成する必要があります。詳細については、「コンソールで証跡を作成する」を参照してください。

> **Important**
>
> コンソールを使用してパイプラインを作成または編集すると、 EventBridge ルールと AWS CloudTrail 証跡が自動的に作成されます。

追跡を作成するには

1. AWS CloudTrail コンソールを開きます。
2. ナビゲーションペインで、[Trails] (追跡) を選択します。
3. [追跡の作成]を選択します。[Trail name] に、証跡の名前を入力します。
4. [保存場所] でログファイルを保存するために使用するバケットを作成あるいは指定します。デフォルトでは、Amazon S3 バケットとオブジェクトはプライベートです。リソース所有者 (バケットを作成した AWS アカウント) のみがバケットとそのオブジェクトにアクセスできます。バケットには、バケット内のオブジェクトへのアクセス AWS CloudTrail を許可するリソースポリシーが必要です。
5. [証跡ログバケットとフォルダ] で、フォルダ内のすべてのオブジェクトのデータイベントを記録するための Amazon S3 バケットとオブジェクトプレフィックス (フォルダ名) を指定します。証跡ごとに、最大 250 個の Amazon S3 オブジェクトを追加できます。必要な暗号化キー情報を入力し、[次へ] を選択します。
6. [イベントタイプ] で [管理イベント] を選択します。
7. [管理イベント] で、[書き込み] を選択します。証跡では、指定したバケットとプレフィックスの Amazon S3 オブジェクトレベルの API アクティビティ (`GetObject` や `PutObject` など) が記録されます。
8. [Write (書き込み)] を選択します。
9. 証跡が適切であることを確認したら、[証跡の作成] を選択します。

Amazon S3 ソースで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. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。

2. 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。[次へ] をクリックします。
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。AWS 経由で API コール CloudTrail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、`my-bucket` というバケットと `my-files.zip` というオブジェクトキーのルールが作成されます。リソースを指定するために [編集] ウィンドウを使用する場合、ルールはカスタムイベントパターンを使用するように更新されます。

   以下に示しているのは、コピーして貼り付けるサンプルイベントパターンです。

```
{
    "source": [
        "aws.s3"
    ],
    "detail-type": [
        "AWS API Call via CloudTrail"
    ],
    "detail": {
        "eventSource": [
            "s3.amazonaws.com"
        ],
        "eventName": [
            "CopyObject",
            "CompleteMultipartUpload",
            "PutObject"
        ],
        "requestParameters": {
            "bucketName": [
                "my-bucket"
            ],
            "key": [
                "my-files.zip"
```

```
            ]
        }
    }
}
```

9. [次へ] をクリックします。
10. [ターゲットタイプ] で、[AWS サービス] を選択します。
11. 「ターゲットを選択」で、「」を選択しますCodePipeline。[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-2:80398例：MyFirstPipeline

12. EventBridge ルールに関連付けられたターゲットを呼び出す EventBridge アクセス許可を付与する IAM サービスロールを作成または指定するには (この場合、ターゲットは CodePipeline) 。
    - この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始するアクセス許可を付与 EventBridgeするサービスロールを作成します。
    - 「既存のロールを使用」を選択して、パイプラインの実行を開始するアクセス EventBridge 許可を付与するサービスロールを入力します。
13. [次へ] をクリックします。
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。

Amazon S3 ソースの EventBridge ルールを作成する (CLI)

AWS CloudTrail 証跡を作成してログ記録を有効にするには

を使用して証跡 AWS CLI を作成するには、 create-trail コマンドを呼び出し、以下を指定します。

- 証跡名。
- AWS CloudTrailにバケットポリシーをすでに適用しているバケットです。

詳細については、AWS「コマンドラインインターフェイス を使用した証跡の作成」を参照してください。

1. create-trail コマンドを呼び出し、--name および --s3-bucket-name パラメータを含めます。

   この変更を行う理由 これにより、 CloudTrailS3 ソースバケットに必要な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、myBucket 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 myBucket
   ```

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 このコマンドは、ソースバケットの CloudTrail ログ記録を開始し、イベントを に送信します EventBridge。

   例：

   次のコマンドでは、--name を使用して、my-trail という名前の証跡のログ記録を開始します。

   ```
   aws cloudtrail start-logging --name my-trail
   ```

3. put-event-selectors コマンドを呼び出し、--trail-name および --event-selectors パラメータを含めます。イベントセレクタを使用して、証跡がソースバケットのデータイベントをログに記録し、イベントを EventBridge ルールに送信するように指定します。

   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 myBucket/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:::myBucket/
   myFolder/file.zip"] }] }]'
   ```

Amazon S3 をイベントソースおよびターゲット CodePipeline とする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. ルールの呼び出し EventBridge に使用するアクセス許可 CodePipeline を に付与します。詳細については、「Amazon のリソースベースのポリシーの使用 EventBridge」を参照してください。

   a. 次のサンプルを使用して、 がサービスロールを引き受けること EventBridge 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 この信頼ポリシーをロールに追加すると、 のアクセス許可が作成されます EventBridge。

```
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
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

次のサンプルコマンドでは、MyS3SourceRule という名前のルールが作成されます。

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"AWS API Call via CloudTrail\"],\"detail\":
{\"eventSource\":[\"s3.amazonaws.com\"],\"eventName\":[\"CopyObject\",\"PutObject
\",\"CompleteMultipartUpload\"],\"requestParameters\":{\"bucketName\":[\"my-bucket
\"],\"key\":[\"my-key\"]}}}
 --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. をターゲット CodePipeline 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
```

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 パラメータのデフォルト設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

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

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. この例に示すように、プレーンテキストエディタでJSONファイルを開き、storage-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": "storage-bucket",
       "PollForSourceChanges": "false",
       "S3ObjectKey": "index.zip"
   },
   ```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除する必要があります。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

   例えば、構造から以下の行を削除します。

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

ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

Amazon S3 ソースの EventBridge ルールを作成する (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。

Amazon S3 をイベントソースおよびターゲット CodePipeline とする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. テンプレートの でResources、 AWS::IAM::Role 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 のアクセス許可を作成できます EventBridge。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": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
```

```
                    },
                    ":",
                    {
                      "Ref": "AppPipeline"
                    }
                  ]
                ]

...
```

2. 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": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. このスニペットを最初のテンプレートに追加して、クロススタック機能を有効にします。

YAML

```
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
  "Outputs" : {
    "SourceBucketARN" : {
      "Description" : "S3 bucket ARN that Cloudtrail will use",
      "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
      "Export" : {
        "Name" : "SourceBucketARN"
      }
    }

...
```

4. 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。
6. 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。
7. [実行] を選択します。

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 パラメータのデフォルト設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 PollForSourceChanges パラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

  JSON

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

```
      }
    ],
    "Configuration": {
      "S3Bucket": {
        "Ref": "SourceBucket"
      },
      "S3ObjectKey": {
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

Amazon S3 パイプラインのリソース用に 2 番目のテンプレートを作成するには CloudTrail

- 別のテンプレートの でResources、、AWS::S3::Bucket、および AWS::CloudTrail::Trail AWS CloudFormation リソースを使用してAWS::S3::BucketPolicy、 のシンプルなバケット定義と証跡を提供します CloudTrail。

  この変更を行う理由 現在の 1 アカウントあたり 5 つの証跡の制限を考慮すると、 CloudTrail 証跡は個別に作成および管理する必要があります。(「 の制限 AWS CloudTrail」を参照してください。) ただし、1 つの証跡に複数の Amazon S3 バケットを含めることができるため、いったん証跡を作成してから、必要に応じて他のパイプライン用に Amazon S3 バケットを追加できます。2 番目のサンプルテンプレートファイルに以下のコードを貼り付けます。

  YAML

```
################################################################################
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist
################################################################################

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
```

```
  AWSCloudTrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: AWSCloudTrailAclCheck
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          -
            Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
```

```
          ReadWriteType: WriteOnly
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:GetBucketAcl",
              "Resource": {
```

```
                "Fn::GetAtt": [
                  "AWSCloudTrailBucket",
                  "Arn"
                ]
              }
            },
            {
              "Sid": "AWSCloudTrailWrite",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:PutObject",
              "Resource": {
                "Fn::Join": [
                  "",
                  [
                    {
                      "Fn::GetAtt": [
                        "AWSCloudTrailBucket",
                        "Arn"
                      ]
                    },
                    "/AWSLogs/",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    "/*"
                  ]
                ]
              },
              "Condition": {
                "StringEquals": {
                  "s3:x-amz-acl": "bucket-owner-full-control"
                }
              }
            }
          ]
        }
      }
    },
    "AwsCloudTrail": {
```

```
      "DependsOn": [
        "AWSCloudTrailBucketPolicy"
      ],
      "Type": "AWS::CloudTrail::Trail",
      "Properties": {
        "S3BucketName": {
          "Ref": "AWSCloudTrailBucket"
        },
        "EventSelectors": [
          {
            "DataResources": [
              {
                "Type": "AWS::S3::Object",
                "Values": [
                  {
                    "Fn::Join": [
                      "",
                      [
                        {
                          "Fn::ImportValue": "SourceBucketARN"
                        },
                        "/",
                        {
                          "Ref": "SourceObjectKey"
                        }
                      ]
                    ]
                  }
                ]
              }
            ],
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": false
          }
        ],
        "IncludeGlobalServiceEvents": true,
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
  }
}
```

```
...
```

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許可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**
>
> 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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

で Bitbucket Cloud ソースアクションを追加するには CodePipeline、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、Bitbucket プロバイダーオプションを選択します。アクションを追加するには Bitbucket Cloud への接続を作成する (コンソール) を参照してください。このコンソールは、接続リソースの作成に役立ちます。

  > **Note**
  >
  > Bitbucket Cloudリポジトリへの接続を作成できます。Bitbucket サーバーなど、インストールされている Bitbucket プロバイダーのタイプはサポートされていません。

- CLI を使用して、次の通りに `CreateSourceConnection` でのアクションのアクション設定を `Bitbucket` プロバイダに追加します。
  - 接続リソースを作成するには、Bitbucket Cloud への接続を作成する (CLI) を参照してCLI で接続リソースを作成します。

- `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用 で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- Bitbucket Cloud など、サードパーティーのリポジトリプロバイダーでアカウントを作成しておく必要があります。
- Bitbucket Cloud リポジトリなどのサードパーティーのコードリポジトリを既に作成しておく必要があります。

> **Note**
>
> Bitbucket Cloud 接続は、接続の作成に使用された Bitbucket Cloud アカウントが所有するリポジトリへのアクセスのみを提供します。
> アプリケーションを Bitbucket Cloud ワークスペースにインストールする場合は、ワークスペースの管理アクセス許可が必要です。アクセス許可がないと、アプリケーションをインストールするオプションは表示されません。

トピック

Bitbucket Cloud への接続を作成する (コンソール)

CodePipeline コンソールを使用して Bitbucket リポジトリの接続アクションを追加するには、次の手順に従います。

> **Note**
>
> Bitbucket Cloudリポジトリへの接続を作成できます。Bitbucket サーバーなど、インストールされている Bitbucket プロバイダーのタイプはサポートされていません。

ステップ 1 ：パイプラインを作成または修正するには

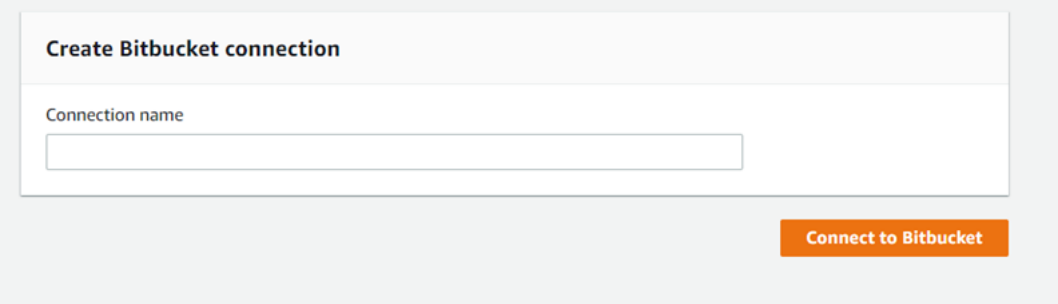
パイプラインを作成または修正するには

1. CodePipeline コンソールにサインインします。
2. 次のいずれかを選択します 。
   - パイプラインの作成を選択します。[パイプラインを作成する] の手順に従い最初の画面を完了し、[次] を選択します。 [ソース] ページの [ソースプロバイダー] の下の [Bitbucket] を選択します。
   - 既存のパイプラインを編集することを選択します。[Edit]、[Edit Stage] の順に選択します。ソースアクションを追加または編集するかを選択します。[アクションの編集] ページで、[Action name (アクション名)] に自分のアクション名を入力します。[アクションプロバイダ] で、[Bitbucket] を選択します。
3. 次のいずれかを行います。
   - [接続] でプロバイダへの接続をまだ作成していない場合は、[Bitbucket への接続] を選択します。ステップ 2 ：Bitbucket に接続に進みます。
   - [接続] でプロバイダへの接続を既に作成している場合は、その接続を選択します。ステップ 3： 接続のソースアクションを保存するに進みます。

ステップ 2: Bitbucket への接続を作成する

Bitbucket Cloud への接続を作成するには

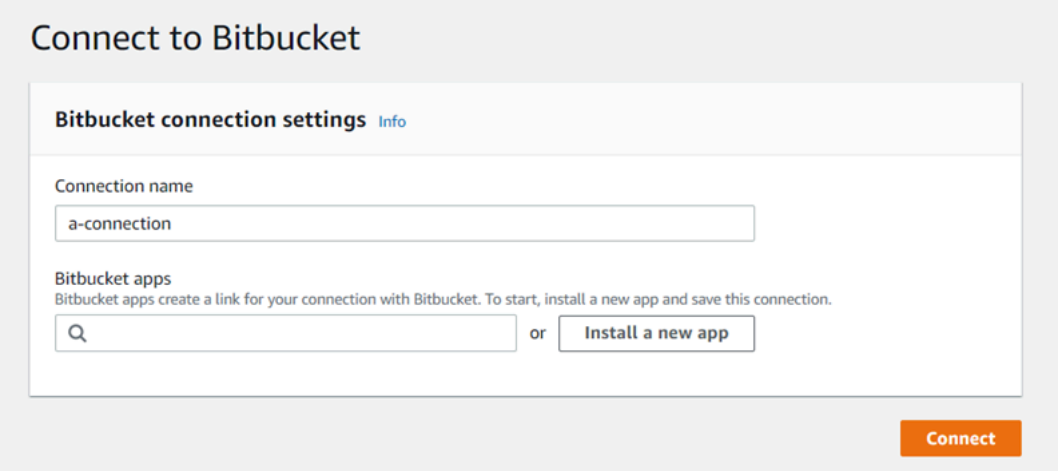
1. リポジトリの [Bitbucket への接続] 設定ページで、接続名を入力し、[Bitbucket への接続] を選択します。



[Bitbucket アプリ] フィールドが表示されます。

2. [Bitbucket apps] (Bitbucket アプリ) で、アプリのインストールを選択するか、アプリを作成するために [Install a new app] (新しいアプリをインストールする) を選択します。

> **Note**
>
> アプリケーションは、Bitbucket Cloud ワークスペースまたはアカウントごとに 1 回だけインストールします。Bitbucket アプリを既にインストールしている場合は、それを選択してステップ 4 に移動します。

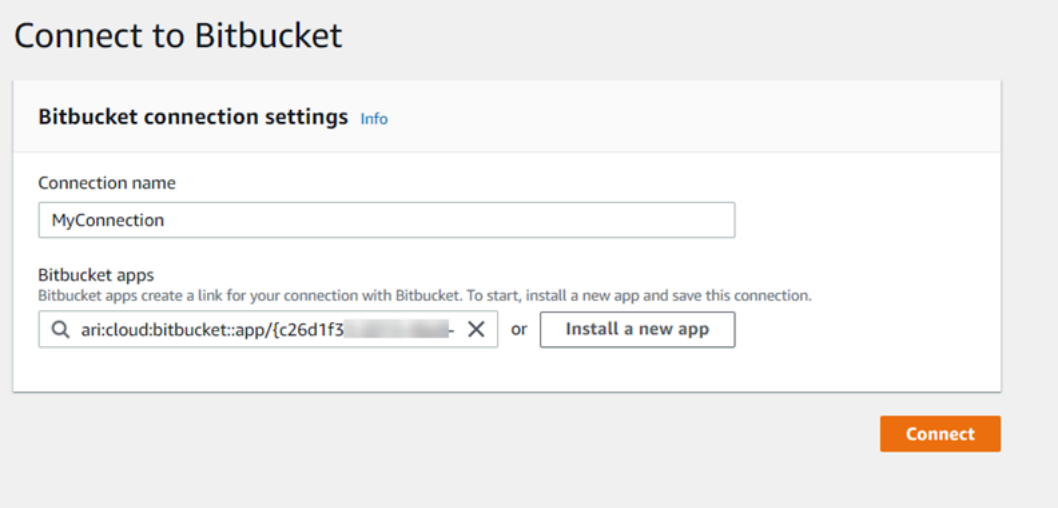


3. Bitbucket Cloud のログインページが表示されたら、認証情報を使用してログインし、続行を選択します。
4. アプリのインストールページで、 AWS CodeStar アプリが Bitbucket アカウントに接続しようとしていることを示すメッセージが表示されます。

   Bitbucket ワークスペースを使用している場合は、[Authorize for] (承認対象) オプションをそのワークスペースに変更します。管理者権限のあるワークスペースのみが表示されます。

[アクセス権の付与] を選択します。

5. Bitbucketアプリには、新規インストールの接続 ID が表示されます。[接続]を選択します。作成された接続が接続リストに表示されます。



ステップ 3: Bitbucket Cloud のソースアクションを保存する

ウィザードで次の手順を使用するか、[アクションを編集] ページで、ソースアクションを接続情報とともに保存します。

接続でソースアクションを完了して保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが CodeConnections アクションの場合にトリガーを追加できます。パイプラインのトリガー設定を設定し、オプションでトリガーでフィルタリングするには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。
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 デフォルトの 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、「」に示すように、 CodeBuild プロジェクトサービスロールのアクセス許可を更新する必要がありますBitbucket、、 GitHub Enterprise Server GitHub、または GitLab.com への接続の CodeBuild GitClone アクセス許可を追加する。

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ステータスになります。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

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameに --provider-typeと を指定します。この例では、サードパーティープロバイダー名は Bitbucket で、指定された接続名は MyConnection です。

```
aws codestar-connections create-connection --provider-type Bitbucket --connection-name MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. コンソールを使用して接続を完了します。詳細については、「Update a pending connection」を参照してください。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - トリガーでフィルタリングするようにパイプライントリガー設定を構成するには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。例えば、次の例では、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"
                                     ]
                                 }
                             }
     ```

```
                ]
            }
        }
    ]
```

CodeCommit ソースアクションと EventBridge

で CodeCommit ソースアクションを追加するには CodePipeline、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 (パイプラインを作成する (コンソール)) またはアクションの編集ページを使用して、CodeCommitプロバイダーオプションを選択します。コンソールは、 EventBridgeソースが変更されたときにパイプラインを開始する ルールを作成します。
- を使用して AWS CLI 、アクションのCodeCommit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。
  - CodeCommit でのアクション設定の例を CodeCommit 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 ソースの EventBridge ルールを作成する (コンソール)、CodeCommit ソースの 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 ソースの EventBridge ルールを作成する (コンソール)
- CodeCommit ソースの 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 ソースの EventBridge ルールを作成する (コンソール)

> **Important**
>
> コンソールを使用してパイプラインを作成または編集すると、 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. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. 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。[次へ] をクリックします。
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 で、フィルタリングするキーワード CodeCommit として を入力します。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、 CodeCommit という名前のブランチを持つMyTestRepoリポジトリのイベントウィンドウのサンプルイベントパターンですmain。

```
{
  "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、 を選択しますCodePipeline。
10. このルールによって開始するパイプラインの、パイプライン ARN を入力します。

> **Note**
>
> get-pipeline コマンドを実行した後、メタデータ出力でパイプライン ARN を見つけることができます。パイプライン ARN はこの形式で作成されます。
> arn:aws:codepipeline:*region*:*account*:pipeline-name
> パイプライン ARN の例:
> `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

11. EventBridge ルールに関連付けられたターゲットを呼び出す EventBridge アクセス許可を付与する IAM サービスロールを作成または指定するには (この場合、ターゲットは CodePipeline) 。
    - この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始するアクセス許可を付与 EventBridgeするサービスロールを作成します。
    - 「既存のロールを使用」を選択して、パイプラインの実行を開始するアクセス EventBridge 許可を付与するサービスロールを入力します。
12. [次へ] をクリックします。
13. [タグ] ページで、[次へ] を選択します
14. [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 ソースの 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 CodePipeline に関連付けられた で作成するすべてのパイプラインで一意である必要があります。
- ルールで使用するソースと詳細フィールドのイベントパターン。詳細については、「Amazon EventBridge とイベントパターン」を参照してください。

をイベントソース CodeCommit 、 をターゲット CodePipeline とする EventBridge ルールを作成するには

1. ルール EventBridge の呼び出しに使用する CodePipeline のアクセス許可を追加します。詳細については、「Amazon でのリソースベースのポリシーの使用 EventBridge」を参照してください。

a. 次のサンプルを使用して、 がサービスロールを引き受け EventBridge ること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 このポリシーをロールに追加すると、 のアクセス許可が作成されます EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

この変更を行う理由 このコマンドでは、 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. をターゲット CodePipeline 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
```

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 パラメータのデフォルト設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

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

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のプレーンテキストエディタで JSON ファイルを開き、以下に示しているように、PollForSourceChanges パラメータを false に変更してソースステージを編集します。

   この変更を行う理由 このパラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

   ```
   "configuration": {
       "PollForSourceChanges": "false",
       "BranchName": "main",
       "RepositoryName": "MyTestRepo"
   },
   ```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除します。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

   例えば、構造から以下の行を削除します。

   ```
   "metadata": {
   ```

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

ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには **start-pipeline-execution** コマンドを使用します。

CodeCommit ソースの EventBridge ルールを作成する (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

1. 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 のアクセス許可を作成できます EventBridge。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": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
```

```
                    "Ref": "AppPipeline"
                  }
                ]

...
```

2. 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。がリポジトリの状態の変更 EventBridge を検出すると、ルールはターゲットパイプラインStartPipelineExecutionで 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": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
```

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

3. 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] を選択します。

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 パラメータのデフォルト設定」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
    }
  ]
},
```

GitHub 接続

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**
>
> 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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

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 CodePipeline、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 （ バージョン 2) 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、パイプラインでフルクローンオプションを使用する方法を説明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: GitHub パイプラインソースでフルクローンを使用する。

- `CodeStarSourceConnection` アクションのアクション設定を `GitHub` プロバイダに追加するには、 パイプラインを作成する (CLI) に記載されている CLI 手順に従います。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- でアカウントを作成しておく必要があります GitHub。
- GitHub コードリポジトリを既に作成しておく必要があります。
- CodePipeline サービスロールが 2019 年 12 月 18 日より前に作成された場合は、 AWS CodeStar 接続`codestar-connections:UseConnection`に使用するアクセス許可を更新する必要がある場合があります。手順については、「 CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

> **Note**
>
> 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

- への接続 GitHubを作成する (コンソール)
- (GitHub CLI) への接続を作成する

への接続 GitHubを作成する (コンソール)

CodePipeline コンソールを使用して、 または GitHub Enterprise Cloud リポジトリの接続アクション GitHubを追加するには、次の手順に従います。

> **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、 によってアクセスまたは表示されません CodePipeline。

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 (バージョン 2) を選択します。

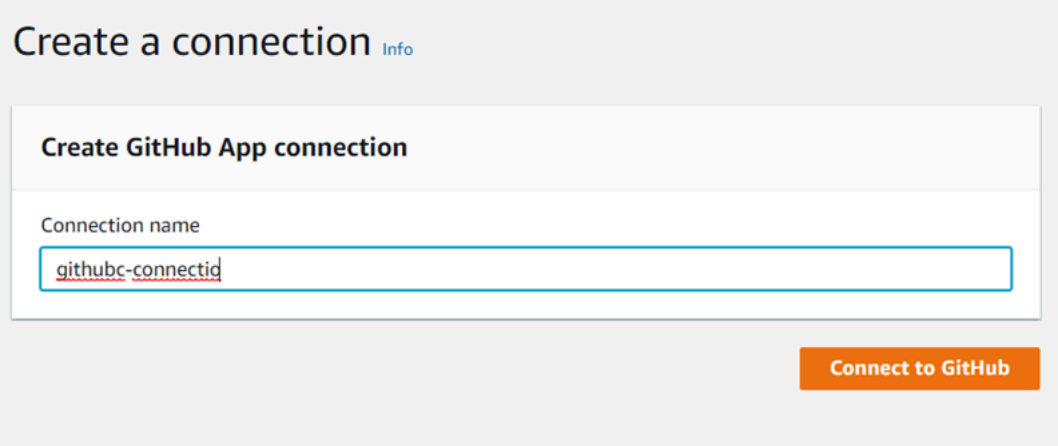
- 既存のパイプラインを編集することを選択します。[Edit]、[Edit Stage] の順に選択します。ソースアクションを追加または編集するかを選択します。[アクションの編集] ページで、[Action name (アクション名)] に自分のアクション名を入力します。アクションプロバイダーで、 GitHub （ バージョン 2) を選択します。

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、「 に接続する GitHub」を選択します。ステップ 2: への接続を作成するに進みます GitHub。
   - [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。

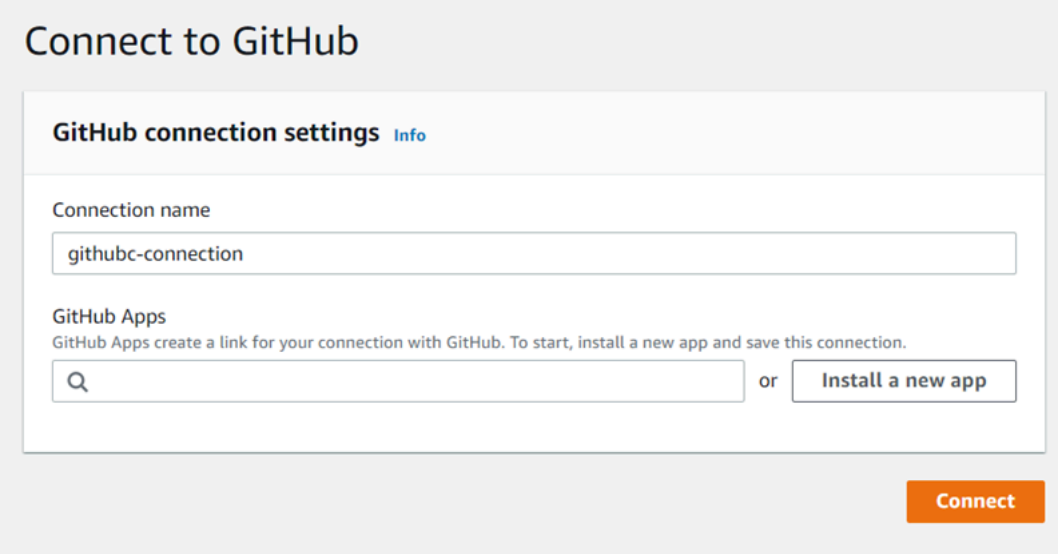
ステップ 2: への接続を作成する GitHub

接続の作成を選択すると、Connect to GitHubページが表示されます。



への接続を作成するには GitHub

1. GitHub 接続設定 で、接続名が接続名 に表示されます。に接続 GitHubを選択します。アクセス要求のページが表示されます。
2. の AWS コネクタの承認を選択します GitHub。接続ページが表示され、GitHub アプリフィールドが表示されます。



3. GitHub アプリ で、アプリのインストールを選択するか、新しいアプリのインストールを選択して作成します。

> **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。AWS Connector for GitHub App 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。

4. 「 用 AWS コネクタのインストール 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。アプリケーションをインストール済みである場合は、[Configure] (設定) を選択してアプリのインストールの変更ページに進むか、戻るボタンでコンソールに戻ることができます。

5. 「 用 AWS コネクタのインストール GitHub」ページで、デフォルトのままにして、「 のインストール」を選択します。
6. Connect to GitHub ページで、新しいインストールの接続 ID が GitHub Apps に表示されます。[接続]を選択します。

ステップ 3: GitHub 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 ソースアクションを保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが CodeConnections アクションの場合にトリガーを追加できます。パイプラインのトリガー設定を設定し、オプションでトリガーでフィルタリングするには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。
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 デフォルトの を選択します。アクションは GitHub リポジトリからファイルにアクセスし、アーティファクトをパイプラインアーティファクトストアの ZIP ファイルに保存します。
   - 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、「」に示すように、 CodeBuild プロジェクトサービスロールのアクセス許可を更新する必要がありますBitbucket、、 GitHub Enterprise Server GitHub、または GitLab.com への接続の CodeBuild GitClone アクセス許可を追加する。[完全クローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: GitHub パイプラインソースでフルクローンを使用する を参照してください。
4. ウィザード上で [次へ] または [保存] を [アクションを編集] ページで選択します。

（ GitHub CLI) への接続を作成する

AWS Command Line Interface （ AWS CLI) を使用して接続を作成できます。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を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

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。 AWS CLI を使用して create-connection コマンドを実行し、接続--connection-nameの --provider-typeと を指定します。この例では、サードパーティープロバイダー名は GitHub で、指定された接続名は MyConnection です。

   ```
   aws codestar-connections create-connection --provider-type GitHub --connection-name
    MyConnection
   ```

   成功した場合、このコマンドは次のような接続 ARN 情報を返します。

   ```
   {
       "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
   aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
   }
   ```

2. コンソールを使用して接続を完了します。詳細については、「Update a pending connection」を参照してください。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - パイプラインのトリガー設定をトリガーでフィルタリングするように設定するには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。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 を追加します。この

例では、release-v0 と release-v1 が包含する Git タグで、release-v2 が除外する Git タグです。

```
"triggers": [
            {
                "providerType": "CodeStarSourceConnection",
                "gitConfiguration": {
                    "sourceActionName": "Source",
                    "push": [
                        {
                            "tags": {
                                "includes": [
                                    "release-v0", "release-v1"
                                ],
                                "excludes": [
                                    "release-v2"
                                ]
                            }
                        }
                    ]
                }
            }
        ]
```

GitHub Enterprise Server 接続

接続を使用すると、サードパーティープロバイダーを AWS リソースに関連付ける設定を許可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**
>
> 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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket

Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

で GitHub Enterprise Server ソースアクションを追加するには CodePipeline、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 エンタープライズサーバー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、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、Enterprise Server (CLI) GitHub へのホストと接続を作成する を参照してCLI でホストリソースと接続リソースを作成します。
  - `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用 で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- Enterprise Server でアカウントを作成し、 GitHub Enterprise Server GitHub インスタンスをインフラストラクチャに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 でコードリポジトリを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)
- Enterprise Server (CLI) GitHub へのホストと接続を作成する

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

CodePipeline コンソールを使用して Enterprise Server リポジトリの接続アクション GitHubを追加するには、次の手順に従います。

> **Note**
>
> GitHub Enterprise Server 接続は、接続の作成に使用された GitHub Enterprise Server 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:

GitHub Enterprise Server へのホスト接続の場合、接続用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、GitHub エンタープライズサーバー 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 エンタープライズサーバー 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、 GitHub エンタープライズサーバーへの接続 を選択します。ステップ 2: GitHub エンタープライズサーバー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。

Enterprise Server への接続 GitHubを作成する

接続の作成を選択すると、エンタープライズサーバーへの接続 GitHubページが表示されます。

> **Important**
>
> AWS CodeConnections リリースで既知の問題により、 は GitHub Enterprise Server バージョン 2.22.0 をサポートしていません。接続するには、バージョン 2.22.1 または入手可能な最新のバージョンにアップグレードします。

GitHub Enterprise Server に接続するには

1. [Connection name] (接続名) に、接続の名前を入力します。
2. [URL] に、サーバーのエンドポイントを入力します。

   > **Note**
   >
   > 指定された URL が接続用の Enterprise Server のセットアップ GitHubに既に使用されている場合は、そのエンドポイント用に以前に作成されたホストリソース ARN を選択するように求められます。

3. Amazon VPC でサーバーを起動し、VPC に接続する場合は、[Use a VPC] (VPC を使用) をクリックして、以下を完了します。

   a. [VPC ID] で、 VPC ID を選択します。 GitHub エンタープライズサーバー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 VPC、または VPN または Direct Connect を介してエンタープライズサーバーインスタンスにアクセスできる GitHub VPC を必ず選択してください。

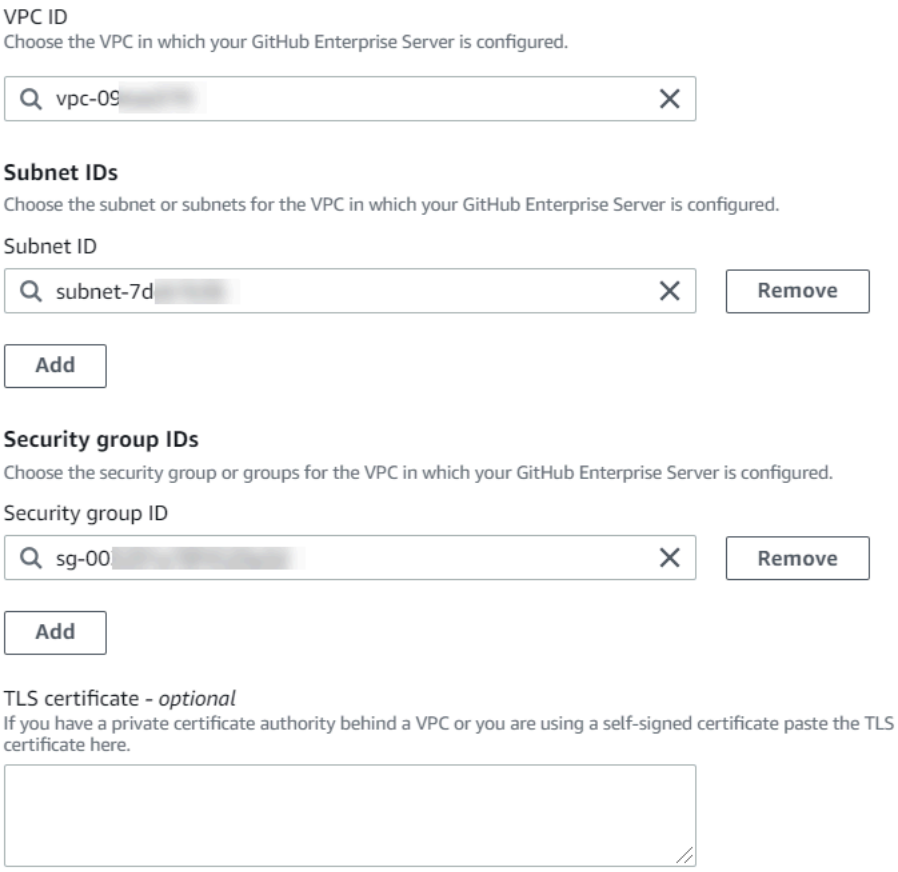
   b. [サブネット ID] で、[Add] を選択します。このフィールドで、ホストに使用するサブネット ID を選択します。最大 10 個のサブネットを選択できます。

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 GitHub 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必ず選択してください。

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。

   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 GitHub 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必ず選択してください。

d. プライベート VPC が設定されていて、非パブリック認証機関を使用して TLS 検証を実行するように GitHub Enterprise Server インスタンスを設定している場合は、TLS 証明書 に証明書 ID を入力します。TLS 証明書の値は、証明書のパブリックキーである必要があります。



4. GitHub 「エンタープライズサーバーに接続」を選択します。作成された接続は、Pending (保留中) のステータスで表示されます。指定したサーバ情報との接続用に、ホストリソースが作成されます。ホスト名には、URL が使用されます。
5. 保留中の接続の更新を選択します。
6. プロンプトが表示されたら、 GitHub エンタープライズログインページで、エンタープライズ認証情報を使用してサインインします GitHub。
7. GitHub アプリの作成ページで、アプリの名前を選択します。
8. GitHub 認証ページで、「<app-name> の承認」を選択します。
9. アプリのインストールページで、コネクタアプリをインストールする準備ができている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。[Install] (インストール) を選択します。
10. 接続ページには、作成された接続が Available (使用可能) ステータスで表示されます。

ステップ 3: GitHub Enterprise Server のソースアクションを保存する

ウィザードで次の手順を使用するか、[アクションを編集] ページで、ソースアクションを接続情報とともに保存します。

接続でソースアクションを完了して保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが CodeConnections アクションの場合にトリガーを追加できます。パイプラインのトリガー設定を設定し、オプションでトリガーでフィルタリングするには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。
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デフォルトの を選択します。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。

Enterprise Server (CLI) GitHub へのホストと接続を作成する

AWS Command Line Interface （ AWS CLI) を使用して接続を作成できます。

これを行うには、create-connection コマンドを使用します。

> **Important**
>
> AWS CLI または を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 回のみ作成します。特定の GitHub Enterprise Server アカウントへのすべての接続は、同じホストを使用します。

ホストを使用して、サードパーティーのプロバイダがインストールされているインフラストラクチャのエンドポイントを表します。CLI でホストの作成を完了すると、ホストは [Pending] ステータスになります。次に、ホストのステータスが [Available] に移行するよう設定もしくは登録します。ホストが使用可能になったら、接続を作成する手順を完了します。

これを行うには、create-host コマンドを使用します。

> **Important**
>
> を通じて作成されたホスト AWS CLI は、デフォルトで Pendingステータスになります。CLI でホストを作成後、コンソールまたは CLI でホストを設定し、ステータスを Available にします。

ホストを作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。 AWS CLI を使用して create-host コマンドを実行し、接続に --name、--provider-type、および --provider-endpointを指定します。この例では、サードパーティープロバイダー名は GitHubEnterpriseServer で、エンドポイントは my-instance.dev です。

   ```
   aws codestar-connections create-host --name MyHost --provider-type
    GitHubEnterpriseServer --provider-endpoint "https://my-instance.dev"
   ```

   成功した場合、このコマンドは次のようなホストの Amazonリソースネーム (ARN) 情報を返します。

   ```
   {
       "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
   Host-28aef605"
   }
   ```

   この手順の後、ホストのステータスは PENDING になります。
2. コンソールでホストのセットアップを完了し、ホストのステータスを Available に移行します。

GitHub Enterprise Server への接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。 AWS CLI を使用して create-connection コマンドを実行し、接続--connection-nameに --host-arnとを指定します。

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. コンソールを使用して、保留中の接続を設定します。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - パイプラインのトリガー設定をトリガーでフィルタリングするように設定するには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。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 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 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許可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**
>
> 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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

GitLab.com ソースアクションを に追加するには CodePipeline、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、GitLabプロバイダーオプションを選択します。アクションを追加するには GitLab.com への接続を作成する (コンソール) を参照してください。このコンソールは、接続リソースの作成に役立ちます。
- CLI を使用して、次の通りに `CreateSourceConnection` でのアクションのアクション設定を `GitLab` プロバイダに追加します。
  - 接続リソースを作成するには、 GitLab.com への接続を作成する (CLI) を参照してCLI で接続リソースを作成します。

- `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用 で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

> **Note**
>
> GitLab.com でこの接続のインストールを承認することで、 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当社のサービスに付与し、アプリケーション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。

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**
  >
  > で所有者ロールを持つリポジトリへの接続を作成し GitLab、その接続を などのリソースを持つリポジトリで使用できます CodePipeline。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)

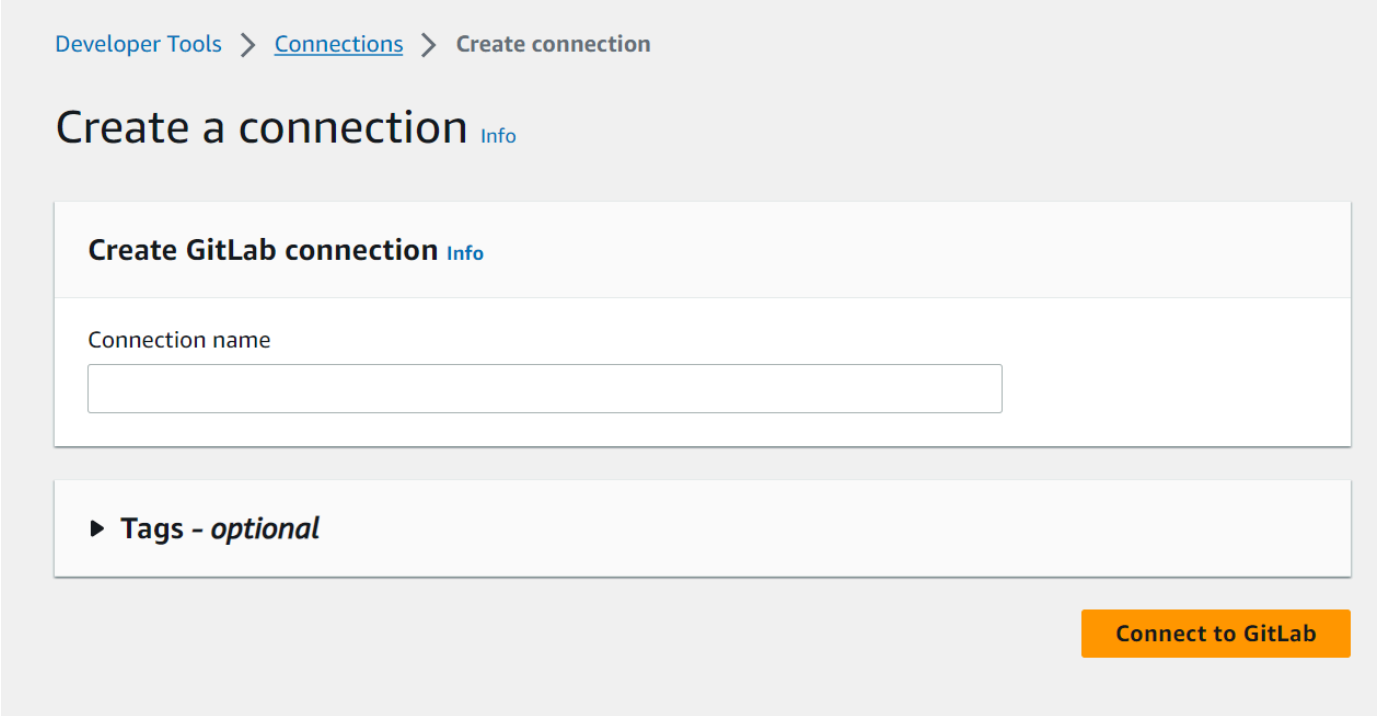
CodePipeline コンソールを使用して、 でプロジェクト (リポジトリ) の接続アクションを追加するには、次の手順に従います GitLab。

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、 を選択しますGitLab。
   - 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、「 に接続する GitLab」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。

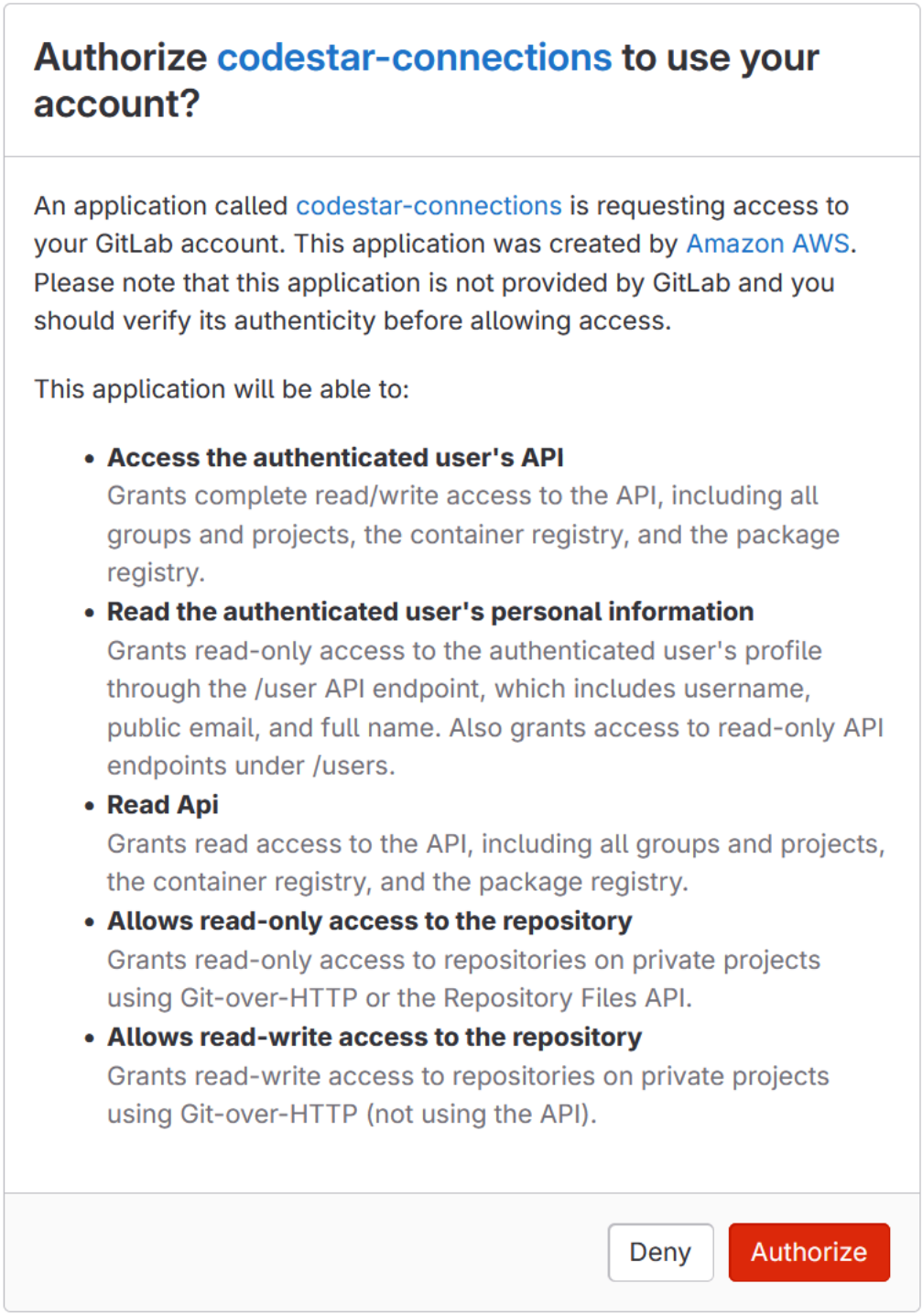
   > **Note**
   >
   > GitLab.com 接続を作成する前にポップアップウィンドウを閉じる場合は、ページを更新する必要があります。

4. GitLab.com リポジトリへの接続を作成するには、「プロバイダーの選択」で、「」を選択しますGitLab。[接続名] に、作成する接続の名前を入力します。 に接続 GitLabを選択します。



5. GitLab.com のサインインページが表示されたら、 認証情報を使用してログインし、「サインイン」を選択します。
6. 接続を初めて承認する場合は、. GitLabcom アカウントにアクセスするための接続の承認をリクエストするメッセージを含む承認ページが表示されます。

   [承認] を選択します。



7. ブラウザは接続コンソールページに戻ります。 GitLab 「接続の作成」で、新しい接続が接続名に表示されます。
8. に接続 GitLabを選択します。

   CodePipeline コンソールに戻ります。

> **Note**
>
> GitLab.com 接続が正常に作成されると、メインウィンドウに成功バナーが表示されます。
> 現在のマシン GitLab で に以前にログインしたことがない場合は、ポップアップウィンドウを手動で閉じる必要があります。

9. リポジトリ名 で、 名前空間でプロジェクトパスを指定 GitLab して、 でプロジェクトの名前を選択します。例えば、グループレベルのリポジトリの場合は、リポジトリ名を `group-name/repository-name` の形式で入力します。パスと名前空間の詳細については、https://docs.gitlab.com/ee/api/projects.html#get-single-project の `path_with_namespace`フィールドを参照してください。の名前空間の詳細については、https://docs.gitlab.com/ee/user/namespace/ GitLabを参照してください。

> **Note**
>
> のグループの場合 GitLab、 名前空間でプロジェクトパス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。名前空間を含むプロジェクトパスは、 の URL にあります GitLab。

10. パイプライントリガーでは、アクションが CodeConnections アクションの場合にトリガーを追加できます。パイプラインのトリガー設定を設定し、オプションでトリガーでフィルタリングするには、「」で詳細を参照してください[コードプッシュまたはプルリクエストでトリガーをフィルタリングする]。

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 デフォルトの 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、「」に示すように、 CodeBuild プロジェクトサービスロールのアクセス許可を更新する必要がありますBitbucket、、 GitHub Enterprise Server GitHub、または GitLab.com への接続の CodeBuild GitClone アクセス許可を追加する。[完全クローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: GitHub パイプラインソースでフルクローンを使用する を参照してください。

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ステータスになります。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

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameに --provider-typeと を指定します。この例では、サードパーティープロバイダー名は GitLab で、指定された接続名は MyConnection です。

```
aws codestar-connections create-connection --provider-type GitLab --connection-name
 MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
}
```

2. コンソールを使用して接続を完了します。詳細については、「Update a pending connection」を参照してください。
3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - トリガーでフィルタリングするようにパイプライントリガー設定を構成するには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。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 を追加します。この例では、`release-v0` と `release-v1` が包含する Git タグで、`release-v2` が除外する Git タグです。

     ```
     "triggers": [
                {
                    "providerType": "CodeStarSourceConnection",
                    "gitConfiguration": {
                        "sourceActionName": "Source",
                        "push": [
                            {
                                "tags": {
                                    "includes": [
                                        "release-v0", "release-v1"
                                    ],
                                    "excludes": [
                                        "release-v2"
                                    ]
                                }
                            }
     ```

```
                ]
            }
        }
    ]
```

GitLab セルフマネージド型 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許可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**
>
> 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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

で GitLab セルフマネージド型のソースアクションを追加するには CodePipeline、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、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 セルフマネージドへの接続 GitLabを作成する (コンソール) を参照してください。このコンソールは、ホストリソースと接続リソースの作成に役立ちます。
- CLI を使用して、次の通りに `CreateSourceConnection` アクションのアクション設定を `GitLabSelfManaged` プロバイダに追加し、リソースを作成します。
  - 接続リソースを作成するには、ホストを作成し、セルフマネージド (CLI) に接続する GitLab を参照してCLI でホストリソースと接続リソースを作成します。
  - `CreateSourceConnection` でのアクション設定の例を CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアク

ション用 で使用し、パイプラインを作成する (CLI) で表示されるようにアクションを追加します。

> **Note**
>
> [設定] からデベロッパーツール コンソールを使用して、接続を作成することもできます。[接続を作成する] を参照してください。

開始する前に:

- でアカウントをすでに作成 GitLab し、セルフマネージドインストールで GitLab Enterprise Edition または GitLab Community Edition を持っている必要があります。詳細については、https://docs.gitlab.com/ee/subscriptions/self_managed/ を参照してください。

  > **Note**
  >
  > Connections は、接続の作成と承認に使用されたアカウント用のアクセスだけを提供します。

  > **Note**
  >
  > で所有者ロールを持つリポジトリへの接続を作成し GitLab、その接続を などのリソースで使用できます CodePipeline。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。

- スコープダウンアクセス許可のみを持つ GitLab 個人アクセストークン (PAT) を既に作成しておく必要があります。API。詳細については、https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html 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 はホストの認証に使用され、それ以外の方法で保存または接続に使用されることはありません。ホストを設定するには、一時的な PAT を作成し、ホストを設定した後に PAT を削除できます。

- ホストを事前に設定することもできます。VPC の有無にかかわらず、ホストをセットアップできます。VPC 設定の詳細とホストの作成に関する追加情報については、「ホストの作成」を参照してください。

トピック

セルフマネージドへの接続 GitLabを作成する (コンソール)

以下のステップを使用して、 CodePipeline コンソールを使用してセルフマネージド型リポジトリの接続アクション GitLab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 セルフマネージド型へのホスト接続の場合、接続用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 に接続する 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。

GitLab セルフマネージドへの接続を作成する

接続の作成を選択すると、セルフマネージド型への接続 GitLabページが表示されます。

GitLab セルフマネージド に接続するには

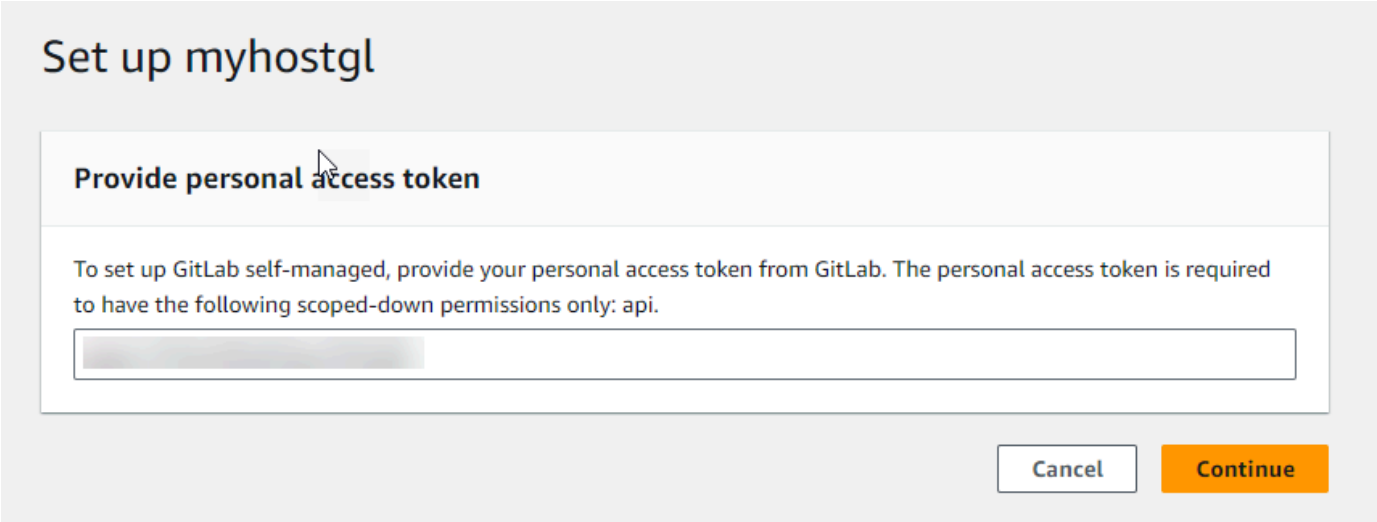
1. [Connection name] (接続名) に、接続の名前を入力します。
2. [URL] に、サーバーのエンドポイントを入力します。

   > **Note**
   >
   > 提供された URL が既に接続用のホストのセットアップに使用されていた場合、その エンドポイント用に以前に作成されたホストリソース ARN を選択するように求められます。

3. Amazon VPC でサーバーを起動し、VPC で接続する場合は、[VPC を使用] を選択して、VPC の情報を指定します。
4. GitLab セルフマネージド 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. プロバイダーにアクセスし続けることを確認するリダイレクトメッセージを示すページが開いた場合は、[続行] を選択します。プロバイダーの承認を入力します。
7. [***host_name*** のセットアップ] ページが表示されます。「個人用アクセストークンの提供」で、スコープダウンされたアクセス許可のみを持つ PAT を指定します GitLab。 `api`

   > **Note**
   >
   > PAT を作成して使用できるのは管理者のみです。

[Continue] を選択します。



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: セルフマネージドソースアクションを保存する GitLab

ウィザードで次の手順を使用するか、[アクションを編集] ページで、ソースアクションを接続情報とともに保存します。

接続でソースアクションを完了して保存するには

1. [リポジトリ名] で、サードパーティーのリポジトリの名前を選択します。
2. パイプライントリガーでは、アクションが CodeConnections アクションの場合にトリガーを追加できます。パイプラインのトリガー設定を設定し、オプションでトリガーでフィルタリングするには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。
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の出力アーティファクトを保存するには、CodePipelineデフォルトの 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。

ホストを作成し、セルフマネージド (CLI) に接続する GitLab

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ステータスです。CLI でホストを作成後、コンソールまたは CLI でホストを設定し、ステータスを Available にします。

ホストを作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-host コマンド AWS CLI を実行し、接続に --name、--provider-type、およ

び --provider-endpointを指定します。この例では、サードパーティープロバイダー名は GitLabSelfManaged で、エンドポイントは my-instance.dev です。

```
aws codestar-connections create-host --name MyHost --provider-type
 GitLabSelfManaged --provider-endpoint "https://my-instance.dev"
```

成功した場合、このコマンドは次のようなホストの Amazonリソースネーム (ARN) 情報を返します。

```
{
    "HostArn": "arn:aws:codestar-connections:us-west-2:account_id:host/My-
Host-28aef605"
}
```

この手順の後、ホストのステータスは PENDING になります。

2. コンソールでホストのセットアップを完了し、ホストのステータスを Available に移行します。

GitLab セルフマネージドへの接続を作成するには

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。を使用して create-connection コマンド AWS CLI を実行し、接続--connection-nameに --host-arnとを指定します。

```
aws codestar-connections create-connection --host-arn arn:aws:codestar-
connections:us-west-2:account_id:host/MyHost-234EXAMPLE --connection-name
 MyConnection
```

成功した場合、このコマンドは次のような接続 ARN 情報を返します。

```
{
    "ConnectionArn": "arn:aws:codestar-connections:us-west-2:account_id:connection/
aEXAMPLE-8aad"
}
```

2. コンソールを使用して、保留中の接続を設定します。

3. パイプラインはデフォルトで、接続ソースリポジトリへのコードのプッシュ時に変更を検出するようになっています。パイプライントリガーを手動リリース用または Git タグ用に設定するには、以下のいずれかを行います。

   - 手動リリースでのみ開始するようにパイプラインのトリガー設定を構成するには、設定に以下の行を追加します。

     ```
     "DetectChanges": "false",
     ```

   - パイプラインのトリガー設定をトリガーでフィルタリングするように設定するには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。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 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"
                                    ]
                                }
                            }
                        ]
                    }
                }
            ]
     ```

でパイプラインを編集する CodePipeline

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を記述します。パイプラインを編集して、要素を追加または削除することができま

す。ただし、パイプラインを編集するときには、パイプライン名またはパイプラインメタデータなどの値を変更することはできません。

パイプライン編集ページを使用して、パイプラインタイプ、変数、およびトリガーを編集できます。パイプラインのステージとアクションを追加または変更することもできます。

パイプラインを作成する場合とは異なり、パイプラインを編集しても、パイプラインを通して最新のリビジョンが実行されることはありません。編集後のパイプラインを通して、最新のリビジョンを実行する場合は、手動で再実行する必要があります。それ以外の場合、編集後のパイプラインはソースステージで設定されているソースの場所の次回変更時に実行されます。詳細については、「パイプラインを手動で開始する」を参照してください。

パイプラインとは異なる AWS リージョン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、「でクロスリージョンアクションを追加する CodePipeline」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、「」を参照してください CodeCommit ソースの EventBridge ルールを作成する (CLI)。CLI を使用した Amazon S3 パイプラインの作成または更新の詳細については、「Amazon S3 ソースの EventBridge ルールを作成する (CLI)」を参照してください。

トピック

- パイプラインを編集する (コンソール)
- パイプラインを編集する (AWS CLI)

パイプラインを編集する (コンソール)

CodePipeline コンソールを使用して、パイプライン内のステージを追加、編集、または削除したり、ステージ内のアクションを追加、編集、または削除したりできます。

パイプラインを更新すると、 は実行中のすべてのアクション CodePipelineを適切に完了し、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を失敗させます。パイプラインが更新されたら、パイプラインを再実行する必要があります。パイプラインの実行に関する詳細については、「パイプラインを手動で開始する」を参照してください。

パイプラインを編集するには

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、編集するパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインの各ステージの各アクションの状態などがわかります。
3. パイプライン詳細ページで、[編集] を選択します。
4. パイプラインタイプを編集するには、[編集: パイプラインのプロパティ] カードで [編集: を選択します。次のいずれかのオプションを選択し、[完了] を選択します。

   - V1 タイプのパイプラインは、標準のパイプライン、ステージ、アクションレベルのパラメータを含む JSON 構造になっています。
   - V2 タイプのパイプラインは、V1 タイプと同じ構造ですが、トリガーとパイプラインレベルの変数など、追加のパラメータがサポートされています。

   パイプラインのタイプによって特徴および価格が異なります。詳細については、「パイプラインのタイプ」を参照してください。
5. パイプライン変数を編集するには、[編集: 変数] カードの[変数の編集] を選択します。パイプラインレベルの変数を追加または変更し、[完了] を選択します。

   パイプラインレベルの変数の詳細については、「変数」を参照してください。パイプラインの実行時に渡されるパイプラインレベルの変数のチュートリアルについては、「チュートリアル: パイプラインレベルの変数を使用する」を参照してください。

> **Note**
>
> パイプラインレベルの変数の追加はオプションですが、パイプラインレベルの変数に値が設定されていない場合、パイプラインの実行は失敗します。

6. パイプライントリガーを編集するには、[編集: トリガー] カードで [トリガーの編集] を選択します。トリガーを追加または変更し、[完了] を選択します。

   トリガーの追加の詳細については、Bitbucket Cloud、 GitHub （ バージョン 2)、 GitHub Enterprise Server、 GitLab.com、または GitLab などの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 接続。

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 リポジトリ名とブランチの場合、このパイプライン用に作成される Amazon CloudWatch Events ルールを示すメッセージが表示されます。 CodeCommit ソースを削除すると、削除する Amazon CloudWatch Events ルールを示すメッセージが表示されます。
     - Amazon S3 ソースバケットの場合、このパイプライン用に作成される Amazon CloudWatch Events ルールと AWS CloudTrail 証跡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、削除する Amazon CloudWatch Events ルールと AWS CloudTrail 証跡を示すメッセージが表示されます。証 AWS CloudTrail 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。
   - 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には、ステージ の編集 を選択し、ステージ失敗時に自動ロールバックを設定する のチェックボックスを選択します。

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のデフォルトは、パイプラインと同じ AWS リージョンです。

   入出力アーティファクトの名前や使用方法など CodePipeline、 のアクションの要件の詳細については、「」を参照してくださいのアクション構造の要件 CodePipeline。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、「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。

   をビルドアクションまたはテストアクション CodeBuild としてステージに追加するには、CodeBuild ユーザーガイドの「 CodePipeline で CodeBuild を使用してコードをテストし、ビルドを実行する」を参照してください。

   > **Note**
   >
   > などの一部のアクションプロバイダーでは GitHub、アクションの設定を完了する前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。

9. アクションをテストするには、[Release change] を選択して、パイプラインによりそのコミットを処理し、パイプラインのソースステージで指定したソースに変更をコミットします。または、パイプラインを手動で開始する「」の手順に従って 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、 は実行中のすべてのアクションを CodePipeline 正常に完了し、実行中のアクションが完了したステージとパイプラインの実行に失敗します。パイプラインが更新されたら、パイプラインを再実行する必要があります。パイプラインの実行に関する詳細については、「パイプラインを手動で開始する」を参照してください。

> **Important**
>
> を使用してパートナーアクションを含むパイプライン AWS CLI を編集できますが、パートナーアクションの JSON を手動で編集しないでください。これを行った場合、パートナーアクションはパイプライン更新後に失敗します。

パイプラインを編集するには

1. ターミナルセッション (Linux、macOS または Unix) またはコマンドプロンプト (Windows) を開き、get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。たとえば、**MyFirstPipeline** という名前のパイプラインに対して、以下のコマンドを入力します。

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

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のプレーンテキストエディタで JSON ファイルを開き、パイプラインに加える変更を反映するようにファイルの構造を変更します。たとえば、ステージを追加または削除すること、または、既存のステージに別のアクションを追加することができます。

以下の例では、pipeline.json ファイルに別のデプロイステージを追加する方法を示しています。このステージは、*Staging* という名前の最初のデプロイステージの後に実行されます。

> **Note**
>
> ここで示しているのは、そのファイルの一部であり、構造全体ではありません。詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

```
,
        {
            "name": "Staging",
            "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyApp"
                            }
                        ],
                        "name": "Deploy-CodeDeploy-Application",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
```

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

CLI を使用して承認アクションをパイプラインに追加する方法については、「でパイプラインに手動承認アクションを追加する CodePipeline 」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、または Amazon S3 ソースバケットの変更を検出します。次のステップには、手動でこれらのリソースを作成する手順が含まれています。フラグを false に設定すると、定期的なチェックが無効になります。これは、推奨される変更検出メソッドを使用する場合には必須ではありません。

3. パイプラインとは異なるリージョンにビルド、テスト、またはデプロイアクションを追加するには、パイプライン構造に以下を追加する必要があります。詳細な手順については、「でクロスリージョンアクションを追加する CodePipeline」を参照してください。

   - Region パラメータをアクションのパイプライン構造に追加します。
   - アクションの作成先のリージョンごとにアーティファクトバケットを指定するには、artifactStores パラメータを使用します。

4. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルの構造を変更する必要があります。metadata コマンドが使用できるように、ファイルから update-pipeline 行を削除する必要があります。JSON ファイルのパイプライン構造からセクションを削除します ("metadata": { } 行と、"created"、"pipelineARN"、および"updated"フィールド)。

   例えば、構造から以下の行を削除します。

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

   ファイルを保存します。

5. パイプラインの編集に CLI を使用する場合には、推奨される変更検出リソースを手動でパイプライン用に管理する必要があります。

   - CodeCommit リポジトリの場合は、 CloudWatch 「」の説明に従ってイベントルールを作成する必要があります CodeCommit ソースの EventBridge ルールを作成する (CLI)。

- Amazon S3 ソースの場合、「」で説明されているように、 CloudWatch イベントルールと AWS CloudTrail 証跡を作成する必要があります EventBridge を使用した Amazon S3 ソースアクション 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。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。

7. CodePipeline コンソールを開き、編集したパイプラインを選択します。

   そのパイプラインには、行った変更が示されます。ソース場所を次に変更すると、修正した構造のパイプラインによりそのリビジョンが実行されます。

8. 修正した構造のパイプラインによりその最新のリビジョンを手動で実行するには、start-pipeline-execution コマンドを実行します。詳細については、「パイプラインを手動で開始する」を参照してください。

パイプラインの構造および想定値の詳細については、「CodePipeline パイプライン構造リファレンス」および AWS CodePipeline API リファレンス を参照してください。

でパイプラインと詳細を表示する CodePipeline

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。

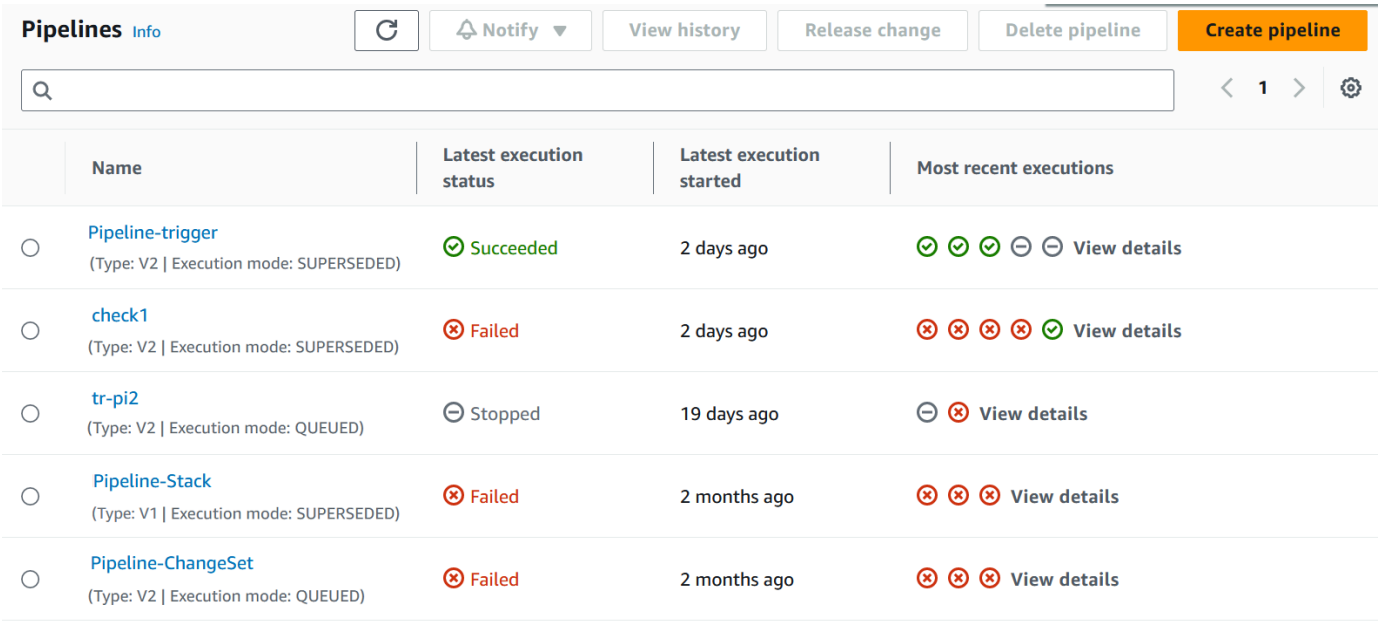
> **Note**
>
> 1 時間後には、パイプラインの詳細ビューがブラウザで自動的に更新されなくなります。現在の情報を表示するには、ページを更新します。

パイプラインを表示するには

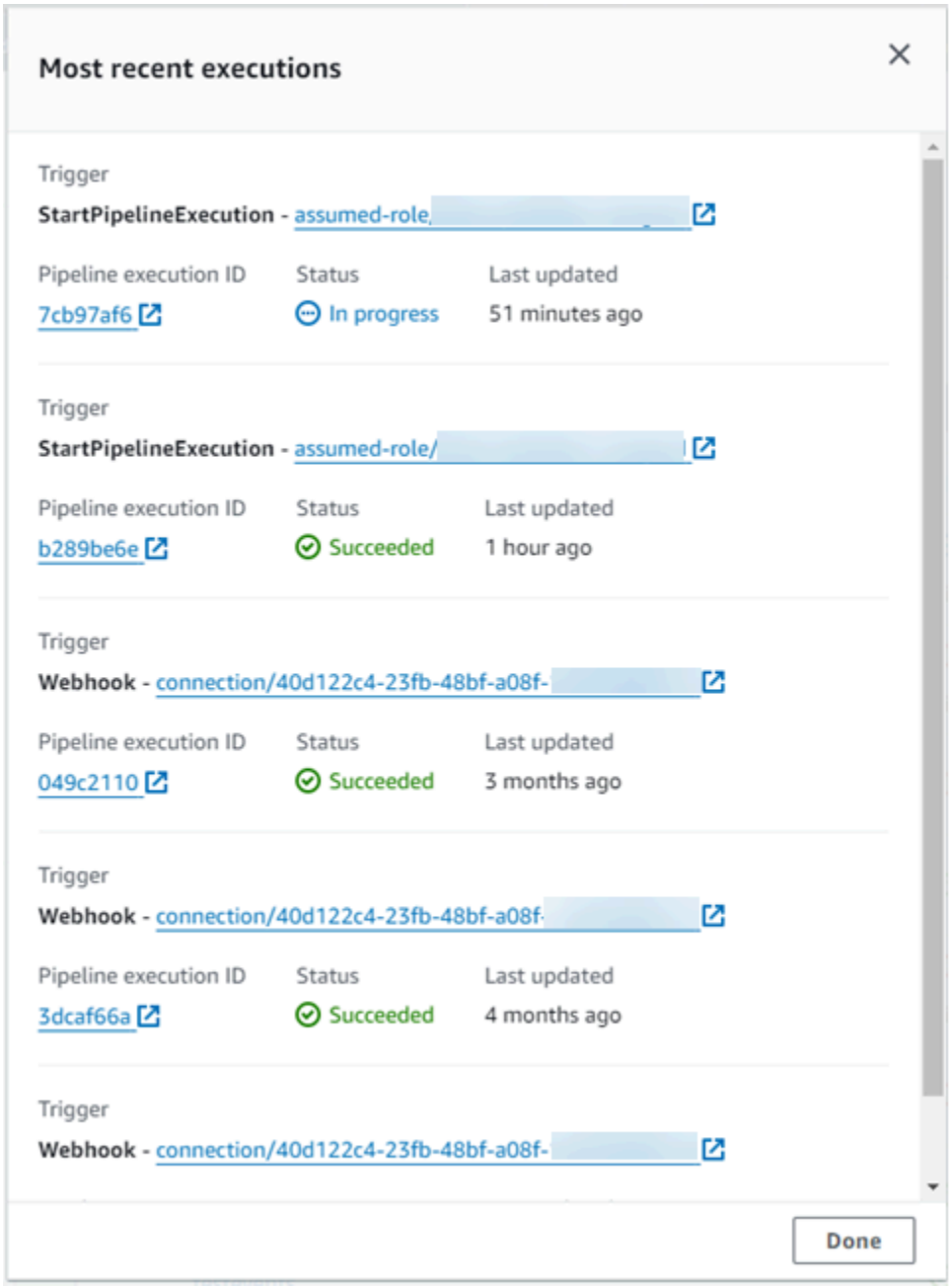
1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   [パイプライン] ページが表示されます。そのリージョンのすべてのパイプラインのリストが表示されます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前、タイプ、ステータス、バージョン、作成日、最終更新日が、最後に開始された実行時刻とともに表示されます。
2. 直近 5 回の実行のステータスが表示されます。



特定の行の横にある [詳細を表示] を選択すると、最新の実行を一覧表示する詳細ダイアログボックスが表示されます。

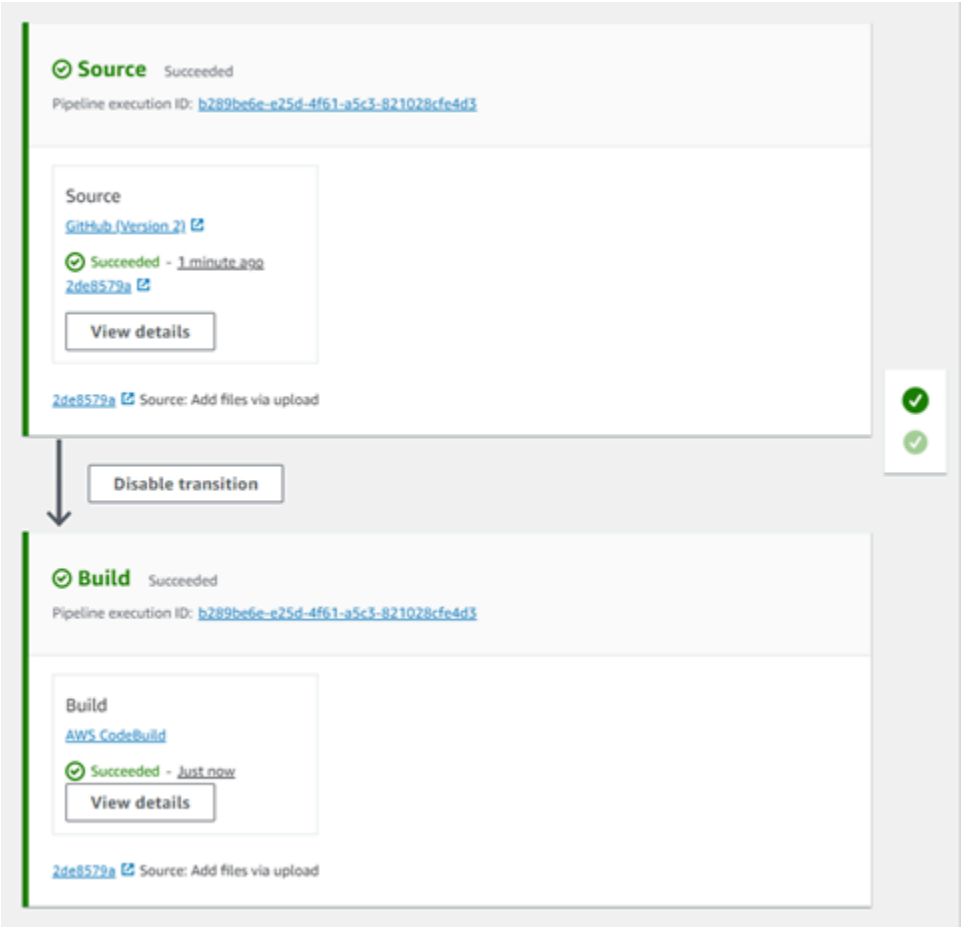


パイプラインの最新の実行の詳細を表示するには、[履歴の表示] を選択します。過去の実行については、実行 ID、ステータス、開始時刻と終了時刻、継続時間、コミット ID、メッセージなど、ソースアーティファクトに関連するリビジョンの詳細を表示できます。

> **Note**
>
> PARALLEL 実行モードのパイプラインの場合、メインパイプラインビューにはパイプライン構造や進行中の実行は表示されません。PARALLEL 実行モードのパイプラインの場合は、実行履歴ページから表示する実行の ID を選択して、パイプライン構造にアク

セスします。左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、視覚化タブでパイプラインを表示します。

3. 1 つのパイプラインの詳細を表示するには、[Name] でパイプラインを選択します。パイプラインの詳細ビューが開いて、各ステージの各アクションの状態や遷移の状態などが表示されます。



グラフィカルビューには、各ステージについて以下の情報が表示されます。

- ステージ名です。
- ステージ用に設定されたすべてのアクション。
- ステージ間の遷移の状態 (有効または無効)。ステージ間の矢印の状態によって示されます。有効な移行は矢印とその横にある [移行を無効にする] ボタンとともに示されます。無効にされた移行は、下に取り消し線が付いた矢印、およびその横の [移行を有効にする] ボタンで示されます。
- ステージのステータスを示すカラーバー:
  - グレー: まだ実行はなし

- 青: 進行中
- 緑: 成功
- 赤: 失敗

グラフィカルビューには、各ステージのアクションについて以下の情報も表示されます。

- アクションの名前。
- など、アクションのプロバイダー CodeDeploy。
- 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)。
- ステージで最新のパイプライン実行を通じて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に関する詳細。
- [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**
>
> Logs タブは、パイプラインのアカウントで実行された CodeBuild および AWS CloudFormation アクションで使用できます。

4. アクションのプロバイダーの詳細を表示するには、プロバイダーを選択します。例えば、前述のパイプラインの例では、ステージングステージまたは本番稼働ステージ CodeDeploy のいずれかで を選択すると、そのステージに設定されたデプロイグループの CodeDeploy コンソールページが表示されます。
5. アクションの進捗状況を表示するには、進行中のアクション ([進行中] メッセージによって示される) の横に表示される [詳細] を選択します。アクションが進行中の場合は、段階的な進行状況と、実行されたステップまたはアクションが表示されます。
6. 手動承認用に設定されたアクションを承認または拒否するには、[確認] を選択します。
7. 正常に完了しなかったステージでアクションを再試行するには、[Retry] を選択します。
8. アクションが最後に実行されたときのステータスが、そのアクションの結果も含めて ([成功] または [失敗]) 表示されます。

パイプラインのアクションの詳細を表示する (コンソール)

各ステージのアクションの詳細など、パイプラインの詳細を表示できます。

> **Note**
>
> 1 時間後には、パイプラインの詳細ビューがブラウザで自動的に更新されなくなります。現在の情報を表示するには、ページを更新します。

パイプラインのアクションの詳細を表示するには

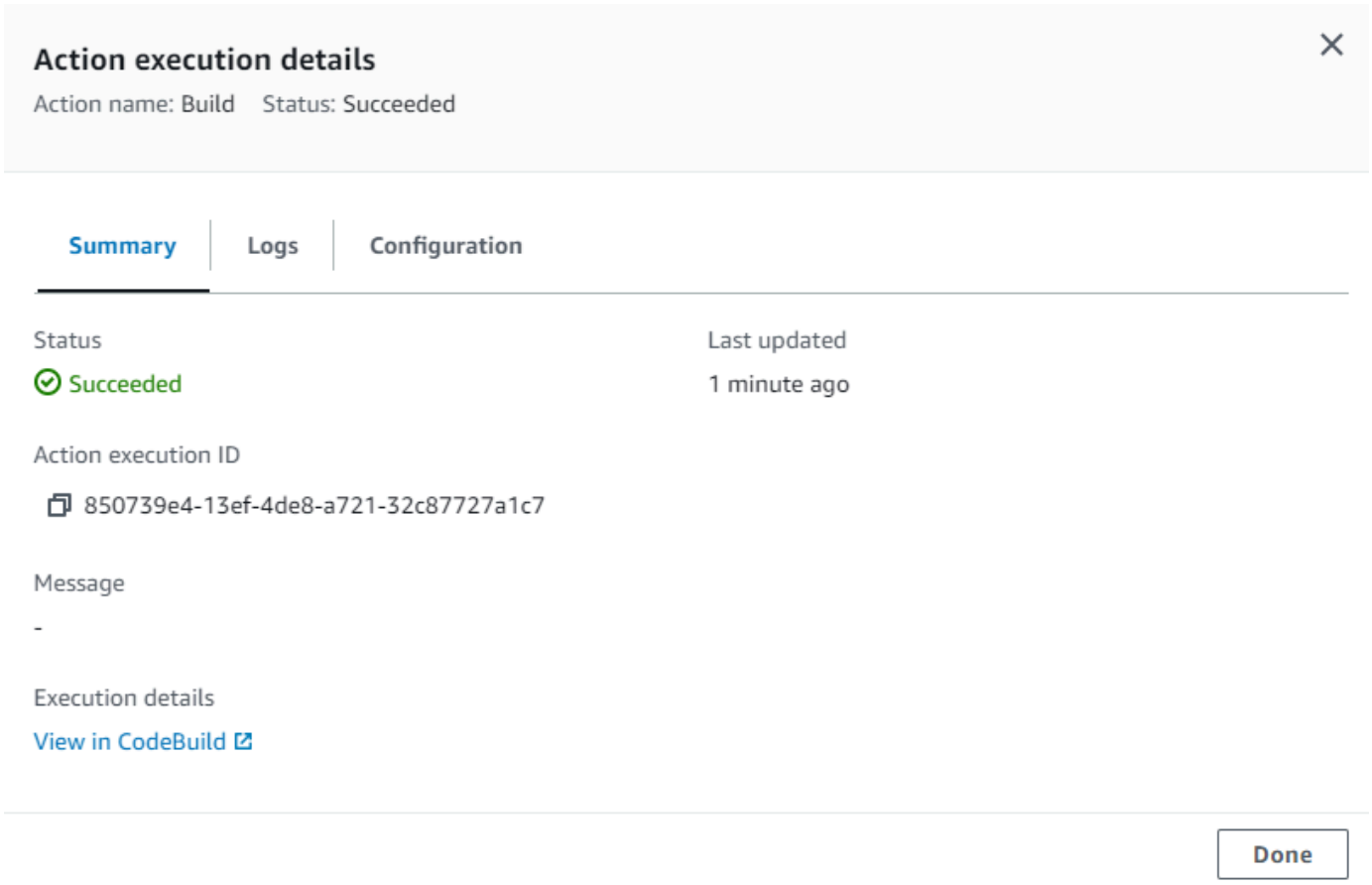
1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   [パイプライン] ページが表示されます。

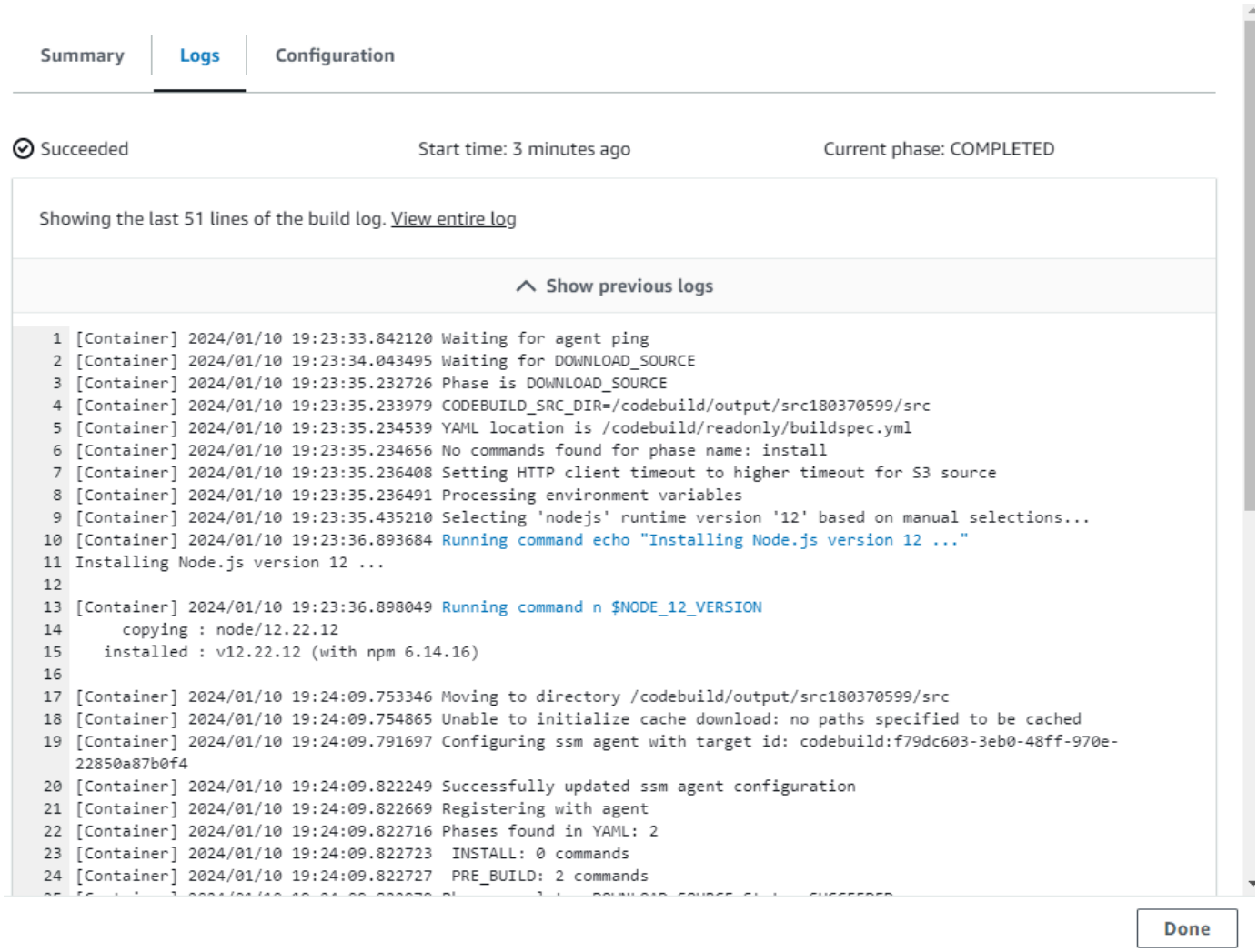
2. どのアクションでも、[詳細を表示] を選択すると、アクションの実行、ログ、およびアクション設定に関する詳細を含むダイアログボックスが開きます。

   > **Note**
   >
   > Logs タブは、 CodeBuild および AWS CloudFormation アクションで使用できます。

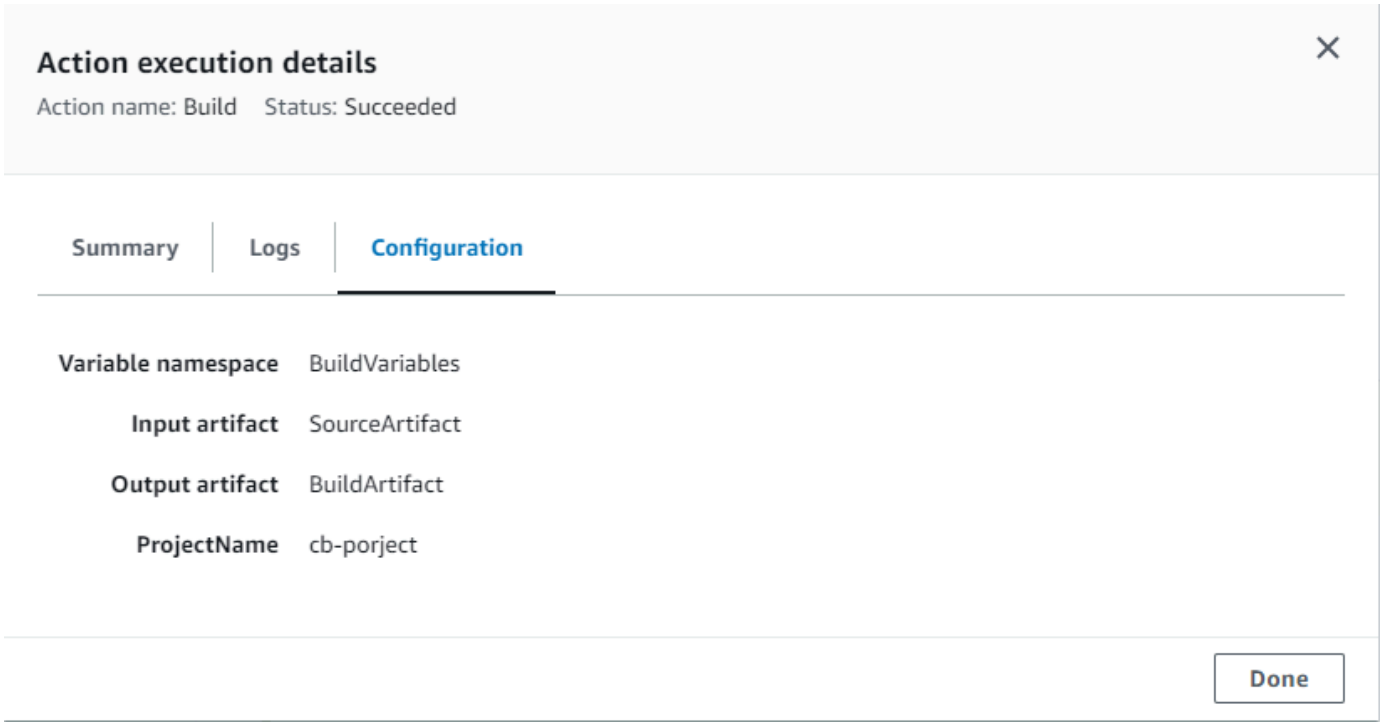
3. パイプラインのステージにあるアクションについてアクションの概要を表示するには、アクションの [詳細を表示] を選択し、[概要] タブを選択します。



4. ログ付きのアクションのアクションログを表示するには、アクションの [詳細を表示] を選択し、[ログ] タブを選択します。



5. アクションの設定の詳細を表示するには、[設定] タブを選択します。



パイプラインの ARN とサービスロール ARN (コンソール) を表示します。

コンソールを使用して、パイプライン ARN、サービスロール ARN、パイプラインアーティファクトストアなどの、パイプライン設定を表示できます。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. パイプラインの名前を選択し、左側のナビゲーションペイン の Settings を選択します。ページでは以下を示します。

   - パイプライン名
   - パイプラインの Amazon リソースネーム (ARN)

     パイプライン ARN はこの形式で作成されます。

     arn:aws:codepipeline:*region*:*account*:pipeline-name

     パイプライン ARN の例:

     `arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline`

- パイプライン CodePipeline のサービスロール ARN
- パイプラインのバージョン
- パイプラインのためのアーティファクトストアの名前と場所

パイプラインの詳細と履歴を表示する (CLI)

パイプラインとパイプラインの実行の詳細を表示するには、以下のコマンドを実行します。

- list-pipelines AWS アカウントに関連付けられているすべてのパイプラインの概要を表示する コマンド。
- get-pipeline コマンドは、1 つのパイプラインの詳細を表示します。
- list-pipeline-executions パイプラインの最新の実行の概要を取得します。
- get-pipeline-execution パイプラインの実行に関する情報 (アーティファクト、パイプラインの実行 ID、パイプラインの名前、バージョン、ステータスなど) を表示します。
- get-pipeline-state コマンドでパイプライン、ステージ、およびアクションのステータスを表示します。
- list-action-executions でパイプラインのアクション実行の詳細を表示します。

1. ターミナル (Linux、macOSまたは Unix) またはコマンドプロンプト (Windows) を開き、 AWS CLI を使用してlist-pipelinesコマンドを実行します。

   ```
   aws codepipeline list-pipelines
   ```

   このコマンドは、 AWS アカウントに関連付けられているすべてのパイプラインのリストを返します。

2. パイプラインの詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline コマンドを実行します。例えば、 という名前のパイプラインの詳細を表示するにはMyFirstPipeline、次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。

でパイプラインを削除する CodePipeline

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 AWS CLI パイプラインを削除できます。

トピック

パイプラインを削除する (コンソール)

パイプラインを削除するには

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [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

1. ターミナル (Linux、macOSUnix) 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。例えば、という名前のパイプラインを削除するには、次のようにします*MyFirstPipeline*。

   ```
   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、コードのデプロイに使用した CodeDeploy や Elastic Beanstalk アプリケーションなど、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。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を保存するために作成された 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。例えば、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、CodePipeline は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する 1 つの Amazon S3 バケットを作成します。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 CodePipeline を使用するパイプラインを に作成する

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、パイプラインに 1 つのアカウントを使用し、 CodeDeploy リソースに別のアカウントを使用する場合があります。

> **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ためのアカウントポリシーとロールを設定する必要があります。 AWS KMS キーの場合、キー ID、キー ARN、またはエイリアス ARN を使用できます。

> **Note**
>
> エイリアスは、カスタマーマスターキー (KMS) を作成したアカウントでのみ認識されます。クロスアカウントアクションの場合、キー ID またはキー ARN のみを使用してキーを識別できます。クロスアカウントアクションには他のアカウント (AccountB) のロールを使用するため、キー ID を指定すると他のアカウント (AccountB) のキーが使用されます。

このウォークスルーおよびサンプルでは、*AccountA*は、パイプラインの作成に使用したアカウントです。パイプラインアーティファクトを保存するために使用される Amazon S3 バケットと、 で使用されるサービスロールにアクセスできます AWS CodePipeline。*AccountB* は、 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の作成に最初に使用されたアカウントです CodeDeploy。

AccountA が AccountB によって作成された CodeDeploy アプリケーションを使用するようにパイプラインを編集するには、*AccountA* は次のことを行う必要があります。 *AccountB*

- *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。
- Amazon S3 バケットへのアクセスを *AccountB* に付与する Amazon S3 バケットポリシーを作成します (例: *codepipeline-us-east-2-1234567890*) 。
- *AccountA* が *AccountB* によって設定されたロールを引き受けることを許可するポリシーを作成し、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*)。
- Amazon S3 バケット (*codepipeline-us-east-2-1234567890*) へのアクセスを許可する CodeDeploy に設定されている Amazon Amazon EC2 インスタンスロールに適用されるポリシーを作成します。 Amazon S3
- *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 CodeDeploy を許可する、 に設定されている Amazon EC2 インスタンスロールに適用されるポリシーを作成します。
- *AccountA* CodePipeline のサービスロールがロールを引き受けることを許可する信頼関係ポリシーを使用して、IAM ロール (*CrossAccount_Role*) を設定してアタッチします。
- パイプラインに必要なデプロイリソースへのアクセスを許可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 (-) へのアクセスを許可するポリシーを作成し、*CrossAccount_Role* にアタッチします。*codepipeline-us-east2-1234567890*

トピック

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のと同じリージョン (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 で にサインインし、コンソールを開きます AWS KMS 。
2. 左側で [カスタマー管理キー]を選択します。
3. [キーの作成] を選択します。[キーの設定] で、[対称] のデフォルトを選択したまま、[次] を選択します。
4. エイリアス で、このキーに使用するエイリアス (たとえば、*PipelineName-Key*) を入力します。必要に応じて、このキーの説明とタグを入力し、[次] を選択します。
5. [キー管理アクセス許可の定義] で、このキーの管理者となるロールを選択し、[次へ] を選択します。
6. 「キー使用許可の定義」で、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 (CodePipeline_Service_Role など) を選択します。「他の AWS アカウント」で、「別の AWS アカウントを追加」を選択します。ARN の一部として *AccountB* のアカウント ID を入力し、[次へ] を選択します。
7. [キーポリシーの確認と編集]で、ポリシーを確認し、[完了]を選択します。
8. キーのリストから、キーのエイリアスを選択し、その ARN (***arn:aws:kms:us-east-2:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***など) にコピーします。この ARN は、パイプラインの編集時とポリシーの設定時に必要になります。

ステップ 1: アカウントポリシーおよびロールをセットアップする

AWS KMS キーを作成したら、クロスアカウントアクセスを有効にするポリシーを作成してアタッチする必要があります。作成するには、*AccountA*および*AccountB*の両方からアクションを行う必要があります。

トピック

- パイプラインを作成するポリシーおよびロールをアカウントに設定する (AccountA)
- AWS リソースを所有するアカウントでポリシーとロールを設定する (AccountB)

パイプラインを作成するポリシーおよびロールをアカウントに設定する (*AccountA*)

別の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 AWS 、*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 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 CodePipeline。

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 で にサインインし、https://console.aws.amazon.com/s3/ で Amazon S3 コンソールを開きます。
2. Amazon S3 バケットのリストで、パイプラインのアーティファクトが保存される Amazon S3バケットを選択します。このバケットの名前は です。ここでcodepipeline-*region*-*1234567EXAMPLE*、*region* はパイプラインを作成した AWS リージョンで、*1234567EXAMPLE* はバケット名が一意であることを保証する 10 桁の乱数です (例: *codepipeline-us-east-2-1234567890*) 。
3. Amazon S3 バケットの詳細ページで、[Properties] を選択します。
4. プロパティペインで、[アクセス許可] を展開し、[バケットポリシーの追加]を選択します。

> **Note**
>
> ポリシーがAmazon S3バケットにすでにアタッチされている場合は、バケットポリシーの編集 を選択します。以下の例のステートメントを既存のポリシーに追加できます。新しいポリシーを追加するには、リンクを選択し、 AWS ポリシージェネレーターの指示に従います。詳細については、「IAMポリシーの概要」を参照してください。

5. [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 *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
  }
 }
  },
{
  "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。

のサービスロールのポリシーを作成するには CodePipeline（コンソール）

1. *AccountA* AWS Management Console で にサインインし、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで、[ロール] を選択します。
3. ロールのリストのロール名 で、 のサービスロールの名前を選択します CodePipeline。
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*)

でアプリケーション、デプロイ、デプロイグループを作成するときは CodeDeploy、Amazon EC2 インスタンスロール も作成します。([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が *AccountB*で作成された CodeDeploy リソースを使用するには、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* CodePipeline のサービスロールがロールを引き受けることを許可する信頼関係ポリシーも含まれて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 リソースを設定することに固有です。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。

用に設定された Amazon EC2 インスタンスロールのポリシーを作成するには CodeDeploy （ コンソール ）

1. *AccountB* AWS Management Console で にサインインし、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. ナビゲーションペインで、[ロール] を選択します。
3. ロールのリストのロール名 で、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されるサービスロールの名前を選択します。 Amazon EC2 CodeDeploy このロール名はさまざまで、デプロイグループで複数のインスタンスロールを使用できます。詳細については、

「Amazon EC2 インスタンスの IAM インスタンスプロファイルを作成する」を参照してください。

4. [Permissions] (アクセス許可) タブで [Add inline policy] (インラインポリシーの追加) を選択します。

5. JSON タブを選択し、次のポリシーを入力して、*AccountA* がパイプラインのアーティファクトを保存するために使用する Amazon S3 バケットへのアクセスを許可します (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. の 2 つ目のポリシーを作成します。 AWS KMS ここで、 ***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***は *AccountA* で作成され、*AccountB* がそのキーを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/2222222-3333333-4444-556677EXAMPLE"
         ]
      }
   ]
}
```

> **Important**
>
> 次に示すように、このポリシーでは *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。

次に、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* CodePipeline の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。このロールには、 CodeDeploy リソースへのアクセスを許可するポリシーと、*AccountA* にアーティファクトを保存するために使用される 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 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。

3. [Select type of trusted entity](信頼できるエンティティのタイプを選択) で、[Another AWS account](別のアカウント) を選択します。「このロールを使用できるアカウントを指定する」の「アカウント ID」に、(AWS アカウント*AccountA*) で CodePipeline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、*AccountA* CodePipeline のサービスロールにスコープダウンすることが CodePipeline 推奨されます。ステップ 16 に従ってアクセス許可を制限します。

4. アクセス許可ポリシー をアタッチ で AmazonS3ReadOnlyAccess を選択し、次へ: タグ を選択します。

> **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。ロール名にこのロールの名前を入力します (例: *CrossAccount_Role)*。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、先ほど作成したロール (*CrossAccount_Role ##*) を選択して、そのロールの概要ページを開きます。
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 タブを選択し、次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:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource": "*"
    }
  ]
}
```

9. [ポリシーの確認]を選択します。
10. [名前] に、このポリシーの名前を入力します。[Create policy]を選択します。
11. [Permissions] (アクセス許可) タブで [Add inline policy] (インラインポリシーの追加) を選択します。
12. [JSON] タブを選択し、 以下のポリシーを入力して、このロールに *AccountA* の Amazon S3 バケットに対する入力アーティファクトの取得と出力アーティファクトの保存を許可します。

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

13. [ポリシーの確認]を選択します。
14. [名前] に、このポリシーの名前を入力します。[Create policy]を選択します。
15. アクセス許可タブで、ポリシー名 のポリシーのリストで AmazonS3ReadOnlyAccess を見つけ、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. 信頼関係タブを選択し、信頼ポリシーの編集 を選択します。左側の列の「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 では、IAM ロール を選択し、*AccountA* のサービス CodePipeline ロールの ARN を指定します。AWS プリンシパル `arn:aws:iam::Account_A:root`のリストから を削除し、ポリシー の更新 を選択します。

ステップ 2: パイプラインを編集する

CodePipeline コンソールを使用して、別の AWS アカウントに関連付けられたリソースを使用するパイプラインを作成または編集することはできません。ただし、 コンソールを使用してパイプラインの一般的な構造を作成し、 を使用してパイプライン AWS CLI を編集し、それらのリソースを追加できます。または、既存のパイプラインの構造を使用して、リソースを手動で追加することもできます。

別の AWS アカウントに関連付けられたリソースを追加するには (AWS CLI)

1. ターミナル (Linux, macOS , or Unix) またはコマンドプロンプト (Windows) で、リソースを追加するパイプラインに対して get-pipeline コマンドを実行します。コマンドの出力を JSON ファイルにコピーします。例えば、 という名前のパイプラインの場合は MyFirstPipeline、次のようなものを入力します。

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

   出力は、*pipeline.json* ファイルを開きます。

2. 任意のプレーンテキストエディタで JSON ファイルを開きます。アーティファクトストア`"type": "S3"`に KMS encryptionKey 、ID、およびタイプ情報を追加します。ここで、*codepipeline-us-east-2-1234567890* はパイプラインのアーティファクトの保存に使用される Amazon S3 バケットの名前であり、 ***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***は先ほど作成したカスタマーマネージドキーの ARN です。

   ```
   {
     "artifactStore”: {
       "location": "codepipeline-us-east-2-1234567890",
       "type": "S3",
       "encryptionKey": {
         "id": "arn:aws:kms:us-
   east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE",
         "type": "KMS"
   ```

```
    }
  },
```

3. ステージにデプロイアクションを追加して、作成したクロスアカウントロール (*CrossAccount_RoleAccountB* に関連付けられた CodeDeploy リソースを使用します。 `roleArn`

   次の例は、 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*ExternalDeploy*。ステージング という名前のステージで *AccountB* で作成された CodeDeploy リソースを使用します。 以下の例では、*AccountB*の ARN は*012ID_ACCOUNT_B*です。

```
,
            {
                "name": "Staging",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyAppBuild"
                            }
                        ],
                        "name": "ExternalDeploy",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "AccountBApplicationName",
                            "DeploymentGroupName": "AccountBApplicationGroupName"
                        },
                        "runOrder": 1,
                        "roleArn":
 "arn:aws:iam::012ID_ACCOUNT_B:role/CrossAccount_Role"
                    }
                ]
            }
```

> **Note**
>
> これは、パイプライン全体の JSON ではなく、ステージでのアクションの構造です。

4. `metadata` コマンドが使用できるように、ファイルから update-pipeline 行を削除する必要があります。JSON ファイルのパイプライン構造からセクションを削除します (`"metadata": { }` 行と、`"created"`、`"pipelineARN"`、および`"updated"`フィールド)。

   例えば、構造から以下の行を削除します。

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

   ファイルを保存します。

5. 変更を適用するには、以下のように、パイプライン JSON ファイルを指定して、 update-pipeline コマンドを実行します。

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

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

   このコマンドは、編集したパイプラインの構造全体を返します。

別の AWS アカウントに関連付けられたリソースを使用するパイプラインをテストするには

1. ターミナル (Linux, macOS , or Unix) またはコマンドプロンプト (Windows) で、以下のように、パイプラインの名前を指定して、start-pipeline-execution コマンドを実行します。

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

   詳細については、「パイプラインを手動で開始する」を参照してください。

2. *AccountA* AWS Management Console で にサインインし、 CodePipelinehttp://console.aws.amazon.com/codesuite/codepipeline/home でコンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。

3. [Name]で、先ほど編集したパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインの各ステージの各アクションの状態などがわかります。
4. パイプラインの進行状況を監視します。別の AWS アカウントに関連付けられたリソースを使用するアクションで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、*AccountB* でサインインして、 でデプロイの詳細を表示します CodeDeploy。

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、完全で end-to-end 継続的な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、 などのパイプラインソースによって決まります CodeCommit。その場合、例えば、ポーリングパイプラインは でイベントベースの変更検出に移行する必要があります EventBridge。

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 ）。	ポーリングパイプラインを CodeCommit ソースに移行する を参照してください。
Amazon S3	EventBridge および バケットがイベント通知に対して有効になっています (推奨 ）。	イベントに対応した S3 ソースを使用してポーリングパイプラインを移行する を参照してください。
Amazon S3	EventBridge および AWS CloudTrail 証跡。	S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する を参照してください。
GitHub バージョン 1	Connections (推奨)	GitHub バージョン 1 のソースアクションのポーリングパイプラインを接続に移行する を参照してください。
GitHub バージョン 1	ウェブフック	GitHub バージョン 1 の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 を参照してください。

Important

バージョン 1 アクションの GitHub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、EventBridge ルールの設定や `PollForSourceChanges`パラメータの省略などによって、イベントベースの変更検出とポーリングの両方を使用して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、「のクォータ AWS CodePipeline」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。

   - 次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、次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. 次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。

   - 次のコードをコピーして PollingPipelinesExtractor.sh スクリプトに貼り付けます。

     ```
     #!/bin/bash

     set +x

     POLLING_PIPELINES=()
     LAST_EXECUTED_DATES=()
     ```

```
NEXT_TOKEN=null
HAS_NEXT_TOKEN=true
if [[ $# -eq 0 ]] ; then
    echo 'Please provide region name'
    exit 0
fi
REGION=$1


while [ "$HAS_NEXT_TOKEN" != "false" ]; do
    if [ "$NEXT_TOKEN" != "null" ];
        then
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION --next-token $NEXT_TOKEN)
        else
            LIST_PIPELINES_RESPONSE=$(aws codepipeline list-pipelines --region
 $REGION)
    fi
    LIST_PIPELINES=$(jq -r '.pipelines[].name' <<< "$LIST_PIPELINES_RESPONSE")
    NEXT_TOKEN=$(jq -r '.nextToken' <<< "$LIST_PIPELINES_RESPONSE")
    if [ "$NEXT_TOKEN" == "null" ];
        then
            HAS_NEXT_TOKEN=false
    fi

    for pipline_name in $LIST_PIPELINES
    do
        PIPELINE=$(aws codepipeline get-pipeline --name $pipline_name --region
 $REGION)
        HAS_POLLABLE_ACTIONS=$(jq '.pipeline.stages[].actions[] |
 select(.actionTypeId.category == "Source") | select(.actionTypeId.owner
 == ("ThirdParty","AWS")) | select(.actionTypeId.provider ==
 ("GitHub","S3","CodeCommit")) | select(.configuration.PollForSourceChanges ==
 ("true",null))' <<< "$PIPELINE")
        if [ ! -z "$HAS_POLLABLE_ACTIONS" ];
        then
            POLLING_PIPELINES+=("$pipline_name")
            PIPELINE_EXECUTIONS=$(aws codepipeline list-pipeline-executions --
pipeline-name $pipline_name --region $REGION)
            LAST_EXECUTION=$(jq -r '.pipelineExecutionSummaries[0]' <<<
 "$PIPELINE_EXECUTIONS")
            if [ "$LAST_EXECUTION" != "null" ];
                then
```

```
                    LAST_EXECUTED_TIMESTAMP=$(jq -r '.startTime' <<<
 "$LAST_EXECUTION")
                    LAST_EXECUTED_DATE="$(date -r ${LAST_EXECUTED_TIMESTAMP%.*})"
                else
                    LAST_EXECUTED_DATE="Not executed in last year"
            fi
            LAST_EXECUTED_DATES+=("$LAST_EXECUTED_DATE")
        fi
    done

done

fileName=$REGION-$(date +%s)
printf "| %-30s | %-30s |\n" "Polling Pipeline Name" "Last Executed Time"
printf "| %-30s | %-30s |\n" "_____________________" "___________________"
for i in "${!POLLING_PIPELINES[@]}"; do
  printf "| %-30s | %-30s |\n" "${POLLING_PIPELINES[i]}"
 "${LAST_EXECUTED_DATES[i]}"
  printf "${POLLING_PIPELINES[i]}," >> $fileName.csv
done

printf "\nSaving Polling Pipeline Names to file $fileName.csv."
```

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

```
import boto3
import sys
import time
import math

hasNextToken = True
nextToken = ""
pollablePipelines = []
lastExecutedTimes = []
if len(sys.argv) == 1:
    raise Exception("Please provide region name.")
session = boto3.Session(profile_name='default', region_name=sys.argv[1])
codepipeline = session.client('codepipeline')

def is_pollable_action(action):
    actionTypeId = action['actionTypeId']
    configuration = action['configuration']
    return actionTypeId['owner'] in {"AWS", "ThirdParty"}
 and actionTypeId['provider'] in {"GitHub", "CodeCommit",
```

```
 "S3"} and ('PollForSourceChanges' not in configuration or
 configuration['PollForSourceChanges'] == 'true')

def has_pollable_actions(pipeline):
    hasPollableAction = False
    pipelineDefinition = codepipeline.get_pipeline(name=pipeline['name'])
['pipeline']
    for action in pipelineDefinition['stages'][0]['actions']:
        hasPollableAction = is_pollable_action(action)
        if hasPollableAction:
            break
    return hasPollableAction

def get_last_executed_time(pipelineName):

 pipelineExecutions=codepipeline.list_pipeline_executions(pipelineName=pipelineName)
['pipelineExecutionSummaries']
    if pipelineExecutions:
        return pipelineExecutions[0]['startTime'].strftime("%A %m/%d/%Y, %H:%M:
%S")
    else:
        return "Not executed in last year"

while hasNextToken:
    if nextToken=="":
        list_pipelines_response = codepipeline.list_pipelines()
    else:
        list_pipelines_response =
 codepipeline.list_pipelines(nextToken=nextToken)
    if 'nextToken' in list_pipelines_response:
        nextToken = list_pipelines_response['nextToken']
    else:
        hasNextToken= False
    for pipeline in list_pipelines_response['pipelines']:
        if has_pollable_actions(pipeline):
            pollablePipelines.append(pipeline['name'])
            lastExecutedTimes.append(get_last_executed_time(pipeline['name']))

fileName="{region}-
{timeNow}.csv".format(region=sys.argv[1],timeNow=math.trunc(time.time()))
file = open(fileName, 'w')

print ("{:<30} {:<30} {:<30}".format('Polling Pipeline Name', '|','Last Executed
 Time'))
```

```
print ("{:<30} {:<30} {:<30}".format('_____________________',
 '|','_____________________'))
for i in range(len(pollablePipelines)):
    print("{:<30} {:<30} {:<30}".format(pollablePipelines[i], '|',
 lastExecutedTimes[i]))
    file.write("{pipeline},".format(pipeline=pollablePipelines[i]))
file.close()
print("\nSaving Polling Pipeline Names to file
 {fileName}".format(fileName=fileName))
```

3. パイプラインがあるリージョンごとに、そのリージョンに対してこのスクリプトを実行する必要があります。スクリプトを実行するには、以下のいずれかの操作を行います。

   - 次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。この例では、リージョンは us-west-2 です。

     ```
     ./PollingPipelinesExtractor.sh us-west-2
     ```

   - Python スクリプトの場合は、次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。この例では、リージョンは us-west-2 です。

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

   スクリプトからの以下のサンプル出力では、リージョン us-west-2 からポーリングパイプラインのリストが返され、各パイプラインの最後の実行時間が表示されています。

   ```
    % ./pollingPipelineExtractor.sh us-west-2

   | Polling Pipeline Name | Last Executed Time           |
   | _____________________ | _____________________        |
   | myCodeBuildPipeline   | Wed Mar 8 09:35:49 PST 2023  |
   | myCodeCommitPipeline  | Mon Apr 24 22:32:32 PDT 2023 |
   | TestPipeline          | Not executed in last year    |

   Saving list of polling pipeline names to us-west-2-1682496174.csv...%
   ```

   スクリプトの出力を分析し、リスト内のパイプラインごとに、ポーリングソースを推奨されるイベントベースの変更検出方法に更新します。

> **Note**
>
> ポーリングパイプラインは、PollForSourceChanges パラメータのパイプラインのアクション設定によって決まります。パイプラインソース設定で PollForSourceChangesパラメータが省略されている場合、 CodePipeline はデフォルトでソースの変更についてリポジトリをポーリングします。この動作は、PollForSourceChanges が含まれており、true に設定されている場合と同じです。詳細については、Amazon S3 ソースアクション 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。

ポーリングパイプラインを CodeCommit ソースに移行する

ポーリングパイプラインを移行して、ソースリポジトリまたは Amazon S3 ソースバケットの変更 CodeCommitを検出 EventBridge できます。

CodeCommit -- CodeCommit ソースを持つパイプラインの場合、変更検出が を介して自動化されるようにパイプラインを変更します EventBridge。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 (CodeCommit または Amazon S3 ソース) を移行する (コンソール)
- CLI: ポーリングパイプラインの移行 (CodeCommit ソース) (CLI)
- AWS CloudFormation: 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)

ポーリングパイプライン (CodeCommit または Amazon S3 ソース) を移行する (コンソール)

CodePipeline コンソールを使用してパイプラインを更新し、 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できます。

> **Note**
>
> コンソールを使用して CodeCommit ソースリポジトリまたは Amazon S3 ソースバケットを持つパイプラインを編集すると、ルールと IAM ロールが自動的に作成されます。を使用してパイプライン AWS CLI を編集する場合は、 EventBridge ルールと IAM ロールを自分で作成する必要があります。詳細については、「CodeCommit ソースアクションと EventBridge」を参照してください。

定期的なチェックを使用しているパイプラインを編集するには、これらの手順を使用します。パイプラインを作成する場合は、「でパイプラインを作成する CodePipeline」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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、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. 変更検出オプションを展開し、 CloudWatch イベントの使用を選択して、変更が発生したときにパイプラインを自動的に開始します (推奨) 。

   このパイプライン用に作成される EventBridge ルールを示すメッセージが表示されます。[更新] を選択します。

   Amazon S3 ソースを含むパイプラインを更新する場合は、以下のメッセージが表示されます。[更新] を選択します。
6. パイプラインの編集が終わったら、[パイプラインの変更を保存] を選択して概要ページに戻ります。

   パイプライン用に作成する EventBridge ルールの名前がメッセージに表示されます。[Save and continue] を選択します。
7. アクションをテストするには、 を使用して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 AWS CLI コミットして、変更をリリースします。

ポーリングパイプラインの移行 (CodeCommit ソース) (CLI)

ポーリング (定期的なチェック) を使用して EventBridge ルールを使用してパイプラインを開始するパイプラインを編集するには、次の手順に従います。パイプラインを作成する場合は、「でパイプラインを作成する CodePipeline」を参照してください。

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 CodeCommit、パイプラインの `PollForSourceChanges`パラメータを編集し、次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 パラメータのデフォルト設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、`MyFirstPipeline` という名前のパイプラインに対して、以下のコマンドを実行します。

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

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。
2. 任意のプレーンテキストエディタで JSON ファイルを開き、以下に示しているように、`PollForSourceChanges` パラメータを `false` に変更してソースステージを編集します。

   この変更を行う理由 このパラメータを `false` に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

   ```
   "configuration": {
   ```

```
    "PollForSourceChanges": "false",
    "BranchName": "main",
    "RepositoryName": "MyTestRepo"
},
```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除します。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

   例えば、構造から以下の行を削除します。

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

   ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには **start-pipeline-execution** コマンドを使用します。

をイベントソース CodeCommit 、 をターゲット CodePipeline とする EventBridge ルールを作成するには

1. ルール EventBridge の呼び出しに使用する CodePipeline のアクセス許可を追加します。詳細については、「Amazon のリソースベースのポリシーの使用 EventBridge」を参照してください。

   a. 次のサンプルを使用して、 がサービスロールを引き受け EventBridge ること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 このポリシーをロールに追加すると、 のアクセス許可が作成されます EventBridge。

```
aws iam put-role-policy --role-name Role-for-MyRule --policy-name CodePipeline-
Permissions-Policy-For-EB --policy-document file://permissionspolicyforEB.json
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

この変更を行う理由 このコマンドでは、 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. をターゲット CodePipeline 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
```

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、次のリソースをテンプレートに追加します。

- 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

1. 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、 は AWS CloudFormation のアクセス許可を作成できます EventBridge。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": [
                  "",
                  [
                    "arn:aws:codepipeline:",
                    {
                      "Ref": "AWS::Region"
                    },
                    ":",
                    {
                      "Ref": "AWS::AccountId"
                    },
                    ":",
                    {
```

```
                    "Ref": "AppPipeline"
                  }
                ]

...
```

2. 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。がリポジトリの状態の変更 EventBridge を検出すると、ルールはターゲットパイプラインStartPipelineExecutionで 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": [
            "",
            [
              "arn:aws:codecommit:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "RepositoryName"
              }
            ]
          ]
        }
      ],
      "detail": {
        "event": [
          "referenceCreated",
          "referenceUpdated"
        ],
        "referenceType": [
          "branch"
        ],
        "referenceName": [
```

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

3. 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] を選択します。

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 パラメータのデフォルト設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 このパラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

  YAML

```
Name: Source
Actions:
  -
    Name: SourceAction
    ActionTypeId:
      Category: Source
      Owner: AWS
      Version: 1
      Provider: CodeCommit
    OutputArtifacts:
      - Name: SourceOutput
    Configuration:
      BranchName: !Ref BranchName
      RepositoryName: !Ref RepositoryName
      PollForSourceChanges: false
    RunOrder: 1
```

JSON

```
{
  "Name": "Source",
  "Actions": [
    {
      "Name": "SourceAction",
      "ActionTypeId": {
        "Category": "Source",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "CodeCommit"
      },
      "OutputArtifacts": [
        {
          "Name": "SourceOutput"
        }
      ],
      "Configuration": {
        "BranchName": {
          "Ref": "BranchName"
        },
        "RepositoryName": {
          "Ref": "RepositoryName"
        },
        "PollForSourceChanges": false
      },
      "RunOrder": 1
    }
  ]
},
```

Example

でこれらのリソースを作成すると AWS CloudFormation、リポジトリ内のファイルが作成または更新されると、パイプラインがトリガーされます。以下に示しているのは、最終的なテンプレートスニペットです。

YAML

```
Resources:
  EventRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Effect: Allow
            Principal:
              Service:
                - events.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        -
          PolicyName: eb-pipeline-execution
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              -
                Effect: Allow
                Action: codepipeline:StartPipelineExecution
                Resource: !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region',
 ':', !Ref 'AWS::AccountId', ':', !Ref AppPipeline ] ]
  EventRule:
    Type: AWS::Events::Rule
    Properties:
      EventPattern:
        source:
          - aws.codecommit
        detail-type:
          - 'CodeCommit Repository State Change'
        resources:
          - !Join [ '', [ 'arn:aws:codecommit:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref RepositoryName ] ]
        detail:
          event:
            - referenceCreated
            - referenceUpdated
          referenceType:
            - branch
```

```
          referenceName:
            - main
      Targets:
        -
          Arn:
            !Join [ '', [ 'arn:aws:codepipeline:', !Ref 'AWS::Region', ':', !Ref
 'AWS::AccountId', ':', !Ref AppPipeline ] ]
          RoleArn: !GetAtt EventRole.Arn
          Id: codepipeline-AppPipeline
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
      Name: codecommit-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: CodeCommit
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                BranchName: !Ref BranchName
                RepositoryName: !Ref RepositoryName
                PollForSourceChanges: false
              RunOrder: 1


...
```

JSON

```
    "Resources": {

...
```

```
        "EventRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "events.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "eb-pipeline-execution",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "codepipeline:StartPipelineExecution",
                                    "Resource": {
                                        "Fn::Join": [
                                            "",
                                            [
                                                "arn:aws:codepipeline:",
                                                {
                                                    "Ref": "AWS::Region"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AWS::AccountId"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AppPipeline"
                                                }
                                            ]
```

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

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

```
                "Name": "codecommit-events-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "AWS",
                                    "Version": 1,
                                    "Provider": "CodeCommit"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "BranchName": {
                                        "Ref": "BranchName"
                                    },
                                    "RepositoryName": {
                                        "Ref": "RepositoryName"
                                    },
                                    "PollForSourceChanges": false
                                },
                                "RunOrder": 1
                            }
                        ]
                    },

...
```

イベントに対応した S3 ソースを使用してポーリングパイプラインを移行する

Amazon S3 ソースを持つパイプラインの場合は、変更検出が EventBridge および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して自動化されるようにパイプラインを変更します。これは、CLI または を使用してパイプラインを移行する場合 AWS CloudFormation に推奨される方法です。

> **Note**
>
> これには、イベント通知が有効になっているバケットの使用が含まれます。この場合、別の CloudTrail 証跡を作成する必要はありません。コンソールを使用している場合は、イベントルールと CloudTrail 証跡が自動的に設定されます。これらの手順については、「S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する」を参照してください。

- CLI: S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する (CLI)
- AWS CloudFormation: 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 (AWS CloudFormation テンプレート)

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 EventBridge でイベントを使用するパイプラインを編集するには、次の手順に従います。パイプラインを作成する場合は、「でパイプラインを作成する CodePipeline」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 イベントルール
- EventBridge 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

Amazon S3 をイベントソースおよびターゲット CodePipeline とする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. ルールの呼び出し EventBridge に使用するアクセス許可 CodePipeline を に付与します。詳細については、「Amazon のリソースベースのポリシーの使用 EventBridge」を参照してください。

a. 次のサンプルを使用して、 がサービスロールを引き受けること EventBridge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 この信頼ポリシーをロールに追加すると、 のアクセス許可が作成されます EventBridge。

```
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
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

   次のサンプルコマンドでは、EnabledS3SourceRule という名前のルールが作成されます。

```
aws events put-rule --name "EnabledS3SourceRule" --event-pattern "{\"source\":[\"aws.s3\"],\"detail-type\":[\"Object Created\"],\"detail\":{\"bucket\":{\"name\":[\"my-bucket\"]}}}" --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. をターゲット CodePipeline 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 パラメータのデフォルト設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

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

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. この例に示すように、プレーンテキストエディタでJSONファイルを開き、storage-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": "storage-bucket",
    "PollForSourceChanges": "false",
    "S3ObjectKey": "index.zip"
},
```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除する必要があります。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

例えば、構造から以下の行を削除します。

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

ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

イベント (AWS CloudFormation テンプレート) で 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

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。

- EventBridge このイベントがパイプラインを開始できるようにする 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


...
```

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 をイベントソースおよびターゲット CodePipeline とする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. テンプレートの でResources、 AWS::IAM::Role 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 のアクセス許可を作成できます EventBridge。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": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]
```

```
...
```

2. 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": [
            "",
            [
              "arn:aws:codepipeline:",
              {
                "Ref": "AWS::Region"
              },
              ":",
              {
                "Ref": "AWS::AccountId"
              },
              ":",
              {
                "Ref": "AppPipeline"
              }
            ]
```

```
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. 更新したテンプレートをローカルコンピュータに保存し、 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] を選択します。

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 パラメータのデフォルト設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

この変更を行う理由 PollForSourceChanges パラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

YAML

```
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              - Name: SourceOutput
            Configuration:
              S3Bucket: !Ref SourceBucket
              S3ObjectKey: !Ref SourceObjectKey
              PollForSourceChanges: false
            RunOrder: 1
```

JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
       "Ref": "SourceBucket"
     },
     "S3ObjectKey": {
```

```
        "Ref": "SourceObjectKey"
      },
      "PollForSourceChanges": false
    },
    "RunOrder": 1
  }
```

Example

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
```

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": [
                                    "",
                                    [
                                        {
                                            "Fn::GetAtt": [
                                                "CodePipelineArtifactStoreBucket",
                                                "Arn"
                                            ]
                                        },
```

```
                                    "/*"
                                ]
                            ]
                        },
                        "Condition": {
                            "StringNotEquals": {
                                "s3:x-amz-server-side-encryption": "aws:kms"
                            }
                        }
                    },
                    {
                        "Sid": "DenyInsecureConnections",
                        "Effect": "Deny",
                        "Principal": "*",
                        "Action": "s3:*",
                        "Resource": {
                            "Fn::Join": [
                                "",
                                [
                                    {
                                        "Fn::GetAtt": [
                                            "CodePipelineArtifactStoreBucket",
                                            "Arn"
                                        ]
                                    },
                                    "/*"
                                ]
                            ]
                        },
                        "Condition": {
                            "Bool": {
                                "aws:SecureTransport": false
                            }
                        }
                    }
                ]
            }
        }
    },
    "CodePipelineServiceRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
```

```
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": [
                                "codepipeline.amazonaws.com"
                            ]
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Path": "/",
            "Policies": [
                {
                    "PolicyName": "AWS-CodePipeline-Service-3",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "codecommit:CancelUploadArchive",
                                    "codecommit:GetBranch",
                                    "codecommit:GetCommit",
                                    "codecommit:GetUploadArchiveStatus",
                                    "codecommit:UploadArchive"
                                ],
                                "Resource": "resource_ARN"
                            },
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "codedeploy:CreateDeployment",
                                    "codedeploy:GetApplicationRevision",
                                    "codedeploy:GetDeployment",
                                    "codedeploy:GetDeploymentConfig",
                                    "codedeploy:RegisterApplicationRevision"
                                ],
                                "Resource": "resource_ARN"
                            },
                            {
                                "Effect": "Allow",
                                "Action": [
```

```
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "devicefarm:ListProjects",
                "devicefarm:ListDevicePools",
                "devicefarm:GetRun",
                "devicefarm:GetUpload",
                "devicefarm:CreateUpload",
                "devicefarm:ScheduleRun"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticbeanstalk:*",
                "ec2:*",
                "elasticloadbalancing:*",
                "autoscaling:*",
                "cloudwatch:*",
                "s3:*",
                "sns:*",
                "cloudformation:*",
                "rds:*",
```

```
                                    "sqs:*",
                                    "ecs:*"
                                ],
                                "Resource": "resource_ARN"
                            }
                        ]
                    }
                }
            ]
        }
    },
    "AppPipeline": {
        "Type": "AWS::CodePipeline::Pipeline",
        "Properties": {
            "Name": "s3-events-pipeline",
            "RoleArn": {
                "Fn::GetAtt": [
                    "CodePipelineServiceRole",
                    "Arn"
                ]
            },
            "Stages": [
                {
                    "Name": "Source",
                    "Actions": [
                        {
                            "Name": "SourceAction",
                            "ActionTypeId": {
                                "Category": "Source",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "S3"
                            },
                            "OutputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "Configuration": {
                                "S3Bucket": {
                                    "Ref": "SourceBucket"
                                },
                                "S3ObjectKey": {
                                    "Ref": "SourceObjectKey"
```

```
                                },
                                "PollForSourceChanges": false
                            },
                            "RunOrder": 1
                        }
                    ]
                },
                {
                    "Name": "Beta",
                    "Actions": [
                        {
                            "Name": "BetaAction",
                            "InputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "ActionTypeId": {
                                "Category": "Deploy",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "CodeDeploy"
                            },
                            "Configuration": {
                                "ApplicationName": {
                                    "Ref": "ApplicationName"
                                },
                                "DeploymentGroupName": {
                                    "Ref": "BetaFleet"
                                }
                            },
                            "RunOrder": 1
                        }
                    ]
                }
            ],
            "ArtifactStore": {
                "Type": "S3",
                "Location": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                }
            }
        }
    },
```

```
        "EventRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "events.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "eb-pipeline-execution",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "codepipeline:StartPipelineExecution",
                                    "Resource": {
                                        "Fn::Join": [
                                            "",
                                            [
                                                "arn:aws:codepipeline:",
                                                {
                                                    "Ref": "AWS::Region"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AWS::AccountId"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AppPipeline"
                                                }
                                            ]
```

```
                                        ]
                                    }
                                }
                            ]
                        }
                    }
                ]
            }
        },
        "EventRule": {
            "Type": "AWS::Events::Rule",

            "Properties": {
                "EventBusName": "default",
          "EventPattern": {
              "source": [
            "aws.s3"
          ],
                "detail-type": [
        "Object Created"
          ],
          "detail": {
        "bucket": {
                        "name": [
                            {
                            "Ref": "SourceBucket"
                            }
                        ]
                    }
                }
},
"Name": "EnabledS3SourceRule",
        "State": "ENABLED",
                "Targets": [
                    {
                        "Arn": {
                            "Fn::Join": [
                                "",
                                [
                                    "arn:aws:codepipeline:",
                                    {
                                        "Ref": "AWS::Region"
                                    },
                                    ":",
```

```
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                ":",
                                {
                                    "Ref": "AppPipeline"
                                }
                            ]
                        ]
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
    }
}
```

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 ソースを持つパイプラインの場合は、変更検出が を介して自動化されるようにパイプラインを変更します EventBridge。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 (CodeCommit または Amazon S3 ソース) を移行する (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 証跡を使用してポーリングパイプラインを移行する (AWS CloudFormation テンプレート)

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 EventBridge でイベントを使用するパイプラインを編集するには、次の手順に従います。パイプラインを作成する場合は、「でパイプラインを作成する CodePipeline」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 イベントがパイプラインを開始できるようにする IAM ロール

AWS CloudTrail 証跡を作成して ログ記録を有効にするには

を使用して証跡 AWS CLI を作成するには、 create-trail コマンドを呼び出し、以下を指定します。

- 証跡名。
- AWS CloudTrailにバケットポリシーをすでに適用しているバケットです。

詳細については、AWS 「コマンドラインインターフェイス を使用した証跡の作成」を参照してください。

1. create-trail コマンドを呼び出し、`--name` および `--s3-bucket-name` パラメータを含めます。

   この変更を行う理由 これにより、 CloudTrailS3 ソースバケットに必要な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、`myBucket` 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 myBucket
   ```

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 このコマンドは、ソースバケットの CloudTrail ログ記録を開始し、イベントを に送信します EventBridge。

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口グ記録を開始します。

```
aws cloudtrail start-logging --name my-trail
```

3. put-event-selectors コマンドを呼び出し、--trail-name および --event-selectors パラメータを含めます。イベントセレクタを使用して、証跡がソースバケットのデータイベントを口グに記録し、イベントを EventBridge ルールに送信するように指定します。

   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 myBucket/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:::myBucket/
myFolder/file.zip"] }] }]'
```

Amazon S3 をイベントソースおよびターゲット CodePipeline とする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. ルールの呼び出し EventBridge に使用するアクセス許可 CodePipeline を に付与します。詳細については、「Amazon のリソースベースのポリシーの使用 EventBridge」を参照してください。

   a. 次のサンプルを使用して、 がサービスロールを引き受けること EventBridge 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 この信頼ポリシーをロールに追加すると、 のアクセス許可が作成されます EventBridge。

```
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
```

2. put-rule コマンドを呼び出し、--name、--event-pattern、--role-arn パラメータを含めます。

   次のサンプルコマンドでは、MyS3SourceRule という名前のルールが作成されます。

```
aws events put-rule --name "MyS3SourceRule" --event-pattern "{\"source\":
[\"aws.s3\"],\"detail-type\":[\"AWS API Call via CloudTrail\"],\"detail\":
{\"eventSource\":[\"s3.amazonaws.com\"],\"eventName\":[\"CopyObject\",\"PutObject
\",\"CompleteMultipartUpload\"],\"requestParameters\":{\"bucketName\":[\"my-bucket
\"],\"key\":[\"my-key\"]}}}
 --role-arn "arn:aws:iam::ACCOUNT_ID:role/Role-for-MyRule"
```

3. をターゲット CodePipeline 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
```

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 パラメータのデフォルト設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインに対して、以下のコマンドを実行します。

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

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. この例に示すように、プレーンテキストエディタでJSONファイルを開き、storage-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": "storage-bucket",
    "PollForSourceChanges": "false",
    "S3ObjectKey": "index.zip"
},
```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから metadata 行を削除する必要があります。それ以外の場合は、update-pipeline コマンドで使用することはできません。"metadata": { } 行と、"created"、"pipelineARN"、"updated" フィールドを削除します。

   例えば、構造から以下の行を削除します。

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

   ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する (AWS CloudFormation テンプレート)

以下の手順を使用して、Amazon S3 ソースを含むパイプラインを、ポーリングからイベントベースの変更検出に編集します。

Amazon S3 でイベント駆動型パイプラインを構築するには、パイプラインの `PollForSourceChanges` パラメータを編集してから、以下のリソースをテンプレートに追加します。

- EventBridge では、すべての Amazon S3 イベントをログに記録する必要があります。発生するイベントのロク記録に Amazon S3 が使用できる AWS CloudTrail 証跡、バケット、バケットポリシーを作成する必要があります。詳細については、「 証跡のデータイベント 」と「 管理イベントのログ記録 」を参照してください。
- EventBridge このイベントがパイプラインを開始できるようにする 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


...
```

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 をイベントソースおよびターゲット CodePipeline とする EventBridge ルールを作成し、アクセス許可ポリシーを適用するには

1. テンプレートの でResources、 AWS::IAM::Role 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 のアクセス許可を作成できます EventBridge。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": [
                    "",
                    [
                      "arn:aws:codepipeline:",
                      {
                        "Ref": "AWS::Region"
                      },
                      ":",
                      {
                        "Ref": "AWS::AccountId"
                      },
                      ":",
                      {
                        "Ref": "AppPipeline"
                      }
                    ]
                  ]
```

```
...
```

2. 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": [
              "",
              [
                "arn:aws:codepipeline:",
                {
                  "Ref": "AWS::Region"
                },
                ":",
                {
                  "Ref": "AWS::AccountId"
                },
                ":",
                {
                  "Ref": "AppPipeline"
                }
              ]
            ]
          },
          "RoleArn": {
            "Fn::GetAtt": [
              "EventRole",
              "Arn"
            ]
          },
          "Id": "codepipeline-AppPipeline"
        }
      ]
    }
  }
},

...
```

3. このスニペットを最初のテンプレートに追加して、クロススタック機能を有効にします。

   YAML

```
Outputs:
  SourceBucketARN:
    Description: "S3 bucket ARN that Cloudtrail will use"
    Value: !GetAtt SourceBucket.Arn
    Export:
      Name: SourceBucketARN
```

JSON

```
  "Outputs" : {
    "SourceBucketARN" : {
      "Description" : "S3 bucket ARN that Cloudtrail will use",
      "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
      "Export" : {
        "Name" : "SourceBucketARN"
      }
    }

...
```

4. 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。
6. 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。
7. [実行] を選択します。

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 パラメータのデフォルト設定」を参照してください

- テンプレートで、PollForSourceChanges を false に変更します。パイプライン定義に PollForSourceChanges が含まれていなかった場合は、追加して false に設定します。

  この変更を行う理由 PollForSourceChanges パラメータを false に変更すると、定期的チェックがオフになるため、イベントベースの変更検出のみ使用することができます。

YAML

```
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: AWS
                Version: 1
                Provider: S3
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                S3Bucket: !Ref SourceBucket
                S3ObjectKey: !Ref SourceObjectKey
                PollForSourceChanges: false
              RunOrder: 1
```

JSON

```
{
   "Name": "SourceAction",
   "ActionTypeId": {
     "Category": "Source",
     "Owner": "AWS",
     "Version": 1,
     "Provider": "S3"
   },
   "OutputArtifacts": [
     {
       "Name": "SourceOutput"
     }
   ],
   "Configuration": {
     "S3Bucket": {
       "Ref": "SourceBucket"
     },
     "S3ObjectKey": {
       "Ref": "SourceObjectKey"
     },
     "PollForSourceChanges": false
```

```
    },
    "RunOrder": 1
  }
```

Amazon S3 パイプラインのリソース用に 2 番目のテンプレートを作成するには CloudTrail

- 別のテンプレートの でResources、、AWS::S3::Bucket、および AWS::CloudTrail::Trail AWS CloudFormation リソースを使用してAWS::S3::BucketPolicy、 のシンプルなバケット定義と証跡を提供します CloudTrail。

  この変更を行う理由 現在の 1 アカウントあたり 5 つの証跡の制限を考慮すると、 CloudTrail 証跡は個別に作成および管理する必要があります。(「 の制限 AWS CloudTrail」を参照してください。) ただし、1 つの証跡に複数の Amazon S3 バケットを含めることができるため、いったん証跡を作成してから、必要に応じて他のパイプライン用に Amazon S3 バケットを追加できます。2 番目のサンプルテンプレートファイルに以下のコードを貼り付けます。

  YAML

```
###########################################################################################
# Prerequisites:
#   - S3 SourceBucket and SourceObjectKey must exist
###########################################################################################

Parameters:
  SourceObjectKey:
    Description: 'S3 source artifact'
    Type: String
    Default: SampleApp_Linux.zip

Resources:
  AWSCloudTrailBucketPolicy:
    Type: AWS::S3::BucketPolicy
    Properties:
      Bucket: !Ref AWSCloudTrailBucket
      PolicyDocument:
        Version: 2012-10-17
        Statement:
          -
            Sid: AWSCloudTrailAclCheck
            Effect: Allow
```

```
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:GetBucketAcl
            Resource: !GetAtt AWSCloudTrailBucket.Arn
          -
            Sid: AWSCloudTrailWrite
            Effect: Allow
            Principal:
              Service:
                - cloudtrail.amazonaws.com
            Action: s3:PutObject
            Resource: !Join [ '', [ !GetAtt AWSCloudTrailBucket.Arn, '/
AWSLogs/', !Ref 'AWS::AccountId', '/*' ] ]
            Condition:
              StringEquals:
                s3:x-amz-acl: bucket-owner-full-control
  AWSCloudTrailBucket:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
  AwsCloudTrail:
    DependsOn:
      - AWSCloudTrailBucketPolicy
    Type: AWS::CloudTrail::Trail
    Properties:
      S3BucketName: !Ref AWSCloudTrailBucket
      EventSelectors:
        -
          DataResources:
            -
              Type: AWS::S3::Object
              Values:
                - !Join [ '', [ !ImportValue SourceBucketARN, '/', !Ref
 SourceObjectKey ] ]
          ReadWriteType: WriteOnly
          IncludeManagementEvents: false
      IncludeGlobalServiceEvents: true
      IsLogging: true
      IsMultiRegionTrail: true


...
```

JSON

```
{
  "Parameters": {
    "SourceObjectKey": {
      "Description": "S3 source artifact",
      "Type": "String",
      "Default": "SampleApp_Linux.zip"
    }
  },
  "Resources": {
    "AWSCloudTrailBucket": {
      "Type": "AWS::S3::Bucket",
        "DeletionPolicy": "Retain"
    },
    "AWSCloudTrailBucketPolicy": {
      "Type": "AWS::S3::BucketPolicy",
      "Properties": {
        "Bucket": {
          "Ref": "AWSCloudTrailBucket"
        },
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Sid": "AWSCloudTrailAclCheck",
              "Effect": "Allow",
              "Principal": {
                "Service": [
                  "cloudtrail.amazonaws.com"
                ]
              },
              "Action": "s3:GetBucketAcl",
              "Resource": {
                "Fn::GetAtt": [
                  "AWSCloudTrailBucket",
                  "Arn"
                ]
              }
            },
            {
              "Sid": "AWSCloudTrailWrite",
              "Effect": "Allow",
```

```
            "Principal": {
              "Service": [
                "cloudtrail.amazonaws.com"
              ]
            },
            "Action": "s3:PutObject",
            "Resource": {
              "Fn::Join": [
                "",
                [
                  {
                    "Fn::GetAtt": [
                      "AWSCloudTrailBucket",
                      "Arn"
                    ]
                  },
                  "/AWSLogs/",
                  {
                    "Ref": "AWS::AccountId"
                  },
                  "/*"
                ]
              ]
            },
            "Condition": {
              "StringEquals": {
                "s3:x-amz-acl": "bucket-owner-full-control"
              }
            }
          }
        ]
      }
    }
  },
  "AwsCloudTrail": {
    "DependsOn": [
      "AWSCloudTrailBucketPolicy"
    ],
    "Type": "AWS::CloudTrail::Trail",
    "Properties": {
      "S3BucketName": {
        "Ref": "AWSCloudTrailBucket"
      },
      "EventSelectors": [
```

```
          {
            "DataResources": [
              {
                "Type": "AWS::S3::Object",
                "Values": [
                  {
                    "Fn::Join": [
                      "",
                      [
                        {
                          "Fn::ImportValue": "SourceBucketARN"
                        },
                        "/",
                        {
                          "Ref": "SourceObjectKey"
                        }
                      ]
                    ]
                  }
                ]
              }
            ],
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": false
          }
        ],
        "IncludeGlobalServiceEvents": true,
        "IsLogging": true,
        "IsMultiRegionTrail": true
      }
    }
  }
}

...
```

Example

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
```

JSON

```
"Resources": {
    "SourceBucket": {
        "Type": "AWS::S3::Bucket",
        "Properties": {
            "VersioningConfiguration": {
                "Status": "Enabled"
            }
        }
    },
    "CodePipelineArtifactStoreBucket": {
        "Type": "AWS::S3::Bucket"
    },
    "CodePipelineArtifactStoreBucketPolicy": {
        "Type": "AWS::S3::BucketPolicy",
        "Properties": {
            "Bucket": {
                "Ref": "CodePipelineArtifactStoreBucket"
            },
            "PolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Sid": "DenyUnEncryptedObjectUploads",
                        "Effect": "Deny",
                        "Principal": "*",
                        "Action": "s3:PutObject",
                        "Resource": {
                            "Fn::Join": [
                                "",
                                [
                                    {
                                        "Fn::GetAtt": [
                                            "CodePipelineArtifactStoreBucket",
                                            "Arn"
                                        ]
                                    },
```

```
                                    "/*"
                                ]
                            ]
                        },
                        "Condition": {
                            "StringNotEquals": {
                                "s3:x-amz-server-side-encryption": "aws:kms"
                            }
                        }
                    },
                    {
                        "Sid": "DenyInsecureConnections",
                        "Effect": "Deny",
                        "Principal": "*",
                        "Action": "s3:*",
                        "Resource": {
                            "Fn::Join": [
                                "",
                                [
                                    {
                                        "Fn::GetAtt": [
                                            "CodePipelineArtifactStoreBucket",
                                            "Arn"
                                        ]
                                    },
                                    "/*"
                                ]
                            ]
                        },
                        "Condition": {
                            "Bool": {
                                "aws:SecureTransport": false
                            }
                        }
                    }
                ]
            }
        }
    },
    "CodePipelineServiceRole": {
        "Type": "AWS::IAM::Role",
        "Properties": {
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
```

```
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "codepipeline.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "AWS-CodePipeline-Service-3",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "codecommit:CancelUploadArchive",
                                        "codecommit:GetBranch",
                                        "codecommit:GetCommit",
                                        "codecommit:GetUploadArchiveStatus",
                                        "codecommit:UploadArchive"
                                    ],
                                    "Resource": "resource_ARN"
                                },
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "codedeploy:CreateDeployment",
                                        "codedeploy:GetApplicationRevision",
                                        "codedeploy:GetDeployment",
                                        "codedeploy:GetDeploymentConfig",
                                        "codedeploy:RegisterApplicationRevision"
                                    ],
                                    "Resource": "resource_ARN"
                                },
                                {
                                    "Effect": "Allow",
                                    "Action": [
```

```
                "codebuild:BatchGetBuilds",
                "codebuild:StartBuild"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "devicefarm:ListProjects",
                "devicefarm:ListDevicePools",
                "devicefarm:GetRun",
                "devicefarm:GetUpload",
                "devicefarm:CreateUpload",
                "devicefarm:ScheduleRun"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "resource_ARN"
        },
        {
            "Effect": "Allow",
            "Action": [
                "elasticbeanstalk:*",
                "ec2:*",
                "elasticloadbalancing:*",
                "autoscaling:*",
                "cloudwatch:*",
                "s3:*",
                "sns:*",
                "cloudformation:*",
                "rds:*",
```

```
                                "sqs:*",
                                "ecs:*"
                            ],
                            "Resource": "resource_ARN"
                        }
                    ]
                }
            }
        ]
    }
},
"AppPipeline": {
    "Type": "AWS::CodePipeline::Pipeline",
    "Properties": {
        "Name": "s3-events-pipeline",
        "RoleArn": {
            "Fn::GetAtt": [
                "CodePipelineServiceRole",
                "Arn"
            ]
        },
        "Stages": [
            {
                "Name": "Source",
                "Actions": [
                    {
                        "Name": "SourceAction",
                        "ActionTypeId": {
                            "Category": "Source",
                            "Owner": "AWS",
                            "Version": 1,
                            "Provider": "S3"
                        },
                        "OutputArtifacts": [
                            {
                                "Name": "SourceOutput"
                            }
                        ],
                        "Configuration": {
                            "S3Bucket": {
                                "Ref": "SourceBucket"
                            },
                            "S3ObjectKey": {
                                "Ref": "SourceObjectKey"
```

```
                                },
                                "PollForSourceChanges": false
                            },
                            "RunOrder": 1
                        }
                    ]
                },
                {
                    "Name": "Beta",
                    "Actions": [
                        {
                            "Name": "BetaAction",
                            "InputArtifacts": [
                                {
                                    "Name": "SourceOutput"
                                }
                            ],
                            "ActionTypeId": {
                                "Category": "Deploy",
                                "Owner": "AWS",
                                "Version": 1,
                                "Provider": "CodeDeploy"
                            },
                            "Configuration": {
                                "ApplicationName": {
                                    "Ref": "ApplicationName"
                                },
                                "DeploymentGroupName": {
                                    "Ref": "BetaFleet"
                                }
                            },
                            "RunOrder": 1
                        }
                    ]
                }
            ],
            "ArtifactStore": {
                "Type": "S3",
                "Location": {
                    "Ref": "CodePipelineArtifactStoreBucket"
                }
            }
        }
    },
```

```
        "EventRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": [
                                    "events.amazonaws.com"
                                ]
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "eb-pipeline-execution",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "codepipeline:StartPipelineExecution",
                                    "Resource": {
                                        "Fn::Join": [
                                            "",
                                            [
                                                "arn:aws:codepipeline:",
                                                {
                                                    "Ref": "AWS::Region"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AWS::AccountId"
                                                },
                                                ":",
                                                {
                                                    "Ref": "AppPipeline"
                                                }
                                            ]
```

```
                                    ]
                                }
                            }
                        ]
                    }
                }
            ]
        }
    },
    "EventRule": {
        "Type": "AWS::Events::Rule",
        "Properties": {
            "EventPattern": {
                "source": [
                    "aws.s3"
                ],
                "detail-type": [
                    "AWS API Call via CloudTrail"
                ],
                "detail": {
                    "eventSource": [
                        "s3.amazonaws.com"
                    ],
                    "eventName": [
                        "PutObject",
                        "CompleteMultipartUpload"
                    ],
                    "resources": {
                        "ARN": [
                            {
                                "Fn::Join": [
                                    "",
                                    [
                                        {
                                            "Fn::GetAtt": [
                                                "SourceBucket",
                                                "Arn"
                                            ]
                                        },
                                        "/",
                                        {
                                            "Ref": "SourceObjectKey"
                                        }
                                    ]
```

```
                                ]
                            }
                        ]
                    }
                }
            },
            "Targets": [
                {
                    "Arn": {
                        "Fn::Join": [
                            "",
                            [
                                "arn:aws:codepipeline:",
                                {
                                    "Ref": "AWS::Region"
                                },
                                ":",
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                ":",
                                {
                                    "Ref": "AppPipeline"
                                }
                            ]
                        ]
                    },
                    "RoleArn": {
                        "Fn::GetAtt": [
                            "EventRole",
                            "Arn"
                        ]
                    },
                    "Id": "codepipeline-AppPipeline"
                }
            ]
        }
    }
},
"Outputs" : {
    "SourceBucketARN" : {
        "Description" : "S3 bucket ARN that Cloudtrail will use",
        "Value" : { "Fn::GetAtt": ["SourceBucket", "Arn"] },
        "Export" : {
```

```
                "Name" : "SourceBucketARN"
            }
        }
    }
}


...
```

GitHub バージョン 1 の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 バージョン 1 の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、バージョン 1 のソースアクションを使用する GitHubパイプラインに推奨される変更検出方法です。

GitHub バージョン 1 のソースアクションを持つパイプラインの場合は、 を使用して変更検出を自動化するように、パイプラインを変更して GitHub バージョン 2 のアクションを使用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、「GitHub 接続」を参照してください。

への接続を作成する GitHub （コンソール）

コンソールを使用して、 への接続を作成できます GitHub。

ステップ 1: バージョン 1 GitHub アクションを置き換える

パイプラインの編集ページを使用して、バージョン 1 GitHub アクションをバージョン 2 GitHub アクションに置き換えます。

バージョン 1 GitHub アクションを置き換えるには

1. CodePipeline コンソールにサインインします。
2. パイプラインを選択し、[編集] を選択します。ソースステージで、[ステージを編集] を選択します。アクションを更新することを推奨するメッセージが表示されます。
3. アクションプロバイダー で、 GitHub （バージョン 2) を選択します。
4. 次のいずれかを行います。

- 「接続」で、プロバイダーへの接続をまだ作成していない場合は、「 に接続する GitHub」を選択します。ステップ 2: への接続を作成するに進みます GitHub。
- [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: への接続を作成する GitHub

接続の作成を選択すると、Connect to GitHubページが表示されます。

への接続を作成するには GitHub

1. GitHub 接続設定 では、接続名 が接続名 に表示されます。

   GitHub アプリ で、アプリのインストールを選択するか、新しいアプリのインストールを選択して作成します。

   > **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 アプリを既にインストールしている場合は、アプリを選択してこのステップをスキップします。

2. の認証ページ GitHub が表示されたら、認証情報を使用してログインし、続行することを選択します。
3. アプリのインストールページで、 AWS CodeStar アプリが GitHub アカウントに接続しようとしていることを示すメッセージが表示されます。

   > **Note**
   >
   > アプリは GitHub アカウントごとに 1 回だけインストールします。アプリケーションをインストール済みである場合は、[Configure] (設定) を選択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 ページに、新しいインストールの接続 ID が表示されます。[接続]を選択します。

ステップ 3: GitHub 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 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デフォルトの を選択します。アクションは GitHub リポジトリからファイルにアクセスし、アーティファクトをパイプラインアーティファクトストアの ZIP ファイルに保存します。
   - 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、「」に示すように、 CodeBuild プロジェクトサービスロールのアクセス許可を更新する必要がありますBitbucket、、 GitHub Enterprise Server GitHub、または GitLab.com への接続の CodeBuild GitClone アクセス許可を追加する。フルクローン オプションの使い方を紹介したチュートリアルは、チュートリアル: GitHub パイプラインソースでフルクローンを使用する をご覧ください。

3. 出力アーティファクト　の場合、`SourceArtifact` のようにこのアクションの出力アーティファクトの名前を保持できます。[Done] を選択して、[アクションを編集] ページを閉じます。
4. [Done] を選択して、ステージの編集ページを閉じます。[Save] を選択して、パイプラインの編集ページを閉じます。

（ GitHub CLI) への接続を作成する

AWS Command Line Interface（ AWS CLI) を使用して、 への接続を作成できます GitHub。

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。

への接続を作成するには GitHub

1. ターミナル (Linux/macOS/Unix) またはコマンドプロンプト (Windows) を開きます。 AWS CLI を使用して create-connection コマンドを実行し、接続--connection-nameに --provider-typeと を指定します。この例では、サードパーティープロバイダー名は GitHub で、指定された接続名は MyConnection です。

   ```
   aws codeconnections create-connection --provider-type GitHub --connection-name
    MyConnection
   ```

   成功した場合、このコマンドは次のような接続 ARN 情報を返します。

   ```
   {
       "ConnectionArn": "arn:aws:codeconnections:us-west-2:account_id:connection/
   aEXAMPLE-8aad-4d5d-8878-dfcab0bc441f"
   }
   ```

2. コンソールを使用して接続を完了します。

GitHub バージョン 1 の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、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。このウェブフックへの移行は、 GitHub バージョン 1 アクションのみを対象としています。

- 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 バージョン 1 のソースアクション) (コンソール)
- CLI: ポーリングパイプラインをウェブフックに移行する (GitHub バージョン 1 のソースアクション) (CLI)
- AWS CloudFormation: プッシュイベント (GitHub バージョン 1 ソースアクション) のパイプラインを更新する (AWS CloudFormation テンプレート)

ポーリングパイプラインをウェブフックに移行する (GitHub バージョン 1 のソースアクション) (コンソール)

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 CodeCommit 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編集して EventBridge 代わりに使用するには、次の手順に従います。パイプラインを作成する場合は、「でパイプラインを作成する CodePipeline」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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、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. 変更検出オプションを展開し、Amazon CloudWatch Events を使用して変更が発生したときにパイプラインを自動的に開始する (推奨) を選択します。

   でソースの変更を検出するためのウェブフック CodePipeline を作成するよう指示 GitHub するメッセージが表示されます。ウェブフックが自動的に AWS CodePipeline 作成されます。以下のオプションでオプトアウトできます。[更新] を選択します。ウェブフックに加えて、 は以下 CodePipeline を作成します。

- シークレット。ランダムに生成され、 への接続を認証するために使用されます GitHub。
- 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 に登録します GitHub。これにより、レポジトリのイベントを受信するための URL がサブスクライブされます。

6. パイプラインの編集が終わったら、[パイプラインの変更を保存] を選択して概要ページに戻ります。

   パイプラインに対して作成されるウェブフックの名前を示すメッセージが表示されます。[Save and continue] を選択します。

7. アクションをテストするには、 を使用して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 AWS CLI コミットして、変更をリリースします。

ポーリングパイプラインをウェブフックに移行する (GitHub バージョン 1 のソースアクション) (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、「でパイプラインを作成する CodePipeline」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 Webhook および認証パラメータ

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 デフォルト

の詳細については、「」を参照してください PollForSourceChanges パラメータのデフォルト設定。

1. テキストエディタで、作成するウェブフックの JSON ファイルを作成して保存します。「my-webhook」という名前のウェブフックには、このサンプルを使用します。

```
{
    "webhook": {
        "name": "my-webhook",
  "targetPipeline": "pipeline_name",
  "targetAction": "source_action_name",
  "filters": [{
      "jsonPath": "$.ref",
      "matchEquals": "refs/heads/{Branch}"
  }],
  "authentication": "GITHUB_HMAC",
  "authenticationConfiguration": {
      "SecretToken": "secret"
  }
    }
}
```

2. put-webhook コマンドを呼び出し、--cli-input および --region パラメータを含めます。

   次のサンプルコマンドは、webhook_json JSON ファイルで ウェブフックを作成します。

```
aws codepipeline put-webhook --cli-input-json file://webhook_json.json --region
 "eu-central-1"
```

3. この例に示す出力では、my-webhook という名前のウェブフックに対して URL と ARN が返されます。

```
{
    "webhook": {
        "url": "https://webhooks.domain.com/
trigger111111111EXAMPLE1111111111111111",
        "definition": {
            "authenticationConfiguration": {
                "SecretToken": "secret"
            },
            "name": "my-webhook",
```

```
            "authentication": "GITHUB_HMAC",
            "targetPipeline": "pipeline_name",
            "targetAction": "Source",
            "filters": [
                {
                    "jsonPath": "$.ref",
                    "matchEquals": "refs/heads/{Branch}"
                }
            ]
        },
        "arn": "arn:aws:codepipeline:eu-central-1:ACCOUNT_ID:webhook:my-webhook"
    },
    "tags": [{
      "key": "Project",
      "value": "ProjectA"
    }]
}
```

この例では、ウェブフックにProjectタグキーとProjectA値を含めることで、ウェブフックにタグ付けを追加します。でのリソースのタグ付けの詳細については CodePipeline、「」を参照してくださいリソースのタグ付け。

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 パラメータのデフォルト設定」を参照してください

1. get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。例えば、MyFirstPipeline という名前のパイプラインの場合は、以下のコマンドを入力します。

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

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のプレーンテキストエディタで JSON ファイルを開き、以下に示しているように、PollForSourceChanges パラメータを変更または追加してソースステージを編集します。この例では、UserGitHubRepo という名前のリポジトリで、パラメータを false に設定します。

この変更を行う理由 このパラメータを変更すると、定期的なチェックがオフになるため、イベントベースの変更検出のみ使用することができます。

```
"configuration": {
    "Owner": "name",
    "Repo": "UserGitHubRepo",
    "PollForSourceChanges": "false",
    "Branch": "main",
    "OAuthToken": "****"
},
```

3. get-pipeline コマンドを使用して取得されたパイプライン構造を操作している場合、ファイルから metadata 行を削除して JSON ファイルの構造を編集する必要があります。それ以外の場合は、update-pipeline コマンドで使用することはできません。JSON ファイルのパイプライン構造から "metadata" セクションを削除します ({ } 行と、"created"、"pipelineARN"、および "updated" フィールド)。

例えば、構造から以下の行を削除します。

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

ファイルを保存します。

4. 変更を適用するには、以下のように、パイプライン JSON ファイルを指定して、 update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには start-pipeline-execution コマンドを使用します。

プッシュイベント (GitHub バージョン 1 ソースアクション) のパイプラインを更新する (AWS CloudFormation テンプレート)

ウェブフックを使用してパイプライン (GitHub ソース付き) を定期的にチェック (ポーリング) からイベントベースの変更検出に更新するには、次の手順に従います。

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、テンプレートに GitHub ウェブフックリソースを追加します。

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"
                                    },
                                    "Branch": {
                                        "Ref": "BranchName"
                                    },
                                    "OAuthToken":
"{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": true
                                },
                                "RunOrder": 1
```

```
                        }
                    ]
                },


...
```

テンプレートにパラメータを追加してウェブフックを作成するには

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、ウェブフックの認証情報に GitHub Secrets Manager への動的な参照を使用します。詳細については、「動的な参照を使用してテンプレート値を指定する」を参照してください。

> **Important**
>
> シークレットパラメータを渡すときは、値をテンプレートに直接入力しないでください。値はプレーンテキストとしてレンダリングされるため、読み取り可能です。セキュリティ上の理由から、 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、`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 デフォルトの詳細については、「」を参照してください PollForSourceChanges パラメータのデフォルト設定。

1. テンプレートの `Resources` に、パラメータを追加します。

YAML

```
Parameters:
        GitHubOwner:
          Type: String

...
```

JSON

```
{
 "Parameters": {
  "BranchName": {
   "Description": "GitHub branch name",
   "Type": "String",
   "Default": "main"
  },
  "GitHubOwner": {
   "Type": "String"
  },

...
```

2. AWS::CodePipeline::Webhook 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 に設定されている場合true、 に関連付けられているユーザーが で必要なスコープを設定OAuthTokenできることを確認します GitHub。トークンとウェブフックには、次の GitHub スコープが必要です。

- 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、 は 404 GitHub を返します。返される 404 の詳細については、「https://help.github.com/articles/about-webhooks」を参照してください

YAML

```
  AppPipelineWebhook:
    Type: AWS::CodePipeline::Webhook
    Properties:
      Authentication: GITHUB_HMAC
      AuthenticationConfiguration:
        SecretToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
      Filters:
        -
          JsonPath: "$.ref"
          MatchEquals: refs/heads/{Branch}
      TargetPipeline: !Ref AppPipeline
      TargetAction: SourceAction
      Name: AppPipelineWebhook
      TargetPipelineVersion: !GetAtt AppPipeline.Version
      RegisterWithThirdParty: true


...
```

JSON

```
"AppPipelineWebhook": {
    "Type": "AWS::CodePipeline::Webhook",
    "Properties": {
        "Authentication": "GITHUB_HMAC",
        "AuthenticationConfiguration": {
            "SecretToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
        },
        "Filters": [{
            "JsonPath": "$.ref",
            "MatchEquals": "refs/heads/{Branch}"
        }],
        "TargetPipeline": {
            "Ref": "AppPipeline"
```

```
        },
        "TargetAction": "SourceAction",
        "Name": "AppPipelineWebhook",
        "TargetPipelineVersion": {
            "Fn::GetAtt": [
              "AppPipeline",
              "Version"
            ]
        },
        "RegisterWithThirdParty": true
    }
},

...
```

3. 更新されたテンプレートをローカルコンピュータに保存し、 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] を選択します。

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 パラメータのデフォルト設定」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

でこれらのリソースを作成すると 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: "$.ref"
          MatchEquals: refs/heads/{Branch}
      TargetPipeline: !Ref AppPipeline
      TargetAction: SourceAction
      Name: AppPipelineWebhook
      TargetPipelineVersion: !GetAtt AppPipeline.Version
      RegisterWithThirdParty: true
  AppPipeline:
    Type: AWS::CodePipeline::Pipeline
    Properties:
```

```
      Name: github-events-pipeline
      RoleArn:
        !GetAtt CodePipelineServiceRole.Arn
      Stages:
        -
          Name: Source
          Actions:
            -
              Name: SourceAction
              ActionTypeId:
                Category: Source
                Owner: ThirdParty
                Version: 1
                Provider: GitHub
              OutputArtifacts:
                - Name: SourceOutput
              Configuration:
                Owner: !Ref GitHubOwner
                Repo: !Ref RepositoryName
                Branch: !Ref BranchName
                OAuthToken:
 {{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}
                PollForSourceChanges: false
              RunOrder: 1

...
```

JSON

```
{
    "Parameters": {
        "BranchName": {
            "Description": "GitHub branch name",
            "Type": "String",
            "Default": "main"
        },
        "RepositoryName": {
            "Description": "GitHub repository name",
            "Type": "String",
            "Default": "test"
        },
        "GitHubOwner": {
            "Type": "String"
```

```
        },
        "ApplicationName": {
            "Description": "CodeDeploy application name",
            "Type": "String",
            "Default": "DemoApplication"
        },
        "BetaFleet": {
            "Description": "Fleet configured in CodeDeploy",
            "Type": "String",
            "Default": "DemoFleet"
        }
    },
    "Resources": {

...

        },
        "AppPipelineWebhook": {
            "Type": "AWS::CodePipeline::Webhook",
            "Properties": {
                "Authentication": "GITHUB_HMAC",
                "AuthenticationConfiguration": {
                    "SecretToken": {

 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
                    }
                },
                "Filters": [
                    {
                        "JsonPath": "$.ref",
                        "MatchEquals": "refs/heads/{Branch}"
                    }
                ],
                "TargetPipeline": {
                    "Ref": "AppPipeline"
                },
                "TargetAction": "SourceAction",
                "Name": "AppPipelineWebhook",
                "TargetPipelineVersion": {
                    "Fn::GetAtt": [
                        "AppPipeline",
                        "Version"
                    ]
                },
```

```
                "RegisterWithThirdParty": true
            }
        },
        "AppPipeline": {
            "Type": "AWS::CodePipeline::Pipeline",
            "Properties": {
                "Name": "github-events-pipeline",
                "RoleArn": {
                    "Fn::GetAtt": [
                        "CodePipelineServiceRole",
                        "Arn"
                    ]
                },
                "Stages": [
                    {
                        "Name": "Source",
                        "Actions": [
                            {
                                "Name": "SourceAction",
                                "ActionTypeId": {
                                    "Category": "Source",
                                    "Owner": "ThirdParty",
                                    "Version": 1,
                                    "Provider": "GitHub"
                                },
                                "OutputArtifacts": [
                                    {
                                        "Name": "SourceOutput"
                                    }
                                ],
                                "Configuration": {
                                    "Owner": {
                                        "Ref": "GitHubOwner"
                                    },
                                    "Repo": {
                                        "Ref": "RepositoryName"
                                    },
                                    "Branch": {
                                        "Ref": "BranchName"
                                    },
                                    "OAuthToken":
"{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}",
                                    "PollForSourceChanges": false
                                },
```

```
                                "RunOrder": 1

...
```

CodePipeline サービスロールを作成する

パイプラインの作成時に、サービスロールを作成するか、既存のサービスロールを使用します。

CodePipeline コンソールまたは を使用して CodePipeline サービスロール AWS CLI 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。

AWS CLI を使用してパイプラインを作成する前に、パイプライン CodePipeline の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チュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。

サービスロールは AWS マネージドロールではありませんが、最初にパイプラインの作成用に作成され、次に新しいアクセス許可がサービスロールポリシーに追加されると、パイプラインのサービスロールを更新する必要がある場合があります。サービスロールを使用してパイプラインを作成すると、このパイプラインに別のサービスロールを適用することはできません。推奨されるポリシーをサービスロールにアタッチします。

サービスロールの詳細については、「 CodePipeline サービスロールを管理する」を参照してください。

CodePipeline サービスロールを作成する (コンソール)

コンソールを使用してパイプラインを作成する場合は、パイプライン作成ウィザードを使用して 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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   [パイプラインの作成] を選択し、パイプライン作成ウィザードの [ステップ 1: パイプラインの設定を選択する] を完了します。

> **Note**
>
> パイプラインを作成したら、その名前を変更することはできません。その他の制限についての詳細については、「のクォータ AWS CodePipeline」を参照してください。

2. サービスロール で、IAM で新しいサービスロールを作成 CodePipeline することを許可する新しいサービスロール を選択します。
3. パイプラインの作成を完了します。パイプラインのサービスロールを使用して IAM ロールを一覧表示できます。また、`get-pipeline` CLI で AWS コマンドを実行して、パイプラインに関連付けられているサービスロールの ARN を表示できます。

CodePipeline サービスロールを作成する (CLI)

AWS CLI または を使用してパイプラインを作成する前に AWS CloudFormation、パイプラインのサービスロールを作成し CodePipeline、サービスロールポリシーと信頼ポリシーをアタッチする必要があります。CLI を使用してサービスロールを作成するには、以下の手順を使用して、まず CLI コマンドを実行するディレクトリに信頼ポリシー JSON とロールポリシー JSON を別々のファイルとして作成します。

> **Note**
>
> 管理者ユーザーのみにサービスロールの作成を許可することをお勧めします。ロールの作成とポリシーのアタッチを許可されたユーザーは、自分のアクセス許可をエスカレートできます。代わりに、彼らが必要とするロールの作成のみを許可したポリシーを作成するか、彼らの代わりに、管理者にサービスロールを作成させます。

1. ターミナルウィンドウで以下のコマンドを入力して、`TrustPolicy.json` という名前のファイルを作成し、そのファイルにロールポリシー JSON を貼り付けます。この例では VIM を使用します。

```
vim TrustPolicy.json
```

2. そのファイルに次の JSON を貼り付けます。

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "codepipeline.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

ファイルを保存して終了するには、次の VIM コマンドを入力します。

```
:wq
```

3. ターミナルウィンドウで以下のコマンドを入力して、RolePolicy.json という名前のファイルを作成し、そのファイルにロールポリシー JSON を貼り付けます。この例では VIM を使用します。

```
vim RolePolicy.json
```

4. そのファイルに次の JSON を貼り付けます。ポリシーステートメントの Resource フィールドにパイプラインの Amazon リソースネーム (ARN) を追加して、アクセス許可の適用範囲をできる限り絞り込んでください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecommit:CancelUploadArchive",
        "codecommit:GetBranch",
        "codecommit:GetCommit",
        "codecommit:GetUploadArchiveStatus",
        "codecommit:UploadArchive"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "codedeploy:CreateDeployment",
        "codedeploy:GetApplicationRevision",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:RegisterApplicationRevision"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "devicefarm:ListProjects",
        "devicefarm:ListDevicePools",
        "devicefarm:GetRun",
        "devicefarm:GetUpload",
        "devicefarm:CreateUpload",
        "devicefarm:ScheduleRun"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:ListFunctions"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*"
    },
    {
```

```
      "Effect": "Allow",
      "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*"
      ],
      "Resource": "resource_ARN"
    }
  ]
}
```

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 --policy-
document file://RolePolicy.json
```

7. 作成されたロール名と信頼ポリシーを表示するには、MyRole という名前のロールに対して次のコマンドを入力します。

```
aws iam get-role --role-name MyRole
```

8. AWS CLI または を使用してパイプラインを作成するときは、サービスロール ARN を使用します AWS CloudFormation。

でパイプラインにタグを付ける CodePipeline

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。でパイプラインにタグを適用できます CodePipeline。 CodePipeline リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプについては、「」を参照してくださいリソースのタグ付け。

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 のいずれかで管理できる唯一の 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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. Pipelines ページで、タグを追加するパイプラインを選択します。
3. ナビゲーションパネルから [Settings (設定)] を選択します。

4. [Pipeline tags] で、[Edit] を選択します。
5. [Key] フィールドと [Value] フィールドに、追加するタグのセットごとにキーペアを入力します。([値] フィールドはオプションです。) 例えば、[キー] では、「**Project**」と入力します。[値] には「**ProjectA**」と入力します。
6. (オプション) [タグを追加] をクリックして行を追加し、さらにタグを入力します。
7. [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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Pipelines] ページで、タグを表示するパイプラインを選択します。
3. ナビゲーションパネルから [Settings (設定)] を選択します。
4. [Pipeline tags] で、[Key] 列と [Value] 列下のパイプラインのタグを表示します。

パイプラインのタグを編集する (コンソール)

コンソールを使用してパイプラインに追加されたタグを編集します。

1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Pipelines] ページで、タグを更新するパイプラインを選択します。
3. ナビゲーションパネルから [Settings (設定)] を選択します。
4. [Pipeline tags] で、[Edit] を選択します。
5. [キー] フィールドと [値] フィールドに、必要に応じて各フィールドの値を更新します。例えば、**Project** キーの場合は、[Value] で、**ProjectA** を **ProjectB** に変更します。
6. [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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Pipelines] ページで、タグを削除するパイプラインを選択します。
3. ナビゲーションパネルから [Settings (設定)] を選択します。
4. [Pipeline tags] で、[Edit] を選択します。
5. 削除する各タグのキーと値の横にある [Remove tag] を選択します。
6. [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に AWS CLI 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、「でパイプラインを作成する CodePipeline」を参照してください。

以下のステップでは、 AWS CLI の最新版を既にインストールしているか、最新版に更新しているものと想定します。詳細については、「AWS Command Line Interfaceのインストール」を参照してください。

ターミナルまたはコマンドラインで、タグを追加するパイプラインのAmazon リソースネーム (ARN)、および追加するタグのキーと値を指定して tag-resource コマンドを実行します。パイプラインに複数のタグを追加できます。例えば、 という名前のパイプライン*MyPipeline*に 2 つのタグ、テスト というタグ値*DeploymentEnvironment*を持つ という名前のタグキー、および *true* というタグ値*IscontainerBased*を持つ という名前のタグキーをタグ付けするには、次のようにします。

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA key=IscontainerBased,value=true
```

成功した場合、このコマンドは何も返しません。

パイプラインのタグを表示する (CLI)

を使用してパイプラインの AWS タグ AWS CLI を表示するには、次の手順に従います。タグが追加されていない場合、返されるリストは空になります。

ターミナルまたはコマンドラインで、list-tags-for-resource コマンドを実行します。例えば、arn:aws:codepipeline:*us-west-2*:*account-id*:*MyPipeline*ARN 値*MyPipeline*を持つ 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には、次のようにします。

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline
```

成功した場合、このコマンドは次のような情報を返します。

```
{
    "tags": {
        "Project": "ProjectA",
        "IscontainerBased": "true"
    }
}
```

パイプラインのタグを編集する (CLI)

を使用してパイプラインのタグ AWS CLI を編集するには、次の手順に従います。既存のキーの値を変更したり、別のキーを追加できます。次のセクションに示すように、パイプラインからタグを削除することもできます。

ターミナルまたはコマンドラインで、タグを更新するパイプラインの ARN を指定して、tag-resource コマンドを実行し、タグキーとタグ値を指定します。

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tags key=Project,value=ProjectA
```

成功した場合、このコマンドは何も返しません。

パイプラインからタグを削除する (CLI)

を使用してパイプラインからタグ AWS CLI を削除するには、次の手順に従います。関連付けられているリソースからタグを削除すると、そのタグが削除されます。

> **Note**
>
> パイプラインを削除すると、削除されたパイプラインからすべてのタグの関連付けが削除されます。パイプラインを削除する前にタグを削除する必要はありません。

ターミナルまたはコマンド行で、タグを削除するパイプラインのARNと削除するタグのタグキーを指定して、untag-resource コマンドを実行します。例えば、タグキー *Project* と *MyPipeline*を使用して という名前のパイプライン上の複数のタグを削除するには、次のようにします*IscontainerBased*。

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-id:MyPipeline --tag-keys Project IscontainerBased
```

成功した場合、このコマンドは何も返しません。パイプラインに関連付けられているタグを確認するには、list-tags-for-resource コマンドを実行します。

通知ルールの作成

通知ルールを使用すると、パイプラインの実行開始時など、重要な変更をユーザーに通知できます。通知ルールは、イベントと、通知の送信に使用される Amazon SNS トピックの両方を指定します。詳細については、「通知とは」を参照してください。

コンソールまたは を使用して AWS CLI 、 の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://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
2. [Pipelines (パイプライン)] を選択し、通知を追加するパイプラインを選択します。
3. パイプラインページで、[Notify (通知)] を選択し、次に、[Create notification rule (通知ルールを作成)] を選択します。パイプラインの [Settings (設定)] ページに移動し、[Create notification rule (通知ルールを作成)] を選択することもできます。

4. [通知名] に、ルールの名前を入力します。
5. 詳細タイプ で、Amazon に提供された情報のみを通知 EventBridge に含める場合は、基本 を選択します。Amazon に提供された情報 EventBridge と、 CodePipeline または通知マネージャーから提供された可能性のある情報を含める場合は、「フル」を選択します。

   詳細については、「通知の内容とセキュリティについて」を参照してください。
6. [Events that trigger notifications (通知をトリガーするイベント)] で、通知を送信するイベントを選択します。詳細については、「パイプラインでの通知ルールのイベント」を参照してください。
7. [Targets (ターゲット)] で、次のいずれかの操作を行います。

   - 通知で使用するリソースを設定済みである場合は、[Choose target type] で、[AWS Chatbot (Slack)] または [SNS topic] を選択します。ターゲットの選択 で、クライアントの名前 (で設定された Slack クライアントの場合 AWS Chatbot) 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 AWS Chatbot クライアントを作成することはできません。 AWS Chatbot (Slack) を選択すると、 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 AWS Chatbot。このオプションを選択すると、 AWS Chatbot コンソールが開きます。詳細については、「通知と の統合を設定する AWS Chatbot」を参照してください。
> - 既存の Amazon SNS トピックをターゲットとして使用する場合は、このトピック用の他のすべてのポリシーに加えて、AWS CodeStar Notifications に必要なポリシーを追加する必要があります。詳細については、「通知用の Amazon SNS トピックを設定する」および「通知の内容とセキュリティについて」を参照してください。

8. ルールの作成を終了するには、[Submit (送信)] を選択します。
9. 通知を受け取るには、そのルールの Amazon SNS トピックにユーザーをサブスクライブする必要があります。詳細については、「ターゲットである Amazon SNS トピックへのユーザーのサブスクライブ」を参照してください。通知と の統合を設定 AWS Chatbot 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、「通知と の統合を設定する AWS Chatbot」を参照してください。

通知ルールを作成するには (AWS CLI)

1. ターミナルまたはコマンドプロンプトで、create-notification rule コマンドを実行して、JSON スケルトンを生成します。

```
aws codestar-notifications create-notification-rule --generate-cli-skeleton
 > rule.json
```

ファイルには任意の名前を付けることができます。この例では、ファイルの名前を *rule.json* とします。

2. プレーンテキストエディタで JSON ファイルを開き、これを編集してルールに必要なリソース、イベントタイプ、ターゲットを含めます。次の例は、ID *123456789012* の AWS アカウント*MyDemoPipeline*で という名前のパイプライン**MyNotificationRule**に対して 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開始されると、通知は完全な詳細タイプで *codestar-notificationsMyNotificationTopic*- という名前の Amazon SNS トピックに送信されます。

```
{
    "Name": "MyNotificationRule",
    "EventTypeIds": [
        "codepipeline-pipeline-pipeline-execution-started"
    ],
    "Resource": "arn:aws:codebuild:us-east-2:123456789012:MyDemoPipeline",
    "Targets": [
        {
            "TargetType": "SNS",
            "TargetAddress": "arn:aws:sns:us-east-2:123456789012:codestar-
notifications-MyNotificationTopic"
        }
    ],
    "Status": "ENABLED",
    "DetailType": "FULL"
```

```
}
```

ファイルを保存します。

3. 先ほど編集したファイルを使用して、ターミナルまたはコマンドラインで、create-notification-rule コマンドを再度実行し、通知ルールを作成します。

```
aws codestar-notifications create-notification-rule --cli-input-json
 file://rule.json
```

4. 成功すると、コマンドは次のような通知ルールの ARN を返します。

```
{
    "Arn": "arn:aws:codestar-notifications:us-east-1:123456789012:notificationrule/
dc82df7a-EXAMPLE"
}
```

でのトリガーの使用 CodePipeline

トリガーを使用すると、特定のブランチまたはプルリクエストの変更が検出されたときなど、特定のイベントタイプまたはフィルタリングされた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、、 GitHubBitbucket CodePipeline、 などの アクションを使用する接続を持つソースCodeStarSourceConnectionアクションに対して設定できます GitLab。

CodeCommit や S3 などのソースアクションは、このセクションでパイプラインの開始について詳述されているように、変更検出を使用します。

パイプラインにトリガーを追加し、特定のイベントをフィルタリングするようにトリガーを設定できます。

コンソールまたは CLI を使用してトリガーを指定します。

コードプッシュまたはプルリクエストでトリガーをフィルタリングする

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 、トリガーのフィルターを設定できます。

次のトリガータイプにフィルターを指定できます。

- プッシュする

  プッシュトリガーは、変更がソースリポジトリにプッシュされ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、送信先ブランチ) からのコミットが使用されます。ブランチ、ファイルパス、または Git タグでプッシュトリガーを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、ソースリポジトリでプルリクエストがオープン、更新、またはクローズされたときにパイプラインを開始します。実行では、プル元のソースブランチ (つまり、ソースブランチ) からのコミットを使用します。ブランチとファイルパスでプルリクエストトリガーをフィルタリングできます。

> **Note**
>
> 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、オープン、更新、またはクローズ (マージ) されます。その他のプルリクエストイベントはすべて無視されます。

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異なる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、「」を参照してください[パイプライン JSON でのトリガーフィルタリング (CLI)](#)。有効な組み合わせは次のとおりです。

- タグ
- ブランチ
- ブランチ + ファイルパス

フィルタータイプは次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部として指定されたデフォルトブランチへのプッシュ時にパイプラインを開始します。
- フィルターを指定する

  コードプッシュのブランチ名など、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、変更時にパイプラインが自動的に開始されないようにも設定されます。
- 変更を検出しない

  これによりトリガーは追加されず、パイプラインは変更時に自動的に開始されません。

次の表は、各イベントタイプの有効なフィルターオプションを示しています。この表は、アクション設定の自動変更検出で、どのトリガー設定がデフォルトで true または false になるかも示しています。

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
トリガーの追加 — フィルターなし	なし	なし	true
トリガーの追加 — コードプッシュでフィルタリングする	プッシュイベント	Git タグ、ブランチ、ファイルパス	false
トリガーの追加 — 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 (Webhookトリガータイプとして) 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、セルフ GitLabマネージド) です。

フィルタリングについては、 で説明されているように、glob 形式の正規表現パターンがサポートされています構文での glob パターンの使用。

> **Note**
>
> 場合によっては、ファイルパスでフィルタリングされたトリガーを持つパイプラインの場合、ファイルパスフィルターを持つブランチが最初に作成されたときにパイプラインが開始されないことがあります。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されない場合があります」を参照してください。

トピック

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場合は、次の考慮事項が適用されます。

- ブランチとファイルパスのフィルターを使用するトリガーの場合、ブランチを初めてプッシュすると、新しく作成されたブランチで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) の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。

トリガーフィルターの例

プッシュリクエストイベントタイプとプルリクエストイベントタイプのフィルターを含む Git 設定では、指定されたフィルターが互いに競合する可能性があります。以下は、プッシュリクエストイベントとプルリクエストイベントの有効なフィルターの組み合わせの例です。

お客様が単一の設定オブジェクト内でフィルターを組み合わせる場合、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、Git 設定を示しています。

```
{
  "filePaths": {
    "includes": ["common/**/*.js"]
  },
  "branches": {
    "includes": ["feature/**"]
  }
```

```
}
```

上記の Git 設定では、この例は、AND オペレーションが成功したためにパイプラインの実行を開始するイベントを示しています。

```
{
  "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "common/app.js"
      ]
      ...
    }
  ]
}
```

この例では、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ないため、パイプラインの実行を開始しないイベントを示します。

```
{
   "ref": "refs/heads/feature/triggers",
  ...
  "commits": [
    {
      ...
      "modified": [
        "src/Main.java"
      ]
      ...
    }
  ]
}
```

同時に、プッシュ配列内のトリガー設定オブジェクトは OR オペレーションを使用します。これにより、同じパイプラインの実行を開始するように複数のトリガーを設定できます。JSON 構造内のフィールド定義のリストについては、「」を参照してくださいパイプライン JSON でのトリガーフィルタリング (CLI)。

プッシュイベントでのフィルタリング (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たり除外したりできます。

プッシュイベントでのフィルタリング (コンソール)

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。

2. [Name] で、編集するパイプラインの名前を選択します。それ以外の場合は、パイプライン作成ウィザードでこれらのステップを使用します。
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、次のオプションからプッシュを選択します。

   - プッシュ を選択して、変更がソースリポジトリにプッシュされたときに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、または Git タグのフィルターを指定できます。
   - プルリクエストを選択して、ソースリポジトリでプルリクエストがオープン、更新、またはクローズされたときにパイプラインを開始します。これを選択すると、フィールドで送信先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. フィルタータイプ で、次のいずれかのオプションを選択します。

   - Branch を選択して、トリガーが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、ワークフロー実行を開始するタイミングを確認します。を含める で、指定したブランチの変更時にパイプラインを開始するためのトリガー設定に指定するブランチ名のパターンを glob 形式で入力します。除外 で、指定したブランチの変更時にトリガー設定が無視し、パイプラインを開始しないように指定するブランチ名の正規表現パターンを glob 形式で入力します。詳細については、「構文での glob パターンの使用」を参照してください。

> **Note**
>
> include と exclude の両方に同じパターンがある場合、デフォルトはパターンを除外します。

正規表現パターンを glob 形式で使用して、ブランチ名を定義できます。例えば、 `main.*`を使用して、 で始まるすべてのブランチを照合します`main.*`。詳細については、「構文での glob パターンの使用」を参照してください。

プッシュトリガーでは、 にプッシュするブランチ、つまり送信先ブランチを指定します。プルリクエストトリガーには、プルリクエストを開く送信先ブランチを指定します。

- (オプション) ファイルパス で、トリガーのファイルパスを指定します。必要に応じて、Include と Exclude に名前を入力します。

  glob 形式の正規表現パターンを使用して、ファイルパス名を定義できます。例えば、`prod.*`を使用して、 で始まるすべてのファイルパスを照合します`prod.*`。詳細については、「構文での glob パターンの使用」を参照してください。
- タグ を選択して、パイプライントリガー設定を Git タグで開始するように設定します。を含める で、指定したタグのリリース時にパイプラインを開始するためのトリガー設定に指定するタグ名のパターンを glob 形式で入力します。除外 で、指定したタグのリリース時にトリガー設定が無視し、パイプラインを開始しないように指定するタグ名の正規表現パターンを glob 形式で入力します。[包含] と [除外] 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。

プルリクエストのフィルタリング (コンソール)

コンソールを使用して、指定したイベントを含むプルリクエス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たり除外したりできます。

プルリクエストのフィルタリング (コンソール)

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。

2. [Name] で、編集するパイプラインの名前を選択します。それ以外の場合は、パイプライン作成ウィザードでこれらのステップを使用します。
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、次のオプションからプルリクエストを選択します。

   - プッシュ を選択して、変更がソースリポジトリにプッシュされたときに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、または Git タグのフィルターを指定できます。
   - プルリクエストを選択して、指定したターゲットブランチに対してプルリクエストがオープン、更新、またはクローズされたときに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を指定できます。

     オプションで、次のプルリクエストイベントを指定してフィルタリングできます。

     - プルリクエストが作成されました
     - プルリクエストに新しいリビジョンが加えられました
     - プルリクエストがクローズされている
6. フィルタータイプ で、次のいずれかのオプションを選択します。

   - Branch を選択して、トリガーが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、ワークフロー実行を開始するタイミングを確認します。を含める で、指定したブランチの変更時にパイプラインを開始するためのトリガー設定に指定するブランチ名のパターンを glob 形式で入力します。除外 で、指定したブランチの変更時にトリガー設定が無視し、パイプラインを開始しないように指定するブランチ名の正規表現パターンを glob 形式で入力します。詳細については、「構文での glob パターンの使用」を参照してください。

     > **Note**
     >
     > include と exclude の両方に同じパターンがある場合、デフォルトはパターンを除外します。

     正規表現パターンを glob 形式で使用して、ブランチ名を定義できます。例えば、 `main.*`を使用して、 で始まるすべてのブランチを照合しますmain.*。詳細については、「構文での glob パターンの使用」を参照してください。

プッシュトリガーでは、 にプッシュするブランチ、つまり送信先ブランチを指定します。プルリクエストトリガーには、プルリクエストを開く送信先ブランチを指定します。

- (オプション) ファイルパス で、トリガーのファイルパス名を指定します。必要に応じて、Include と Exclude に名前を入力します。

  glob 形式の正規表現パターンを使用して、ファイルパス名を定義できます。例えば、`prod.*`を使用して、 で始まるすべてのファイルパスを照合します`prod.*`。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。

パイプライン JSON でのトリガーフィルタリング (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 構造の例は、 のフィールド定義のリファレンスを提供します`create-pipeline`。

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

JSON 構造のフィールドは次のように定義されます。

- sourceActionName: Git 設定のパイプラインソースアクションの名前。
- push: フィルタリングを使用してイベントをプッシュします。これらのイベントは、異なるプッシュフィルター間の OR オペレーションと、フィルター内の AND オペレーションを使用します。
  - branches: フィルタリングするブランチ。ブランチは、包含と除外の間で AND オペレーションを使用します。
    - includes: 含まれるブランチをフィルタリングするパターン。OR オペレーションの使用が含まれます。
    - excludes: 除外されるブランチをフィルタリングするパターン。OR オペレーションの使用を除外します。
  - filePaths: フィルタリングするファイルパス名。
    - includes: 含まれるファイルパスをフィルタリングするパターン。OR オペレーションの使用が含まれます。
    - excludes: 除外されるファイルパスをフィルタリングするパターン。OR オペレーションの使用を除外します。

- tags: フィルタリングするタグ名。
  - includes: 含まれるタグをフィルタリングするパターン。OR オペレーションの使用が含まれます。
  - excludes: 除外されるタグをフィルタリングするパターン。OR オペレーションの使用を除外します。
- 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をフィルタリングして、リクエストイベントをプルします。
  - events: 指定されたオープン、更新、またはクローズされたプルリクエストイベントを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、包含と除外の間で AND オペレーションを使用します。
    - includes: 含まれるブランチをフィルタリングするパターン。OR オペレーションの使用が含まれます。
    - excludes: 除外されるブランチをフィルタリングするパターン。OR オペレーションの使用を除外します。
  - filePaths: フィルタリングするファイルパス名。
    - includes: 含まれるファイルパスをフィルタリングするパターン。OR オペレーションの使用が含まれます。
    - excludes: 除外されるファイルパスをフィルタリングするパターン。OR オペレーションの使用を除外します。

AWS CloudFormation テンプレートでのトリガーフィルタリング

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。

次のサンプルテンプレートスニペットは、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、「」を参照してくださいパイプライン JSON でのトリガーフィルタリング (CLI)。

```
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
```

で実行を管理する CodePipeline

パイプラインの進行状況を分析するために、エラーログの表示、パイプラインとアクションの実行履歴の表示、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

- で実行を表示する CodePipeline
- パイプライン実行モードを設定または変更する
- 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
- ステージロールバックの設定

で実行を表示する CodePipeline

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、アカウント内のすべてのパイプラインのリストを表示できます。パイプラインで最後に実行されたアクション、ステージ間の移行が有効か無効か、失敗したアクションの有無、その他の情報など、各パイプラインの詳細を表示することもできます。履歴が記録されたすべてのパイプライン実行の詳細を示す履歴ページを表示することもできます。

> **Note**
>
> 特定の実行モードを切り替えると、パイプラインビューと履歴が変わる可能性があります。詳細については、「パイプライン実行モードを設定または変更する」を参照してください。

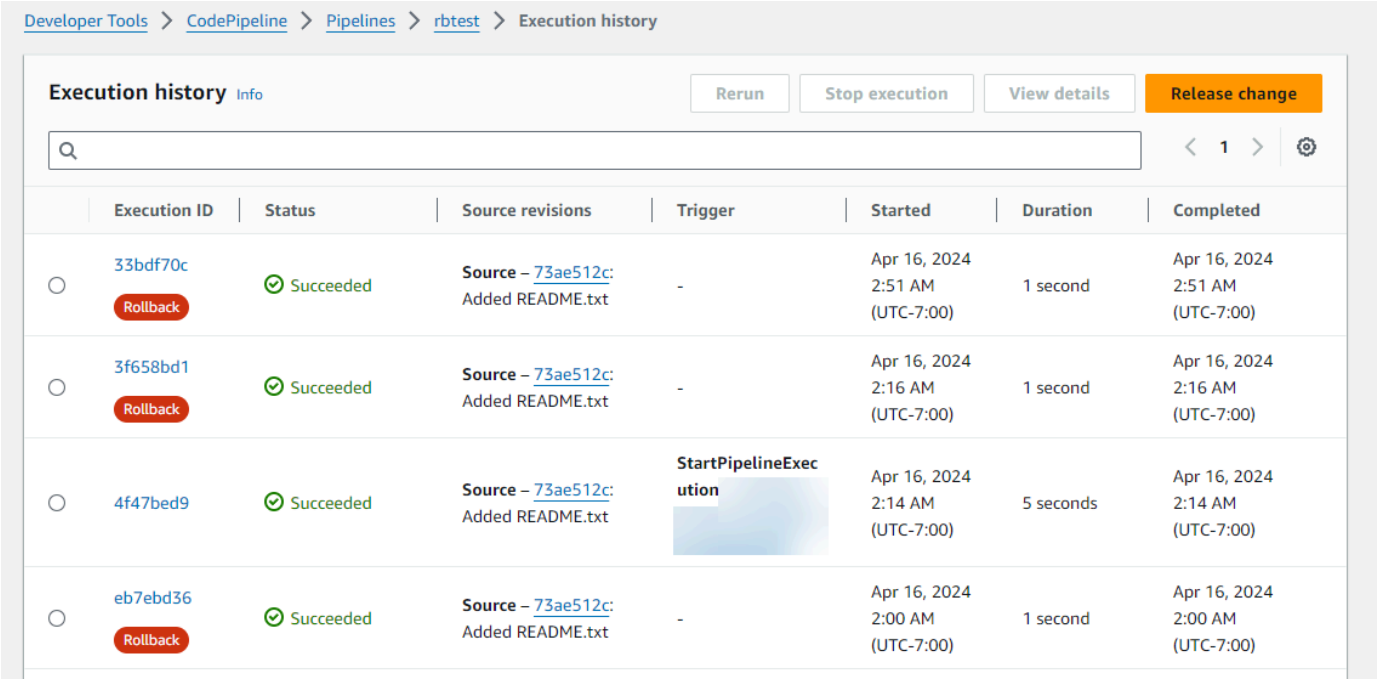
実行履歴は、最大 12 か月間保持されます。

コンソールを使用して、パイプラインの実行履歴を表示できます。履歴には、各実行のステータス、ソースのリビジョン、タイミングの詳細が含まれます。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [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は、実行履歴ページから表示する実行の ID を選択して、パイプライン構造にアクセスします。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、視覚化タブでパイプラインを表示します。



4. パイプラインの各実行に関連するステータス、ソースリビジョン、変更の詳細、トリガーが表示されます。ロールバックされたパイプラインの実行では、コンソールの詳細画面に実行タイプのロールバックが表示されます。自動ロールバックをトリガーした失敗した実行の場合、失敗した実行 ID が表示されます。
5. 実行を選択します。詳細ビューには、実行の詳細、[タイムライン] タブ、[視覚化] タブ、[変数] タブが表示されます。パイプラインレベルの変数の値はパイプラインの実行時に解決され、実行ごとに実行履歴で確認できます。

> **Note**
>
> パイプラインアクションからの出力変数は、アクションの実行ごとの履歴の下にある [出力変数] タブで確認できます。

実行のステータスを表示する (コンソール)

パイプラインのステータスは、実行履歴ページの [ステータス] で確認できます。実行 ID リンクを選択し、次にアクションのステータスを表示します。

パイプライン、ステージ、およびアクションの有効な状態は以下のとおりです。

> **Note**
>
> 次のパイプライン状態は、インバウンド実行であるパイプライン実行にも適用されます。受信実行とそのステータスを表示するには、インバウンドの実行(コンソール)を表示します。を参照してください。

パイプラインレベルの状態

パイプラインの状態	説明
InProgress	パイプライン実行が現在実行中です。
停止中	パイプライン実行は、パイプライン実行を [Stop and wait (停止して待機)] するか、[Stop and abandon (停止して中止)] する要求により、停止しています。
停止	停止プロセスが完了し、パイプラインの実行が停止します。
成功	パイプライン実行が正常に完了しました。
置換	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。

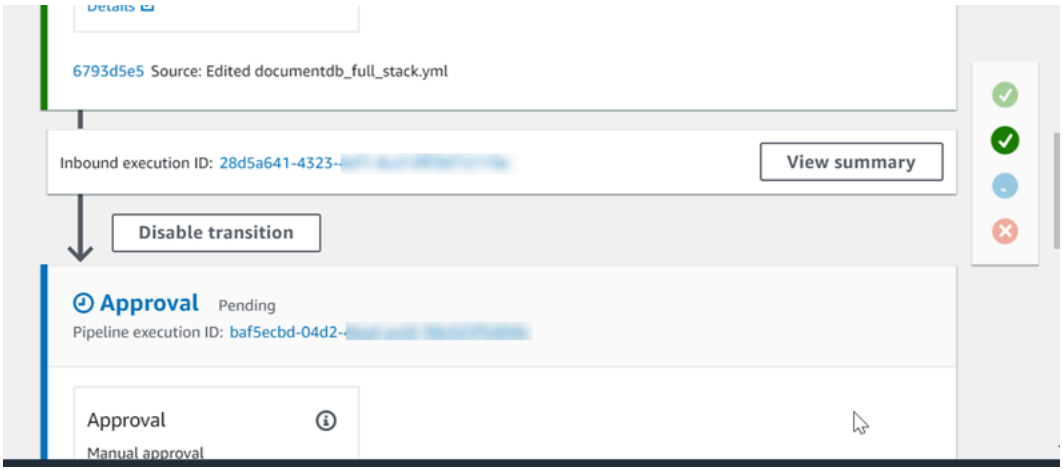
受信実行のステータスと詳細を表示するには、コンソールを使用します。トランジションが有効な場合、またはステージが使用可能になると、`InProgress` であるインバウンド実行が続き、ステージを入力します。`Stopped` のステータスを用いたインバウンド実行は、ステージを入力しません。パイプラインが編集されている場合、インバウンド実行ステータスは `Failed` に変わります。パイプラインを編集すると、進行中のすべての実行は続行されず、実行ステータスは `Failed` に変わります。

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. インバウンドの実行を表示したいパイプラインの名前を選択し、以下のいずれかの操作を行います：

- [View (表示)] を選択します。パイプラインダイアグラムでは、無効なトランジションの前にある Inbound execution ID フィールドで、インバウンド実行 ID を表示できます。



  View summary を選択し、実行 ID、ソーストリガー、次のステージの名前などの実行の詳細を表示します。

- パイプラインを選択し、View history を選択します。

パイプライン実行ソースのリビジョンを表示する (コンソール)

パイプラインの実行に使用するソースアーティファクト (パイプラインの最初のステージで生成された出力アーティファクト) に関する詳細を表示できます。この詳細には、コミット ID などの識別子、チェックインコメント、および CLI を使用した場合は、パイプラインのビルドアクションのバージョン番号などが含まれます。一部のリビジョンタイプでは、コミットの URL を表示して開くことができます。ソースのリビジョンは、以下で構成されます。

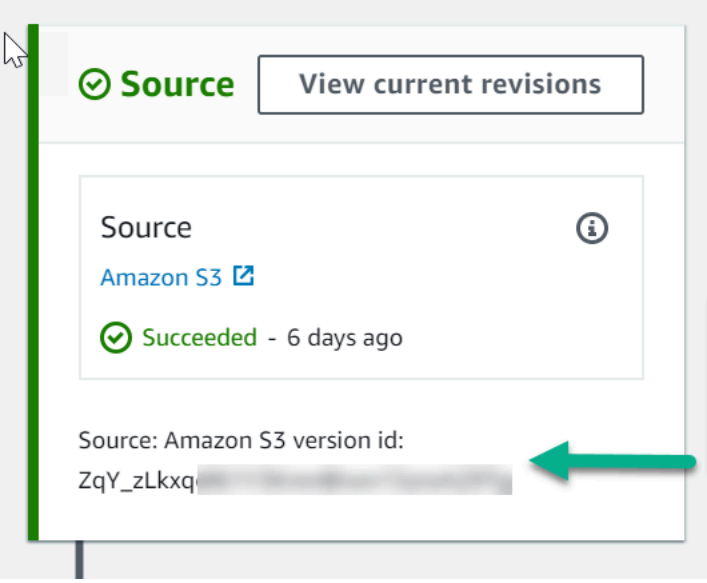
- Summary: アーティファクトの最新のリビジョンに関する概要。 GitHub および CodeCommit リポジトリの場合、コミットメッセージ。Amazon S3 バケットまたはアクションの場合、オブジェクトメタデータで指定された codepipeline-artifact-revision-summary キーのユーザー提供コンテンツ。
- revisionUrl: アーティファクトリビジョンのリビジョン URL (例: 外部リポジトリ URL)。
- revisionId: アーティファクトリビジョンのリビジョン ID。例えば、 CodeCommit または GitHub リポジトリのソース変更の場合、これはコミット ID です。 GitHub または 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。

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

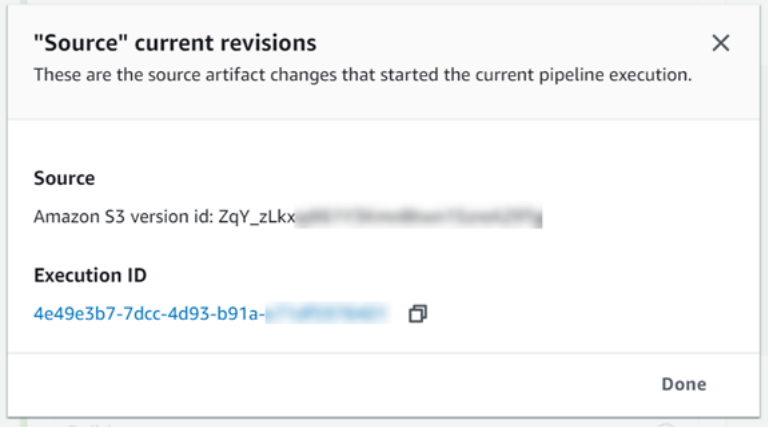
   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。

2. ソースリビジョンの詳細を表示するパイプラインの名前を選択します。次のいずれかを行います。

   - [View history (履歴の表示)] を選択します。[Source revisions (ソースリビジョン)] に、各実行のソース変更が一覧表示されます。
   - ソースリビジョンの詳細を表示するアクションを見つけ、そのステージの下部にあるリビジョン情報を確認します。



   ソース情報を表示するには、[View current revisions (現行リビジョンを表示)] を選択します。Amazon S3 バケットに格納されているアーティファクトを除いて、この情報詳細ビューのコミット ID などの識別子は、アーティファクトのソース情報ページにリンクされています。



アクション実行を表示する (コンソール)

パイプラインのアクションの詳細として、アクションの実行 ID、入力アーティファクト、出力アーティファクト、ステータスなどを表示できます。アクションの詳細を表示するには、コンソールでパイプラインを選択し、次に実行 ID を選択します。

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. アクションの詳細を表示するパイプラインの名前を選択し、次に [View history (履歴の表示)] を選択します。
3. [Execution ID (実行 ID)] で、アクションの実行の詳細を表示する実行 ID を選択します。
4. [Timeline (タイムライン)] タブで以下の情報を確認できます。

   a. [アクション名] でリンクを選択し、アクションの詳細ページを開きます。このページで、ステータス、ステージ名、アクション名、設定データ、およびアーティファクト情報を確認できます。

   b. [プロバイダー] で、アクションプロバイダーの詳細を表示するリンクを選択します。例えば、前述のパイプラインの例では、ステージングステージまたは本番稼働ステージ CodeDeploy のいずれかで を選択すると、 CodeDeployそのステージに設定された CodeDeploy アプリケーションの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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. アクションの詳細を表示するパイプラインの名前を選択し、次に [View history (履歴の表示)] を選択します。
3. [Execution ID (実行 ID)] で、アクションの詳細を表示する実行 ID を選択します。
4. [Timeline (タイムライン)] タブの [アクション名] で、アクションの詳細ページを開くリンクを選択します。
5. 詳細ページの [実行] で、アクションの実行のステータスとタイミングを表示します。
6. 設定タブで、 アクションのリソース設定 (CodeBuild ビルドプロジェクト名など) を表示します。
7. [アーティファクト] タブで、[アーティファクトタイプ] と [アーティファクトプロバイダー] のアーティファクト詳細を表示します。[アーティファクト名] のリンクを選択してアーティファクトストアのアーティファクトを表示します。
8. [出力変数] タブで、アクションの実行のためにパイプライン内のアクションから解決された変数を確認できます。

パイプラインの詳細と履歴を表示する (CLI)

パイプラインとパイプラインの実行の詳細を表示するには、以下のコマンドを実行します。

- list-pipelines コマンドを使用して、 に関連付けられているすべてのパイプラインの概要を表示します AWS アカウント。
- get-pipeline コマンドは、1 つのパイプラインの詳細を表示します。
- list-pipeline-executions パイプラインの最新の実行の概要を取得します。

- get-pipeline-execution パイプラインの実行に関する情報 (アーティファクト、パイプラインの実行 ID、パイプラインの名前、バージョン、ステータスなど) を表示します。
- get-pipeline-state コマンドでパイプライン、ステージ、およびアクションのステータスを表示します。
- list-action-executions でパイプラインのアクション実行の詳細を表示します。

トピック

`list-pipeline-executions` (CLI) で実行履歴を表示する

パイプラインの実行履歴を表示できます。

- パイプラインの過去の実行の詳細を表示するには、パイプラインの一意の名前を指定して、list-pipeline-executions コマンドを実行します。例えば、 という名前のパイプラインの現在の状態に関する詳細を表示するには*MyFirstPipeline*、次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の実行には、実行タイプ が表示されますRollback。自動ロールバックをトリガーした失敗した実行の場合、失敗した実行 ID が表示されます。

  次の例は、3 つの実行*MyFirstPipeline*がある という名前のパイプラインに対して返されたデータを示しています。

  ```
  {
      "pipelineExecutionSummaries": [
          {
              "pipelineExecutionId": "eb7ebd36-353a-4551-90dc-18ca5EXAMPLE",
              "status": "Succeeded",
  ```

```
            "startTime": "2024-04-16T09:00:28.185000+00:00",
            "lastUpdateTime": "2024-04-16T09:00:29.665000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console-URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "trigger_ARN"
            },
            "executionMode": "SUPERSEDED"
        },
        {
            "pipelineExecutionId": "fcd61d8b-4532-4384-9da1-2aca1EXAMPLE",
            "status": "Succeeded",
            "startTime": "2024-04-16T08:58:56.601000+00:00",
            "lastUpdateTime": "2024-04-16T08:59:04.274000+00:00",
            "sourceRevisions": [
                {
                    "actionName": "Source",
                    "revisionId": "revision_ID",
                    "revisionSummary": "Added README.txt",
                    "revisionUrl": "console_URL"
                }
            ],
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "trigger_ARN"
            },
            "executionMode": "SUPERSEDED"
        }
```

パイプラインの実行について詳細を表示するには、パイプライン実行の一意の ID を指定して、get-pipeline-execution コマンドを実行します。例えば、前の例の最初の実行の詳細を表示するには、以下のように入力します。

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-execution-id 7cf7f7cb-3137-539g-j458-d7eu3EXAMPLE
```

このコマンドは、パイプラインの実行の概要情報 (アーティファクトの詳細、パイプラインの実行 ID、名前、バージョン、パイプラインのステータスなど) を返します。

次の例は、 という名前のパイプラインに対して返されたデータを示しています*MyFirstPipeline*。

```
{
    "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 (CLI) を使用してパイプラインの状態を表示する

CLI を使って パイプライン、ステージ、およびアクションのステータスを 表示できます。

- パイプラインの現在の状態の詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline-state コマンドを実行します。例えば、 という名前のパイプラインの現在の状態に関する詳細を表示するには*MyFirstPipeline*、次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。

次の例は、 という名前の 3 つの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。*MyFirstPipeline*最初の 2 つのステージとアクションは成功を示し、3 つ目は失敗を示し、2 つ目と 3 つめのステージ間の移行は無効になっています。

```
{
    "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**CLI) でインバウンド実行ステータスを表示する

インバウンドの実行のステータスを表示するのに、CLI を使用できます。トランジションが有効な場合、またはステージが使用可能な場合、InProgress であるインバウンド実行が続き、ステージを入力します。Stopped のステータスを用いたインバウンド実行は、ステージを入力しません。パイプラインが編集されている場合、インバウンド実行ステータスは Failed に変わります。パイプラインを編集すると、進行中のすべての実行は続行されず、実行ステータスは Failed に変わります。

- パイプラインの現在の状態の詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline-state コマンドを実行します。例えば、 という名前のパイプラインの現在の状態に関する詳細を表示するには*MyFirstPipeline*、次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。

次の例は、 という名前の 2 ステージパイプラインに対して返されたデータを示しています*MyFirstPipeline*。最初のステージは有効な移行とパイプライン実行の成功を示し、2 番目のステージは という名前の無効な移行とインバウンド実行 ID Beta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` (CLI) を使用してステータスとソースリビジョンを表示する

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、オブジェクトメタデータで指定された codepipeline-artifact-revision-summary キーのユーザー提供コンテンツ。
- revisionUrl: アーティファクトリビジョンのコミット ID。 GitHub または AWS CodeCommit リポジトリに保存されているアーティファクトの場合、コミット ID はコミットの詳細ページにリンクされます。

get-pipeline-execution コマンドを実行して、パイプライン実行で追加された最新のソースリビジョンに関する情報を表示できます。get-pipeline-state コマンドを実行して、パイプラインのすべてのステージに関する詳細を取得したら、ソースリビジョンの詳細を必要とするステージに適用される実行 ID を特定します。次に、その実行 ID を get-pipeline-execution コマンドで使用します (パイプラインのステージは、別のパイプラインの実行時に正常に完了している場合があるため、別の実行 ID が割り当てられていることがあります。)

つまり、Staging ステージに現在あるアーティファクトに関する詳細を表示する場合は、get-pipeline-state コマンドを実行し、Staging ステージの現在の実行 ID を特定してから、その実行 ID を使用して get-pipeline-execution コマンドを実行します。

パイプラインのステータスとソースリビジョンを表示するには

1. ターミナル (Linux、macOS、Unix) またはコマンドプロンプト (Windows) を開き、 AWS CLI を使用して get-pipeline-state のコマンドを実行します。という名前のパイプラインの場合は*MyFirstPipeline*、次のように入力します。

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

   このコマンドは、各ステージの最新のパイプライン実行 ID を含む、パイプラインの最新の状態を返します。

2. パイプライン実行の詳細を表示するには、get-pipeline-execution コマンドを実行します。この場合、パイプラインの一意の名前と、アーティファクトの詳細を表示する特定の実行のパイプライン実行 ID を指定します。例えば、実行 ID が 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE *MyFirstPipeline*の という名前のパイプラインの実行に関する詳細を表示するには、次のように入力します。 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-
execution-id 3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE
```

このコマンドは、パイプライン実行の一部である各ソースリビジョンに関する情報と、パイプラインを識別する情報を返します。実行に含まれていたパイプラインステージに関する情報のみが含まれます。パイプライン実行の一部ではなかった、パイプラインの他のステージが存在する可能性があります。

次の例は、「」という名前のアーティファクトが GitHub リポジトリに保存MyAppされている*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。

3.

```
{
    "pipelineExecution": {
        "artifactRevisions": [
            {
                "created": 1427298837.7689769,
                "name": "MyApp",
                "revisionChangeIdentifier": "1427298921.3976923",
                "revisionId": "7636d59f3c461cEXAMPLE8417dbc6371",
                "revisionSummary": "Updating the application for feature 12-4820",
                "revisionUrl": "https://api.github.com/repos/anycompany/MyApp/git/
commits/7636d59f3c461cEXAMPLE8417dbc6371"
            }
        ],
        "pipelineExecutionId": "3137f7cb-7cf7-039j-s83l-d7eu3EXAMPLE",
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 2,
        "status": "Succeeded",
        "executionMode": "SUPERSEDED",
        "executionType": "ROLLBACK",
        "rollbackMetadata": {
            "rollbackTargetPipelineExecutionId": "4f47bed9-6998-476c-a49d-
e60beEXAMPLE"
         }
```

```
    }
}
```

`list-action-executions` (CLI) を使用してアクション実行を表示する

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、 という名前のパイプラインでアクションの実行を表示するには*MyFirstPipeline*、次のように入力します。

    ```
    aws codepipeline list-action-executions --pipeline-name MyFirstPipeline
    ```

    以下は、このコマンドの出力例の一部です。

    ```
    {
        "actionExecutionDetails": [
            {
                "actionExecutionId": "ID",
                "lastUpdateTime": 1552958312.034,
                "startTime": 1552958246.542,
                "pipelineExecutionId": "Execution_ID",
                "actionName": "Build",
                "status": "Failed",
                "output": {
                    "executionResult": {
                        "externalExecutionUrl": "Project_ID",
                        "externalExecutionSummary": "Build terminated with state:
     FAILED",
                        "externalExecutionId": "ID"
                    },
    ```

```
                "outputArtifacts": []
            },
            "stageName": "Beta",
            "pipelineVersion": 8,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ],
                "actionTypeId": {
                    "version": "1",
                    "category": "Build",
                    "owner": "AWS",
                    "provider": "CodeBuild"
                }
            }
        },

. . .
```

- パイプラインの実行についてすべてのアクションの実行を表示するには、パイプラインの一意の名前と実行 ID を指定して、list-action-executions コマンドを実行します。例えば、*Execution_ID* のアクションの実行を表示するには、以下のように入力します。

```
aws codepipeline list-action-executions --pipeline-name MyFirstPipeline --filter
 pipelineExecutionId=Execution_ID
```

- 以下は、このコマンドの出力例の一部です。

```
{
    "actionExecutionDetails": [
        {
            "stageName": "Beta",
            "pipelineVersion": 8,
```

```
            "actionName": "Build",
            "status": "Failed",
            "lastUpdateTime": 1552958312.034,
            "input": {
                "configuration": {
                    "ProjectName": "java-project"
                },
                "region": "us-east-1",
                "actionTypeId": {
                    "owner": "AWS",
                    "category": "Build",
                    "provider": "CodeBuild",
                    "version": "1"
                },
                "inputArtifacts": [
                    {
                        "s3location": {
                            "bucket": "codepipeline-us-east-1-ID",
                            "key": "MyFirstPipeline/MyApp/Object.zip"
                        },
                        "name": "MyApp"
                    }
                ]
            },


. . .
```

パイプライン実行モードを設定または変更する

パイプラインの実行モードを設定して、複数の実行の処理方法を指定できます。

パイプライン実行モードの詳細については、「」を参照してくださいパイプライン実行の仕組み。

> **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 に編集しても、パイプラインの状態には更新された状態が PARALLEL として表示されません。詳細については、「PARALLEL モードから変更されたパイプラインには、以前の実行モードが表示されます。」を参照してください。

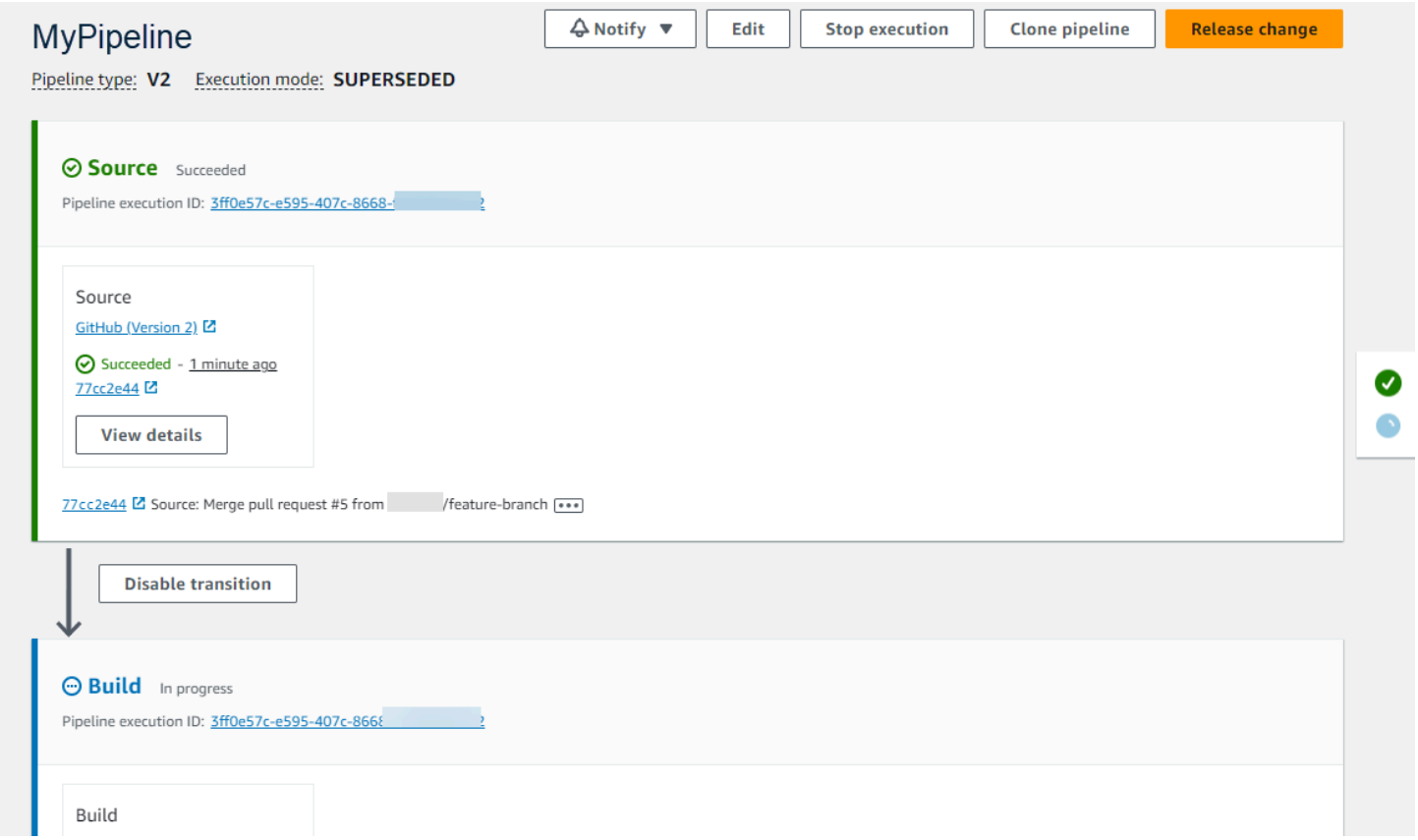
> **Important**
>
> PARALLEL モードのパイプラインの場合、パイプライン実行モードを QUEUED または SUPERSEDED に編集しても、各モードのパイプライン定義は更新されません。詳細については、「PARALLEL モードのパイプラインは、QUEUED モードまたは SUPERSEDED モードに変更したときに編集された場合、古いパイプライン定義になります。」を参照してください。

実行モードの表示に関する考慮事項

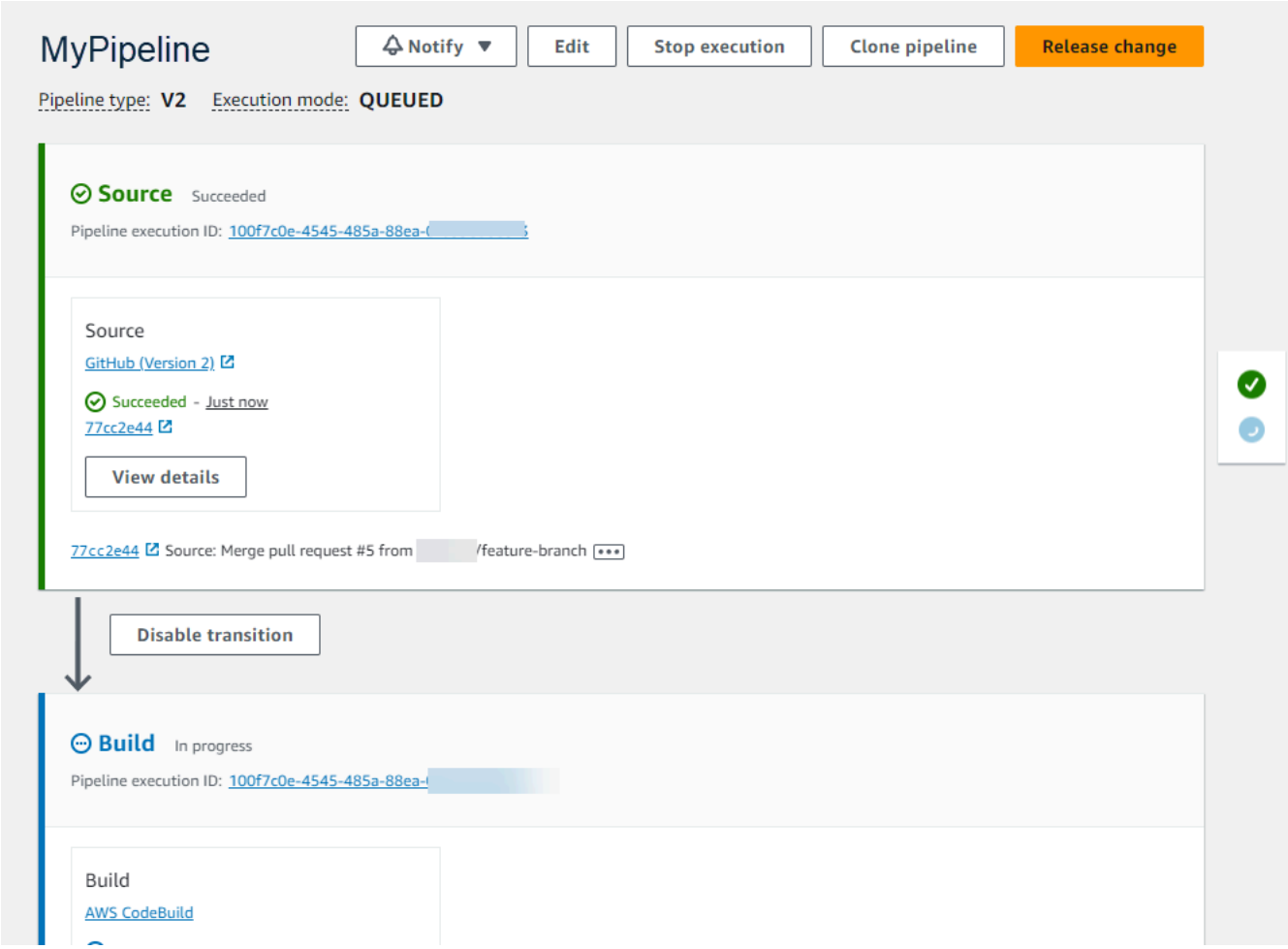
特定の実行モードでパイプラインを表示する際には、考慮事項があります。

SUPERSEDED モードと QUEUED モードの場合、パイプラインビューを使用して進行中の実行を確認し、実行 ID をクリックして詳細と履歴を表示します。PARALLEL モードの場合は、実行 ID をクリックして、視覚化タブで進行中の実行を表示します。

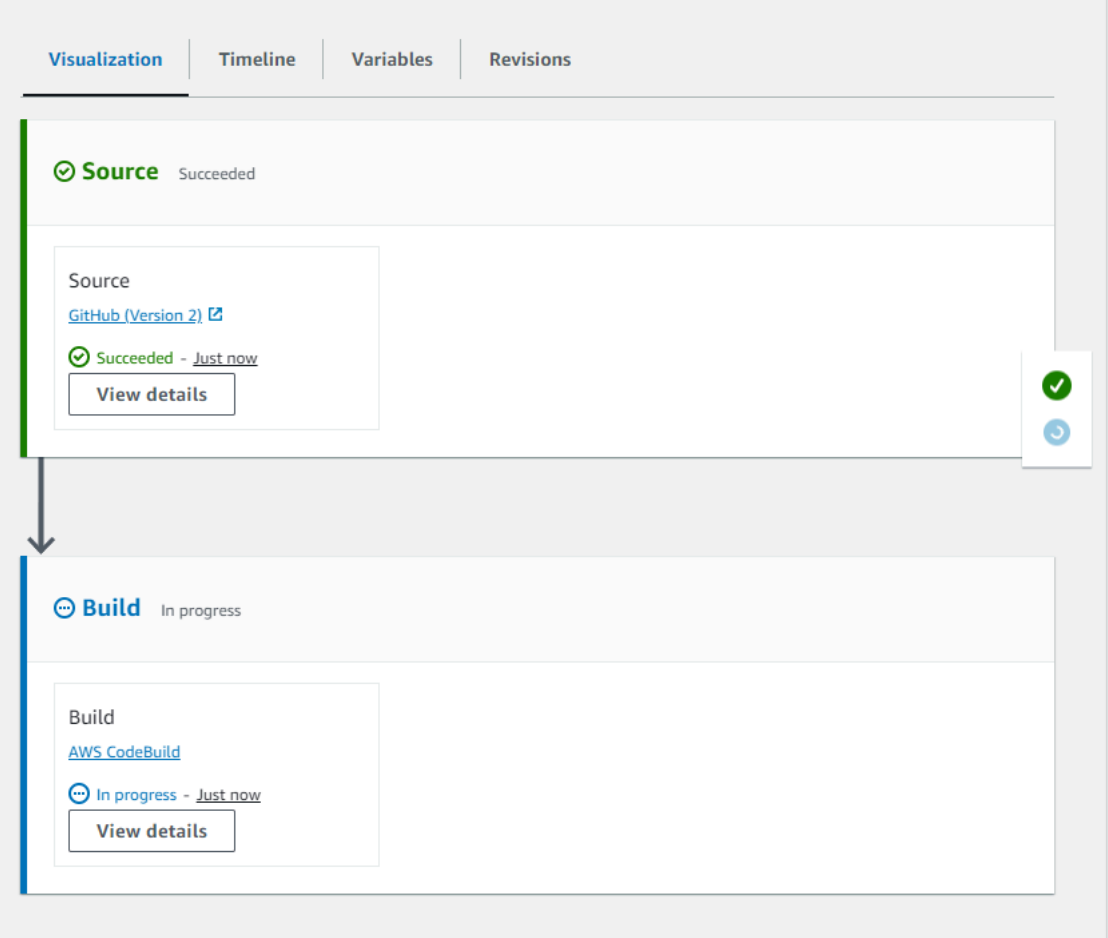
の SUPERSEDED モードのビューを次に示します CodePipeline。



の QUEUED モードのビューを次に示します CodePipeline。



の PARALLEL モードのビューを次に示します CodePipeline。



実行モードを切り替える際の考慮事項

以下は、パイプラインのモードを変更する際のパイプラインに関する考慮事項です。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、特定のビューまたは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 モードで開始された実行は引き続き実行されます。これらは実行履歴ページで表示できます。パイプラインビューには、QUEUED モードまたは SUPERSEDED モードより前または空で実行された実行が表示されます。

別の例として、QUEUED モードまたは SUPERSEDED モードから PARALLEL モードに切り替えると、パイプラインビュー/ステート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、実行の詳細ページの視覚化タブを使用します。SUPERSEDED モードまたは QUEUED モードで開始された実行はキャンセルされます。

次の表に詳細を示します。

モードの変更	保留中およびアクティブな実行の詳細	パイプライン状態の詳細
SUPERSED から SUPERSED / SUPERSED から QUEUED	• アクティブな実行は、進行中のアクションが完了した後にキャンセルされます。 • 保留中の実行はキャンセルされます。	キャンセルされた などのパイプラインの状態は、最初のモードのバージョンと 2 番目のモードの間で保持されます。
QUEUED to QUEUED / QUEUED to SUPERSEDED	• アクティブな実行は、進行中のアクションが完了した後にキャンセルされます。 • 保留中の実行はキャンセルされます。	キャンセルなどのパイプラインの状態は、最初のモードのバージョンと 2 番目のモードの間で保持されます。
PARALLEL から PARALLEL	すべての実行は、パイプライン定義の更新とは別に実行できます。	空。並列モードにはパイプライン状態はありません。
PARALLEL に継承/PARALLEL にキューに入れる	• アクティブな実行は、進行中のアクションが完了した後にキャンセルされます。 • 保留中の実行はキャンセルされます。	空。並列モードにはパイプライン状態はありません。

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [Name] 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。
   - 置き換え済み
   - キューに登録済み (パイプラインタイプ V2 が必要)
   - 並列 (パイプラインタイプ V2 が必要)
6. 編集ページで、完了 を選択します。

パイプライン実行モードを設定する (CLI)

を使用してパイプライン実行モード AWS CLI を設定するには、 `create-pipeline`または `update-pipeline` コマンドを使用します。

1. ターミナルセッション (Linux、macOS または Unix) またはコマンドプロンプト (Windows) を開き、get-pipeline コマンドを実行して、パイプライン構造を JSON ファイルにコピーします。たとえば、**MyFirstPipeline** という名前のパイプラインに対して、以下のコマンドを入力します。

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

   このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。
2. 任意のプレーンテキストエディタで JSON ファイルを開き、QUEUED など、設定するパイプライン実行モードを反映するようにファイルの構造を変更します。

   ```
   "executionMode": "QUEUED"
   ```

   次の例は、2 つのステージを持つサンプルパイプラインで実行モードを QUEUED に設定する方法を示しています。

   ```
   {
       "pipeline": {
           "name": "MyPipeline",
           "roleArn": "arn:aws:iam::111122223333:role/service-role/
   AWSCodePipelineServiceRole-us-east-1-dkpippe",
           "artifactStore": {
               "type": "S3",
   ```

```
            "location": "bucket"
        },
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "provider": "CodeCommit",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BranchName": "main",
                            "OutputArtifactFormat": "CODE_ZIP",
                            "PollForSourceChanges": "true",
                            "RepositoryName": "MyDemoRepo"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "inputArtifacts": [],
                        "region": "us-east-1",
                        "namespace": "SourceVariables"
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "Build",
                        "actionTypeId": {
                            "category": "Build",
                            "owner": "AWS",
                            "provider": "CodeBuild",
                            "version": "1"
                        },
                        "runOrder": 1,
```

```
                        "configuration": {
                            "ProjectName": "MyBuildProject"
                        },
                        "outputArtifacts": [
                            {
                                "name": "BuildArtifact"
                            }
                        ],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1",
                        "namespace": "BuildVariables"
                    }
                ]
            }
        ],
        "version": 1,
        "executionMode": "QUEUED"
    }
}
```

3. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルの構造を変更する必要があります。metadata コマンドが使用できるように、ファイルから update-pipeline 行を削除する必要があります。JSON ファイルのパイプライン構造からセクションを削除します ("metadata": { } 行と、"created"、"pipelineARN"、および"updated"フィールド)。

   例えば、構造から以下の行を削除します。

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

   ファイルを保存します。

4. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。

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

パイプラインを最初から再実行することなく、失敗したステージを再試行できます。そのためには、ステージ内の失敗したアクションを再試行するか、ステージ内の最初のアクションから始めてステージ内のすべてのアクションを再試行します。ステージ内の失敗したアクションを再試行する場合、進行中のすべてのアクションはそのまま動作し続け、失敗したアクションは再度トリガーされます。失敗したアクションのあるステージ内で最初のアクションから再試行する場合、ステージに進行中のアクションがないことが必要です。ステージを再試行するには、すべてのアクションが失敗しているか、一部のアクションが失敗して他のアクションが成功している必要があります。

> **Important**
>
> 失敗したステージを再試行すると、そのステージ内の最初のアクションからすべてのアクションが再試行されます。失敗したアクションを再試行すると、ステージ内のすべての失敗したアクションが再試行されます。これにより、同じ実行内で以前に成功したアクションの出力アーティファクトは上書きされます。

アーティファクトが上書きされても、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、アクションが失敗したかどうかを判断できます。

> **Note**
>
> 次の場合は、ステージを再試行できないことがあります。
>
> - ステージ内のすべてのアクションが成功したため、ステージが失敗ステータスになっていない。
> - ステージが失敗した後、パイプライン全体の構造が変更された。
> - ステージで別の再試行がすでに進行中です。

トピック

失敗したステージを再試行する (コンソール)

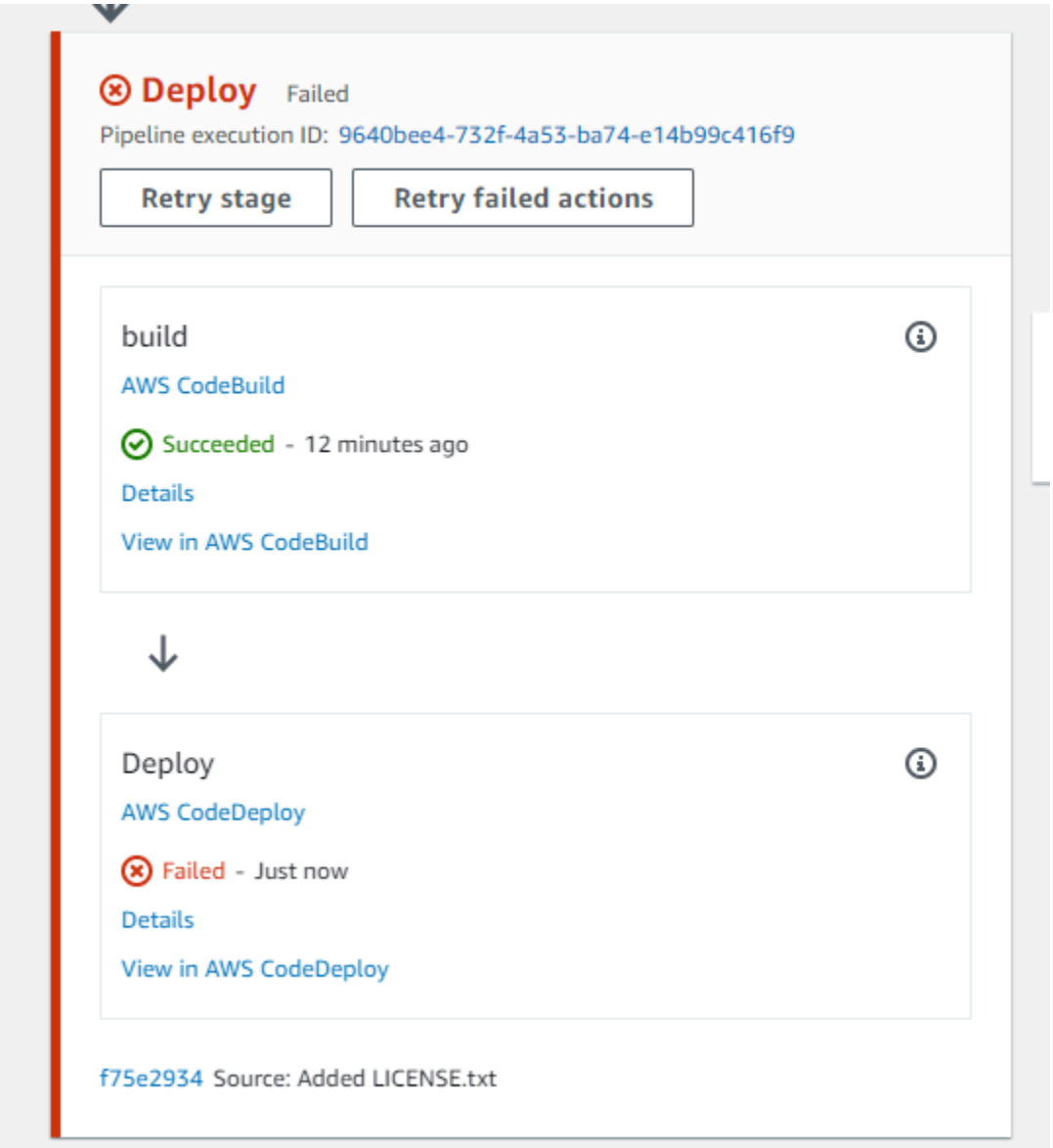
失敗したステージまたはステージ内の失敗したアクションを再試行するには

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、パイプラインの名前を選択します。
3. 失敗したアクションのあるステージを見つけ、次のいずれかを選択します。

   - ステージ内のすべてのアクションを再試行するには、[ステージの再試行] を選択します。

- ステージ内の失敗したアクションのみを再試行するには、[失敗したアクションの再試行] を選択します。



再試行したアクションがすべて正常に完了した場合、パイプラインは続行されます。

失敗したステージを再試行する (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を AWS CLI 再試行するには、次のパラメータを指定して retry-stage-execution コマンドを実行します。

```
--pipeline-name <value>
--stage-name <value>
--pipeline-execution-id <value>
--retry-mode ALL_ACTIONS/FAILED_ACTIONS
```

> **Note**
>
> retry-mode に使用できる値は FAILED_ACTIONS と ALL_ACTIONS です。

1. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、retry-stage-execution コマンドを実行します。次の例では、MyPipeline という名前のパイプラインに対して実行しています。

   ```
   aws codepipeline retry-stage-execution --pipeline-name MyPipeline --stage-name
    Deploy --pipeline-execution-id b59babff-5f34-EXAMPLE --retry-mode FAILED_ACTIONS
   ```

   出力は実行 ID を返します。

   ```
   {
       "pipelineExecutionId": "b59babff-5f34-EXAMPLE"
   }
   ```

2. JSON 入力ファイルを使用してコマンドを実行することもできます。まず、パイプライン、失敗したアクションを含むステージ、そのステージでの最新のパイプラインの実行を識別する JSON ファイルを作成します。その後、retry-stage-execution コマンドに --cli-input-json パラメータを指定して実行します。JSON ファイルに必要な詳細を取得するには、get-pipeline-state コマンドを使用するのが最も簡単です。

   a. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、パイプラインの get-pipeline-state コマンドを実行します。例えば、 という名前のパイプラインの場合は MyFirstPipeline、次のようなものを入力します。

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

      コマンドに対する応答には、各ステージのパイプラインの状態情報が含まれます。以下の例では、応答は [Staging] ステージで 1 つ以上のアクションが失敗したことを示しています。

```
{
    "updated": 1427245911.525,
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [...],
            "stageName": "Source",
            "latestExecution": {
                "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
                "status": "Succeeded"
            }
        },
        {
            "actionStates": [...],
            "stageName": "Staging",
            "latestExecution": {
                "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
                "status": "Failed"
            }
        }
    ]
}
```

b. プレーンテキストエディタで、JSON 形式で以下の情報を記録するファイルを作成します。

- 失敗したアクションを含むパイプラインの名前
- 失敗したアクションを含むステージの名前
- ステージでの最新のパイプラインの実行 ID
- 再試行モード

前の MyFirstPipeline 例では、ファイルは次のようになります。

```
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

c. **retry-failed-actions.json** のような名前でファイルを保存します。

d. retry-stage-execution コマンドを実行したときに作成したファイルを呼び出します。例:

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

```
aws codepipeline retry-stage-execution --cli-input-json file://retry-failed-actions.json
```

e. 再試行の結果を表示するには、 CodePipeline コンソールを開き、失敗したアクションを含むパイプラインを選択するか、 get-pipeline-state コマンドを再度使用します。詳細については、「でパイプラインと詳細を表示する CodePipeline」を参照してください。

ステージロールバックの設定

ステージを、そのステージで成功した実行にロールバックできます。失敗時にロールバックするようにステージを事前設定することも、ステージを手動で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、新しい実行が行われます。ロールバック用に選択されたターゲットパイプラインの実行は、ソースリビジョンと変数を取得するために使用されます。

実行のタイプは、標準またはロールバックのいずれかで、パイプライン履歴、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
- 自動ロールバックのステージを設定する
- 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次のとおりです。

- 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た場合にのみ、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。

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た場合にのみ、以前の実行にロールバックできます。

「」で説明されているように、障害発生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自動ロールバックのステージを設定する。

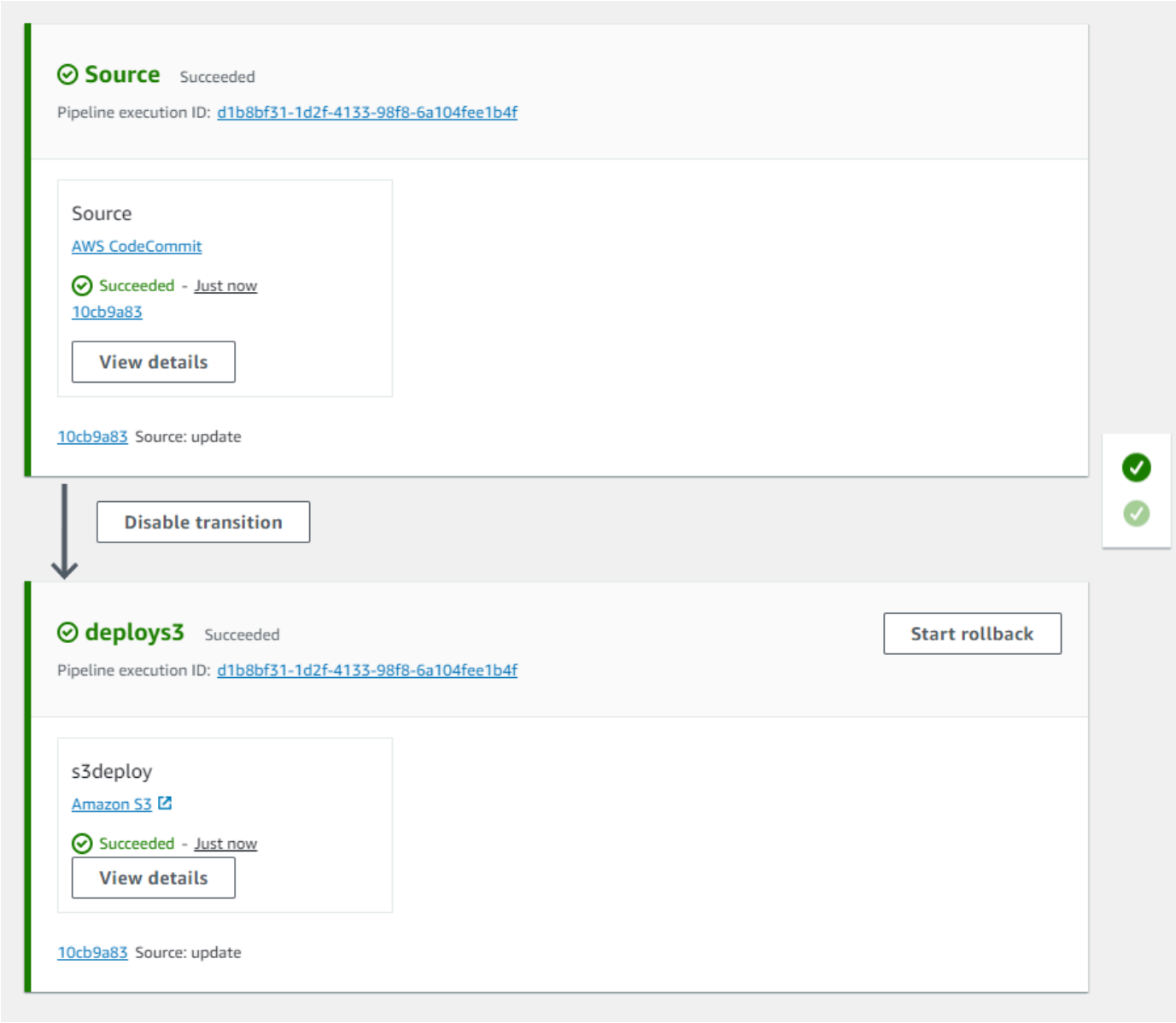
ステージを手動でロールバックする (コンソール)

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の実行に手動でロールバックできます。ステージがロールバックされると、ロールバックラベルがコンソールのパイプラインの視覚化に表示されます。

ステージを手動でロールバックする (コンソール)

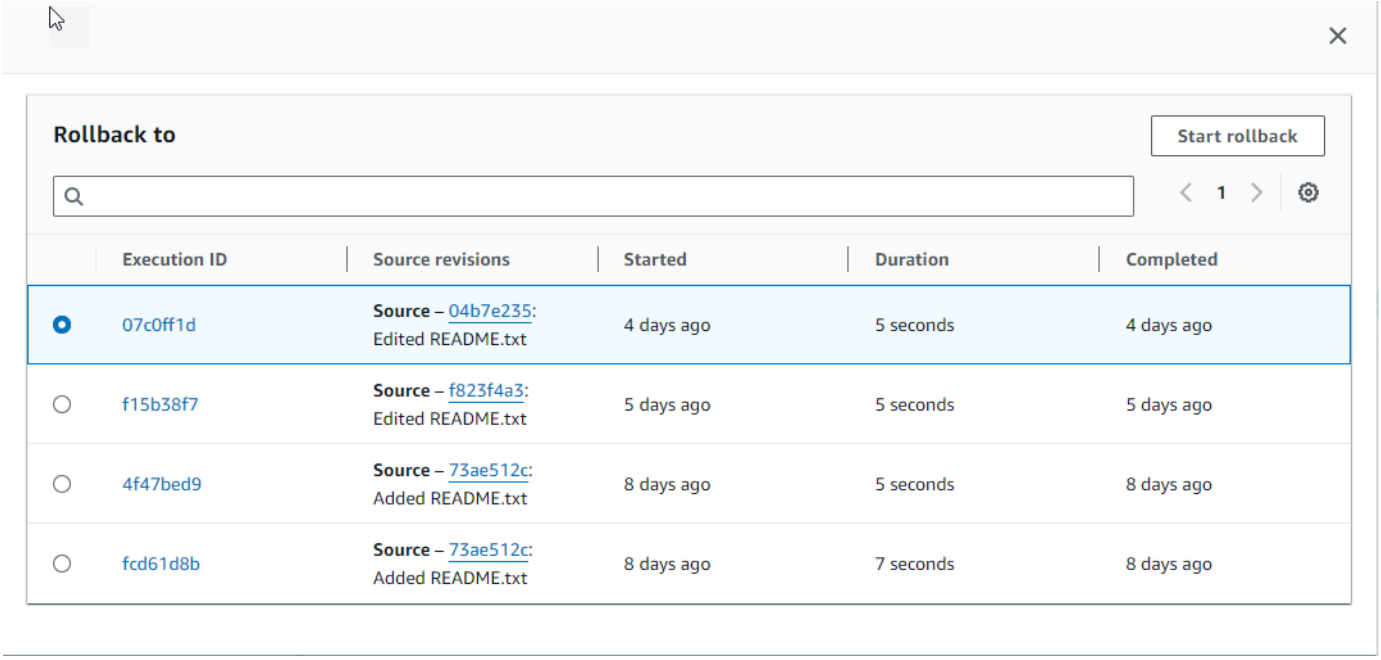
1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. 名前 で、ロールバックするステージを持つパイプラインの名前を選択します。

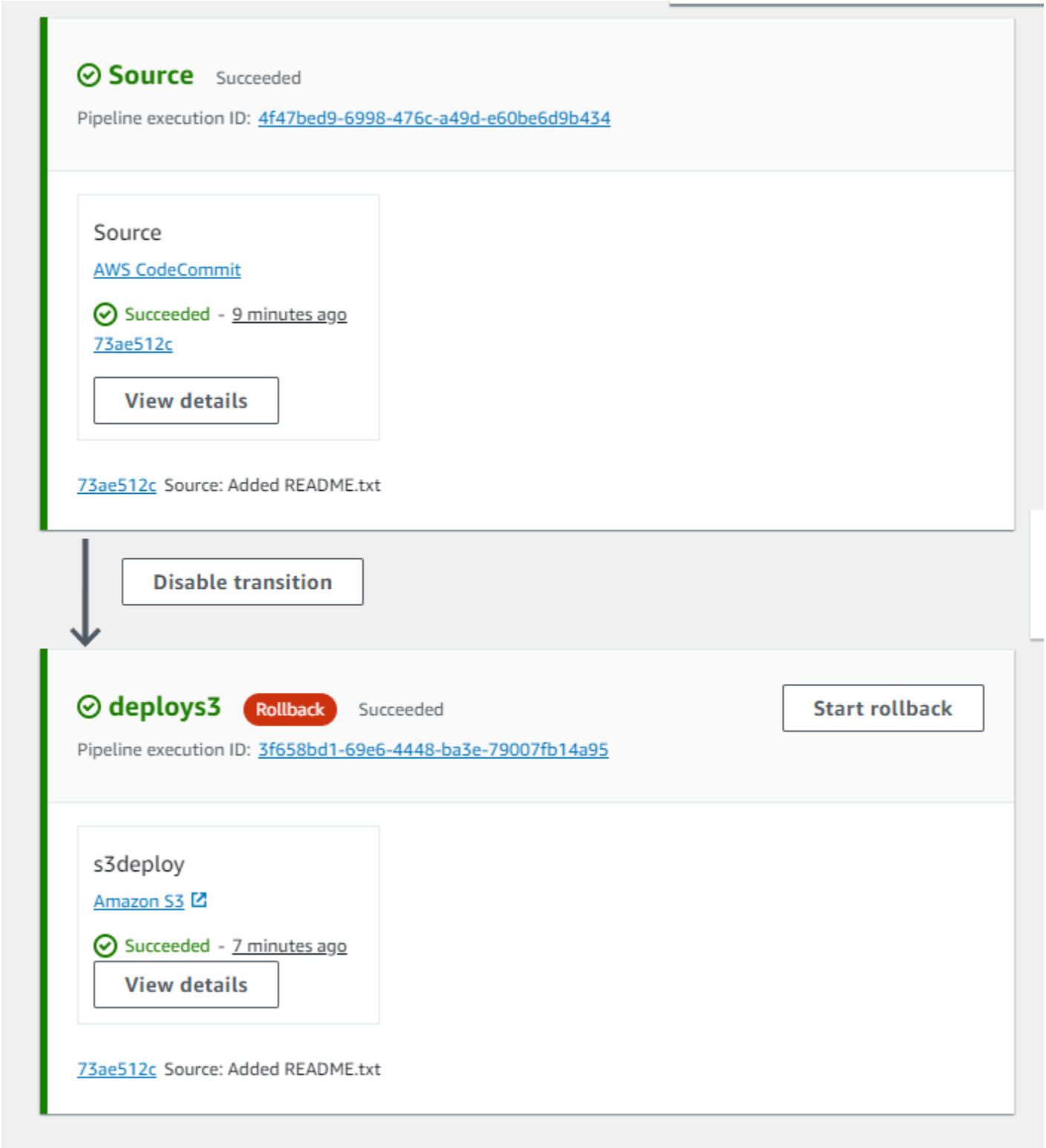


3. ステージで、ロールバックの開始 を選択します。ロールバックページが表示されます。
4. ステージをロールバックするターゲット実行を選択します。

> **Note**
>
> 使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でのすべての実行になります。



次の図は、新しい実行 ID を持つロールバックステージ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。

で説明されているように、ステージを手動でロールバックすることもできますステージを手動でロールバックする。

> **Note**
>
> 使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でのすべての実行になります。

ステージを手動でロールバックするには (CLI)

1. 手動ロールバックの CLI コマンドには、ステージで以前に成功したパイプライン実行の実行 ID が必要です。指定するターゲットパイプラインの実行 ID を取得するには、 ステージで成功した実行を返すフィルターで list-pipeline-executions コマンドを使用します。ターミナル (Linux、macOS または Unix) またはコマンドプロンプト (Windows) を開き、 AWS CLI を使用してlist-pipeline-executionsコマンドを実行し、パイプラインの名前とステージで正常に実行するためのフィルターを指定します。この例では、出力には、 という名前のパイプラインの実行 MyFirstPipeline と、 という名前のステージで正常に実行されたパイプラインの実行が一覧表示されますdeploys3。

   ```
   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。これは、次のステップでターゲット実行 ID として使用します。

2. 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、を使用して AWS CLI rollback-stageコマンドを実行し、パイプラインの名前、ステージの名前、ロールバックするターゲット実行を指定します。例えば、 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、次のようにしますMyFirstPipeline。

   ```
   aws codepipeline rollback-stage --pipeline-name MyFirstPipeline --stage-name Deploy
    --target-pipeline-execution-id bc022580-4193-491b-8923-9728dEXAMPLE
   ```

   出力は、新しいロールバック実行の実行 ID を返します。これは、指定され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。

自動ロールバックのステージを設定する

パイプラインのステージは、障害発生時に自動的にロールバックするように設定できます。ステージが失敗すると、ステージは最後に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た場合にのみ、以前の実行にロールバックできます。自動ロールバック設定はパイプライン定義の一部であるため、パイプラインステージは、パイプラインステージでパイプラインが正常に実行された後にのみ自動ロールバックされます。

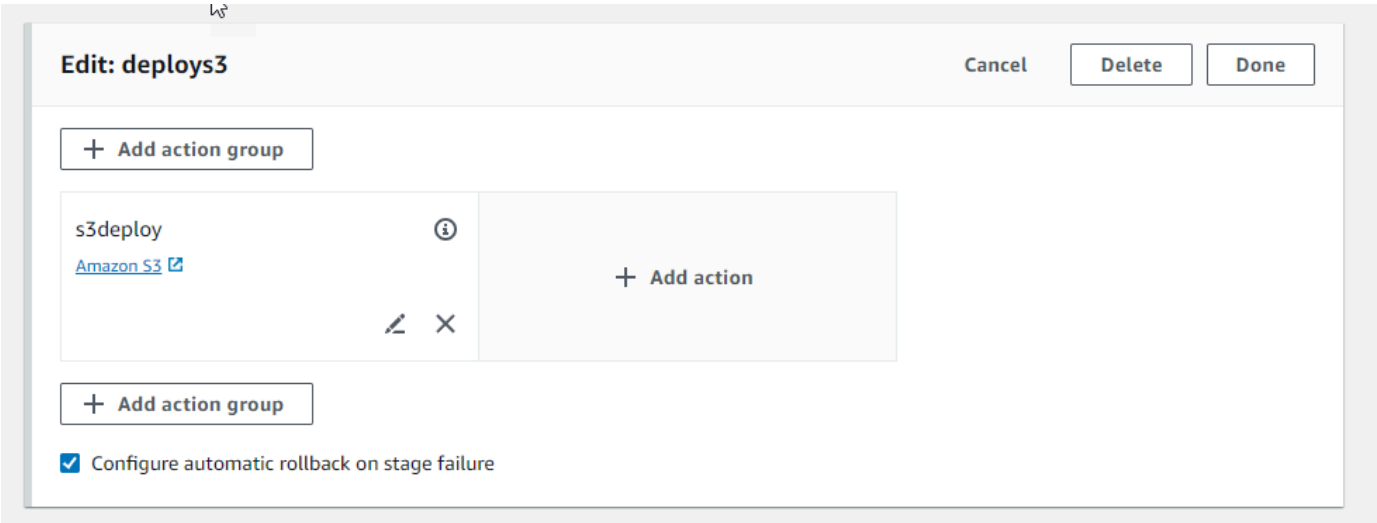
自動ロールバックのステージを設定する (コンソール)

ステージは、指定した前回の正常な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RollbackStage」の「」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 (コンソール)

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

   AWS アカウントに関連付けられているすべてのパイプラインの名前とステータスが表示されます。
2. [Name] 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ついて、ステージの編集 を選択します。
5. ステージ障害時に自動ロールバックを設定する を選択します。パイプラインに変更を保存します。



自動ロールバックのステージを設定する (CLI)

を使用して、最後に正常に実行され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、 でパイプラインを作成する CodePipeline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でパイプラインを編集する CodePipeline。

- 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update-pipelineコマンドを実行し、パイプライン構造で障害条件を指定します。次の例では、 という名前のステージングされた の自動ロールバックを設定しますS3Deploy。

```
{
                "name": "S3Deploy",
                "actions": [
                    {
                        "name": "s3deployaction",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "provider": "S3",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "BucketName": "static-website-bucket",
                            "Extract": "false",
                            "ObjectKey": "SampleApp.zip"
                        },
                        "outputArtifacts": [],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "region": "us-east-1"
                    }
                ],
                "onFailure": {
                    "result": "ROLLBACK"
                }
            }
```

ステージロールバックの障害条件の設定の詳細については、 API リファレンスFailureConditionsの「」を参照してください。 CodePipeline

自動ロールバックのステージを設定する (AWS CloudFormation)

AWS CloudFormation を使用して、障害発生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後に成功した実行にロールバックされます。

```
OnFailure:
     Result: ROLLBACK
```

- 次のスニペットに示すように、テンプレートを更新します。次の例では、 という名前のステージングされた の自動ロールバックを設定しますRelease。

```
AppPipeline:
  Type: AWS::CodePipeline::Pipeline
  Properties:
    RoleArn:
      Ref: CodePipelineServiceRole
    Stages:
      -
        Name: Source
        Actions:
          -
            Name: SourceAction
            ActionTypeId:
              Category: Source
              Owner: AWS
              Version: 1
              Provider: S3
            OutputArtifacts:
              -
                Name: SourceOutput
            Configuration:
              S3Bucket:
                Ref: SourceS3Bucket
              S3ObjectKey:
                Ref: SourceS3ObjectKey
            RunOrder: 1
```

```
      -
        Name: Release
        Actions:
          -
            Name: ReleaseAction
            InputArtifacts:
              -
                Name: SourceOutput
            ActionTypeId:
              Category: Deploy
              Owner: AWS
              Version: 1
              Provider: CodeDeploy
            Configuration:
              ApplicationName:
                Ref: ApplicationName
              DeploymentGroupName:
                Ref: DeploymentGroupName
            RunOrder: 1
       OnFailure:
            Result: ROLLBACK
    ArtifactStore:
      Type: S3
      Location:
        Ref: ArtifactStoreS3Location
      EncryptionKey:
        Id: arn:aws:kms:useast-1:ACCOUNT-ID:key/KEY-ID
        Type: KMS
    DisableInboundStageTransitions:
      -
        StageName: Release
        Reason: "Disabling the transition until integration tests are completed"
    Tags:
      - Key: Project
        Value: ProjectA
      - Key: IsContainerBased
        Value: 'true'
```

ステージロールバックの障害条件の設定の詳細については、「 ユーザーガイドStageDeclaration」のOnFailure「」のAWS CloudFormation 「」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。

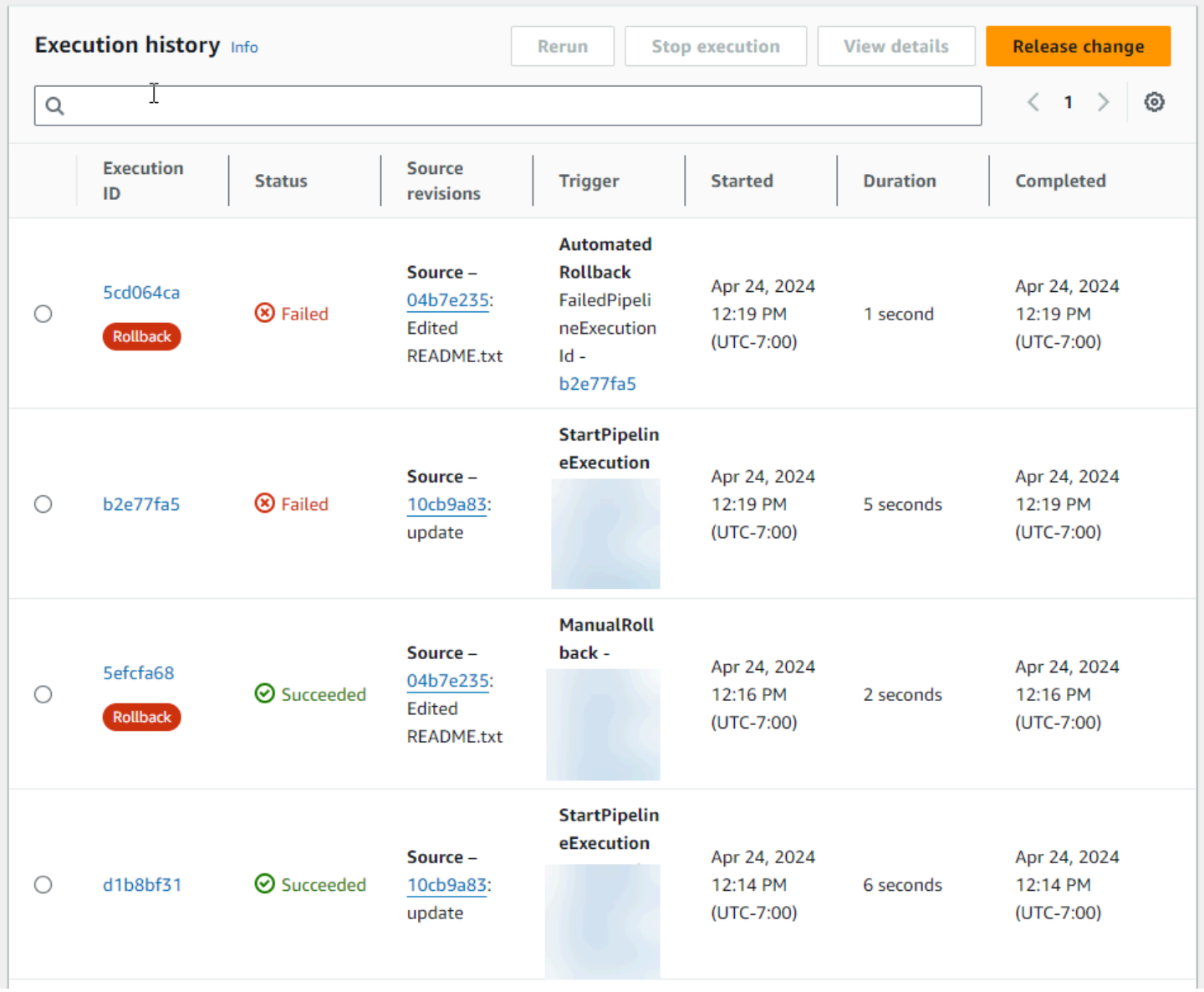
実行のリストでロールバックステータスを表示する (コンソール)

コンソールを使用して、ロールバック実行のステータスとターゲット実行 ID を実行リストに表示できます。

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   に関連付けられている AWS アカウント 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. [History (履歴)] を選択します。実行のリストには、ロールバック というラベルが表示されます。



詳細を表示する実行 ID を選択します。

`list-pipeline-executions` (CLI) でロールバックステータスを表示する

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完了したすべての実行のリストを返します。

次の例は、ロールバック実行*MyFirstPipeline*によってパイプラインが開始された という名前のパイプラインに対し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"
                }
            ],
```

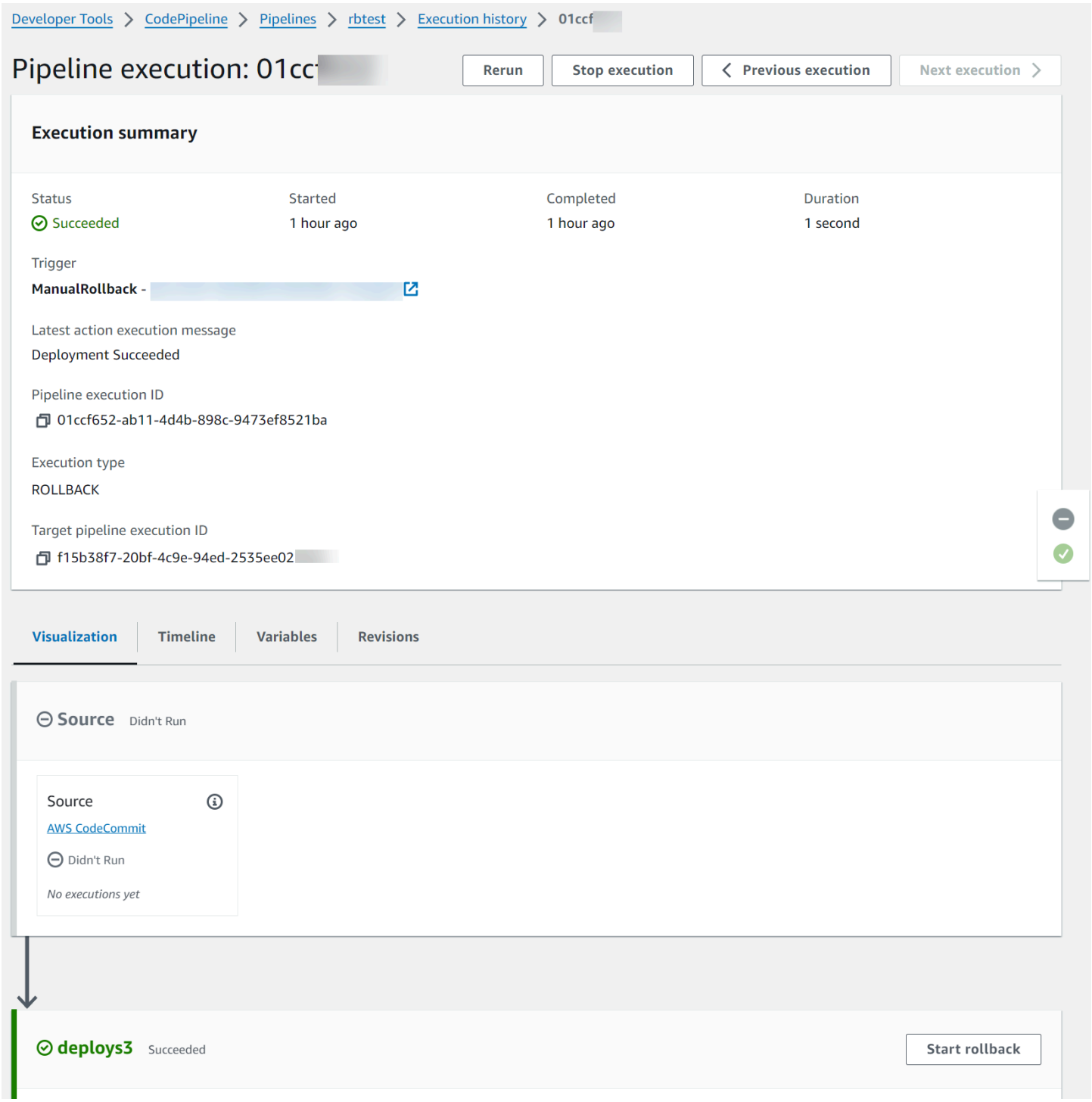
```
            "trigger": {
                "triggerType": "StartPipelineExecution",
                "triggerDetail": "arn:aws:sts::<account_ID>:assumed-role/<role>"
            },
            "executionMode": "SUPERSEDED"
        },
        {
            "pipelineExecutionId": "5cd064ca-bff7-425f-8653-f41d9EXAMPLE",
            "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、ロールバック実行のステータスとターゲットパイプライン実行 ID を表示できます。



get-pipeline-execution (CLI) を使用してロールバックの詳細を表示する

ロールバックされたパイプライン実行は、パイプライン実行を取得するための出力に表示されます。

- パイプラインの詳細を表示するには、パイプラインの一意の名前を指定して、get-pipeline-execution コマンドを実行します。例えば、 という名前のパイプラインの詳細を表示するには*MyFirstPipeline*、次のように入力します。

```
aws codepipeline get-pipeline-execution --pipeline-name MyFirstPipeline --pipeline-
execution-id 3f658bd1-69e6-4448-ba3e-79007EXAMPLE
```

このコマンドは、パイプラインの構造を返します。

次の例は、ロールバック実行 ID とメタデータが表示されている*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。

```
{
    "pipelineExecution": {
        "pipelineName": "MyFirstPipeline",
        "pipelineVersion": 6,
        "pipelineExecutionId": "2004a94e-8b46-4c34-a695-c8d20EXAMPLE",
        "status": "Succeeded",
        "artifactRevisions": [
            {
                "name": "SourceArtifact",
                "revisionId": "<ID>",
                "revisionSummary": "Added README.txt",
                "revisionUrl": "<console_URL>"
            }
        ],
        "trigger": {
            "triggerType": "ManualRollback",
            "triggerDetail": "arn:aws:sts::<account_ID>:assumed-role/<role>"
        },
        "executionMode": "SUPERSEDED",
        "executionType": "ROLLBACK",
        "rollbackMetadata": {
            "rollbackTargetPipelineExecutionId": "4f47bed9-6998-476c-a49d-
e60beEXAMPLE"
        }
    }
}
```

(**get-pipeline-state**CLI) でロールバック状態を表示する

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、 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*MyFirstPipeline*、次のように入力します。

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

次の例は、返されたデータをロールバック実行タイプで示しています。

```
{
    "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"
}
```

でのアクションの操作 CodePipeline

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、次の 6 種類のアクションがサポートされています。

- 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

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、 で利用可能になった後にパイプラインに追加できます CodePipeline。

> **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、「でカスタムアクションを作成して追加する CodePipeline」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、統合を Lambda 関数として書き込みます。Lambda 関数は、アクションに使用できるジョブがある CodePipeline たびに によって呼び出されます。
  - JobWorker： 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使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統合を書き込みます。次に、ジョブワーカーはジョブを実行し、API CodePipeline を使用してジョブ結果を に送信します。 CodePipeline APIs

> **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

サードパーティープロバイダーによって新しい CodePipeline アクションタイプがリクエストされると、 のアクションタイプの所有者に対してアクションタイプが作成され CodePipeline、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パブリックアクションに変更するように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、エグゼキュターリソース、およびアクションタイプのリクエストを作成する前に、統合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 Lambda 関数にアクセス許可を追加して、 CodePipeline サービスプリンシパル を使用して呼び出すアクセス許可を CodePipeline サービスに提供しますcodepipeline.amazonaws.com。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
- ジョブワーカー統合モデルでは、ジョブワーカーが APIs を使用してジョブをポーリングするCodePipeline許可されたアカウントのリストとの統合を作成します。

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、 CodePipeline API リファレンスUpdateActionTypeのActionTypeDeclaration「」および「」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 、お客様のアカウントのアクセス許可を指定するポリシーステートメント。
  - type: Lambda または JobWorker のアクションタイプの作成と更新に使用されるインテグレーションモデル。
- id: アクションタイプのカテゴリ、所有者、プロバイダー、およびバージョン ID。
  - category: ソース、ビルド、デプロイ、テスト、呼び出し、承認のステージで実行できるアクションの種類。
  - provider: プロバイダ会社や製品名など、呼び出されるアクションタイプのプロバイダ。プロバイダ名は、アクションタイプの作成時に指定されます。
  - owner: AWS または ThirdParty の呼び出されているアクションの作成者。
  - version: アクションタイプのバージョン設定に使用する文字列。最初のバージョンでは、バージョン番号を 1 に設定します。
- inputArtifactDetails: パイプラインの前のステージから期待されるアーティファクトの数。
- outputArtifactDetails: アクションタイプステージの結果から期待されるアーティファクトの数。
- permissions: アクションタイプを使用する許可のあるアカウントを識別する詳細。

- properties: プロジェクトタスクを完了するために必要なパラメータ。
  - description: ユーザーに表示されるプロパティの説明。
  - optional: 設定プロパティがオプションであるかどうか。
  - noEcho: 顧客が入力したフィールド値をログから除外するかどうか。の場合true、 GetPipeline API リクエストで返された値は編集されます。
  - 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: URLs CodePipeline のリストがユーザーに表示されます。
  - 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: コンソールに表示される URL で CodePipeline、ユーザー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が表示されます。
  - thirdPartyConfigurationUrl: ユーザーが外部サービスにサインアップし、そのサービスによって提供されるアクションの初期設定を実行できるページへの URL。

次のコードは、アクションタイプ定義ファイルの例を示しています。

```
{
   "actionType": {
      "description": "string",
      "executor": {
         "configuration": {
            "jobWorkerExecutorConfiguration": {
               "pollingAccounts": [ "string" ],
               "pollingServicePrincipals": [ "string" ]
            },
            "lambdaExecutorConfiguration": {
               "lambdaFunctionArn": "string"
            }
         },
         "jobTimeout": number,
         "policyStatementsTemplate": "string",
         "type": "string"
      },
```

```
      "id": {
         "category": "string",
         "owner": "string",
         "provider": "string",
         "version": "string"
      },
      "inputArtifactDetails": {
         "maximumCount": number,
         "minimumCount": number
      },
      "outputArtifactDetails": {
         "maximumCount": number,
         "minimumCount": number
      },
      "permissions": {
         "allowedAccounts": [ "string" ]
      },
      "properties": [
         {
            "description": "string",
            "key": boolean,
            "name": "string",
            "noEcho": boolean,
            "optional": boolean,
            "queryable": boolean
         }
      ],
      "urls": {
         "configurationUrl": "string",
         "entityUrlTemplate": "string",
         "executionUrlTemplate": "string",
         "revisionUrlTemplate": "string"
      }
   }
}
```

ステップ 3: との統合を登録する CodePipeline

アクションタイプを に登録するには CodePipeline、 CodePipeline サービスチームに連絡してリクエストしてください。

CodePipeline サービスチームは、サービス codebase に変更を加えて新しいアクションタイプ統合を登録します。 は、パブリックアクション とプライベートアクション の 2 つの新しいアクション

CodePipeline を登録します。プライベートアクションをテストに使用し、準備ができたら、顧客トラフィックを処理する公開アクションを起動します。

Lambda インテグレーションのリクエストを登録するには

- 次のフォームを使用して、 CodePipeline サービスチームにリクエストを送信します。

```
This issue will track the onboarding of [Name] in CodePipeline.


[Contact engineer] will be the primary point of contact for this integration.

Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:

1. Attach an action type definition file in JSON format. This includes the schema
 for the action type

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. The Lambda function ARN

4. List of AWS ##### where your action will be available

5. Will this be available as a public action?
```

ジョブワーカーインテグレーションのリクエストを登録するには

- 次のフォームを使用して、 CodePipeline サービスチームにリクエストを送信します。

```
This issue will track the onboarding of [Name] in CodePipeline.

[Contact engineer] will be the primary point of contact for this integration.


Name of the action type as you want it to appear to customers: Example.com Testing

Initial onboard checklist:
```

```
1. Attach an action type definition file in JSON format. This includes the schema
 for the action type.

2. A list of test accounts for the allowlist which can access the new action type
 [{account, account_name}]

3. URL information:
Website URL: https://www.example.com/%TestThirdPartyName%/%TestVersionNumber%

Example URL pattern where customers will be able to review their configuration
 information for the action: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%CustomerActionConfiguration%

Example runtime URL pattern: https://www.example.com/%TestThirdPartyName%/
%customer-ID%/%TestRunId%

4. List of AWS ##### where your action will be available

5. Will this be available as a public action?
```

ステップ 4: 新しいインテグレーションを起動する

新しい統合をパブリックに使用する準備ができたら、 CodePipeline サービスチームに連絡してください。

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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

2. パイプラインのリストで、アクションタイプを追加したいパイプラインを選択します。
3. パイプラインの概要ビューページで、[編集] を選択します。
4. ステージの編集を選択します。アクションタイプを追加したいステージで、[Add action group (アクショングループの追加)] を選択します。[アクションの編集] ページが表示されます。
5. [アクションの編集] ページで、[Action name (アクション名)] にアクションの名前を入力します。これは、パイプラインのステージに表示される名前です。
6. [アクションプロバイダ] で、リストからアクションタイプを選択します。

   リスト内の値は、アクションタイプ定義ファイルで指定された `provider` に基づいています。
7. [入力アーティファクト] で、次の形式でアーティファクト名を入力します。

   `Artifactname``::`*`FileName`*

   許容される最小数量と最大数量は、アクションタイプ定義ファイルで指定される `inputArtifactDetails` に基づいて定義されます。
8. [<Action_Name> への接続] を選択します。

   ブラウザウィンドウが開き、アクションタイプ用に作成したウェブサイトに接続します。
9. 顧客としてウェブサイトにログインし、顧客がアクションタイプを使用するための手順を完了します。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。
10. CodePipeline 「アクションの編集」ページに、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、でパイプラインを作成する CodePipeline のステップに従い、テストをするステージの数だけアクションタイプを選択します。

アクションタイプを表示する

CLI を使用して、アクションタイプを表示できます。get-action-type コマンドを使用して、インテグレーションモデルを使用して作成されたアクションタイプを表示します。

アクションタイプを表示するには

1. 入力 JSON ファイルを作成し、ファイルの名前を `file.json` にします。アクションタイプ ID を次の例に示すように JSON 形式で追加します。

```
{
    "category": "Test",
    "owner": "ThirdParty",
    "provider": "TestProvider",
    "version": "1"
}
```

2. ターミナルウィンドウまたはコマンドラインで、get-action-type コマンドを実行します。

```
aws codepipeline get-action-type --cli-input-json file://file.json
```

このコマンドは、アクションタイプのアクション定義の出力を返します。この例では、Lambda インテグレーションモデルで作成されたアクションタイプを表示します。

```
{
    "actionType": {
        "executor": {
            "configuration": {
```

```
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda"
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
            "allowedAccounts": [
                "<account-id>"
            ]
        },
        "properties": []
    }
}
```

アクションタイプを更新する

CLI を使用して、インテグレーションモデルで作成されたアクションタイプを編集できます。

公開アクションタイプの場合、所有者を更新することはできず、オプションのプロパティを必須事項に変更することはできません。また、新しいオプションプロパティのみを追加できます。

1. get-action-type コマンドを使用して、アクションタイプの構造を取得します。構造をコピーします。

2. 入力 JSON ファイルを作成し、ファイルの名前を action.json にします。前のステップでコピーしたアクションタイプ構造を、そこに貼り付けます。変更したいパラメータを更新します。オプションのパラメータを追加することもできます。

   入力ファイルのパラメータの詳細については、ステップ 2: アクションタイプ定義ファイルを作成する の「アクション定義ファイルの説明」を参照してください。

   次の例では、Lambda インテグレーションモデルで作成されたアクションタイプの例を更新する方法を表示します。この例では、以下の変更が発生します。

   - provider の名前を TestProvider1 に変更します。
   - 900 秒のジョブタイムアウト制限を追加します。
   - Host という名前のアクション設定プロパティを追加します。これは、アクションを使用する顧客に表示されます。

```
{
    "actionType": {
        "executor": {
            "configuration": {
                "lambdaExecutorConfiguration": {
                    "lambdaFunctionArn": "arn:aws:lambda:us-west-2:<account-
id>:function:my-function"
                }
            },
            "type": "Lambda",
            "jobTimeout": 900
        },
        "id": {
            "category": "Test",
            "owner": "ThirdParty",
            "provider": "TestProvider1",
            "version": "1"
        },
        "inputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "outputArtifactDetails": {
            "minimumCount": 0,
            "maximumCount": 1
        },
        "permissions": {
```

```
            "allowedAccounts": [
                "account-id"
            ]
        },
        "properties": {
         "description": "Owned build action parameter description",
         "optional": true,
         "noEcho": false,
         "key": true,
         "queryable": false,
         "name": "Host"
         }
    }
}
```

3. ターミナルまたはコマンドラインで、update-action-type コマンドを実行します。

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

このコマンドは、更新されたパラメータに適合するアクションタイプの出力を返します。

でカスタムアクションを作成して追加する CodePipeline

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多数のアクションが含まれています。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。

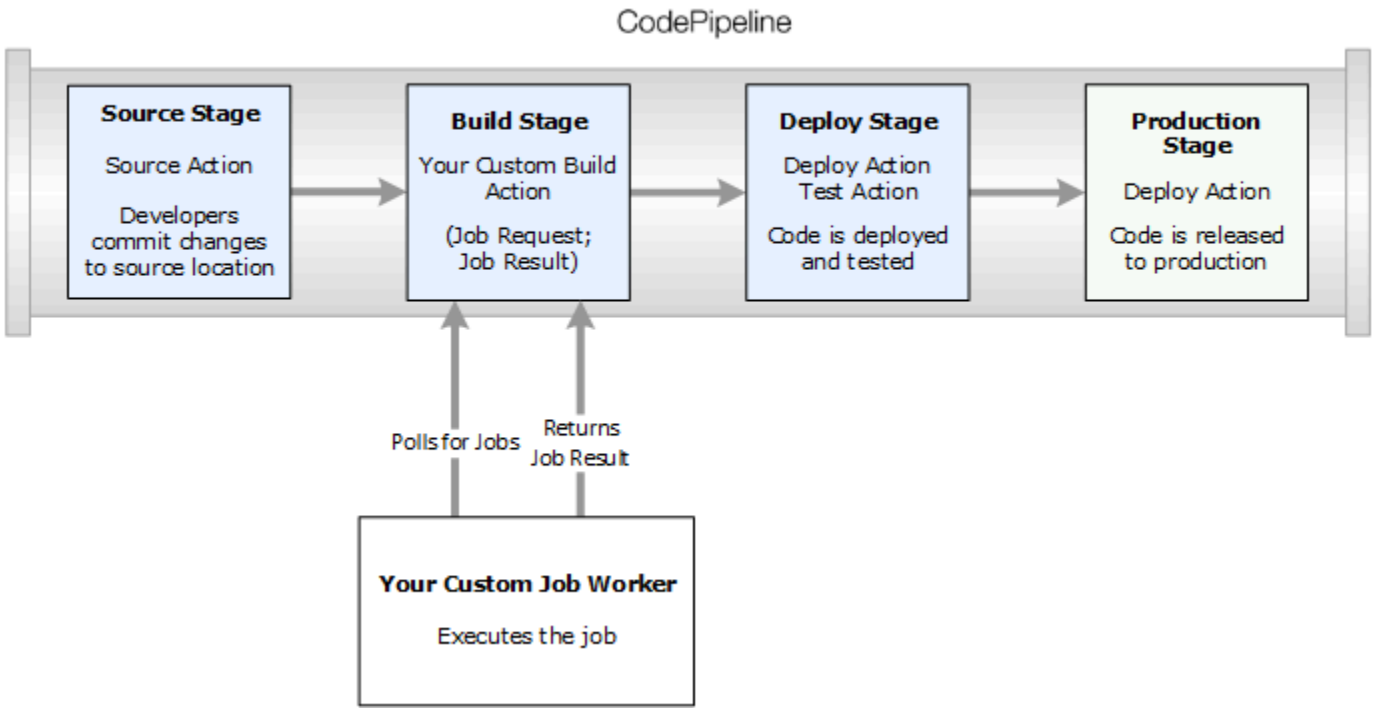
次のアクションカテゴリのカスタム AWS CodePipeline アクションを作成できます。

- 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ときは、このカスタムアクション CodePipeline のジョブリクエストをポーリングし、ジョブを実行し、ステータス結果を に返すジョブワーカーも作成する必要があり

ます CodePipeline。このジョブワーカーは、 のパブリックエンドポイントにアクセスできる限り、任意のコンピュータまたはリソースに配置できます CodePipeline。簡単にアクセスおよびセキュリティを管理するために、ジョブワーカーを Amazon EC2 インスタンスにホストすることを考慮してください。

次の図では、カスタム構築アクションを含むパイプラインの高レベルビューを示します:



パイプラインにステージの一部としてカスタムアクションが含まれる場合、パイプラインはジョブリクエストを作成します。カスタムジョブワーカーはそのリクエストを検出し、そのジョブを実行します (この例では、サードパーティー構築ソフトウェアを使用するカスタムプロセス)。アクションが完了すると、ジョブワーカーは成功結果または失敗結果を返します。成功結果を受け取ると、パイプラインはリビジョンと次のアクションのアーティファクトを提供します。失敗が返された場合、パイプラインはリビジョンを次のアクションに渡しません。

> **Note**
>
> これらの手順では、すでにの開始方法 CodePipelineのステップを完了していることを前提としています。

トピック

- カスタムアクションを作成する

- カスタムアクションのジョブワーカーを作成する
- パイプラインにカスタムアクションを追加する

カスタムアクションを作成する

を使用してカスタムアクションを作成するには AWS CLI

1. テキストエディタを開き、アクションカテゴリ、アクションプロバイダー、およびカスタムアクションで必要な設定を含む、カスタムアクションの JSON ファイルを作成します。例えば、1 つのプロパティのみを必要とするカスタムビルドアクションを作成する場合、JSON ファイルは次のようになります。

```
{
    "category": "Build",
    "provider": "My-Build-Provider-Name",
    "version": "1",
    "settings": {
        "entityUrlTemplate": "https://my-build-instance/job/{Config:ProjectName}/",
        "executionUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/lastSuccessfulBuild/{ExternalExecutionId}/"
    },
    "configurationProperties": [{
        "name": "ProjectName",
        "required": true,
        "key": true,
        "secret": false,
        "queryable": false,
        "description": "The name of the build project must be provided when this
 action is added to the pipeline.",
        "type": "String"
    }],
    "inputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "outputArtifactDetails": {
        "maximumCount": integer,
        "minimumCount": integer
    },
    "tags": [{
      "key": "Project",
```

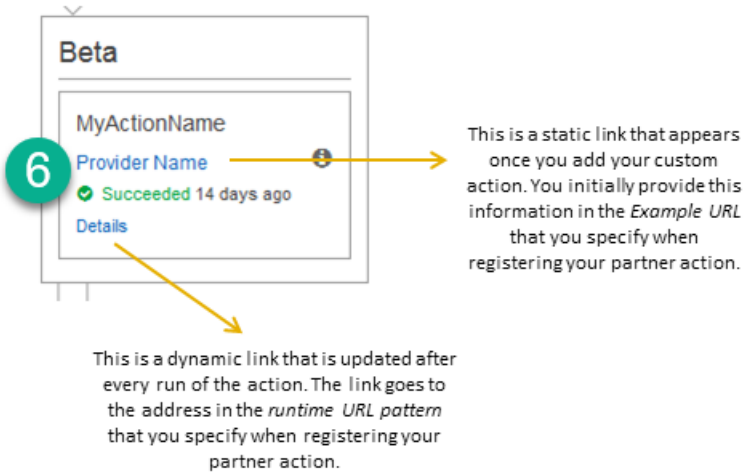
```
      "value": "ProjectA"
    }]
}
```

この例では、カスタムアクションで Project タグキーと ProjectA 値を含めることで、カスタムアクションにタグ付けを追加します。でのリソースのタグ付けの詳細については CodePipeline、「」を参照してくださいリソースのタグ付け。

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、上記のサンプルでは、entityUrlTemplate値は設定プロパティ を参照します*ProjectName*。

- 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と `RevisionURLTemplate`および の詳細については、CodePipeline API リファレンスの`ThirdPartyURL`ActionTypeSettings「」およびCreateCustomActionType「」を参照してください。アクション構造の要件とアクションの作成方法の詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

2. JSON ファイルを保存し、覚えやすい名前 (.json など) を付けます*`MyCustomAction`*。
3. AWS CLIをインストールしたコンピュータで、ターミナルセッション (Linux、OS X、Unix) またはコマンドプロンプト (Windows) を開きます。
4. AWS CLI を使用して aws codepipeline create-custom-action-type コマンドを実行し、先ほど作成した JSON ファイルの名前を指定します。

   例えば、ビルドカスタムアクションを作成するには以下のようにします。

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

```
aws codepipeline create-custom-action-type --cli-input-json
 file://MyCustomAction.json
```

5. このコマンドは、作成したカスタムアクションの構造全体、および追加された JobList アクション設定プロパティを返します。パイプラインにカスタムアクションを追加するときは、JobList を使用して、プロバイダーからのプロジェクトのうちジョブをポーリングできるものを指定できます。これを設定しない場合、カスタムジョブワーカーがジョブをポーリングするときに、使用可能なすべてのジョブが返されます。

   例えば、前のコマンドは、次のような構造を返します。

```
{
    "actionType": {
        "inputArtifactDetails": {
            "maximumCount": 1,
            "minimumCount": 1
       },
       "actionConfigurationProperties": [
            {
                "secret": false,
                "required": true,
                "name": "ProjectName",
                "key": true,
                "description": "The name of the build project must be provided when
 this action is added to the pipeline."
            }
        ],
        "outputArtifactDetails": {
            "maximumCount": 0,
            "minimumCount": 0
        },
        "id": {
            "category": "Build",
            "owner": "Custom",
            "version": "1",
```

```
            "provider": "My-Build-Provider-Name"
        },
        "settings": {
            "entityUrlTemplate": "https://my-build-instance/job/
{Config:ProjectName}/",
            "executionUrlTemplate": "https://my-build-instance/job/mybuildjob/
lastSuccessfulBuild/{ExternalExecutionId}/"
        }
    }
}
```

> **Note**
>
> create-custom-action-type コマンドの出力の一部として、 idセクションには "owner": "Custom". CodePipeline automatically がカスタムアクションタイプの所有者Customとして割り当てられます。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には、カスタムアクション CodePipeline のジョブリクエストをポーリングし、ジョブを実行し、ステータス結果を に返すジョブワーカーが必要です CodePipeline。ジョブワーカーは、 のパブリックエンドポイントにアクセスできる限り、任意のコンピュータまたはリソースに配置できます CodePipeline。

ジョブワーカーを設計する方法は複数あります。以下のセクションでは、 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 CodePipeline。

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

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 CodePipeline、ユーザーとアクセス許可の管理を統合するための戦略が必要です。

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。組み込みの との統合を使用して AWS 、カスタムジョブワーカーと 間のやり取りを簡素化できます CodePipeline。

Amazon EC2 インスタンスをセットアップする

1. Amazon EC2 の詳細を参照し、統合に適しているかどうかを判断します。詳細については、「Amazon EC2 - 仮想サーバーのホスティング」を参照してください。
2. Amazon EC2 インスタンスの作成を開始します。詳細については、「Amazon EC2 Linux インスタンスの開始方法」を参照してください。

他に考慮すべき戦略は、ID フェデレーションと IAM を使用した既存の ID プロバイダーシステムおよびリソースとの統合です。この戦略は、お客様がすでに企業 ID プロバイダーを持っているか、ウェブ ID プロバイダーを使用するユーザーをサポートできるよう設定されている場合に、特に便利です。ID フェデレーションを使用すると、IAM ユーザーを作成または管理しなくても CodePipeline、 を含む AWS リソースへの安全なアクセスを許可できます。パスワードのセキュリティ要件や認証情報の更新に機能やポリシーを活用できます。サンプルアプリケーションをお客様自身の設計のテンプレートとして使用できます。

ID フェデレーションをセットアップするには

1. IAM 認証フェデレーションの詳細について学習します。詳細については、「フェデレーションの管理」を参照してください。
2. 「一時的なアクセス権を付与するシナリオ」の例を参照して、カスタムアクションのニーズに最適な一時アクセスのシナリオを確認します。
3. インフラストラクチャに関連する ID フェデレーションのコード例を確認します。例えば、以下の参照先をご覧ください。
   - Active Directory ユースケースのための ID フェデレーションのサンプルアプリケーション
4. ID フェデレーションの設定を開始します。詳細については、「IAM ユーザーガイド」の「ID プロバイダーとフェデレーション」を参照してください。

カスタムアクションとジョブワーカーを実行するときに、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、 に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 Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の AWS CLI 「 を使用するための の設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」

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
		の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使用した認証」を参照してください。 AWS SDKs • AWS APIs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。

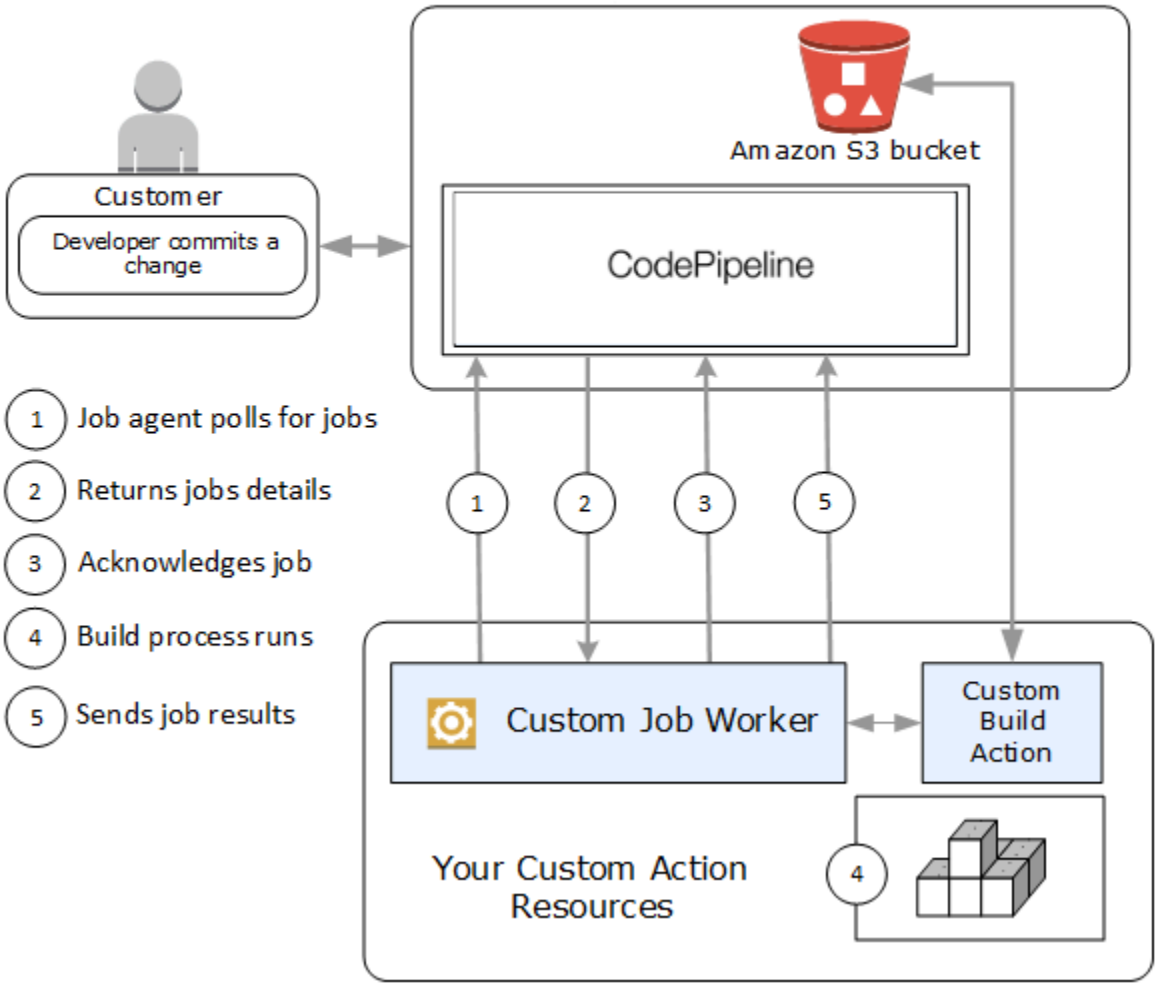
次は、カスタムジョブワーカーで使用するために作成する可能性があるポリシーの例です。このポリシーは例に過ぎず、そのまま提供されています。

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

> **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

アクセス許可管理戦略を選択したら、ジョブワーカーが とどのようにやり取りするかを検討する必要があります CodePipeline。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、 を使用してジョブ CodePipeline をポーリングします`PollForJobs`。

2. パイプラインがソースステージでの変更によってトリガーされる際 (例えば、開発者が変更をコミットした際)、自動リリースプロセスが開始します。プロセスは、カスタムアクションが設定されたステージまで継続します。このステージでアクションに達すると、 はジョブを CodePipeline キューに入れます。このジョブは、ジョブワーカーがステータスを取得するために PollForJobs を再度呼び出すと、表示されます。PollForJobs からジョブ詳細を取得し、ジョブワーカーに渡します。
3. ジョブワーカーが AcknowledgeJobを呼び出して CodePipeline ジョブ確認を送信します。ジョブワーカーがジョブを続行する必要があることを示す CodePipeline 確認応答が返されます (InProgress) 。または、ジョブのポーリングが複数あり、別のジョブワーカーが既にジョブを申請している場合、InvalidNonceException工ラー応答が返されます。確認後、InProgress は結果が返されるまで CodePipeline 待機します。
4. ジョブワーカーはリビジョンでカスタムアクションを開始し、その後にアクションが実行されます。他のアクションとともに、カスタムアクションは結果をジョブワーカーに返します。ビルドカスタムアクションの例では、アクションが Amazon S3 バケットからアーティファクトを引き出し、構築して、構築されたアーティファクトを Amazon S3 バケットに正常にプッシュします。
5. そのアクションが実行されている間、ジョブワーカーは、継続トークン (ジョブワーカーによって生成されたジョブの状態のシリアル化、例えば JSON 形式のビルド識別子や Amazon S3 オブジェクトキー) を使用して PutJobSuccessResult を呼び出すことができ、さらに ExternalExecutionId 情報 (executionUrlTemplate のリンクの入力に使用される) も呼び出すことができます。これは、進行中に特定のアクション詳細への有効なリンクとともにパイプラインのコンソールビューを更新します。必要ではありませんが、ユーザーがカスタムアクションの実行中にそのステータスを確認することを可能にするため、ベストプラクティスです。

   PutJobSuccessResult が呼び出されると、ジョブは完了したと見なされます。継続トークン CodePipeline を含む新しいジョブが 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。(のクォータ AWS CodePipeline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。最後のステップは成功または失敗を に返し CodePipeline、最後のステップでは継続トークンは返しません。

継続トークンの使用の詳細については、CodePipeline 「 API リファレンス」の「 PutJobSuccessResultの仕様」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 API のいずれかを呼び出し CodePipeline て、カスタムアクションの結果を に返します。 APIs
   - 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:

- CodePipeline を使用した ジョブのポーリングPollForJobs。
- ジョブを承認し、AcknowledgeJob、、PutJobSuccessResultおよび CodePipeline を使用して結果を に返しますPutJobFailureResult。
- 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。には Sig V4 が必要です 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 PutObject が暗号化を使用するよう設定する必要があります。現在、 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。アーティファクトをアップロードするために AWS マネージドキー またはカスタマーマネージドキーのどちらを使用するかを知るには、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チェック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、 を設定するとき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。キープロパ

ティが null の場合、特に設定 AWS マネージドキー されていない限り、 AWS マネージドキー. CodePipeline uses を使用します。

Java または .NET で AWS KMS パラメータを作成する方法を示す例については、「SDK を使用して Amazon S3 AWS Key Management Service で を指定する AWS SDKs」を参照してください。の Amazon S3 バケットの詳細については CodePipeline、「」を参照してくださいCodePipeline の概念。

カスタムジョブワーカーのより複雑な例は、 で入手できます GitHub。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。

- のサンプルジョブワーカー: CodePipeline GitHub リポジトリからサンプルをダウンロードします。

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を使用するか、既存のパイプラインを編集してカスタムアクションを追加するか AWS CLI、、 SDKs、または APIs を使用して、カスタムアクションをパイプラインに追加できます。

> **Note**
>
> ビルドまたはデプロイアクションであれば、パイプラインの作成ウィザードでカスタムアクションを含むパイプラインを作成できます。カスタムアクションがテスト カテゴリーにある場合は、既存のパイプラインを編集して追加する必要があります。

トピック

- 既存のパイプラインにカスタムアクションを追加する (CLI)

既存のパイプラインにカスタムアクションを追加する (CLI)

を使用して AWS CLI 、既存のパイプラインにカスタムアクションを追加できます。

1. ターミナルセッション (Linux、macOSまたはUnix) またはコマンドプロンプト (Windows) を開き、get-pipeline コマンドを実行して、編集するパイプライン構造を JSON ファイルにコピーし

ます。例えば、**MyFirstPipeline** という名前のパイプラインの場合は、以下のコマンドを入力します。

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

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. 任意のテキストエディタで JSON ファイルを開き、ファイルの構造を変更して、カスタムアクションを既存のステージに追加します。

> **Note**
>
> そのステージでアクションを別のアクションと並行して実行する場合は、そのアクションと同じ `runOrder` 値を割り当てます。

例えば、Build という名前のステージを追加し、パイプラインの構造を変更してそのステージにビルドカスタムアクションを追加する場合は次のように JSON を変更して、デプロイステージの前にビルドステージを追加します。

```
,
    {
      "name": "MyBuildStage",
      "actions":  [
              {
                "inputArtifacts": [
                {
                   "name": "MyApp"
                 }
                   ],
                    "name": "MyBuildCustomAction",
                    "actionTypeId": {
                        "category": "Build",
                        "owner": "Custom",
                        "version": "1",
                        "provider": "My-Build-Provider-Name"
                    },
                    "outputArtifacts": [
                        {
                          "name": "MyBuiltApp"
```

```
                        }
                    ],
                    "configuration": {
                        "ProjectName": "MyBuildProject"
                    },
                    "runOrder": 1
                }
            ]
        },
        {
            "name": "Staging",
            "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "MyBuiltApp"
                            }
                        ],
                        "name": "Deploy-CodeDeploy-Application",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "CodePipelineDemoApplication",
                            "DeploymentGroupName": "CodePipelineDemoFleet"
                        },
                        "runOrder": 1
                    }
                ]
             }
    ]
}
```

3. 変更を適用するには、以下のように、パイプライン JSON ファイルを指定して、 update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。

4. CodePipeline コンソールを開き、編集したパイプラインの名前を選択します。

   そのパイプラインには、行った変更が示されます。ソース場所を次に変更すると、修正した構造のパイプラインによりそのリビジョンが実行されます。

でカスタムアクションにタグを付ける CodePipeline

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、のカスタムアクションにタグを適用できます CodePipeline。 CodePipeline リソースのタグ付け、ユースケース、タグのキーと値の制約、サポートされているリソースタイプについては、「」を参照してくださいリソースのタグ付け。

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、「でカスタムアクションを作成して追加する CodePipeline」を参照してください。

以下のステップでは、 AWS CLI の最新版を既にインストールしているか、最新版に更新しているものと想定します。詳細については、「AWS Command Line Interfaceのインストール」を参照してください。

ターミナルまたはコマンドラインで、タグを追加するカスタムアクションの Amazon リソースネーム (ARN)、および追加するタグのキーと値を指定して tag-resource コマンドを実行します。1 つのカスタムアクションに複数のタグを追加できます。例えば、カスタムアクションに 2 つのタグ、タグ値が *TestActionType*の という名前のタグキー*UnitTest*、タグ値が の という名前のタグキー*ApplicationName*をタグ付けするには、次のようにします*MyApplication*。

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-
id:actiontype:Owner/Category/Provider/Version --tags key=TestActionType,value=UnitTest
 key=ApplicationName,value=MyApplication
```

成功した場合、このコマンドは何も返しません。

カスタムアクションのタグを表示する

を使用してカスタムアクションの AWS タグ AWS CLI を表示するには、次の手順に従います。タグが追加されていない場合、返されるリストは空になります。

ターミナルまたはコマンドラインで、list-tags-for-resource コマンドを実行します。たとえば、ARN arn:aws:codepipeline:*us-west-2*:*account-id*:actiontype:*Owner*/*Category*/*Provider*/*Version* を持つカスタムアクションのタグキーとタグ値のリストを表示するには、次のように入力します。

```
aws codepipeline list-tags-for-resource --resource-arn arn:aws:codepipeline:us-
west-2:account-id:actiontype:Owner/Category/Provider/Version
```

成功した場合、このコマンドは次のような情報を返します。

```
{
    "tags": {
        "TestActionType": "UnitTest",
        "ApplicationName": "MyApplication"
    }
}
```

カスタムアクションのタグを編集する

を使用してカスタムアクションのタグ AWS CLI を編集するには、次の手順に従います。既存のキーの値を変更したり、別のキーを追加できます。次のセクションに示すように、カスタムアクションからタグを削除することもできます。

端末またはコマンドラインで、tag-resource コマンドを実行して、タグを更新するカスタムアクションの Amazon リソースネーム (ARN) を指定し、タグキーとタグ値を指定します。

```
aws codepipeline tag-resource --resource-arn arn:aws:codepipeline:us-
west-2:account-id:actiontype:Owner/Category/Provider/Version --tags
 key=TestActionType,value=IntegrationTest
```

カスタムアクションからタグを削除する

を使用してカスタムアクションからタグ AWS CLI を削除するには、次の手順に従います。関連付けられているリソースからタグを削除すると、そのタグが削除されます。

> **Note**
>
> カスタムアクションを削除すると、削除されたカスタムアクションからすべてのタグの関連付けが削除されます。カスタムアクションを削除する前にタグを削除する必要はありません。

ターミナルまたはコマンド行で、タグを削除するカスタムアクションの ARN と削除するタグのタグキーを指定して、untag-resource コマンドを実行します。例えば、タグキー を持つカスタムアクションのタグを削除するには、次のようにします*TestActionType*。

```
aws codepipeline untag-resource --resource-arn arn:aws:codepipeline:us-west-2:account-
id:actiontype:Owner/Category/Provider/Version --tag-keys TestActionType
```

成功した場合、このコマンドは何も返しません。カスタムアクションに関連付けられているタグを確認するには、list-tags-for-resource コマンドを実行します。

のパイプラインで AWS Lambda 関数を呼び出す CodePipeline

AWS Lambda はサーバーのプロビジョニングや管理をする必要がなく、コードを実行できるコンピューティングサービスです。Lambda 関数を作成し、アクションとしてパイプラインに追加できま

す。Lambda は、ほぼ全てのタスクを実行する関数の書き込みができるため、パイプラインの操作方法をカスタマイズできます。

> **Important**
>
> が Lambda CodePipeline に送信する JSON イベントはログに記録しないでください。ログにユーザー認証情報が記録される可能性があるため CloudWatch です。 CodePipeline ロールは JSON イベントを使用して、 `artifactCredentials`フィールドの Lambda に一時的な認証情報を渡します。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 値を入れ替える 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、「 の料金」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 実行ロールと、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アクセス許可を設定します。

- 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**
>
> 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 CodePipeline、PutJobSuccessResultおよび を使用した Lambda 実行のステータスは、Lambda 関数が存在する AWS リージョンに送信PutJobFailureResultされ、 CodePipeline が存在するリージョンには送信されません。

このトピックでは、 で Lambda 関数を使用する柔軟性を示すサンプル関数について説明します CodePipeline。

- Basic Lambda function
  - で使用する基本的な Lambda 関数の作成 CodePipeline。
  - 成功または失敗を返すと、 アクションの詳細リンク CodePipeline で になります。
- AWS CloudFormation テンプレートを使用するサンプル Python 関数
  - 複数の設定値を関数に渡すために JSON でエンコードされたユーザーパラメータを使用する (`get_user_params`)。
  - アーティファクトバケットで、.zip アーティファクトと相互作用する (`get_template`)。
  - 長時間実行の非同期処理を監視する継続トークンを使用する (`continue_job_later`)。これにより、15 分のランタイム (Lambda の制限) を超えてもアクションを続行し、関数を成功させることができます。

各サンプル関数には、ロールに追加する必要がある権限についての情報が含まれます。の制限の詳細については AWS Lambda、「 AWS Lambda デベロッパーガイド」の「 の制限」を参照してください。

> **Important**
>
> このトピックに含まれるサンプルコード、ロール、およびポリシーは単なる例であり、現状のまま提供されます。

トピック

ステップ 1: パイプラインを作成する

このステップでは、後で Lambda 関数を追加するパイプラインを作成します。これは、「CodePipeline チュートリアル」で作成したものと同じパイプラインです。このパイプラインがまだアカウントに設定されていて、Lambda 関数を作成するのと同じリージョンにある場合、このステップは省略できます。

パイプラインを作成するには

1. の最初の 3 つのステップチュートリアル: シンプルなパイプラインを作成する (S3 バケット)に従って、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、このトピックのステップでは を使用します MyLambdaTestPipeline。
2. パイプラインのステータスページで、 CodeDeploy アクションで詳細 を選択します。デプロイグループのデプロイの詳細ページで、リストからインスタンス ID を選択します。
3. Amazon EC2 コンソールのインスタンスの [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**
>
> のデフォルトのサービスロールポリシー CodePipeline には、関数の呼び出しに必要な Lambda アクセス許可が含まれています。ただし、デフォルトサービスロールを変更、または別のものを選択した場合、ロールのポリシーが `lambda:InvokeFunction` および

lambda:ListFunctions 権限を許可していることを確認してください。そうしない場合、Lambda アクションを含むパイプラインは失敗します。

ステップ 2 : Lambda 関数を作成する

このステップでは、HTTP リクエストを生成し、ウェブページ上のテキスト行をチェックする Lambda 関数を作成します。このステップの一環として、IAM ポリシーおよび Lambda 実行ロールを作成する必要があります。詳細については、[AWS Lambda デベロッパーガイド]の [権限モデル」を参照してください。

実行ロールを作成するには

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. [Policies] を選択してから、[Create Policy] を選択します [JSON] タブを選択して、次のポリシーをフィールドに貼り付けます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:*"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Action": [
        "codepipeline:PutJobSuccessResult",
        "codepipeline:PutJobFailureResult"
        ],
        "Effect": "Allow",
        "Resource": "*"
     }
  ]
}
```

3. [ポリシーの確認] を選択します。

4. [ポリシーの確認] ページで、[名前] に、ポリシー名 (**CodePipelineLambdaExecPolicy** など) を入力します。[説明] で **Enables Lambda to execute code** を入力します。

   [ポリシーの作成] を選択します。

   > **Note**
   >
   > これらは、Lambda 関数が CodePipeline および Amazon と相互運用するために必要な最小限のアクセス許可です CloudWatch。このポリシーを拡張して、他の AWS リソースとやり取りする関数を許可する場合は、これらの Lambda 関数に必要なアクションを許可するようにこのポリシーを変更する必要があります。

5. ポリシーダッシュボードページで、[ロール]、[ロールの作成] の順に選択します。
6. [ロールの作成] ページで、[AWS のサービス] を選択します。[Lambda] を選択し、[Next: Permissions (次へ: アクセス許可)] を選択します。
7. アクセス許可ポリシーのアタッチページで、 の横にあるチェックボックスを選択しCodePipelineLambdaExecPolicy、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。

で使用するサンプル Lambda 関数を作成するには CodePipeline

1. にサインイン AWS Management Console し、https://console.aws.amazon.com/lambda/ で AWS Lambda コンソールを開きます。
2. [関数] ページで、[関数の作成] を選択します。

   > **Note**
   >
   > [Lambda] のページの代わりに [Welcome] のページが表示された場合は、[今すぐ始める] を選択します。

3. [関数の作成] ページで、[一から作成] を選択します。[関数の名前] に、Lambda 関数の名前を入力します (例 : **MyLambdaFunctionForAWSCodePipeline**)。ランタイム で、Node.js 20.x を選択します。
4. [Role (ロール)] で、[既存のロールを選択] を選択します。[Existing role (既存のロール)] でロールを選択し、[Create function (関数の作成)] を選択します。

作成した関数の詳細ページが開きます。

5. 次のコードを関数コードボックスに貼り付けます。

> **Note**
>
> CodePipeline.job キーの下にあるイベントオブジェクトには、ジョブの詳細 が含まれます。JSON イベントが Lambda に CodePipeline 返す完全な例については、「」を参照してくださいJSON イベントの例。

```
import { CodePipelineClient, PutJobSuccessResultCommand,
 PutJobFailureResultCommand } from "@aws-sdk/client-codepipeline";
import http from 'http';
import assert from 'assert';

export const handler = (event, context) => {

    const codepipeline = new CodePipelineClient();

    // Retrieve the Job ID from the Lambda action
    const jobId = event["CodePipeline.job"].id;

    // Retrieve the value of UserParameters from the Lambda action configuration in
 CodePipeline, in this case a URL which will be
    // health checked by this function.
    const url =
 event["CodePipeline.job"].data.actionConfiguration.configuration.UserParameters;

    // Notify CodePipeline of a successful job
    const putJobSuccess = async function(message) {
        const command = new PutJobSuccessResultCommand({
            jobId: jobId
        });
         try {
             await codepipeline.send(command);
             context.succeed(message);
         } catch (err) {
             context.fail(err);
         }
    };
```

```
    // Notify CodePipeline of a failed job
    const putJobFailure = async function(message) {
        const command = new PutJobFailureResultCommand({
            jobId: jobId,
            failureDetails: {
                message: JSON.stringify(message),
                type: 'JobFailed',
                externalExecutionId: context.awsRequestId
            }
        });
        await codepipeline.send(command);
        context.fail(message);
    };

    // Validate the URL passed in UserParameters
    if(!url || url.indexOf('http://') === -1) {
        putJobFailure('The UserParameters field must contain a valid URL address to
test, including http:// or https://');
        return;
    }

    // Helper function to make a HTTP GET request to the page.
    // The helper will test the response and succeed or fail the job accordingly
    const getPage = function(url, callback) {
        var pageObject = {
            body: '',
            statusCode: 0,
            contains: function(search) {
                return this.body.indexOf(search) > -1;
            }
        };
        http.get(url, function(response) {
            pageObject.body = '';
            pageObject.statusCode = response.statusCode;

            response.on('data', function (chunk) {
                pageObject.body += chunk;
            });

            response.on('end', function () {
                callback(pageObject);
            });

            response.resume();
```

```
        }).on('error', function(error) {
            // Fail the job if our request failed
            putJobFailure(error);
        });
    };

    getPage(url, function(returnedPage) {
        try {
            // Check if the HTTP response has a 200 status
            assert(returnedPage.statusCode === 200);
            // Check if the page contains the text "Congratulations"
            // You can change this to check for different text, or add other tests
 as required
            assert(returnedPage.contains('Congratulations'));

            // Succeed the job
            putJobSuccess("Tests passed.");
        } catch (ex) {
            // If any of the assertions failed then fail the job
            putJobFailure(ex);
        }
    });
};
```

6. [Handler (ハンドラ)] はデフォルト値のままにし、[Role (ロール)] もデフォルトの **CodePipelineLambdaExecRole** のままにします。
7. 基本設定のタイムアウトに **20** 秒と入力します。
8. [保存] を選択します。

ステップ 3: CodePipeline コンソールで Lambda 関数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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. [Welcome (ようこそ)] ページで、作成したパイプラインを選択します。

3. パイプラインビューページで、[編集] を選択します。
4. 編集ページで、ステージを追加 を選択して、デプロイステージの後に CodeDeploy アクションでステージを追加します。ステージの名前を入力し (たとえば、**LambdaStage**)、[Add stage (ステージの追加)] を選択します。

> **Note**
>
> Lambda アクションを既存のステージに追加することもできます。デモンストレーション用に、ステージでの唯一のアクションとして Lambda 関数を追加し、パイプラインでアーティファクトが進行するにつれてその進行状況を簡単に表示できるようにしています。

5. [+ Add action group (+ アクションの追加)] を選択します。[アクションの編集] の、[アクション名] に、Lambda アクション (例: **MyLambdaAction**) の名前を入力します。[プロバイダ] で、[AWS Lambda] を選択します。[関数名] に Lambda 関数の名前 (例えば、**MyLambdaFunctionForAWSCodePipeline**) を選択または入力します。[ユーザーパラメータ] で、先ほどコピーした Amazon EC2 インスタンスの IP アドレス (例: **http://*192.0.2.4***) を指定し、[完了] を選択します。

> **Note**
>
> このトピックでは、IP アドレスを使用していますが、実際のシナリオでは、代わりに登録済みのウェブサイト名 (**http://*www.example.com*** など) を指定できます。のイベントデータとハンドラーの詳細については AWS Lambda、「 AWS Lambda デベロッパーガイド」の「プログラミングモデル」を参照してください。

6. [アクションの編集] ページで、[保存] を選択します。

ステップ 4 : Lambda 関数でパイプラインをテストする

関数をテストするには、パイプラインを通して最新の変更をリリースします。

コンソールを使用してパイプラインによりアーティファクトの最新バージョンを実行するには

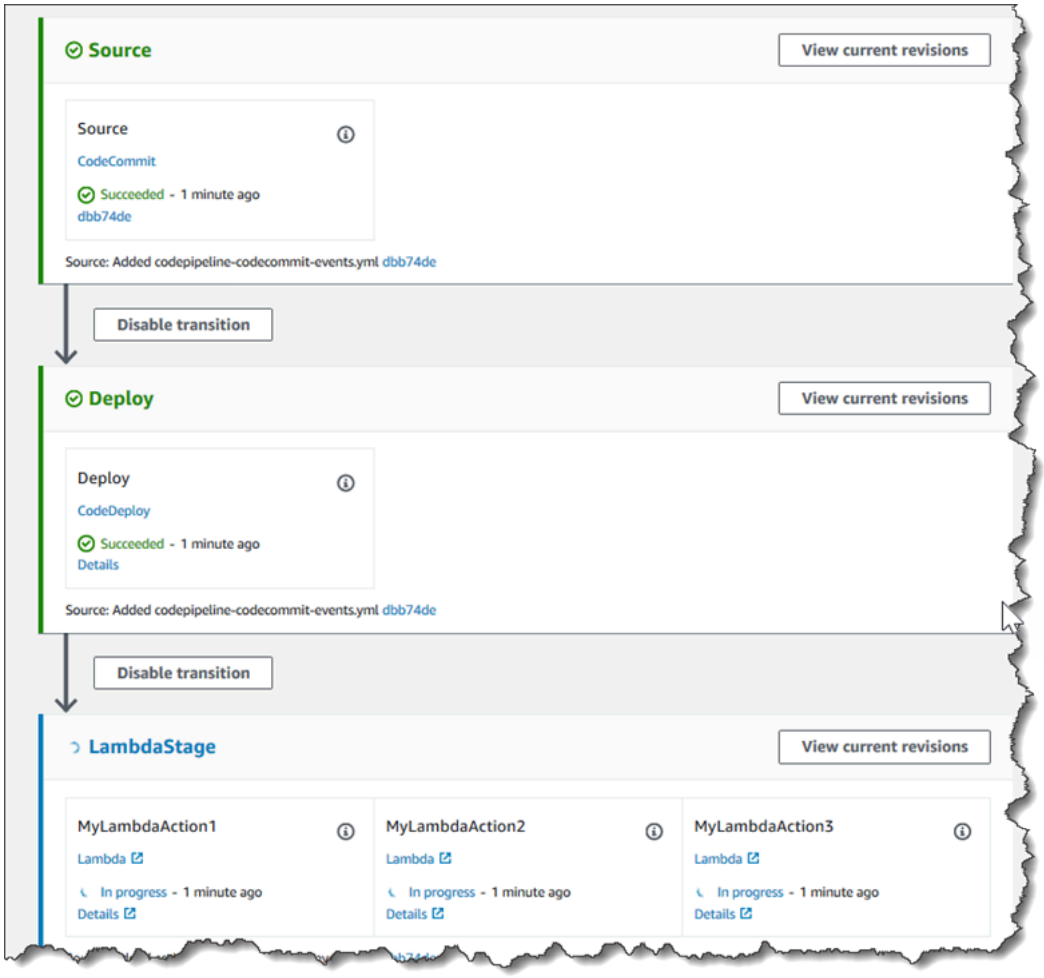
1. パイプラインの詳細ページで、[Release change] を選択します。これにより、ソースアクションで指定した各ソース場所における最新のリビジョンがパイプラインで実行されます。

2. Lambda アクションが完了したら、詳細リンクを選択して CloudWatch、イベントの請求期間を含む Amazon の関数のログストリームを表示します。関数が失敗した場合、 CloudWatch ログは原因に関する情報を提供します。

ステップ 5: 次のステップ

Lambda 関数を作成し、アクションとしてパイプラインに追加したので、次を実行できます。

- さらに Lambda アクションをステージに追加して他のウェブページをチェックします。
- Lambda 関数を変更し、別の文字列をチェックします。
- [Lambda 関数を試し]、パイプラインに独自の Lambda 関数を作成して追加します。



Lambda 関数の実験が完了したら、料金が発生しないように、パイプラインから削除し、 から削除し AWS Lambda、IAM から ロールを削除することを検討してください。詳細については、でパイプ

ラインを編集する CodePipeline、でパイプラインを削除する CodePipeline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

次の例は、 によって Lambda に送信される JSON イベントの例を示しています CodePipeline。このイベントの構造は、GetJobDetails API へのレスポンスと似ていますが、actionTypeId および pipelineContext データタイプがありません。2 つのアクション設定の詳細、FunctionName および UserParameters は、JSON イベントと GetJobDetails API へのレスポンスの両方に含まれます。############ の値は例または説明であり、実際の値ではありません。

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunctionForAWSCodePipeline",
                    "UserParameters": "some-input-such-as-a-URL"
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "the name of the bucket configured as the
 pipeline artifact store in Amazon S3, for example codepipeline-us-east-2-1234567890",
                            "objectKey": "the name of the application, for example
 CodePipelineDemoApplication.zip"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
                "sessionToken": "MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
```

```
WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
3rrszlaEXAMPLE=",
                "accessKeyId": "AKIAIOSFODNN7EXAMPLE"
            },
            "continuationToken": "A continuation token if continuing job",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

追加のサンプル関数

次の Lambda 関数のサンプルは、 のパイプラインに使用できる追加機能を示しています CodePipeline。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 CodePipeline、Lambda 実行ロールのポリシーに AWS CloudFormation、Amazon S3、および `Allow`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名前を付けます*MyTemplate*。詳細については、「でパイプラインを編集する CodePipeline」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、このサンプルには独自の 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."
```

6. [ハンドラー] をデフォルト値のままにし、[ロール] 以前に選択または作成した名前 **CodePipelineLambdaExecRole** のままにします。
7. 基本設定のタイムアウトで、デフォルトの 3 秒を **20** に置き換えます。
8. [保存] を選択します。
9. CodePipeline コンソールから、パイプラインを編集して、パイプラインのステージで 関数を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。ではUserParameters、次の 3 つのパラメータを含む JSON 文字列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、 という名前のスタックを作成するには*MyTestStack*、入力アーティファクト を持つパイプラインに対して*MyTemplate*、 にUserParameters「{"stack""*MyTestStack*file"template-package/template.json"artifact"*MyTemplate*"}」と入力します。

> **Note**
>
> で入力アーティファクトを指定してもUserParameters、入力アーティファクト の アクションにこの入力アーティファクトを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。

ステージ内の失敗したアクションを再試行する

パイプラインを最初から再実行することなく、失敗したステージを再試行できます。そのためには、ステージ内の失敗したアクションを再試行するか、ステージ内の最初のアクションから始めてステージ内のすべてのアクションを再試行します。ステージ内の失敗したアクションを再試行する場合、進行中のすべてのアクションはそのまま動作し続け、失敗したアクションは再度トリガーされます。失敗したアクションのあるステージ内で最初のアクションから再試行する場合、ステージに進行中のアクションがないことが必要です。ステージを再試行するには、すべてのアクションが失敗しているか、一部のアクションが失敗して他のアクションが成功している必要があります。

> **Important**
>
> 失敗したステージを再試行すると、そのステージ内の最初のアクションからすべてのアクションが再試行されます。失敗したアクションを再試行すると、ステージ内のすべての失敗したアクションが再試行されます。これにより、同じ実行内で以前に成功したアクションの出力アーティファクトは上書きされます。
> アーティファクトが上書きされても、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、アクションが失敗したかどうかを判断できます。

> **Note**
>
> 次の場合は、ステージを再試行できないことがあります。
>
> - ステージ内のすべてのアクションが成功したため、ステージが失敗ステータスになっていない。
> - ステージが失敗した後、パイプライン全体の構造が変更された。
> - ステージで別の再試行がすでに進行中です。

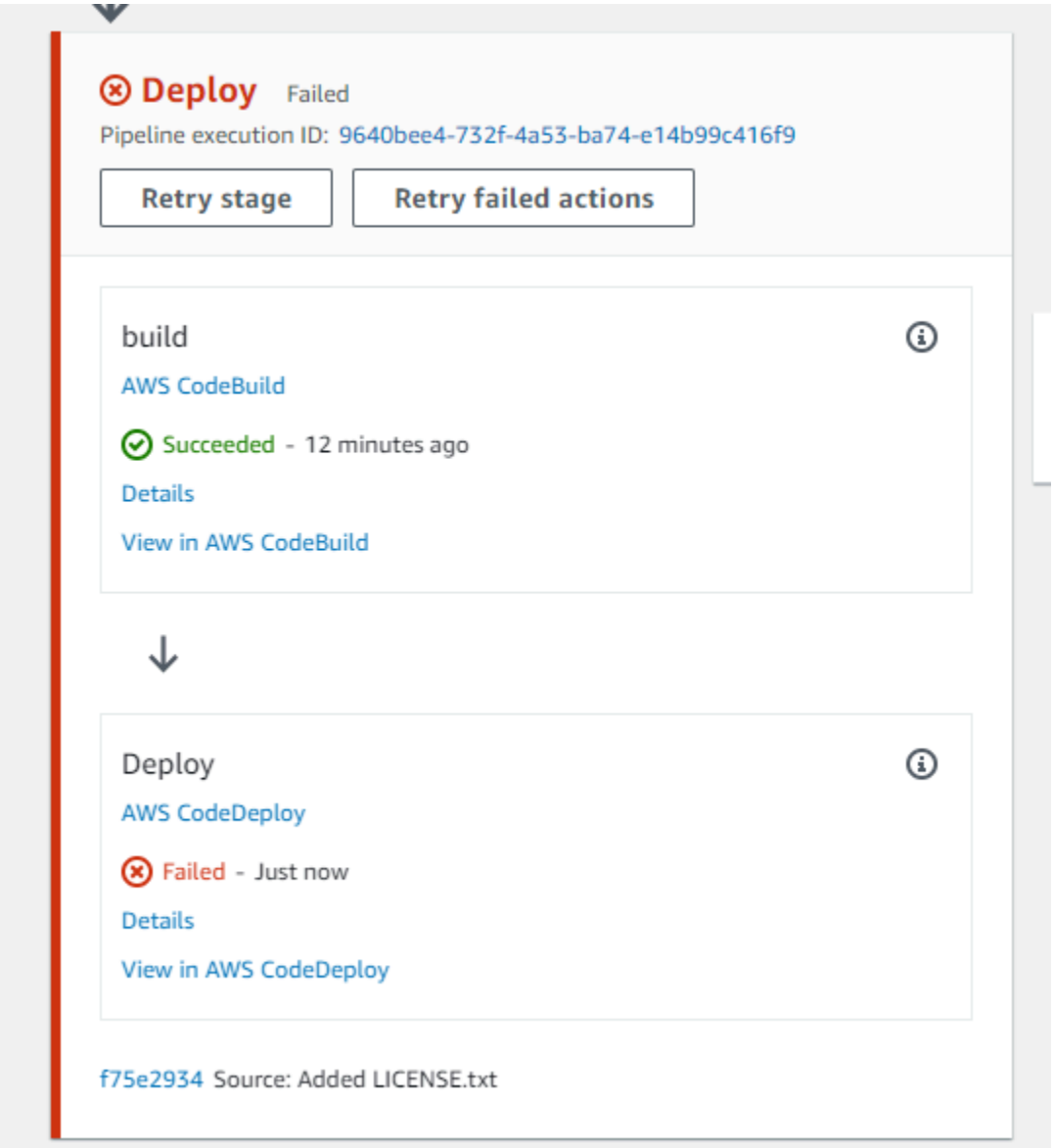
トピック

失敗したアクションを再試行する (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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、パイプラインの名前を選択します。
3. 失敗したアクションのあるステージを見つけ、次のいずれかを選択します。
   - ステージ内のすべてのアクションを再試行するには、[ステージの再試行] を選択します。
   - ステージ内の失敗したアクションのみを再試行するには、[失敗したアクションの再試行] を選択します。



再試行したアクションがすべて正常に完了した場合、パイプラインは続行されます。

失敗したアクションを再試行する (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を AWS CLI 再試行するには、次のパラメータを指定して retry-stage-execution コマンドを実行します。

```
--pipeline-name <value>
```

```
--stage-name <value>
--pipeline-execution-id <value>
--retry-mode ALL_ACTIONS/FAILED_ACTIONS
```

> **Note**
>
> retry-mode に使用できる値は FAILED_ACTIONS と ALL_ACTIONS です。

1. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、retry-stage-execution コマンドを実行します。次の例では、MyPipeline という名前のパイプラインに対して実行しています。

   ```
   aws codepipeline retry-stage-execution --pipeline-name MyPipeline --stage-name
    Deploy --pipeline-execution-id b59babff-5f34-EXAMPLE --retry-mode FAILED_ACTIONS
   ```

   出力は実行 ID を返します。

   ```
   {
       "pipelineExecutionId": "b59babff-5f34-EXAMPLE"
   }
   ```

2. JSON 入力ファイルを使用してコマンドを実行することもできます。まず、パイプライン、失敗したアクションを含むステージ、そのステージでの最新のパイプラインの実行を識別する JSON ファイルを作成します。その後、retry-stage-execution コマンドに --cli-input-json パラメータを指定して実行します。JSON ファイルに必要な詳細を取得するには、get-pipeline-state コマンドを使用するのが最も簡単です。

   a. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、パイプラインの get-pipeline-state コマンドを実行します。例えば、 という名前のパイプラインの場合 MyFirstPipeline、次のようなものを入力します。

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

      コマンドに対する応答には、各ステージのパイプラインの状態情報が含まれます。以下の例では、応答は [Staging] ステージで 1 つ以上のアクションが失敗したことを示しています。

      ```
      {
          "updated": 1427245911.525,
      ```

```
    "created": 1427245911.525,
    "pipelineVersion": 1,
    "pipelineName": "MyFirstPipeline",
    "stageStates": [
        {
            "actionStates": [...],
            "stageName": "Source",
            "latestExecution": {
                "pipelineExecutionId": "9811f7cb-7cf7-SUCCESS",
                "status": "Succeeded"
            }
        },
        {
            "actionStates": [...],
            "stageName": "Staging",
            "latestExecution": {
                "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
                "status": "Failed"
            }
        }
    ]
}
```

b. プレーンテキストエディタで、JSON 形式で以下の情報を記録するファイルを作成します。

- 失敗したアクションを含むパイプラインの名前
- 失敗したアクションを含むステージの名前
- ステージでの最新のパイプラインの実行 ID
- 再試行モード

前の MyFirstPipeline 例では、ファイルは次のようになります。

```
{
    "pipelineName": "MyFirstPipeline",
    "stageName": "Staging",
    "pipelineExecutionId": "3137f7cb-7cf7-EXAMPLE",
    "retryMode": "FAILED_ACTIONS"
}
```

c. **retry-failed-actions.json** のような名前でファイルを保存します。

d. retry-stage-execution コマンドを実行したときに作成したファイルを呼び出します。例:

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

```
aws codepipeline retry-stage-execution --cli-input-json file://retry-failed-actions.json
```

e. 再試行の結果を表示するには、 CodePipeline コンソールを開き、失敗したアクションを含むパイプラインを選択するか、 get-pipeline-state コマンドを再度使用します。詳細については、「でパイプラインと詳細を表示する CodePipeline」を参照してください。

での承認アクションの管理 CodePipeline

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

での手動承認アクションの設定オプション CodePipeline

CodePipeline には、承認アクションを承認者に伝えるために使用できる 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、「の手動承認通知の JSON データ形式 CodePipeline」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。

での承認アクションのセットアップとワークフローの概要 CodePipeline

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は、 CodePipeline オペレーションで使用するサービスロールに Amazon SNS リソースへのアクセス許可があることを確認します。
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、「」のARNsと AWS のサービス 名前空間」を参照してくださいAmazon Web Services 全般のリファレンス。)
5. 承認アクションに達すると、パイプラインは停止します。Amazon SNS トピックの ARN がアクション設定に含まれている場合、通知は Amazon SNS トピックに発行され、メッセージは、コンソールの承認アクションをレビューするリンクと合わせて、トピックの受信者またはサブスクライブされたエンドポイントに送信されます。
6. 承認者は、必要に応じて、ターゲット URL およびコメントを確認します。
7. 承認者は、コンソール、CLI、SDK を使用して、概要コメントを確認し、次のようなレスポンスを送信します。
   - 承認済み: パイプラインの実行が再開されます。
   - 拒否: ステージステータスが「失敗」に変更され、パイプラインの実行は再開されません。

   7 日以内にレスポンスが送信されない場合、アクションは「失敗」とマークされます。

で IAM ユーザーに承認許可を付与する CodePipeline

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。で使用できる マネージドポリシーの詳細については CodePipeline、「」を参照してくださいAWS の マネージドポリシー AWS CodePipeline。

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

で承認アクションを実行する必要があるユーザーには CodePipeline、`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。

- 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「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」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する。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。

で承認アクションを実行する必要があるユーザーには CodePipeline、次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**
>
> アクセスcodepipeline:ListPipelines許可は、IAM ユーザーがこのパイプラインのリストを表示するために CodePipeline ダッシュボードにアクセスする必要がある場合にのみ必要です。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、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [ポリシーエディタ] セクションで、[JSON] オプションを選択します。
5. 次の JSON ポリシードキュメントを入力します。

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Effect": "Allow",
           "Action": [
               "codepipeline:ListPipelines"
           ],
           "Resource": [
               "*"
           ]
       },
       {
           "Effect": "Allow",
           "Action": [
               "codepipeline:GetPipeline",
               "codepipeline:GetPipelineState",
               "codepipeline:GetPipelineExecution"
           ],
```

```
            "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline"
        },
        {
            "Effect": "Allow",
            "Action": [
                "codepipeline:PutApprovalResult"
            ],
            "Resource": "arn:aws:codepipeline:us-
east-2:80398EXAMPLE:MyFirstPipeline/MyApprovalStage/MyApprovalAction"
        }
    ]
}
```

6. [次へ] をクリックします。

> **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] 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。

サービスロールに Amazon SNS アクセス許可を付与する CodePipeline

承認アクションでレビューが必要なときに Amazon SNS を使用してトピックに通知を発行する場合は、 CodePipeline オペレーションで使用するサービスロールに Amazon SNS リソースへのアクセス許可を付与する必要があります。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の開始方法 CodePipeline。

```
{
    "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、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. 左側のナビゲーションペインで、[ポリシー] を選択します。

   初めて [ポリシー] を選択する場合には、[管理ポリシーにようこそ] ページが表示されます。[今すぐ始める] を選択します。
3. ページの上部で、[ポリシーを作成] を選択します。
4. [ポリシーエディタ] セクションで、[JSON] オプションを選択します。
5. JSON ポリシードキュメントを入力するか貼り付けます。IAM ポリシー言語の詳細については、「IAM JSON ポリシーリファレンス」を参照してください。
6. ポリシーの検証中に生成されたセキュリティ警告、エラー、または一般的な警告を解決してから、[Next] (次へ) を選択します。

> **Note**
>
> いつでも [Visual] と [JSON] エディタオプションを切り替えることができます。ただし、[Visual] エディタで [次] 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、テンプレートの生成 CloudFormation を選択します。の詳細については AWS CloudFormation、「 ユーザーガイド」の「 AWS Identity and Access Management リソースタイプのリファレンスAWS CloudFormation 」を参照してください。
8. ポリシーにアクセス権限を追加し終えたら、[次へ] を選択します。
9. [確認と作成] ページで、作成するポリシーの [ポリシー名] と [説明] (オプション) を入力します。[このポリシーで定義されているアクセス許可] を確認して、ポリシーによって付与されたアクセス許可を確認します。
10. (オプション) タグをキー - 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 でのタグの使用に関する詳細については、『IAM ユーザーガイド』の「IAM リソースにタグを付ける」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。

でパイプラインに手動承認アクションを追加する CodePipeline

CodePipeline パイプラインのステージに承認アクションを追加し、パイプラインを停止して、誰かが手動でアクションを承認または拒否できるようにします。

> **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するためのアクセス許可を CodePipeline サービスロールに付与します。詳細については、「サービスロールに Amazon SNS アクセス許可を付与する CodePipeline 」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、「で IAM ユーザーに承認許可を付与する CodePipeline」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。

手動承認アクションを CodePipeline パイプラインに追加する (コンソール)

CodePipeline コンソールを使用して、既存の CodePipelineパイプラインに承認アクションを追加できます。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. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
2. [Name] で、パイプラインを選択します。
3. パイプライン詳細ページで、[編集] を選択します。
4. 承認アクションを新しいステージに追加する場合は、承認リクエストを追加するパイプラインのポイントで [+Add stage (+ ステージの追加)] を選択し、ステージの名前を入力します。[ステージの追加 (Add stage)] ページの [Stage name (ステージ名)] に、新しいステージ名を入力します。たとえば、新しいステージを追加し、`Manual_Approval` という名前を付けます。

   承認アクションを既存のステージに追加する場合は、[ステージの編集] を選択します。
5. 承認アクションを追加するステージで、[+ Add action group (アクショングループの追加)] を選択します。
6. [アクションの編集] ページで、次の作業を行います。
   1. [アクション名] に、アクションを識別する名前を入力します。
   2. [アクションプロバイダー] の、[承認] で、[手動承認] を選択します。
   3. (オプション) [SNS トピックの ARN] で、承認アクションの通知を送るために使用するトピックの名前を選択します。
   4. (オプション) [URL for review] に、承認者が検討するページまたはアプリケーションの URL を入力します。承認者は、パイプラインのコンソールビューに含まれるリンクからこの URL にアクセスできます。
   5. (オプション) [コメント] に、レビュー者と共有するその他の情報を入力します。
   6. [保存] を選択します。

手動承認アクションをパイプラインに追加する CodePipeline (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、「でパイプラインを作成する CodePipeline」および「でパイプラインを編集する CodePipeline」を参照してください。

承認アクションを含むパイプラインにステージを追加するには、パイプライン作成時または更新時に、次の例のように追加します。

> **Note**
>
> configuration セクションはオプションです。このセクションはファイルの一部であり、構造全体を示したものではありません。詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

```
{
    "name": "MyApprovalStage",
    "actions": [
        {
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "inputArtifacts": [],
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."},
            "runOrder": 1
        }
    ]
}
```

承認アクションが他のアクションを含むステージに存在する場合、このステージを含む JSON ファイルのセクションは、次の例のようになります。

> **Note**
>
> configuration セクションはオプションです。このセクションはファイルの一部であり、構造全体を示したものではありません。詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

```
,
{
    "name": "Production",
    "actions": [
        {
            "inputArtifacts": [],
            "name": "MyApprovalAction",
            "actionTypeId": {
                "category": "Approval",
                "owner": "AWS",
                "version": "1",
                "provider": "Manual"
            },
            "outputArtifacts": [],
            "configuration": {
                "NotificationArn": "arn:aws:sns:us-
east-2:80398EXAMPLE:MyApprovalTopic",
                "ExternalEntityLink": "http://example.com",
                "CustomData": "The latest changes include feedback from Bob."
            },
            "runOrder": 1
        },
        {
            "inputArtifacts": [
                {
                    "name": "MyApp"
                }
            ],
            "name": "MyDeploymentAction",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
```

```
                "version": "1",
                "provider": "CodeDeploy"
            },
            "outputArtifacts": [],
            "configuration": {
                "ApplicationName": "MyDemoApplication",
                "DeploymentGroupName": "MyProductionFleet"
            },
            "runOrder": 2
        }
    ]
}
```

での承認アクションを承認または拒否する CodePipeline

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。

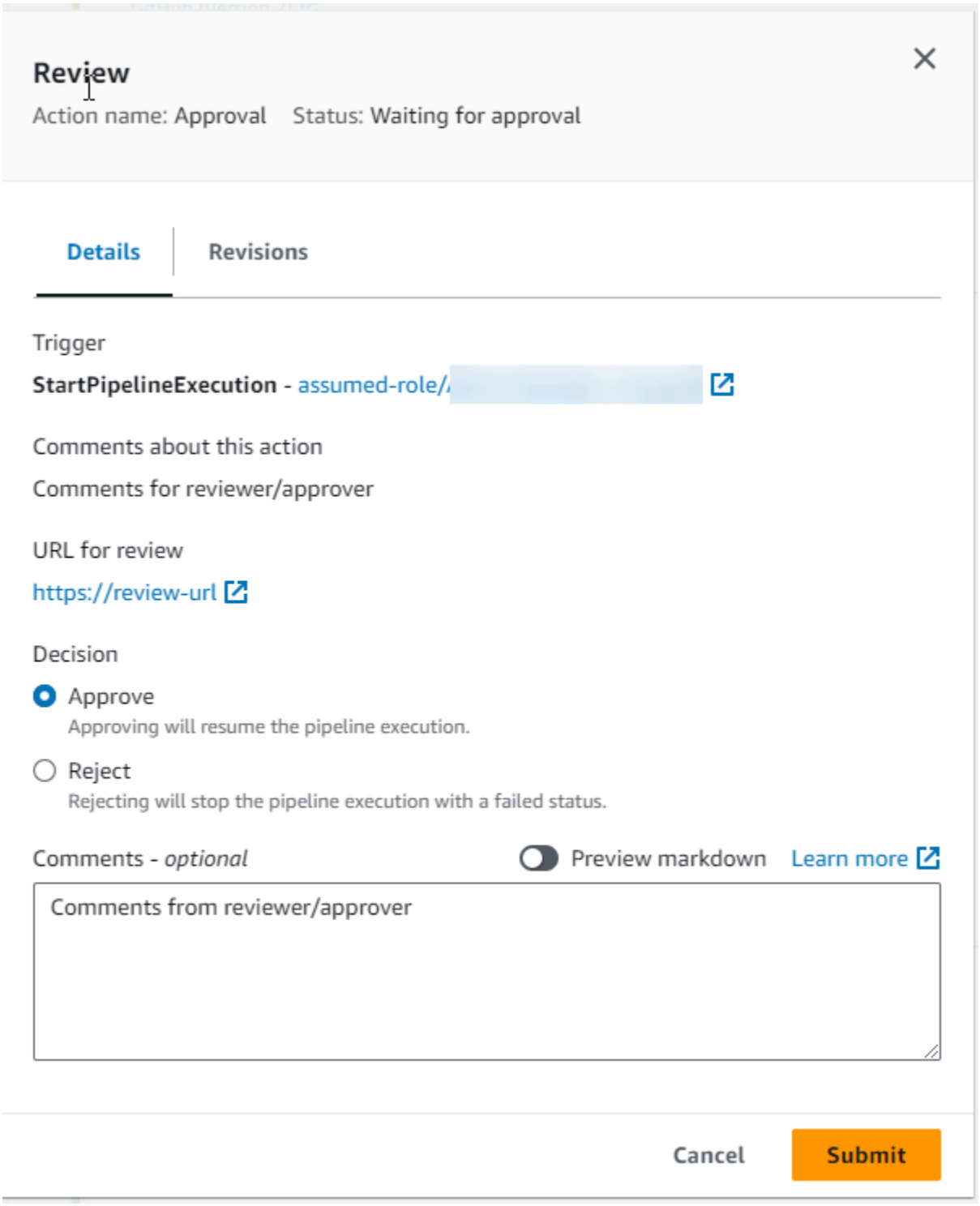
パイプラインに承認アクションを追加した人が通知を設定していると、パイプラインの情報と承認のステータスを含む E メールを受け取る場合があります。

承認アクションを承認または拒否する (コンソール)

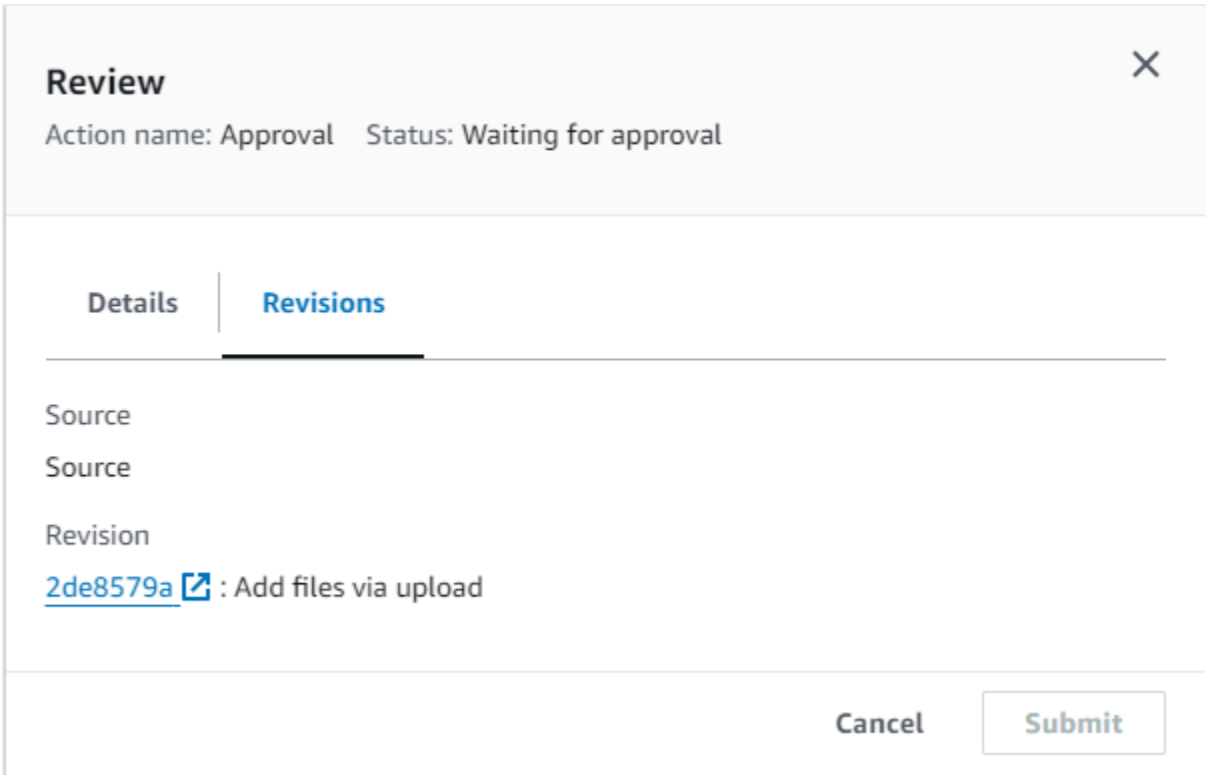
承認アクションへの直接リンクを含む通知を受け取った場合は、[Approve or reject (承認または却下)] リンクを選択し、コンソールにサインインし、このステップ 7 に進みます。そうでない場合は、以下のすべての手順を実行します。

1. https://console.aws.amazon.com/codepipeline/ で CodePipeline コンソールを開きます。
2. [All Pipelines (すべてのパイプライン)] ページで、パイプラインの名前を選択します。
3. 承認アクションが含まれるステージを見つけます。[Review] (レビュー) を選択します。

   [レビュー] ダイアログボックスが表示されます。[詳細] タブには、レビューの内容とコメントが表示されます。



[リビジョン] タブには、実行のソースリビジョンが表示されます。



4. [詳細] タブには、コメントと URL (ある場合) が表示されます。メッセージに、レビューするコンテンツの URL があれば、それも表示されます。
5. URL が表示された場合は、アクションの [レビューの URL] リンクを選択して、ターゲットウェブページを開き、コンテンツをレビューします。
6. [レビュー] ウィンドウで、アクションを承認または拒否した理由など、レビューコメントを入力し、[承認] または [拒否] を選択します。
7. [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。

1. ターミナル (Linux、macOS または Unix) またはコマンドプロンプト (Windows) で、承認アクションを含むパイプラインで get-pipeline-state コマンドを実行します。例えば、 という名前のパイプラインの場合は*MyFirstPipeline*、次のように入力します。

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

2. コマンドに対する応答で、次に示すような承認アクションの actionStates セクションの latestExecution にある token 値を見つけます。

```
{
    "created": 1467929497.204,
    "pipelineName": "MyFirstPipeline",
    "pipelineVersion": 1,
    "stageStates": [
        {
            "actionStates": [
                {
                    "actionName": "MyApprovalAction",
                    "currentRevision": {
                        "created": 1467929497.204,
                        "revisionChangeId": "CEM7d6Tp7zfelUSLCPPwo234xEXAMPLE",
                        "revisionId": "HYGp7zmwbCPPwo23xCMdTeqIlEXAMPLE"
                    },
                    "latestExecution": {
                        "lastUpdatedBy": "identity",
                        "summary": "The new design needs to be reviewed before
 release.",
                        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN"
                    }
                }
//More content might appear here
}
```

3. プレーンテキストエディタで、JSON 形式で以下の情報を記録するファイルを追加します。

- 承認アクションを含むパイプラインの名前。
- 承認アクションを含むステージの名前。
- 承認アクションの名前。
- 前の手順で収集したトークン値。
- アクションに対する応答 (承認済みまたは却下)。応答の先頭は大文字であることが必要です。
- 概要コメント。

前の*MyFirstPipeline*例では、ファイルは次のようになります。

```
{
  "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. コマンドを実行しput-approval-result、次のような承認 JSON ファイルの名前を指定します。

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

```
aws codepipeline put-approval-result --cli-input-json file://approvalstage-approved.json
```

の手動承認通知の JSON データ形式 CodePipeline

Amazon SNS 通知を使用する承認アクションの場合、アクションに関する JSON データは、パイプライン停止時に Amazon SNS に対して作成し、発行されます。この JSON 出力を使用して、メッセージを Amazon SQS のキューに送信するか、 AWS Lambdaの関数を呼び出します。

> **Note**
>
> このガイドでは、JSON を使用して通知を設定する方法については説明していません。詳細については、[Amazon SNS デベロッパーガイド] の [Amazon SQS キューへの Amazon SNS メッセージの送信] と [Amazon SNS 通知を使った Lambda 関数の呼び出し] を参照してください。

次の例は、承認で使用できる CodePipeline JSON 出力の構造を示しています。

```
{
    "region": "us-east-2",
```

```
    "consoleLink": "https://console.aws.amazon.com/codepipeline/home?region=us-east-2#/
view/MyFirstPipeline",
    "approval": {
        "pipelineName": "MyFirstPipeline",
        "stageName": "MyApprovalStage",
        "actionName": "MyApprovalAction",
        "token": "1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
        "expires": "2016-07-07T20:22Z",
        "externalEntityLink": "http://example.com",
        "approvalReviewLink": "https://console.aws.amazon.com/codepipeline/
home?region=us-east-2#/view/MyFirstPipeline/MyApprovalStage/MyApprovalAction/
approve/1a2b3c4d-573f-4ea7-a67E-XAMPLETOKEN",
        "customData": "Review the latest changes and approve or reject within seven
 days."
    }
}
```

でクロスリージョンアクションを追加する CodePipeline

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多数のアクションが含まれています。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。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 CodePipeline、「」を参照してくださいのクォータ AWS CodePipeline。

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。

> **Note**
>
> の特定のアクションタイプは、特定の AWS リージョン CodePipeline 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、「CodePipeline パイプライン構造リファレンス」をご参照ください。

> **Note**
>
> CodePipeline は、クロス AWS リージョンアクションを実行するときに、あるリージョンから他のリージョンへのアーティファクトのコピーを処理します。

コンソールを使用してパイプラインまたはクロスリージョンアクションを作成する場合、デフォルトのアーティファクトバケットは、アクションがあるリージョン CodePipeline で 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、アクションがある各リージョンの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**
>
> 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 CodePipeline、PutJobSuccessResultおよび を使用した Lambda 実行のステータスは、Lambda 関数が存在する AWS リージョンではなく、 CodePipeline が存在するリージョンに送信PutJobFailureResultする必要があります。

パイプラインにステージの一部としてクロスリージョンアクションが含まれている場合、 はクロスリージョンアクションの入力アーティファクトのみをパイプラインリージョンからアクションのリージョンに CodePipeline レプリケートします。

> **Note**
>
> パイプラインリージョンと、 CloudWatch イベント変更検出リソースが維持されているリージョンは変わりません。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、「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 はアクションのリージョンで別のアーティファクトバケットを使用します。クロスリージョンのアーティファクトバケットの詳細については、「CodePipeline パイプライン構造リファレンス」を参照してください。

パイプラインステージにクロスリージョンアクションを追加する (コンソール)

コンソールを使用してパイプラインにクロスリージョンアクションを追加します。

> **Note**
>
> 変更を保存する際にパイプラインが実行中の場合、その実行は完了しません。

クロスリージョンアクションを追加するには

1. コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. パイプラインを選択し、[編集] を選択します。
3. 図の下部で、新しいステージを追加する場合は [+ Add stage (+ ステージの追加)] を選択します。既存のステージにアクションを追加する場合は、[Edit stage (ステージの編集)] を選択します。
4. [Edit: <Stage> (編集: <ステージ>)] で、シリアルアクションを追加する場合は [+ Add action group (+ アクショングループの追加)] を選択します。または、パラレルアクションを追加する場合は、[+Add action (+アクションの追加)] を追加します。
5. [アクションの編集] ページで、以下の操作を行います。
   a. [アクション名] に、クロスリージョンアクションの名前を入力します。
   b. [Action provider (アクションプロバイダー)] で、アクションプロバイダーを選択します。
   c. リージョン で、アクションのリソースを作成または作成する予定の AWS リージョンを選択します。リージョンを選択すると、このリージョンで使用できるリソースが一覧表示されて選択できるようになります。リージョンフィールドは、このアクションタイプとプロバイダータイプに対して AWS リソースが作成される場所を指定します。このフィールドには、アクションプロバイダーが AWS のサービスであるアクションのみが表示されます。[リージョン] フィールドは、デフォルトで、パイプラインと同じ AWS リージョン になります。
   d. [入力アーティファクト] で、前のステージからの適切な入力を選択します。例えば、前のステージがソースステージの場合は、 を選択しますSourceArtifact。
   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。例えば、次のステージがデプロイステージの場合は、 を選択しますBuildArtifact。
   g. [保存] を選択します。
6. [Edit: <Stage> (編集: <ステージ>)] で、完了] を選択します。
7. [保存] を選択します。

パイプラインステージ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 (https://console.aws.amazon.com/codesuite/codepipeline/home.) にサインインします。
2. パイプラインを選択し、[編集] を選択します。
3. [Edit stage (ステージの編集)] を選択します。
4. [Edit: <Stage> (編集: <ステージ>)] で、既存のアクションを編集するためのアイコンを選択します。
5. [アクションの編集] ページで、必要に応じてフィールドを変更します。
6. [Edit: <Stage> (編集: <ステージ>)] で、完了] を選択します。
7. [保存] を選択します。

パイプラインステージからクロスリージョンアクションを削除する (コンソール)

コンソールを使用してパイプラインから既存のクロスリージョンアクションを削除します。

> **Note**
> 変更を保存する際にパイプラインが実行中の場合、その実行は完了しません。

クロスリージョンアクションを削除するには

1. コンソール (http://console.aws.amazon.com/codesuite/codepipeline/home) にサインインします。
2. パイプラインを選択し、[編集] を選択します。
3. [Edit stage (ステージの編集)] を選択します。
4. [Edit: <Stage> (編集: <ステージ>)] で、既存のアクションを削除するためのアイコンを選択します。
5. [Edit: <Stage> (編集: <ステージ>)] で、完了] を選択します。
6. [保存] を選択します。

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。単一リージョンパイプラインの `artifactStore` パラメータを指定する代わりに、`artifactStores` パラメータを使用して各リージョンのアーティファクトバケットのリストを含めます。

> **Note**
>
> このチュートリアルおよびその例では、*##### A* がパイプラインの作成先のリージョンです。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*RegionA* Amazon S3 バケットと、 で使用されるサービスロールにアクセスできます CodePipeline。*RegionB* は、で使用される CodeDeploy アプリケーション、デプロイグループ、およびサービスロールが作成されるリージョン CodeDeploy です。

前提条件

以下を作成しておく必要があります。

- *##### A* のパイプライン。
- *##### B* の Amazon S3 アーティファクトバケット
- リージョン *RegionB* のクロスリージョンデプロイアクションの CodeDeploy アプリケーションやデプロイグループなど、アクションのリソース。

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

1. *##### A* のパイプラインで、get-pipeline コマンドを実行し、パイプライン構造を JSON ファイルにコピーします。例えば、`MyFirstPipeline` という名前のパイプラインに対して、以下のコマンドを実行します。

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

このコマンドは何も返しませんが、作成したファイルは、コマンドを実行したディレクトリにあります。

2. region フィールドを追加して、アクションのリージョンとリソースを含むクロスリージョンアクションを関連付けた新しいステージを追加します。次の JSON サンプルは、プロバイダーが新しいリージョン で CodeDeployであるクロスリージョンデプロイアクションでデプロイステージを追加しますus-east-1。

```
{
                "name": "Deploy",
                "actions": [
                    {
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "name": "Deploy",
                        "region": "RegionB",
                        "actionTypeId": {
                            "category": "Deploy",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeDeploy"
                        },
                        "outputArtifacts": [],
                        "configuration": {
                            "ApplicationName": "name",
                            "DeploymentGroupName": "name"
                        },
                        "runOrder": 1
                    }
```

3. パイプライン構造で、artifactStore フィールドを削除し、新しいクロスリージョンアクションの artifactStores 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、`my-storage-bucket-us-east-1` という名前の新しい us-east-1 バケットを追加します。

```
        "artifactStores": {
            "us-west-2": {
                "type": "S3",
                "location": "my-storage-bucket"
            },
            "us-east-1": {
                "type": "S3",
                "location": "my-storage-bucket-us-east-1"
            }
        },
```

4. get-pipeline コマンドを使用して取得したパイプライン構造を使用している場合、JSON ファイルから `metadata` 行を削除します。それ以外の場合は、update-pipeline コマンドで使用することはできません。`"metadata": { }` 行と、`"created"`、`"pipelineARN"`、`"updated"` フィールドを削除します。

   例えば、構造から以下の行を削除します。

```
"metadata": {
```

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

ファイルを保存します。

5. 変更を適用するには、パイプライン JSON ファイルを指定して、update-pipeline コマンドを実行します。

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

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

このコマンドは、編集したパイプラインの構造全体を返します。出力は以下のようになります。

```
{
    "pipeline": {
        "version": 4,
        "roleArn": "ARN",
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "inputArtifacts": [],
                        "name": "Source",
                        "actionTypeId": {
                            "category": "Source",
                            "owner": "AWS",
                            "version": "1",
                            "provider": "CodeCommit"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "configuration": {
```

```
                        "PollForSourceChanges": "false",
                        "BranchName": "main",
                        "RepositoryName": "MyTestRepo"
                    },
                    "runOrder": 1
                }
            ]
        },
        {
            "name": "Deploy",
            "actions": [
                {
                    "inputArtifacts": [
                        {
                            "name": "SourceArtifact"
                        }
                    ],
                    "name": "Deploy",
                    "region": "us-east-1",
                    "actionTypeId": {
                        "category": "Deploy",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeDeploy"
                    },
                    "outputArtifacts": [],
                    "configuration": {
                        "ApplicationName": "name",
                        "DeploymentGroupName": "name"
                    },
                    "runOrder": 1
                }
            ]
        }
    ],
    "name": "AnyCompanyPipeline",
    "artifactStores": {
        "us-west-2": {
            "type": "S3",
            "location": "my-storage-bucket"
        },
        "us-east-1": {
            "type": "S3",
            "location": "my-storage-bucket-us-east-1"
```

```
            }
        }
    }
}
```

> **Note**
>
> update-pipeline コマンドは、パイプラインを停止します。update-pipeline コマンドを実行したときにパイプラインによりリビジョンが実行されている場合、その実行は停止します。更新されたパイプラインによりそのリビジョンを実行するには、パイプラインを手動で開始する必要があります。パイプラインを手動で開始するには **start-pipeline-execution** コマンドを使用します。

6. パイプラインを更新したら、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。

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)

AWS CloudFormation を使用して、既存のパイプラインにクロスリージョンアクションを追加できます。

でクロスリージョンアクションを追加するには AWS CloudFormation

1. この例に示すように、`Region` パラメータをテンプレートの `ActionDeclaration` リソースに追加します。

```
ActionDeclaration:
              Type: Object
              Properties:
                ActionTypeId:
                  Type: ActionTypeId
                  Required: true
                Configuration:
                  Type: Map
                InputArtifacts:
                  Type: Array
                  ItemType:
                    Type: InputArtifact
                Name:
                  Type: String
                  Required: true
                OutputArtifacts:
                  Type: Array
                  ItemType:
                    Type: OutputArtifact
                RoleArn:
                  Type: String
                RunOrder:
                  Type: Integer
                Region:
                  Type: String
```

2. Mappings で、この例で示しているように、キー SecondRegionMap および RegionA の値をマップする RegionB という名前のマッピング用のリージョンマップを追加します。Pipeline リソースの artifactStore フィールドで、新しいクロスリージョンアクションの artifactStores マップを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 eu-central-1 とします。

```
Mappings:
  SecondRegionMap:
    us-west-2:
      SecondRegion: "eu-central-1"
    eu-central-1:
      SecondRegion: "us-west-2"

...

          Properties:
            ArtifactStores:
              -
                Region: eu-central-1
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-eu-central-1
              -
                Region: us-west-2
                ArtifactStore:
                  Type: "S3"
                  Location: test-cross-region-artifact-store-bucket-us-west-2
```

3. 更新したテンプレートをローカルコンピュータに保存し、 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] を選択します。

変数の操作

の一部のアクションは変数 CodePipeline を生成します。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。

変数の使用 step-by-step 例を表示するには：

- 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、「」を参照してくださいチュートリアル: Lambda 呼び出しアクションで変数を使用する。
- アップストリーム AWS CloudFormation アクションのスタック出力変数を参照する CloudFormation アクションのチュートリアルについては、「」を参照してくださいチュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。
- CodeCommit コミット ID とコミットメッセージに解決される出力変数を参照するメッセージテキストを含む手動承認アクションの例については、「」を参照してください例: 手動承認で変数を使用する。
- GitHub ブランチ名に解決される環境変数を使用した CodeBuild アクションの例については、「」を参照してください例: CodeBuild 環境変数で変数を使用する BranchName 。
- 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 アクション出力変数」を参照してください。で使用できる環境変数のリストについては CodeBuild、「 ユーザーガイド」の「ビルド環境の環境変数AWS CodeBuild 」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 環境変数で変数を使用する BranchName

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使用して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使用するように設定されています。

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

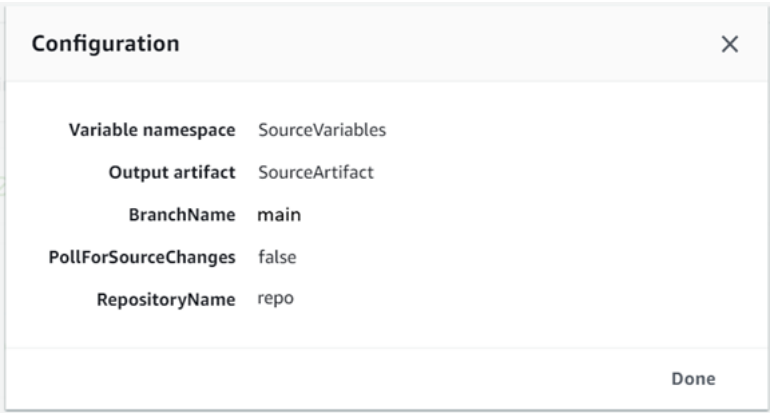
1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. パイプラインの作成 を選択します。パイプラインに名前を入力し、[Next (次へ)] を選択します。
3. ソース で、プロバイダー で、 を選択しますCodeCommit。ソースアクションの CodeCommit リポジトリとブランチを選択し、次へ を選択します。
4. ビルド で、プロバイダー で を選択しますCodeBuild。既存の CodeBuild ビルドプロジェクト名を選択するか、プロジェクトの作成 を選択します。ビルドプロジェクトの作成 でビルドプロジェクトを作成し、 に戻る CodePipelineを選択します。

   [環境変数] の下で、[Add environment variables (環境変数を追加)] を選択します。例えば、実行 ID を変数構文 `#{codepipeline.PipelineExecutionId}` で入力し、コミット ID を変数構文 `#{SourceVariables.CommitId}` で入力します。

   > **Note**
   >
   > 変数構文は、ウィザードの任意のアクション設定フィールドに入力できます。

5. [作成] を選択します。

6. パイプラインが作成されたら、ウィザードによって作成された名前空間を表示できます。パイプラインで、名前空間を表示するステージのヘルプペインアイコンを選択します。この例では、ソースアクションの自動生成された名前空間、SourceVariables が表示されます。



既存のアクションの名前空間を編集するには

1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. 編集するパイプラインを選択して [編集] を選択します。ソースステージで、[Edit stage (ステージを編集)] を選択します。 CodeCommit アクションを追加します。
3. [アクションの編集] で、[Variable namespace (変数の名前空間)] フィールドを表示します。既存のアクションが以前に作成された場合、またはウィザードを使用せずに作成された場合は、名前空間を追加する必要があります。[Variable namespace (変数の名前空間)] に名前空間名を入力し、[Save (保存)] を選択します。

出力変数を表示するには

1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。
2. パイプラインが作成され、正常に実行できたら、アクションの実行詳細ページで変数の値を表示することもできます。詳細については、「変数を表示する (コンソール)」を参照してください。

変数のアクションを設定する (CLI)

create-pipeline コマンドを使用してパイプラインを作成するか、update-pipeline コマンドを使用してパイプラインを編集する場合は、アクションの設定で変数を参照したり、使用したりできます。

名前空間が指定されていない場合、アクションによって生成された変数は、ダウンストリームアクション設定で参照することはできません。

名前空間でアクションを設定するには

1. 「でパイプラインを作成する CodePipeline」の手順 に従って、CLI を使用してパイプラインを作成します。入力ファイルを起動して、create-pipeline コマンドに `--cli-input-json` パラメータを指定します。パイプライン構造で、`namespace` パラメータを追加し、`SourceVariables` などの名前を指定します。

```
. . .
{
          "inputArtifacts": [],
          "name": "Source",
          "region": "us-west-2",
          "namespace": "SourceVariables",
          "actionTypeId": {
            "category": "Source",
            "owner": "AWS",
            "version": "1",
            "provider": "CodeCommit"
          },
          "outputArtifacts": [

. . .
```

2. **MyPipeline.json** のような名前でファイルを保存します。
3. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、create-pipelineコマンドを実行し、パイプラインを作成します。

   create-pipeline コマンドを実行したときに作成したファイルを呼び出します。例:

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

変数を消費するダウンストリームアクションを設定するには

1. 入力ファイルを編集して、update-pipeline コマンドに `--cli-input-json` パラメータを指定します。ダウンストリームアクションで、そのアクションの設定に変数を追加します。変数は、ピリオドで区切られた名前空間とキーで構成されます。たとえば、パイプライン実行 ID とソースコミット ID の変数を追加するには、変数 `#{codepipeline.PipelineExecutionId}` に

変数の名前空間 codepipeline を指定します。変数の #{SourceVariables.CommitId} 名前空間 SourceVariables を指定します。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifacts"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Execution_ID\",\"value
\":\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":
\"Commit_ID\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

2. **MyPipeline.json** のような名前でファイルを保存します。
3. ターミナル (Linux、macOS、UNIX) またはコマンドプロンプト (Windows) で、create-pipelineコマンドを実行し、パイプラインを作成します。

   create-pipeline コマンドを実行したときに作成したファイルを呼び出します。例:

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

出力変数を表示する

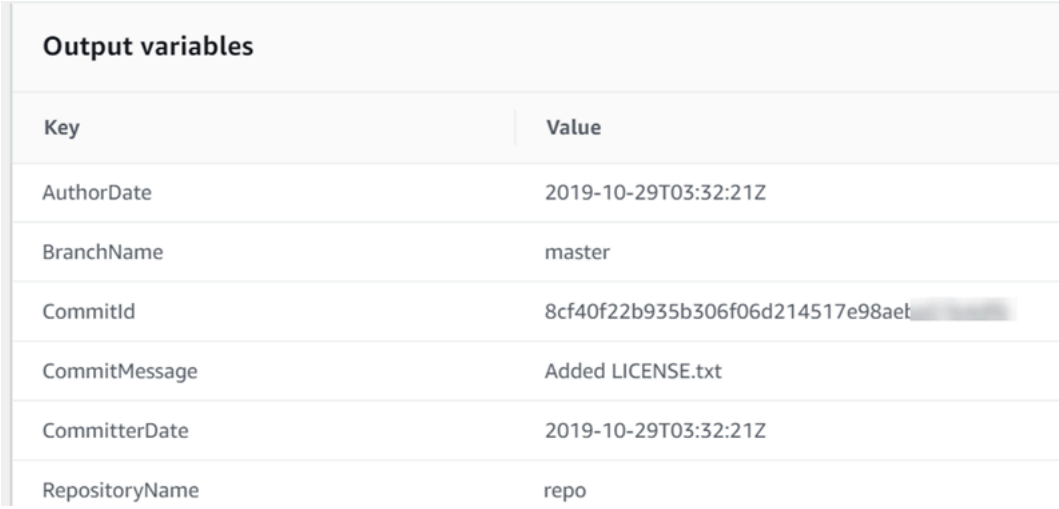
アクション実行の詳細を表示して、各実行に固有のアクションの変数を表示できます。

変数を表示する (コンソール)

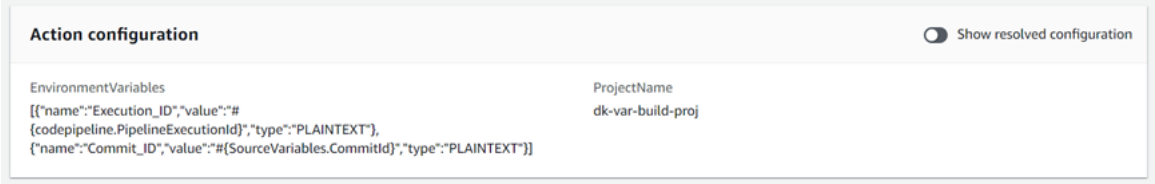
コンソールを使用して、アクションの変数を表示できます。

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

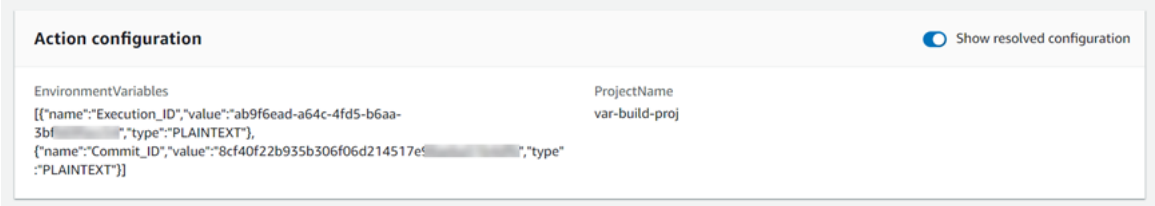
   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、パイプラインの名前を選択します。
3. [View history (履歴の表示)] を選択します。
4. パイプラインが正常に実行されると、ソースアクションによって生成された変数を表示できます。[View history (履歴の表示)] を選択します。パイプライン実行のアクションリストでソースを選択すると、アクションの実行の詳細が表示されます CodeCommit 。アクションの詳細画面で、[Output variables (出力変数)] の下の変数を表示します。



5. パイプラインが正常に実行されると、ビルドアクションによって消費される変数を表示できます。[View history (履歴の表示)] を選択します。パイプライン実行のアクションリストで、ビルド を選択して、アクションの実行の詳細を表示します CodeBuild 。アクションの詳細ページで、[アクション設定] の下にある変数を表示します。自動生成された名前空間が表示されます。



デフォルトでは、[アクション設定] には変数の構文が表示されます。[Show resolved configuration (解決された設定を表示)] を選択して、アクションの実行中に生成された値を表示するようにリストを切り替えることができます。



変数を表示する (CLI)

list-action-executions コマンドを使用して、アクションの変数を表示できます。

1. 以下のコマンドを使用します。

```
aws codepipeline list-action-executions
```

出力には、次に示すように outputVariables パラメータが表示されます。

```
"outputVariables": {
                    "BranchName": "main",
                    "CommitMessage": "Updated files for test",
                    "AuthorDate": "2019-11-08T22:24:34Z",
                    "CommitId": "d99b0083cc10EXAMPLE",
                    "CommitterDate": "2019-11-08T22:24:34Z",
                    "RepositoryName": "variables-repo"
                },
```

2. 以下のコマンドを使用します。

```
aws codepipeline get-pipeline --name <pipeline-name>
```

アクションの CodeBuild アクション設定では、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、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。承認を追加するパイプラインを選択します。
2. パイプラインを編集するには、[編集] を選択します。ソースアクションの後に、手動承認を追加します。[アクション名] に、承認処理の名前を入力します。
3. [アクションプロバイダ] で、[手動承認] を選択します。
4. レビュー用の URL で、 の変数構文`CommitId`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を持つアクションの変数構文は CodeCommit `SourceVariables`です`#{SourceVariables.CommitId}`。

   [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 環境変数で変数を使用する BranchName

パイプラインに CodeBuild アクションを追加すると、 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の`BranchName`出力変数を参照できます。の アクションからの出力変数を使用すると CodePipeline、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 CodeBuild 環境変数に出力変数構文を追加する方法を示します。この例の出力変数構文は、 の GitHub ソースアクション出力変数を表します`BranchName`。アクションが正常に実行されると、 変数は GitHub ブランチ名を表示するように解決されます。

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

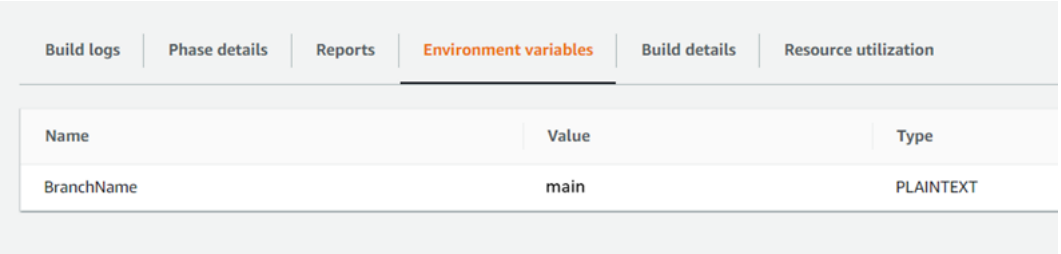
   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。承認を追加するパイプラインを選択します。
2. パイプラインを編集するには、[編集] を選択します。 CodeBuild アクションを含むステージで、ステージの編集 を選択します。
3. アイコンを選択して CodeBuild アクションを編集します。

4. [編集アクション] ページの 環境変数 で、次のように入力します。

   - [名前]に、環境変数の名前を入力します。
   - [値]に、パイプライン出力変数の変数構文を入力します。これには、ソースアクションに割り当てられた名前空間が含まれます。例えば、デフォルトの名前空間を持つアクションの出力変数構文 GitHubは `SourceVariables`です`#{SourceVariables.BranchName}`。
   - [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。以下のうちのひとつを選択します。

   - CodePipeline コンソール： パイプラインを選択し、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。これは出力 GitHub 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、 GitHub ソースアクションから解決された`BranchName`出力変数です。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 では、解決された出力変数は CodeBuild 環境変数の値です。この例では、環境変数の名前は `BranchName`で、値は GitHub ソースアクションから解決された`BranchName`出力変数です。この例では、この値は `main` です。



でのステージ移行の操作 CodePipeline

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、「での承認アクションの管理 CodePipeline」を参照してください。

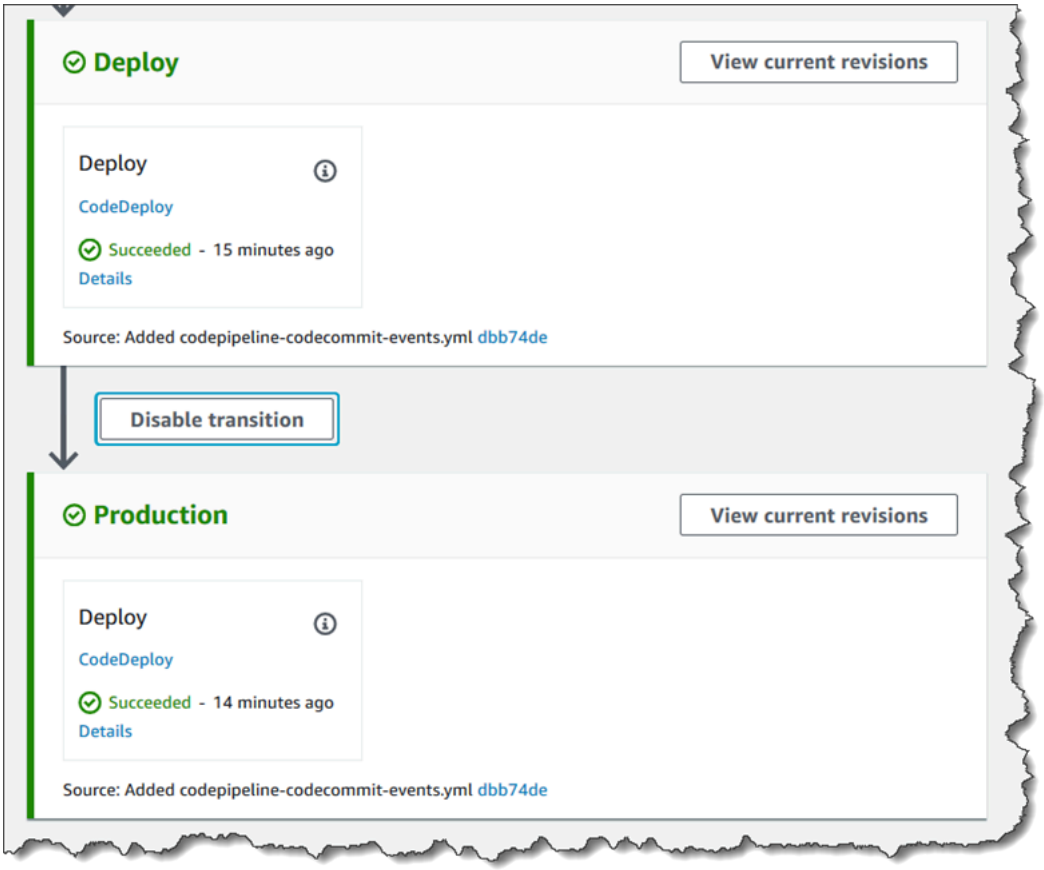
トピック

移行を無効化または有効化する (コンソール)

パイプラインで遷移を無効/有効にするには

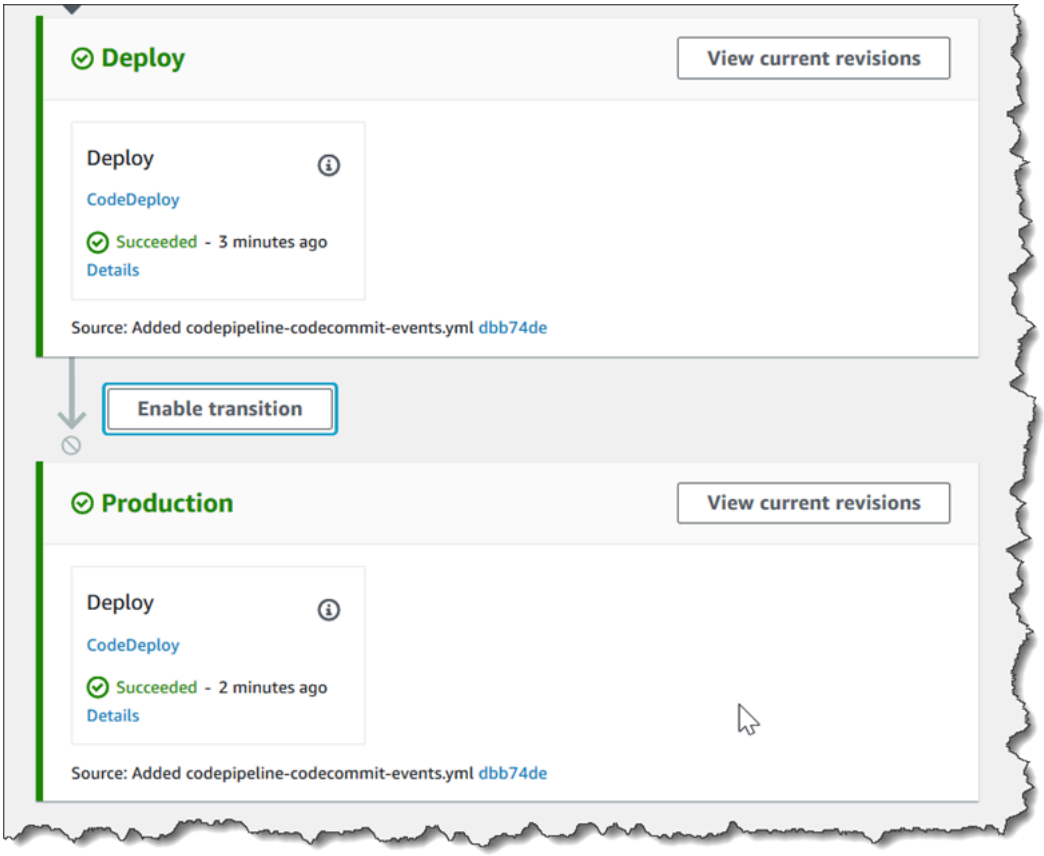
1. にサインイン AWS Management Console し、http://console.aws.amazon.com/codesuite/codepipeline/home で CodePipeline コンソールを開きます。

   AWS アカウントに関連付けられているすべてのパイプラインの名前が表示されます。
2. [Name] で、遷移を有効または無効にするパイプラインの名前を選択します。これにより、パイプラインの詳細ビューが開いて、パイプラインのステージ間の遷移がわかります。
3. 実行する最後のステージの後の矢印を見つけ、その横にあるボタンを選択します。たとえば、以下のパイプラインの例で、[Staging (ステージング)] ステージのアクションを実行するが、[Production (本番稼働用)] という名前のステージのアクションを実行しないようにするには、その 2 つのステージの間の [Disable transition (移行を無効にする)] ボタンを選択します。



4. [移行を無効にする] ダイアログボックスで、移行を無効にする理由を入力し、[無効化] を選択します。

   ボタンは、矢印の前のステージと矢印の後のステージとの間で遷移が無効にされたことを示すように変わります。無効にされた移行の後のステージですでに実行されていたリビジョンは、パイプラインにより続行されますが、それ以降のリビジョンは、無効にされた移行の後には続行されません。



5. 矢印の横にある [移行を有効にする] ボタンを選択します。[Enable transition] ダイアログボックスで、[Enable] を選択します。パイプラインはすぐに 2 つのステージ間の移行を有効にします。移行が無効にされた後、初期ステージで実行されていたリビジョンがある場合、パイプラインはすぐに、以前の無効にされた移行の後のステージで最新のリビジョンの実行を開始します。パイプラインは、そのリビジョンをパイプラインの残りのすべてのステージで実行します。

> **Note**
>
> 移行を有効にした後、変更が CodePipeline コンソールに表示されるまでに数秒かかる場合があります。

移行を無効化または有効化する (CLI)

を使用してステージ間の移行を無効にするには AWS CLI、 disable-stage-transition コマンドを実行します。無効にされた遷移を有効にするには、enable-stage-transition コマンドを実行します。

遷移を無効にするには

1. ターミナル (Linux、macOS または Unix) またはコマンドプロンプト (Windows) を開き、 を使用してdisable-stage-transitionコマンド AWS CLI を実行し、パイプラインの名前、移行を無効にするステージの名前、移行タイプ、および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。

   例えば、 という名前のパイプラインで *Staging* という名前のステージへの移行を無効にするには*MyFirstPipeline*、次のようなコマンドを入力します。

   ```
   aws codepipeline disable-stage-transition --pipeline-name MyFirstPipeline --stage-
   name Staging --transition-type Inbound --reason "My Reason"
   ```

   このコマンドは何も返しません。
2. 移行が無効になっていることを確認するには、 CodePipeline コンソールでパイプラインを表示するか、 get-pipeline-state コマンドを実行します。詳細については、「パイプラインを表示する (コンソール)」および「パイプラインの詳細と履歴を表示する (CLI)」を参照してください。

遷移を有効にするには

1. ターミナル (Linux、macOS または Unix) またはコマンドプロンプト (Windows) を開き、 を使用してenable-stage-transitionコマンド AWS CLI を実行し、パイプラインの名前、移行を有効にするステージの名前、移行タイプを指定します。

   例えば、 という名前のパイプラインで *Staging* という名前のステージへの移行を有効にするには*MyFirstPipeline*、次のようなコマンドを入力します。

   ```
   aws codepipeline enable-stage-transition --pipeline-name MyFirstPipeline --stage-
   name Staging  --transition-type Inbound
   ```

   このコマンドは何も返しません。
2. 移行が無効になっていることを確認するには、 CodePipeline コンソールでパイプラインを表示するか、 get-pipeline-state コマンドを実行します。詳細については、「パイプラインを表示する (コンソール)」および「パイプラインの詳細と履歴を表示する (CLI)」を参照してください。

パイプラインのモニタリング

モニタリングは、 AWS CodePipelineの信頼性、可用性、パフォーマンスを維持する上で重要な部分です。マルチポイント障害が発生した場合は、その障害をより簡単にデバッグできるように、 AWS ソリューションのすべての部分からモニタリングデータを収集する必要があります。ただし、モニタリングを開始する前に、以下の質問に回答するモニタリング計画を作成する必要があります。

- どのような目的でモニタリングしますか?
- どのリソースをモニタリングしますか?
- どのくらいの頻度でこれらのリソースをモニタリングしますか?
- どのモニタリングツールが使用可能ですか?
- 誰がモニタリングタスクを実行しますか?
- 問題が発生した場合、だれが通知を受け取りますか?

次のツールを使用して、 CodePipeline パイプラインとその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 — でイベントをモニタリング CodePipelineし EventBridge、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更を検出できます。EventBridge は、そのデータを AWS Lambda や Amazon Simple Notification Service. EventBridge events などのターゲットにルーティングします。これは、Amazon CloudWatch Events に表示されるものと同じです。
- デベロッパー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で CodePipeline イベントをモニタリングし、Amazon Simple Notification Service 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 - AWS アカウント CodePipeline で によって、または 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 CloudTrail するために使用します。新しいログファイルが配信されたときに に Amazon SNS 通知 CloudWatch を発行するように選択して、すばやくアクションを実行できます。
- 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に関する詳細を表示できます。

トピック

- CodePipeline イベントのモニタリング

- イベントのプレースホルダーバケットに関するリファレンス
- を使用した CodePipeline API コールのログ記録 AWS CloudTrail

CodePipeline イベントのモニタリング

で CodePipeline イベントをモニタリングすることで EventBridge、独自のアプリケーション software-as-a-service (SaaS) アプリケーションからリアルタイムデータのストリームを配信し、そのデータを AWS Lambda や Amazon Simple Notification Service などのターゲットに AWS のサービス EventBridge ルーティングできます。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す。Amazon Events は、 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 ユーザーガイド」の「Amazon とは EventBridge」を参照してください。 EventBridge

> **Note**
>
> イベントを管理する EventBridge には、Amazon が推奨されます。Amazon CloudWatch Events と EventBridge は基盤となるサービスと API と同じですが、より多くの機能 EventBridge を提供します。 CloudWatch イベントまたは のいずれかで行った変更は EventBridge 、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イベントソースとして などのルールを作成します CodePipeline。例えば、パイプライン、ステージ、またはアクションの状態が変化したときにルールをトリガーするイベントソース CodePipeline として を使用するイベントパターンを持つルールを作成できます。
- ターゲット。新しいルールは選択したサービスをイベントターゲットとして受け取ります。ターゲットサービスを設定することで、通知を送り、状態情報を取得し、是正措置を講じ、イベントを開始し、その他のアクションを実行します。ターゲットを追加するときは、選択したターゲットサービスを呼び出せるよう EventBridge に、 にアクセス許可も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、 内のリソースの状態が変化する EventBridge たびに、 にイベントを報告します AWS アカウント 。イベントは、次のリソースの保証された at-least-once ベースで出力されます。

- 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 EventBridge を使用して によって出力されます。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 PutEvents API リファレンス」の「 API コール」を参照してください。 EventBridge

次の例は、 のイベントを示しています CodePipeline。可能な場合、各例は生成されたイベントのスキーマと、処理されたイベントのスキーマを示しています 。

トピック

詳細タイプ

モニタリングするイベントを設定する場合、イベントの詳細タイプを選択できます。

以下の状態が変わった時に通知が送信されるように設定できます。

- 指定したパイプラインまたはすべてのパイプライン。これを制御するには、"detail-type": "CodePipeline Pipeline Execution State Change" を使用します。
- 指定したパイプラインまたはすべてのパイプライン内の、指定したステージまたはすべてのステージ。これを制御するには、"detail-type": "CodePipeline Stage Execution State Change" を使用します。
- 指定したパイプラインまたはすべてのパイプライン内の、指定したステージまたはすべてのステージ内の、指定したアクションまたはすべてのアクション。これを制御するには、"detail-type": "CodePipeline Action Execution State Change" を使用します。

> **Note**
>
> によって出力されるイベントには detail-typeパラメータ EventBridge が含まれており、イベントが処理detailTypeされると に変換されます。

Detail-type	状態	説明
CodePipeline パイプライン実行状態の変更	CANCELED	パイプライン構造が更新されたため、パイプライン実行がキャンセルされました。
	FAILED	パイプライン実行が正常に完了しませんでした。
	再開	失敗したパイプライン実行が RetryStageExecution API コールに応じて再試行されました。
	開始	パイプライン実行が現在実行中です。
	停止	停止プロセスが完了し、パイプラインの実行が停止します。
	停止中	パイプライン実行は、パイプライン実行を [Stop and wait (停止して待機)] するか、[Stop and abandon (停止して中止)] する要求により、停止しています。
	成功	パイプライン実行が正常に完了しました。

Detail-type	状態	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 ステージ実行状態の変更	CANCELED	パイプライン構造が更新されたため、ステージはキャンセルされました。
	FAILED	ステージは正常に完了しませんでした。
	再開	失敗したステージが `RetryStageExecution` API コールに応じて再試行されました。
	開始	このステージは現在実行中です。
	停止	停止プロセスが完了し、ステージの実行が停止します。
	停止中	ステージ実行は、パイプライン実行を [Stop and wait (停止して待機)] するか、[Stop and abandon (停止して中止)] する要求により、停止しています。
	成功	ステージは正常に完了しました。
CodePipeline アクション実行状態の変更	中止	パイプラインの実行を [Stop and abandon (停止して中止)] する要求により、アクションは中止されます。
	CANCELED	パイプライン構造が更新されたため、アクションはキャンセルされました。
	FAILED	承認アクションでは、失敗の状態は、アクションがレビュー者により拒否されたか、または誤ったアクション設定のために失敗したことを意味します。
	開始	アクションは現在実行中です。
	成功	アクションは正常に完了しました。

パイプラインレベルのイベント

パイプラインレベルのイベントは、パイプライン実行の状態が変更されたときに発生します。

トピック

パイプライン開始イベント

パイプライン実行が開始すると、以下の内容の通知が送信されます。この例は、"myPipeline" リージョンの us-east-1 という名前のパイプラインです。id フィールドはイベント ID を表し、account フィールドは、パイプラインが作成されるアカウント ID を表します。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
```

```
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:44:50Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
east-1:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "trigger-type": "StartPipelineExecution",
            "trigger-detail": "arn:aws:sts::123456789012:assumed-role/Admin/my-user"
        },
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

パイプライン停止イベント

パイプライン実行が停止すると、以下の内容の通知が送信されます。この例は、myPipeline リージョンの us-west-2 という名前のパイプラインです。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
```

```
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "STOPPING",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
        "stop-execution-comments": "Stopping the pipeline for an update"
    }
}
```

パイプライン成功イベント

パイプライン実行が成功すると、以下の内容の通知が送信されます。この例は、myPipeline us-east-1 リージョンの という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:44Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
        "version": 3.0,
```

```
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-30T22:13:51Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "SUCCEEDED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

パイプラインが成功しました (Git タグを使用した例)

パイプラインの実行が再試行され、成功したステージがあると、以下の内容の通知を送信するイベントが発生します。この例は、eu-central-1 リージョンの myPipeline という名前のパイプラインのもので、execution-trigger が Git タグに設定されています。

> **Note**
>
> execution-trigger} フィールドには、パイプラインをトリガーしたイベントの種類に応じて、tag-name または branch-name のいずれかが表示されます。

```
{
    "version": "0",
    "id": "b128b002-09fd-4574-4eba-27152726c777",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2023-10-26T13:50:53Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "e17b5773-cc0d-4db2-9ad7-594c73888de8",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "SUCCEEDED",
        "version": 32.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

パイプライン失敗イベント

パイプライン実行が成功すると、以下の内容の通知が送信されます。この例は、"myPipeline" リージョンの us-west-2 という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
```

```
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "state": "FAILED",
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
```

```
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
    }
```

パイプラインが失敗しました (Git タグを使用した例)

トリガーで設定されたパイプラインがソースステージで失敗しない限り、以下の内容の通知を送信するイベントが発生します。この例は、eu-central-1 リージョンの myPipeline という名前のパイプラインのもので、execution-trigger が Git タグに設定されています。

> **Note**
>
> execution-trigger} フィールドには、パイプラインをトリガーしたイベントの種類に応じて、tag-name または branch-name のいずれかが表示されます。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Pipeline Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
```

```
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Pipeline Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "execution-trigger": {
            "author-display-name": "Mary Major",
            "full-repository-name": "mmajor/sample-project",
            "provider-type": "GitLab",
            "author-email": "email_address",
            "commit-message": "Update file README.md",
            "author-date": "2023-08-16T21:08:08Z",
            "tag-name": "gitlab-v4.2.1",
            "commit-id": "commit_ID",
            "connection-arn": "arn:aws:codestar-connections:eu-
central-1:123456789012:connection/0f5b706a-1a1d-46c5-86b6-f177321bcfb2",
            "author-id": "Mary Major"
        },
        "state": "FAILED",
```

```
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "failedActionCount": 1,
        "failedActions": [
            {
                "action": "Deploy",
                "additionalInformation": "Deployment <ID> failed"
            }
        ],
        "failedStage": "Deploy"
    }
```

ステージレベルのイベント

ステージレベルのイベントは、ステージ実行の状態が変更されたときに発生します。

トピック

- ステージ開始イベント
- ステージ停止イベント
- ステージ停止イベント
- ステージ再試行後のステージ再開イベント

ステージ開始イベント

ステージ実行が開始すると、以下の内容の通知が送信されます。この例は、"myPipeline" ステージの us-east-1 リージョンの Prod という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
```

```
    "account": 123456789012,
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "version": 1.0,
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Prod",
        "state": "STARTED",
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Stage Execution State Change",
    "region": "us-east-1",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:40Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Source",
        "state": "STARTED",
        "version": 1.0,
        "pipeline-execution-attempt": 0.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "sourceActions": [
            {
                "sourceActionName": "Source",
```

```
                "sourceActionProvider": "CodeCommit",
                "sourceActionVariables": {
                    "BranchName": "main",
                    "CommitId": "<ID>",
                    "RepositoryName": "my-repo"
                }
            }
        ]
    }
}
```

ステージ停止イベント

ステージ実行が停止すると、以下の内容の通知が送信されます。この例は、myPipeline ステージの us-west-2 リージョンの Deploy という名前のパイプラインです。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:02:20Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPING",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

ステージ停止イベント

ステージ実行が停止すると、以下の内容の通知が送信されます。この例は、myPipeline ステージの us-west-2 リージョンの Deploy という名前のパイプラインです。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:49:39.208Z",
        "stage": "Deploy",
        "state": "STOPPED",
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

ステージ再試行後のステージ再開イベント

ステージ実行が再開され、ステージが再試行されると、以下の内容の通知を送信するイベントが発生します。

ステージが再試行されると、例に示すように `stage-last-retry-attempt-time` フィールドが表示されます。再試行が行われた場合、そのフィールドはすべてのステージイベントで表示されます。

> **Note**
>
> `stage-last-retry-attempt-time` フィールドは、ステージが再試行された後、以降のすべてのステージイベントに存在します。

```
{
    "version": "0",
    "id": "38656bcd-a798-5f92-c738-02a71be484e1",
    "detail-type": "CodePipeline Stage Execution State Change",
    "source": "aws.codepipeline",
```

```
    "account": "123456789012",
    "time": "2023-10-26T14:14:56Z",
    "region": "eu-central-1",
    "resources": [
        "arn:aws:codepipeline:eu-central-1:123456789012:BuildFromTag"
    ],
    "detail": {
        "pipeline": "BuildFromTag",
        "execution-id": "05dafb6a-5a56-4951-a858-968795364846",
        "stage-last-retry-attempt-time": "2023-10-26T14:14:56.305Z",
        "stage": "Build",
        "state": "RESUMED",
        "version": 32.0,
        "pipeline-execution-attempt": 2.0
    }
}
```

アクションレベルのイベント

アクションレベルのイベントは、アクション実行の状態が変更されたときに発生します。

トピック

- アクション開始イベント
- アクション成功イベント
- アクション失敗イベント
- アクション放棄イベント

アクション開始イベント

アクション実行が開始すると、以下の内容の通知が送信されます。この例は、デプロイアクション myPipeline の us-east-1 リージョンの myAction という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
    "id": 01234567-EXAMPLE,
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": 123456789012,
```

```
    "time": "2020-01-24T22:03:07Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": 12345678-1234-5678-abcd-12345678abcd,
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Prod",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "myAction",
        "state": "STARTED",
        "type": {
            "owner": "AWS",
            "category": "Deploy",
            "provider": "CodeDeploy",
            "version": "1"
        },
        "version": 2.0
        "pipeline-execution-attempt": 1.0
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ]
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
```

```
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "input-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-east-1-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
        ],
        "state": "STARTED",
        "region": "us-east-1",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-east-1:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

アクション成功イベント

アクション実行が成功すると、以下の内容の通知が送信されます。この例は、ソースアクション "myPipeline" の us-west-2 リージョンの "Source" という名前のパイプラインです。このイベントタイプには、2 つの異なる region フィールドがあります。イベント region フィールドは、パイプラインイベントのリージョンを指定します。region セクション下の detail フィールドは、アクションのリージョンを指定します。

Emitted event


```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-24T22:03:11Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Added LICENSE.txt",
            "external-execution-id": "8cf40fEXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-BUCKETEXAMPLE",
                    "key": "myPipeline/SourceArti/KEYEXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
        },
        "version": 3.0,
```

```
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:45:44Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:ACCOUNT:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "arn:aws:codepipeline:us-west-2:123456789012:myPipeline",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Source",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codecommit/home#/repository/my-repo/commit/8cf40f2EXAMPLE",
            "external-execution-summary": "Edited index.html",
            "external-execution-id": "36ab3ab7EXAMPLE"
        },
        "output-artifacts": [
            {
                "name": "SourceArtifact",
                "s3location": {
                    "bucket": "codepipeline-us-west-2-EXAMPLE",
                    "key": "myPipeline/SourceArti/EXAMPLE"
                }
            }
        ],
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Source",
        "state": "SUCCEEDED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeCommit",
            "category": "Source",
            "version": "1"
```

```
        },
        "version": 1.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {}
}
```

アクション失敗イベント

アクション実行が成功すると、以下の内容の通知が送信されます。この例は、"myPipeline" アクションの us-west-2 リージョンの "Deploy" という名前のパイプラインです。

Emitted event

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:55:43Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "start-time": "2023-10-26T13:51:09.981Z",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://us-west-2.console.aws.amazon.com/
codedeploy/home?#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
```

```
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
        "version": 4.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

Processed event

```
{
    "account": "123456789012",
    "detailType": "CodePipeline Action Execution State Change",
    "region": "us-west-2",
    "source": "aws.codepipeline",
    "time": "2021-06-24T00:46:16Z",
    "notificationRuleArn": "arn:aws:codestar-notifications:us-
west-2:123456789012:notificationrule/a69c62c21EXAMPLE",
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "execution-result": {
            "external-execution-url": "https://console.aws.amazon.com/codedeploy/
home?region=us-west-2#/deployments/<ID>",
            "external-execution-summary": "Deployment <ID> failed",
            "external-execution-id": "<ID>",
            "error-code": "JobFailed"
        },
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "FAILED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
```

```
        },
        "version": 13.0,
        "pipeline-execution-attempt": 1.0
    },
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "additionalAttributes": {
        "additionalInformation": "Deployment <ID> failed"
    }
}
```

アクション放棄イベント

アクション実行が放棄されると、以下の内容の通知が送信されます。この例は、"myPipeline" アクションの us-west-2 リージョンの "Deploy" という名前のパイプラインです。

```
{
    "version": "0",
    "id": "01234567-EXAMPLE",
    "detail-type": "CodePipeline Action Execution State Change",
    "source": "aws.codepipeline",
    "account": "123456789012",
    "time": "2020-01-31T18:21:39Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:codepipeline:us-west-2:123456789012:myPipeline"
    ],
    "detail": {
        "pipeline": "myPipeline",
        "execution-id": "12345678-1234-5678-abcd-12345678abcd",
        "stage": "Deploy",
        "action-execution-id": "47f821c5-a902-44b2-ae61-b878d31ecd21",
        "action": "Deploy",
        "state": "ABANDONED",
        "region": "us-west-2",
        "type": {
            "owner": "AWS",
            "provider": "CodeDeploy",
            "category": "Deploy",
            "version": "1"
        },
```

```
        "version": 3.0,
        "pipeline-execution-attempt": 1.0
    }
}
```

パイプラインイベントで通知を送信するルールを作成する

ルールは特定のイベントを監視し、選択した AWS ターゲットにルーティングします。別の AWS アクションが発生したときに自動的に AWS アクションを実行するルール、または設定されたスケジュールで定期的に AWS アクションを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)

これらのステップは、 EventBridge コンソールを使用して、 で変更の通知を送信するルールを作成する方法を示しています CodePipeline。

Amazon S3 ソースで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. https://console.aws.amazon.com/events/ で Amazon EventBridge コンソールを開きます。
2. 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。[次へ] をクリックします。
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。
   - パイプラインレベルのイベントに適用されるルールについては、CodePipelineパイプライン実行状態の変更 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については、CodePipelineステージ実行状態の変更 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については、CodePipeline「アクション実行状態の変更」を選択します。

7. ルールを適用する状態変更を指定します。

   - すべての状態変更に適用されるルールには、[Any state] を選択します。
   - いくつかの状態変更のみに適用されるルールには、[Specific state(s)] を選択してから、リストから 1 つ以上の値を選択します。

8. セレクタが許可するよりも詳細なイベントパターンには、[イベントパターン] ウィンドウの [パターンの編集] オプションを使用して、JSON 形式のイベントパターンを指定することもできます。

   > **Note**
   >
   > 指定されていない場合、イベントパターンは、すべてのパイプライン/ステージ/アクションの状態に対して作成されます。

   より詳細なイベントパターンについては、以下のイベントパターンの例をコピーして [イベントパターン] ウィンドウに貼り付けることができます。

   - Example

     このイベントパターンのサンプルを使用して、すべてのパイプラインで失敗したデプロイとビルドアクションをキャプチャします。

```
{
"source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Deploy", "Build"]
    }
  }
}
```

- Example

  このイベントパターンのサンプルを使用して、すべてのパイプラインで、拒否された、または失敗したすべての承認アクションをキャプチャします。

```
{
 "source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Action Execution State Change"
  ],
  "detail": {
    "state": [
      "FAILED"
    ],
    "type": {
      "category": ["Approval"]
    }
  }
}
```

- Example

  このイベントパターンのサンプルを使用して、指定したパイプラインからすべてのイベントをキャプチャします。

```
{
"source": [
    "aws.codepipeline"
  ],
  "detail-type": [
    "CodePipeline Pipeline Execution State Change",
    "CodePipeline Action Execution State Change",
    "CodePipeline Stage Execution State Change"
  ],
  "detail": {
    "pipeline": ["myPipeline", "my2ndPipeline"]
  }
}
```

9. [次へ] をクリックします。

10. [ターゲットタイプ] で、[AWS サービス] を選択します。
11. 「ターゲットを選択」で、「」を選択しますCodePipeline。[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-2:80398例：MyFirstPipeline

12. EventBridge ルールに関連付けられたターゲットを呼び出す EventBridge アクセス許可を付与する IAM サービスロールを作成または指定するには (この場合、ターゲットは CodePipeline) 。
    - この特定のリソースの新しいロールを作成する を選択して、パイプラインの実行を開始する EventBridge アクセス許可を付与するサービスロールを作成します。
    - 「既存のロールを使用」を選択して、パイプラインの実行を開始するアクセス EventBridge 許可を付与するサービスロールを入力します。
13. [次へ] をクリックします。
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)

これらのステップは、 CLI を使用して、 で変更の通知を送信する CloudWatch イベントルールを作成する方法を示しています CodePipeline。

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、アカウント CodePipeline AWS に関連付けられた で作成するすべてのパイプラインで一意である必要があります。
- ルールで使用するソースと詳細フィールドのイベントパターン。詳細については、「Amazon EventBridge とイベントパターン」を参照してください。

をイベントソース CodePipeline 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。)

   次のサンプルコマンドは--event-pattern、 を使用して、“MyPipelineStateChanges”「myPipeline」という名前のパイプラインの実行が失敗した場合に CloudWatch イベントを発行する というルールを作成します。myPipeline."

   ```
   aws events put-rule --name "MyPipelineStateChanges" --event-pattern "{\"source\":
   [\"aws.codepipeline\"],\"detail-type\":[\"CodePipeline Pipeline Execution State
    Change\"],\"detail\":{\"pipeline\":[\"myPipeline\"],\"state\":[\"FAILED\"]}}"
   ```

2. put-targets コマンドを呼び出し、次のパラメータを含めます:

   - `--rule` パラメータは、put-rule を使用して作成した `rule_name` で使用されます。
   - `--targets` パラメータは、ターゲットリストのリスト `Id` と Amazon SNS トピックの ARN で使用されます。

   次のサンプルコマンドでは、`MyPipelineStateChanges` と呼ばれるルールに対して指定し、ターゲット `Id` は 1 番で構成されています。これは、ルールのターゲットのリストが何であるかを示し、この場合は ターゲット 1 です。このサンプルコマンドでは、Amazon SNS トピックの例 ARN も指定されます。

   ```
   aws events put-targets --rule MyPipelineStateChanges --targets
    Id=1,Arn=arn:aws:sns:us-west-2:11111EXAMPLE:MyNotificationTopic
   ```

3. が指定されたターゲットサービス EventBridge を使用して通知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、「Amazon でのリソースベースのポリシーの使用 EventBridge」を参照してください。

イベントのプレースホルダーバケットに関するリファレンス

このセクションは参照のみを目的としています。イベント検出リソースを使用するパイプラインを作成する方法については、「 ソースアクションと変更検出方法」を参照してください。

Amazon S3 によって提供されるソースアクションと は、イベントベースの変更検出リソース CodeCommit を使用して、ソースバケットまたはリポジトリで変更が行われたときにパイプライン

をトリガーします。これらのリソースは、 CodeCommit リポジトリへのコード変更など、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 イベントルールです。Amazon S3 ソースに CloudWatch Events を使用する場合は、イベントがログに記録 CloudTrail されるようにオンにする必要があります。 CloudTrail には、ダイジェストを送信できる S3 バケットが必要です。イベント CloudWatch 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が、プレースホルダーバケットからデータにアクセスすることはできません。

- CLI または を使用して CloudWatch イベントリソースを 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。
- コンソールを使用して S3 ソースでパイプラインをセットアップした場合、コンソールは CloudWatch イベントリソースを作成するときに 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、「ログファイルの取得と表示」を参照してください CloudTrail 。

トピック

- イベントのプレースホルダーバケット名 (リージョン別)

イベントのプレースホルダーバケット名 (リージョン別)

次の表は、Amazon S3 ソースアクションでパイプラインの変更検出イベントを追跡するログファイルが含まれている S3 プレースホルダーバケットの名前の一覧です。

リージョン名	プレースホルダーバケット名	リージョン識別子
米国東部 (オハイオ)	codepipeline-cloudtrail-placeholder-bucket-us-east-2	us-east-2

リージョン名	プレースホルダーバケット名	リージョン識別子
米国東部 (バージニア北部)	codepipeline-cloudtrail-placeholder-bucket-us-east-1	us-east-1
米国西部 (北カリフォルニア)	codepipeline-cloudtrail-placeholder-bucket-us-west-1	us-west-1
米国西部 (オレゴン)	codepipeline-cloudtrail-placeholder-bucket米国西部 2	us-west-2
カナダ (中部)	codepipeline-cloudtrail-placeholder-bucket-ca-central-1	ca-central-1
欧州 (フランクフルト)	codepipeline-cloudtrail-placeholder-bucket-eu-central-1	eu-central-1
欧州 (アイルランド)	codepipeline-cloudtrail-placeholder-bucket-eu-west-1	eu-west-1
欧州 (ロンドン)	codepipeline-cloudtrail-placeholder-bucketeu-west-2	eu-west-2
欧州 (パリ)	codepipeline-cloudtrail-placeholder-bucketeu-west-3	eu-west-3
欧州 (ストックホルム)	codepipeline-cloudtrail-placeholder-bucket-eu-north-1	eu-north-1
アジアパシフィック (香港)	codepipeline-cloudtrail-placeholder-bucket-ap-east-1	ap-east-1
アジアパシフィック (ハイデラバード)	codepipeline-cloudtrail-placeholder-bucket-ap-south-2	ap-south-2
アジアパシフィック (ジャカルタ)	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-3	ap-southeast-3

リージョン名	プレースホルダーバケット名	リージョン識別子
アジアパシフィック (メルボルン)	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-4	ap-southeast-4
アジアパシフィック (ムンバイ)	codepipeline-cloudtrail-placeholder-bucket-ap-south-1	ap-south-1
アジアパシフィック (大阪)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-3-prod	ap-northeast-3
アジアパシフィック (東京)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-1	ap-northeast-1
アジアパシフィック (ソウル)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-2	ap-northeast-2
アジアパシフィック (シンガポール)	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-1	ap-southeast-1
アジアパシフィック (シドニー)	codepipeline-cloudtrail-placeholder-bucket-ap-southeast-2	ap-southeast-2
アジアパシフィック (東京)	codepipeline-cloudtrail-placeholder-bucket-ap-northeast-1	ap-northeast-1
カナダ (中部)	codepipeline-cloudtrail-placeholder-bucket-ca-central-1	ca-central-1
欧州 (フランクフルト)	codepipeline-cloudtrail-placeholder-bucket-eu-central-1	eu-central-1

リージョン名	プレースホルダーバケット名	リージョン識別子
欧州 (アイルランド)	codepipeline-cloudtrail-placeholder-bucket-eu-west-1	eu-west-1
欧州 (ロンドン)	codepipeline-cloudtrail-placeholder-bucketeu-west-2	eu-west-2
欧州 (ミラノ)	codepipeline-cloudtrail-placeholder-bucket-eu-south-1	eu-south-1
欧州 (パリ)	codepipeline-cloudtrail-placeholder-bucketeu-west-3	eu-west-3
欧州 (スペイン)	codepipeline-cloudtrail-placeholder-bucket-eu-south-2	eu-south-2
欧州 (ストックホルム)	codepipeline-cloudtrail-placeholder-bucket-eu-north-1	eu-north-1
欧州 (チューリッヒ)*	codepipeline-cloudtrail-placeholder-bucket-eu-中央-2	eu-central-2
イスラエル (テルアビブ)	codepipeline-cloudtrail-placeholder-bucket-il-central-1	il-central-1
中東 (バーレーン)*	codepipeline-cloudtrail-placeholder-bucket-me-south-1	me-south-1
中東 (アラブ首長国連邦)	codepipeline-cloudtrail-placeholder-bucket-me-central-1	me-central-1
南米 (サンパウロ)	codepipeline-cloudtrail-placeholder-bucket-sa-east-1	sa-east-1

を使用した CodePipeline API コールのログ記録 AWS CloudTrail

AWS CodePipeline は、 のユーザー AWS CloudTrail、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 CodePipelineである と統合 AWS のサービス されています。 は、 のすべて

の API コールをイベント CodePipeline として CloudTrail キャプチャします。キャプチャされた呼び出しには、 CodePipeline コンソールからの呼び出しと CodePipeline API オペレーションへのコード呼び出しが含まれます。証跡を作成する場合は、 の CloudTrail イベントなど、Amazon S3 バケットへのイベントの継続的な配信を有効にすることができます CodePipeline。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。によって収集された情報を使用して CloudTrail、 に対するリクエスト CodePipeline、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。

の詳細については CloudTrail、「 AWS CloudTrail ユーザーガイド」を参照してください。

CodePipeline の情報 CloudTrail

CloudTrail アカウントを作成する AWS アカウント と、 で が有効になります。でアクティビティが発生すると CodePipeline、そのアクティビティは CloudTrail イベント履歴 の他の AWS のサービスイベントとともにイベントに記録されます。 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、「イベント履歴を使用した CloudTrail イベントの表示」を参照してください。

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 CodePipeline、証跡を作成します。証跡により CloudTrail 、 はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで証跡を作成すると、証跡はすべての AWS リージョンに適用されます。証跡は、 AWS パーティション内のすべてのリージョンからのイベントをログに記録し、指定した Amazon S3 バケットにログファイルを配信します。さらに、他の を設定 AWS のサービス して、CloudTrail ログで収集されたイベントデータをさらに分析し、それに基づく対応を行うことができます。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 サポートされているサービスと統合
- の Amazon SNS 通知の設定 CloudTrail
- 複数のリージョンからの CloudTrail ログファイルの受信と複数のアカウントからの CloudTrail ログファイルの受信

すべての CodePipeline アクションは によってログに記録 CloudTrail され、 CodePipeline API リファレンス に記載されています。例えば、、、および UpdatePipelineアクションを呼び出すGetPipelineExecutionとCreatePipeline、 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) 認証情報のどちらを使用して行われたか。
- 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 Element」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 Amazon S3 バケットにイベントをログファイルとして配信できるようにする設定です。 CloudTrail ログファイルには 1 つ以上のログエントリが含まれます。イベントは任意のソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストパラメータなどに関する情報が含まれます。 CloudTrail ログファイルはパブリック API コール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。

次の例は、更新パイプラインイベントの CloudTrail ログエントリを示しています。 という名前のパイプライン MyFirstPipeline は、アカウント ID が 80398EXAMPLE の JaneDoe-CodePipeline というユーザー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。ログ内の 要素`requestParameters`と`responseElements`要素の両方に CloudTrail編集されたパイプラインの構造全体が含まれているため、これらの要素は次の例で省略されています。強調 が、変更されたパイプラインの `requestParameters` 部分、以前のバージョン番号のパイプライン、`responseElements` 部分に追加されています。これは、バージョン番号が 1 ずつインクリメントすることを意味します。実際のログエントリでより多くのデータが表示された場合、編集した部分は省略記号 (...) でマークされます。

```
{
 "eventVersion":"1.03",
  "userIdentity": {
   "type":"IAMUser",
   "principalId":"AKIAI44QH8DHBEXAMPLE",
   "arn":"arn:aws:iam::80398EXAMPLE:user/JaneDoe-CodePipeline",
   "accountId":"80398EXAMPLE",
   "accessKeyId":"AKIAIOSFODNN7EXAMPLE",
   "userName":"JaneDoe-CodePipeline",
```

```
    "sessionContext": {
    "attributes":{
     "mfaAuthenticated":"false",
     "creationDate":"2015-06-17T14:44:03Z"
     }
    },
  "invokedBy":"signin.amazonaws.com"},
  "eventTime":"2015-06-17T19:12:20Z",
  "eventSource":"codepipeline.amazonaws.com",
  "eventName":"UpdatePipeline",
  "awsRegion":"us-east-2",
  "sourceIPAddress":"192.0.2.64",
  "userAgent":"signin.amazonaws.com",
  "requestParameters":{
    "pipeline":{
    "version":1,
    "roleArn":"arn:aws:iam::80398EXAMPLE:role/CodePipeline_Service_Role",
    "name":"MyFirstPipeline",
    "stages":[
      {
      "actions":[
       {
           "name":"MySourceStage",
         "actionType":{
         "owner":"AWS",
         "version":"1",
         "category":"Source",
         "provider":"S3"
      },
     "inputArtifacts":[],
     "outputArtifacts":[
      {"name":"MyApp"}
      ],
     "runOrder":1,
     "configuration":{
      "S3Bucket":"awscodepipeline-demobucket-example-date",
      "S3ObjectKey":"sampleapp_linux.zip"
       }
       }
      ],
        "name":"Source"
      },
      (...)
                },
```

```
 "responseElements":{
   "pipeline":{
     "version":2,
     (...)
           },
     "requestID":"2c4af5c9-7ce8-EXAMPLE",
     "eventID":""c53dbd42-This-Is-An-Example"",
     "eventType":"AwsApiCall",
     "recipientAccountId":"80398EXAMPLE"
     }
      ]
}
```

トラブルシューティング CodePipeline

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

- への接続時に接続エラー： GitHub「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「組織の所有者が GitHub アプリをインストールする必要があります」
- 実行モードが QUEUED または PARALLEL モードに変更されたパイプラインは、実行制限に達すると失敗します。
- PARALLEL モードのパイプラインは、QUEUED モードまたは SUPERSEDED モードに変更したときに編集された場合、古いパイプライン定義になります。
- PARALLEL モードから変更されたパイプラインには、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されない場合があります
- 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されない場合があります
- CodeCommit または PARALLEL モードでの S3 ソースリビジョンが EventBridge イベントと一致しない可能性があります
- 別の問題があるため問い合わせ先を教えてください。

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。指定されたロールには、十分なアクセス許可がありません: サービス：AmazonElasticLoadBalancing」

問題： のサービスロールに AWS Elastic Beanstalk、Elastic Load Balancing の一部のオペレーションなど、 に対する十分なアクセス許可 CodePipeline がありません。の サービスロールは、この問題に対応するために 2015 年 8 月 6 日に更新 CodePipeline されました。この日付より前にサービスロールを作成したお客様は、サービスロールのポリシーステートメントを変更して必要なアクセス権限を追加する必要があります。

解決方法: 最も簡単な解決方法は、「 CodePipeline サービスロールにアクセス許可を追加する」で記載されているようにサービスロールに対するポリシーステートメントを編集することです。

編集済みのポリシーを適用したら、パイプラインを手動で開始する のステップに従って Elastic Beanstalk を使用するパイプラインを手動で再実行します。

セキュリティのニーズに応じて、他の方法でアクセス権限を変更することもできます。

デプロイエラー: AWS Elastic Beanstalk デプロイアクションで設定されたパイプラインは、「」アクセスDescribeEvents許可がない場合に失敗するのではなくハングします

問題： のサービスロールには、 を使用するパイプラインの `"elasticbeanstalk:DescribeEvents"`アクションが含まれている CodePipeline 必要があります AWS Elastic Beanstalk。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。

解決方法： CodePipeline サービスロールを確認します。`"elasticbeanstalk:DescribeEvents"` アクションが欠落している場合は、「CodePipeline サービスロールにアクセス許可を追加する」の手順に従い、IAM コンソールで [ポリシーの編集] 機能を使用してこのアクションを追加します。

編集済みのポリシーを適用したら、パイプラインを手動で開始する のステップに従って Elastic Beanstalk を使用するパイプラインを手動で再実行します。

パイプラインエラー: ソースアクションは、 CodeCommit 「リポジトリ にアクセスできませんでした」というアクセス許可不足のメッセージを返します**repository-name**。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： のサービスロールに に対する十分なアクセス許可 CodePipeline がないため、2016 年 4 月 18 日にリポジトリの使用 CodeCommitのサポートが追加される前に作成された CodeCommit 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。

解決方法： CodePipeline のサービスロールのポリシー CodeCommit に に必要なアクセス許可を追加します。詳細については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

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、 に必要なアクセス許可を持つインスタンスロールでインスタンスが作成されていない可能性がありますCodePipeline。Jakins サーバー、オンプレミスインスタンス、または必要な IAM ロールなしで作成された Amazon EC2 インスタンスで IAM ユーザーを使用している場合、IAM ユーザーには必要な権限がないか、Jenkins サーバーがサーバー上に設定されたプロファイルを介してこれらの認証情報にアクセスできません。

修正案: Amazon EC2 インスタンスロールまたは IAM ユーザーが、`AWSCodePipelineCustomActionAccess` 管理されたポリシーまたは同等の権限で設定されていることを確認します。詳細については、「AWS の マネージドポリシー AWS CodePipeline」を参照してください。

IAM ユーザーを使用している場合は、インスタンスで設定された AWS プロファイルで、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 Jenkins UI との統合用に設定した IAM ユーザー認証情報 CodePipeline を直接提供する必要がある場合があります。これは推奨されるベストプラクティスではありません。その必要がある場合は、Jenkins サーバーが保護されており、HTTP ではなく HTTPS を使用していることを確認してください。

パイプラインエラー: 別の AWS リージョンで作成されたバケットを使用してある AWS リージョンで作成されたパイプラインは、「」というコードInternalErrorで「」を返しますJobFailed。

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 リージョンに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 に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 から 404 が見つかりませんというエラーが報告されます。

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 「 のサンプル CodeBuild」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： でパイプラインアーティファクト名を表示すると CodePipeline、名前が切り捨て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、 がジョブワーカーの一時的な認証情報 CodePipeline を生成するときに、完全な Amazon S3 パスがポリシーサイズの制限を超えないように、アーティファクト名を切り捨てます。

アーティファクト名が切り捨てられているように見えても、 は切り捨てられた名前のアーティファクトの影響を受けない方法でアーティファクトバケットに CodePipeline 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 Enterprise Server GitHub、または GitLab.com への接続の CodeBuild GitClone アクセス許可を追加する

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、ダウンロードするコード CodeBuild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管理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 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。接続 ARN を CodeBuild サービスロールポリシーに追加します。

   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
   ```

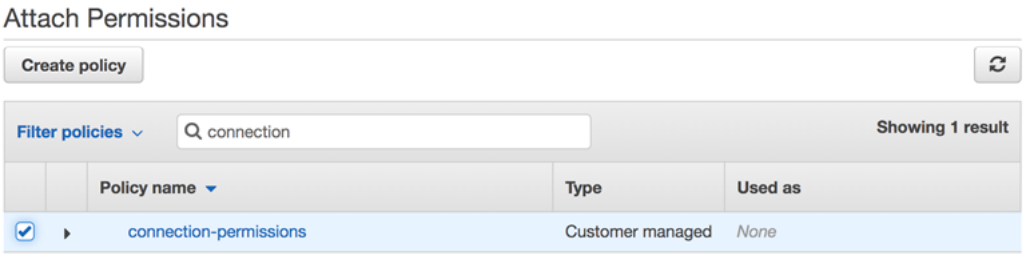
2. CodeBuild サービスロールを検索するには、パイプラインで使用されているビルドプロジェクトを開き、ビルドの詳細タブに移動します。

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。
5. [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。
6. アクセス許可をアタッチしたページに戻り、ポリシーリストを更新して、作成したポリシーを選択します。[ポリシーのアタッチ] を選択します。



ソースアクションの CodeBuild GitClone CodeCommitアクセス許可を追加する

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 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、環境が CodeBuildリポジトリからプルするアクセス許可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管理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 アクセス許可を追加するには

1. CodeBuild サービスロールを見つけるには、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、ビルドの詳細タブに移動します。
2. [サービスロール] リンクを選択します。これにより IAM コンソールが開き、リポジトリへのアクセスを許可する新しいポリシーを追加できます。
3. IAM コンソールで [ポリシーのアタッチ] を選択し、[ポリシーの作成] を選択します。
4. [JSON] タブに、以下のサンプルポリシーを貼り付けます。

```
{
    "Action": [
        "codecommit:GitPull"
    ],
    "Resource": "*",
    "Effect": "Allow"
},
```

5. [ポリシーの確認] を選択します。ポリシーの名前 (例: **codecommit-gitpull**) を入力し、[ポリシーの作成] を選択します。
6. アクセス許可をアタッチしたページに戻り、ポリシーリストを更新して、作成したポリシーを選択します。[ポリシーのアタッチ] を選択します。

パイプラインエラー: CodeDeployToECS アクションを含むデプロイは、「< ソースアーティファクト名 >」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を返します。

問題:

タスク定義ファイルは、 CodeDeploy (アクション) を介して Amazon ECS に CodePipeline デプロイするCodeDeployToECSアクションに必要なアーティファクトです。CodeDeployToECS デプロイアクションのアーティファクト ZIP の最大サイズは 3 MB です。ファイルが見つからないかアーティファクトのサイズが 3 MB を超える場合は、次のエラーメッセージが返されます。

<source artifact name> からタスク定義アーティファクトファイルを読み取ろうとしたときに例外が発生しました

解決方法: タスク定義ファイルがアーティファクトとして含まれていることを確認してください。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 バージョン 1 のソースアクション: リポジトリリストに異なるリポジトリが表示される

問題:

CodePipeline コンソールで GitHub バージョン 1 アクションの認証に成功したら、 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 組織に基づいています。認証に使用しているアカウント GitHub が、リポジトリが作成された GitHub 組織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 バージョン 2 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 を使用するため GitHub、接続を作成するには、リポジトリに対する組織所有者のアクセス許可または管理者のアクセス許可が必要です。

解決方法： GitHub リポジトリのアクセス許可レベルの詳細については、https://docs.github.com/en/free-pro-team@latest/github/setting-up-and-managing-organizations-and-teams/permission-levels-for-anorganization を参照してください。

Amazon S3 エラー： CodePipeline サービスロール <ARN> が S3 バケット <BucketName> の S3 アクセスを拒否されました

問題:

進行中、 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 CodePipeline ことを確認します。アクションにチェックするアクセス許可がない場合、Amazon S3 で`AccessDenied`エラーが発生し、次のエラーメッセージが に表示されます CodePipeline。

CodePipeline サービスロール「arn:aws:iam::*`AccountID`*:role/service-role/*`RoleID`*」が S3 バケットの S3 アクセスを拒否されました*`BucketName`*」

アクションの CloudTrail ログには、`AccessDenied`エラーも記録されます。

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は、ポリシー内のアクションのリスト`s3:ListBucket`に を追加します。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で、アーティファクトバケットポリシー とも呼ばれる場合は、 ステートメントを追加して、CodePipeline サービスロールによるアクセス`s3:ListBucket`許可の使用を許可します。

  アーティファクトバケットにポリシーを追加するには

  1. パイプラインの ARN とサービスロール ARN (コンソール) を表示します。 の手順を行い、パイプラインの [設定] ページでアーティファクトバケットを選択し、Amazon S3 コンソールに表示させます。
  2. [Permissions (アクセス許可)] を選択します。
  3. [バケットポリシー] で [編集] を選択します。

4. [ポリシー] テキストフィールドで、新しいバケットポリシーを入力するか、次の例に示すように既存のポリシーを編集します。バケットポリシーは JSON ファイルであるため、有効な JSON を入力する必要があります。

   次の例は、サービスロールのロール ID の例が *AROAEXAMPLEID* であるアーティファクトバケットのバケットポリシーステートメントを示しています。

```
{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:ListBucket",
    "Resource": "arn:aws:s3:::BucketName",
    "Condition": {
        "StringLike": {
            "aws:userid": "AROAEXAMPLEID:*"
        }
    }
}
```

   次の例は、アクセス許可が追加された後の同じバケットポリシーステートメントを示しています。

```
{
    "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、https://aws.amazon.com/blogs/security/writing-iam-policies-how-to-grant-access-to-an-amazon-s3-bucket/のステップ を参照してください。

5. [保存] を選択します。

編集済みのポリシーを適用したら、パイプラインを手動で開始する のステップに従ってパイプラインを手動で再実行します。

Amazon S3、Amazon ECR、または CodeCommitソースを持つパイプラインは自動的に起動しなくなりました

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、コンソールはソーストリガー識別子が類似していて、初期文字が同じである変更を検出しない場合があります。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。

のパラメータ名の末尾にあるマイナーな変更の例は、 CodeCommit ブランチ名を `MyTestBranch-1`に変更 CodeCommit することです`MyTestBranch-2`。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:

次のいずれかを行います。

- CodeCommit/S3/ECR 構成設定を変更して、パラメータ値の開始部分に変更を加えます。

  例: ブランチ名を `release-branch` から `2nd-release-branch` に変更します。`release-branch-2` など、名前の末尾の変更は避けてください。
- 各パイプラインの CodeCommit/S3/ECR 設定を変更します。

  例: ブランチ名を `myRepo/myBranch` から `myDeployRepo/myDeployBranch` に変更します。`myRepo/myBranch2` など、名前の末尾の変更は避けてください。
- コンソールの代わりに CLI または を使用して AWS CloudFormation 、変更検出イベントルールを作成および更新します。S3 ソースアクションのイベントルールを作成する手順については、「EventBridge を使用した Amazon S3 ソースアクション AWS CloudTrail」を参照してください。Amazon ECR アクションのイベントルールを作成する手順については、「Amazon ECR ソースアクションと EventBridge リソース」を参照してください。 CodeCommit アクションのイベン

トルールを作成する手順については、「」を参照してください CodeCommit ソースアクションと EventBridge。

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。

への接続時に接続エラー： GitHub「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「組織の所有者が GitHub アプリをインストールする必要があります」

問題:

で GitHub ソースアクションの接続を作成するには CodePipeline、組織の所有者である必要があります GitHub。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 アプリをインストールする必要があります

解決方法： 組織内のリポジトリの場合 GitHub、組織所有者は GitHub リポジトリへの接続を作成する必要があります。組織のリポジトリでない場合、ユーザーがリポジトリの所有者である必要があります。

実行モードが QUEUED または PARALLEL モードに変更されたパイプラインは、実行制限に達すると失敗します。

問題： QUEUED モードでのパイプラインの同時実行の最大数は 50 回です。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。

解決方法： 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 または PARALLEL 実行モードの詳細については、「」を参照してくださいCodePipeline の概念。

PARALLEL モードのパイプラインは、QUEUED モードまたは SUPERSEDED モードに変更したときに編集された場合、古いパイプライン定義になります。

問題： 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 に編集しても、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、SUPERSEDED モードまたは QUEUED モードでは使用されません。

解決方法： 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 に編集するときは、パイプライン定義を同時に更新しないでください。

QUEUED または PARALLEL 実行モードの詳細については、「」を参照してくださいCodePipeline の概念。

PARALLEL モードから変更されたパイプラインには、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の場合、パイプライン実行モードを QUEUED または SUPERSEDED に編集しても、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。パイプラインが PARALLEL から QUEUED または SUPERSEDED に変更された場合、SUPERSEDED モードまたは QUEUED モードのパイプラインの状態は、これらのモードのいずれかで最後に既知の状態になります。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は、状態は空になります。

解決方法： 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、実行モードの表示に PARALLEL 状態は表示されないことに注意してください。

QUEUED または PARALLEL 実行モードの詳細については、「」を参照してくださいCodePipeline の概念。

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されない場合があります

説明： ソースアクションなど、接続を使用する BitBucket ソースアクションを持つパイプラインの場合、Git 設定でトリガーを設定して、ファイルパスでフィルタリングしてパイプラインを開始できます。場合によっては、ファイルパスでフィルタリングされたトリガーを持つパイプラインの場合、ファイルパスフィルターを持つブランチが最初に作成されたときにパイプラインが開始されないことがあります。これは、変更されたファイルを CodeConnections 接続で解決できないためです。トリガーの Git 設定がファイルパスでフィルタリングするように設定されている場合、フィルターを持つブランチがソースリポジトリに作成されたばかりのときにパイプラインが開始されません。ファイルパスでのフィルタリングの詳細については、「」を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。

結果： 例えば、ブランチ「B」にファイルパスフィルター CodePipeline がある のパイプラインは、ブランチ「B」の作成時にトリガーされません。ファイルパスフィルターがない場合でも、パイプラインは開始されます。

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、ファイル制限に達したときに開始されない場合があります

説明： ソースアクションなど、接続を使用する BitBucket ソースアクションを持つパイプラインの場合、ファイルパスでフィルタリングしてパイプラインを開始できる Git 設定でトリガーを設定できます。 CodePipeline したがって、トリガーの Git 設定がファイルパスでフィルタリングするように設定されている場合、100 を超えるファイルがある場合にパイプラインが開始されないことがあります。ファイルパスでのフィルタリングの詳細については、「」を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。

結果： 例えば、差分に 150 個のファイルが含まれている場合、 CodePipeline は最初の 100 個のファイル (特定の順序なし) を調べて、指定され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に一致するファイルが、 によって取得された 100 個のファイルに含まれていない場合 CodePipeline、パイプラインは呼び出されません。

CodeCommit または PARALLEL モードでの S3 ソースリビジョンが EventBridge イベントと一致しない可能性があります

説明： PARALLEL モードで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される場合がありますが、これは EventBridge イベントの変更と同じではない可能性があります。場合によっては、パイプラインを開始するコミットまたはイメージタグの間に分割秒がある場合、 が CodePipelineイベントを受信してその実行を開始すると、別のコミットまたはイメージタグがプッシュされ CodePipeline (CodeCommit アクションなど) 、その時点で HEAD コミットのクローンが作成されます。

結果： または S3 ソースを持つ CodeCommit PARALLEL モードのパイプラインの場合、パイプラインの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 のクローンを作成します。例えば、PARALLEL モードのパイプラインの場合、コミットがプッシュされ、実行 1 のパイプラインが開始され、2 番目のパイプラインの実行では 2 番目のコミットが使用されます。

別の問題があるため問い合わせ先を教えてください。

これらの他のリソースを試してください。

- AWS Support にお問い合わせください。
- CodePipeline フォーラム で質問してください。
- クォータの引き上げをリクエストする。詳細については、「のクォータ AWS CodePipeline」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 bui lt であるデータセンターとネットワークアーキテクチャからメリットを得られます。

セキュリティは、 AWS とユーザーの間で共有される責任です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティとして説明しています。

- クラウドのセキュリティ — AWS 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。また、 は、お客様が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。に適用されるコンプライアンスプログラムの詳細については AWS CodePipeline、「コンプライアンスAWS のサービス プログラムによる対象範囲内の 」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 AWS のサービス 、使用する によって決まります。また、お客様は、 お客様のデータの機密性、企業の要件、および適用可能な法律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、 の使用時に責任共有モデルを適用する方法を理解するのに役立ちます CodePipeline。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ために CodePipeline を設定する方法を示します。また、 CodePipeline リソースのモニタリングや保護 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
- での口グ記録とモニタリング CodePipeline
- のコンプライアンス検証 AWS CodePipeline
- の耐障害性 AWS CodePipeline
- のインフラストラクチャセキュリティ AWS CodePipeline
- セキュリティ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 共有モデル、 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。お客様は、このインフラストラクチャでホストされてい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、「データプライバシーのよくある質問」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記事「AWS 責任共有モデルおよび GDPR」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 は必須であり TLS 1.3 がお勧めです。
- 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。
- 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 などの高度なマネージド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 140-2 検証済みの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) 140-2」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は、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配置しないことを強くお勧めします。これは、コンソール、API、CodePipeline または SDK を使用して AWS CLIまたは他の AWS のサービス を操作する場合も同様です。 AWS SDKs 名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断ログに使用される場合があります。外部サーバーへの URL を提供する場合は、そのサーバーへのリクエストを検証するための認証情報を URL に含めないように強くお勧めします。

以下のセキュリティのベストプラクティスも、 でのデータ保護に対処します CodePipeline。

- の Amazon S3 に保存されているアーティファクトのサーバー側の暗号化を設定する CodePipeline

- 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

インターネットトラフィックのプライバシー

Amazon VPC は AWS のサービス 、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、 を搭載した Amazon VPC エンドポイントをサポートしています。これは AWS PrivateLink、Elastic Network Interface とプライベート IP アドレス AWS のサービス 間のプライベート通信を容易にする AWS テクノロジーです。つまり、VPC のプライベートエンドポイント 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では、 への接続にインターネットアクセスが必要でした CodePipeline。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 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 CodePipeline。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。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、または VPN 接続を必要と CodePipeline せずに、信頼性が高くスケーラブルな 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

のデータは CodePipeline 、 を使用して保管時に暗号化されます 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、「の Amazon S3 に保存されているアーティファクトのサーバー側の暗号化を設定する CodePipeline」を参照してください。

転送中の暗号化

すべての service-to-service 通信は、転送中に SSL/TLS を使用して暗号化されます。

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オプションを選択した場合、 は CodePipeline を使用します AWS マネージドキー。この 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。

> **Important**
>
> CodePipeline は対称 KMS キーのみをサポートします。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。

の Amazon S3 に保存されているアーティファクトのサーバー側の暗号化を設定する CodePipeline

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ともに暗号化され、 によって管理されます AWS。
- 独自の カスタマーマネージドキー を作成して管理できます。

> **Important**
>
> CodePipeline は対称 KMS キーのみをサポートします。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。

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 暗号化または復号化する場合は、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と の詳細については AWS KMS、「サーバー側の暗号化を使用したデータの保護」および「 (SSE-KMS) に保存 AWS Key Management Service されている KMS キーを使用したサーバー側の暗号化を使用したデータの保護」を参照してください。

の詳細については AWS KMS、「 AWS Key Management Service デベロッパーガイド」を参照してください。

トピック

の表示 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、 は Amazon S3 (aws/s3キー) AWS マネージドキー の AWS KMS を使用してサーバー側の暗号化 CodePipeline を使用します。 Amazon S3 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。S3 バケットからアーティファクトを取得すると、 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。
4. パイプラインのリージョンを選択します。例えば、パイプラインが us-east-2 に作成されている場合は、フィルタが 米国西部 (オハイオ州) に設定されていることを確認します。

   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 CodePipeline、「 AWS CodePipeline エンドポイントとクォータ」を参照してください。
5. リスト内のパイプラインに使用されるエイリアスがあるキー (デフォルトは aws/s3) を選択します。キーの基本情報が表示されます。

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 CodePipeline を使用するパイプラインを に作成する」を参照してください。

暗号化のベストプラクティスでは、暗号化キーの広範な再利用を推奨していません。ベストプラクティスとして、キーを定期的にローテーションします。 AWS KMS キーの新しい暗号化マテリアルを作成するには、カスタマーマネージドキーを作成し、新しいカスタマーマネージドキーを使用するようにアプリケーションまたはエイリアスを変更します。または、既存の カスタマー管理キー の自動キーローテーションを有効にすることができます。

カスタマーマネージドキーをローテーションするには、「キーローテーション」を参照してください。

> **Important**
>
> CodePipeline は対称 KMS キーのみをサポートします。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、その他のシークレット AWS Secrets Manager をライフサイクル全体でローテーション、管理、取得することをお勧めします。Secrets Manager を使用すると、コード内のハードコードされた認証情報 (パスワードを含む) を、Secrets Manager への API コールで置き換えて、プログラムでシークレットを取得することができます。詳細については、AWS Secrets Manager ユーザーガイドの「Secrets Manager AWS とは」を参照してください。

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 AWS CloudFormation 、Secrets Manager に保存したシークレットにアクセスする動的な参照をテンプレートに含める必要があります。参照 ID パターンと例については、「Secrets Manager のシークレットのAWS CloudFormation ユーザーガイド」を参照してください。パイプラインの GitHub ウェブフックにテンプレートスニペットで動的参照を使用する例については、「Webhook リソース設定」を参照してください。

以下も参照してください。

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AWS 」を参照してください。
- 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、https://aws.amazon.com/blogs/security/how-to-create-and-retrieve-secrets-managed-in-aws-secrets-manager-using-aws-cloudformation-template/ を参照してください。

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 AWS のサービス 使用できる です。

トピック

対象者

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

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、管理者から必要な認証情報とアクセス許可が与えられます。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しておくと、管理者に適切な許可をリクエストするうえで役立ちます。の機能にアクセスできない場合は、CodePipeline「」を参照してくださいAWS CodePipeline ID とアクセスのトラブルシューティング。

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、 へのフルアクセスがあります CodePipeline。サービスユーザーがどの CodePipeline 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 IAM を で使用する方法の詳細については、 CodePipeline「」を参照してくださいが IAM と AWS CodePipeline 連携する方法。

IAM 管理者 – IAM 管理者は、 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 CodePipeline。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、「」を参照してくださいAWS CodePipeline アイデンティティベースポリシーの例。

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 の認証情報は、フェデレーティッド ID の例です。フェデレーティッドアイデンティティ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、「 ユーザーガイド」の「 へのサインイン AWS アカウント方法AWS サインイン 」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 Software Development Kit (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に暗号で署名します。 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。推奨される方法を使用

してリクエストを自分で署名する方法の詳細については、IAM ユーザーガイドの API AWS リクエストの署名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。例えば、 AWS では、多要素認証 (MFA) を使用してアカウントのセキュリティを向上させる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「Multi-factor authentication」(多要素認証) および『IAM ユーザーガイド』の「AWSにおける多要素認証 (MFA) の使用」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「ルートユーザー認証情報が必要なタスク」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、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 内のアイデンティティです。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権限を与えることができます。

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、ロールでは一時的な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 (ロールではなく) の作成が適している場合」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 内のアイデンティティです。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。ロール を切り替える AWS Management Console ことで、 で IAM ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API AWS CLI オペレーションを呼び出すか、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「IAM ロールの使用」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アイデンティティに権限を割り当てるには、ロールを作成してそのロールの権限を定義します。フェデレーティッドアイデンティティが認証されると、そのアイデンティティはロールに関連付けられ、ロールで定義されている権限が付与されます。フェデレーションの詳細については、『IAM ユーザーガイド』の「サードパーティーアイデンティティプロバイダー向けロールの作成」 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する場合、権限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。権限セットの詳細については、『AWS IAM Identity Center ユーザーガイド』の「権限セット」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、 (ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。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 ロールとリソースベースのポリシーとの相違点」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。サービスでは、呼び出し元プリンシパルの権限、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 でアクションを実行する場合 AWS、ユーザーは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 へのリクエストのリクエスト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、「転送アクセスセッション」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 のサービスにアクセス許可を委任するロールの作成」を参照してください。
- サービス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、サービスにリンクされたロールの権限を表示できますが、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。 AWS ロールを EC2 インスタンスに割り当て、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「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用して権限を付与する」を参照してください。

IAM ロールと IAM ユーザーのどちらを使用するかについては、『IAM ユーザーガイド』の「(IAM ユーザーではなく) IAM ロールをいつ作成したら良いのか?」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、 AWS ID またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられ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) AWS 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 AWS JSON 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んなリソースにどんな条件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継ぐことができます。

IAM ポリシーは、オペレーションの実行方法を問わず、アクションの権限を定義します。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、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 権限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「IAM ポリシーの作成」を参照してください。

アイデンティティベースポリシーは、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「マネージドポリシーとインラインポリシーの比較」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、一般的ではない追加のポリシータイプをサポートします。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のポリシーによって 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。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) に対する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、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するサービスです。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「SCP の仕組み」を参照してください。
- 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 へのアクセスを管理する前に CodePipeline、 で使用できる IAM 機能を理解しておく必要があります CodePipeline。 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AWS のサービス「IAM と連携する 」を参照してください。

トピック

- CodePipeline アイデンティティベースのポリシー
- CodePipeline リソースベースのポリシー
- CodePipeline タグに基づく認可
- 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、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 AWS JSON 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んなリソースにどんな条件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権限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権限を付与するポリシーで使用されます。

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codepipeline:。

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 element. CodePipeline defines のいずれかを含める必要があります。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。

単一ステートメントに複数アクションを指定するには、次のようにカンマで区切ります:

```
"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については、「IAM ユーザーガイド」の「 で定義されるアクション AWS CodePipeline」を参照してください。

リソース

管理者は AWS JSON 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どのような条件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、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの権限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 リソースとオペレーション

では CodePipeline、プライマリリソースはパイプラインです。ポリシーでは、Amazon リソースネーム (ARN) を使用して、ポリシーが適用されるリソースを識別します。 は、ステージ、アクション、カスタムアクションなど、プライマリリソースで使用できる他のリソース CodePipeline をサポートします。これらは サブリソースと呼ばれます。これらのリソースとサブリソースには、一意の Amazon リソースネーム (ARN) が関連付けられています。ARNs「」の「Amazon リソースネー

ム (ARN) と AWS のサービス 名前空間」を参照してくださいAmazon Web Services 全般のリファレンス。パイプラインに関連付けられたパイプラインの ARN を取得するには、「設定コンソールに表示されるパイプラインの ARN 」を参照してください。詳細については、「パイプラインの ARN とサービスロール ARN (コンソール) を表示します。」を参照してください。

リソースタイプ	ARN 形式
パイプライン	arn:aws:コードパイプライン:[##]:[#####]:[#######]
ステージ	arn:aws:コードパイプライン:[##]:[#####]:[#######/#####]
アクション	arn:aws:codepipeline:*region*:*account*:*pipeline-name* /*stage-name* /*action-name*
カスタムアクション	arn:aws:codepipeline:*region*:*account*:actionty pe:*owner*/*category*/*provider*/*version*
すべての CodePipeline リソース	arn:aws:codepipeline:*
指定されたリージョン内の指定されたアカウントが所有するすべての CodePipeline 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、イベントパターンとルールでは完全一致 CodePipeline を使用します。イベントパターンの作成時に正しい ARN 文字を使用して、一致させるリソース内の ARN 構文とそれらの文字が一致する必要があります。

には CodePipeline、リソースレベルのアクセス許可をサポートする API コールがあります。リソースレベルのアクセス許可は、API コールでリソース ARN を指定できるかどうか、または API コールでワイルドカードを使用したすべてのリソースの呼び出しのみが可能かどうかを示します。リソースレベルのアクセス許可の詳細な説明と、リソースレベルのアクセス許可をサポートする

CodePipeline API コールのリストCodePipeline アクセス許可リファレンスについては、「」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 (など) を受け入れますGetPipeline。単一のステートメントに複数のリソースを指定するには、以下のようにコンマで ARN を区切ります。

```
"Resource": ["arn1", "arn2"]
```

CodePipeline は、 CodePipeline リソースを操作するための一連のオペレーションを提供します。使用可能なオペレーションのリストについては、「CodePipeline アクセス許可リファレンス」を参照してください。

条件キー

管理者は AWS JSON 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んなリソースにどんな条件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 条件演算子 を使用して条件式を作成することで、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 要素を指定するか、1 つの Condition 要素に複数のキーを指定すると、 AWS 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「IAM ポリシーの要素: 変数およびタグ」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ます。すべての AWS グローバル条件キーを確認するには、「IAM ユーザーガイド」のAWS 「 グローバル条件コンテキストキー」を参照してください。

CodePipeline は独自の条件キーのセットを定義し、一部のグローバル条件キーの使用もサポートします。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AWS 」の「グローバル条件コンテキストキー」を参照してください。

すべての Amazon EC2 アクションは、aws:RequestedRegion および ec2:Region 条件キーをサポートします。詳細については、「例: 特定のリージョンへのアクセスの制限」を参照してください。

CodePipeline 条件キーのリストを確認するには、「IAM ユーザーガイド」の「 の条件キー AWS CodePipeline」を参照してください。条件キーを使用できるアクションとリソースについては、「 で定義されるアクション AWS CodePipeline」を参照してください。

例

CodePipeline アイデンティティベースのポリシーの例を表示するには、「」を参照してくださいAWS CodePipeline アイデンティティベースポリシーの例。

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をサポートしていません。ただし、 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 CodePipeline 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には、「」、「」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。

CodePipeline タグに基づく認可

CodePipeline リソースにタグをアタッチしたり、 へのリクエストでタグを渡すことができます CodePipeline。タグに基づいてアクセスを管理するには、codepipeline:ResourceTag/*key-name*、aws:RequestTag/*key-name*、または aws:TagKeys の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。 CodePipeline リソースのタグ付けの詳細については、「」を参照してくださいリソースのタグ付け。

リソースのタグに基づいてリソースへのアクセスを制限するためのアイデンティティベースポリシーの例を表示するには、「タグを使用して リソースへのアクセス CodePipelineを制御する」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 アカウント内のエンティティです。

での一時的な認証情報の使用 CodePipeline

一時的な認証情報を使用して、フェデレーションでサインインする、IAM ロールを引き受ける、またはクロスアカウントロール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 などの AWS STS API オペレーションを呼び出しますGetFederationToken。

CodePipeline では、一時的な認証情報の使用がサポートされています。

サービスロール

CodePipeline は、サービスがユーザーに代わってサービスロールを引き受けることを許可します。このロールにより、サービスがお客様に代わって他のサービスのリソースにアクセスし、アクションを完了することが許可されます。サービスロールは、IAM アカウントに表示され、アカウントによって所有されます。つまり、IAM 管理者は、このロール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 はサービスロールをサポートします。

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 ユーザーとロールには 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ポリシーを作成する方法については、「IAM ユーザーガイド」の「JSON タブでのポリシーの作成」を参照してください。

別のアカウントのリソースを使用するパイプラインを作成する方法、および関連するポリシーの例については、「」を参照してください別の AWS アカウントのリソース CodePipeline を使用するパイプラインを に作成する。

トピック

- ポリシーの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
- タグを使用して リソースへのアクセス CodePipelineを制御する
- CodePipeline コンソールの使用に必要な許可
- AWS の マネージドポリシー AWS CodePipeline
- カスタマーマネージドポリシーの例

ポリシーの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で誰かが CodePipeline リソースを作成、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を付与するには、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して、アクセス許可をさらに減らすことをお勧めします。詳細については、IAM ユーザーガイドの「AWS マネージドポリシー」または「AWS ジョブ機能の管理ポリシー」を参照してください。

- 最小特権を適用する – IAM ポリシーで権限を設定するときは、タスクの実行に必要な権限のみを付与します。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、最小特権権限とも呼ばれています。IAM を使用して権限を適用する方法の詳細については、『IAM ユーザーガイド』の「IAM でのポリシーと権限」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、 などの特定の を介してサービスアクションが使用される場合に 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 policy elements: Condition] (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「IAM Access Analyzer ポリシーの検証」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、セキュリティを強化するために MFA を有効にします。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 保護 API アクセスの設定」を参照してください。

IAM でのベストプラクティスの詳細については、『IAM ユーザーガイド』の「IAM でのセキュリティのベストプラクティス」を参照してください。

コンソールでのリソースの表示

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ための アクセス`ListRepositories`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、次のオペレーションを実行できます。

- アカウントのユーザーまたはグループに許可ポリシーをアタッチする – CodePipeline コンソールでパイプラインを表示する許可をユーザーに付与するには、ユーザーが属するユーザーまたはグループに許可ポリシーをアタッチできます。
- アクセス権限ポリシーをロールにアタッチする (クロスアカウントの許可を付与) - ID ベースのアクセス権限ポリシーを IAM ロールにアタッチして、クロスアカウントの権限を付与することができます。例えば、アカウント A の管理者は、次のように別の AWS アカウント (アカウント B など) または にクロスアカウントアクセス許可を付与するロールを作成できます AWS のサービス 。
  1. アカウント A の管理者は、IAM ロールを作成して、アカウント A のリソースに許可を付与するロールに許可ポリシーをアタッチします。
  2. アカウント A の管理者は、アカウント B をそのロールを引き受けるプリンシパルとして識別するロールに、信頼ポリシーをアタッチします。
  3. アカウント B の管理者は、アカウント B の任意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 のユーザーはアカウント A のリソースを作成またはアクセスできます。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 AWS のサービス プリンシパルにすることができます。

  IAM を使用した許可の委任の詳細については、「IAM ユーザーガイド」の「アクセス管理」を参照してください。

以下に示しているアクセス許可ポリシーの例では、us-west-2 region で MyFirstPipeline という名前のパイプライン内のすべてのステージ間の移行を無効または有効にするアクセス許可を付与しています。

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

次の例は、111222333444 アカウントのポリシーを示しています。このポリシーでは、コンソールで という名前のパイプラインを表示することはできますが、変更することはできませんMyFirstPipeline CodePipeline 。このポリシーは、AWSCodePipeline_ReadOnlyAccess マネージドポリシーに基づいていますが、パイプライン MyFirstPipeline に固有であるため、このマネージドポリシーを直接使用することはできません。ポリシーを特定のパイプラインに制限しない場合は、 によって作成および保守されている管理ポリシーのいずれかを使用することを検討してください CodePipeline。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": [
    {
      "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、*111111111111* AWS アカウントで作成されたポリシーで、ユーザーが 111222333444 アカウント*CrossAccountPipelineViewers*で という名前の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",
            "Action": "sts:AssumeRole",
```

```
            "Resource": "arn:aws:iam::111222333444:role/CrossAccountPipelineViewers"
        }
    ]
}
```

アカウントのユーザーがアクセスを許可される通話とリソースを制限する IAM ポリシーを作成し、管理者ユーザーにそれらのポリシーをアタッチできます。IAM ロールの作成方法と の IAM ポリシーステートメントの例については CodePipeline、「」を参照してくださいカスタマーマネージドポリシーの例。

タグを使用して リソースへのアクセス CodePipelineを制御する

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については、「」を参照してくださいリソースのタグ付け。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するか、タグ付けをサポートするサービスへのリクエストに渡すことができます。では CodePipeline、リソースにタグを付けることができ、一部のアクションにタグを含めることができます。IAM ポリシーを作成するときに、タグ条件キーを使用して以下をコントロールできます。

- どのユーザーがパイプラインリソースに対してアクションを実行できるか (リソースに既に付けられているタグに基づいて)。
- どのタグをアクションのリクエストで渡すことができるか。
- リクエストで特定のタグキーを使用できるかどうか。

文字列条件演算子では、キーと文字列値の比較に基づいてアクセスを制限する Condition 要素を構築できます。Null 条件以外の条件演算子名の末尾に IfExists を追加できます。「ポリシーキーがリクエストのコンテキストで存在する場合、ポリシーで指定されたとおりにキーを処理します。キーが存在しない場合、条件要素は true と評価されます。」 例えば、StringEqualsIfExists を使用

して、他のタイプのリソースには存在しない可能性のある条件キーに基づいた制限を行うことができます。

タグ条件キーの完全な構文と意味については、「タグを使用したアクセスコントロール」を参照してください。条件キーの詳細については、以下のリソースを参照してください。このセクションの CodePipelineポリシー例は、条件キーに関する以下の情報と一致し、リソースのネスト CodePipeline などの のニュアンスの例で拡張されています。

- 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
- の条件キー CodePipeline

次の例は、 CodePipeline ユーザー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、任意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、任意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

次のポリシーは、 で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 CodePipeline。

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。管理ポリシーは、任意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、任意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 コンソールの使用に必要な許可

CodePipeline コンソール CodePipeline で を使用するには、次のサービスからの最小限のアクセス許可のセットが必要です。

- AWS Identity and Access Management
- Amazon Simple Storage Service

これらのアクセス許可により、アカウントの他の AWS リソースを記述できます AWS 。

他のサービスをパイプラインに取り入れた場合は、次のアクセス許可が 1 つ以上必要になることがあります。

- AWS CodeCommit
- 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。これらのユーザーが CodePipeline 引き続きコンソールを使用できるようにするには、「」で説明されているように、 AWSCodePipeline_ReadOnlyAccess管理ポリシーもユーザーにアタッチしますAWS の マネージドポリシー AWS CodePipeline。

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。

AWS の マネージドポリシー 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、多くの一般的なユースケースにアクセス許可を付与するように設計されているため、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。ユースケース別にカスタマーマネージドポリシーを定義して、マネージドポリシー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) が更新されます。 は、新しい AWS のサービス が起動されるか、既存のサービスで新しい API AWS オペレーションが使用可能になると、 AWS 管理ポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS マネージドポリシー」を参照してください。

> **Important**
>
> AWS マネージドポリシー AWSCodePipelineFullAccess および AWSCodePipelineReadOnlyAccess は置き換えられました。AWSCodePipeline_FullAccess および AWSCodePipeline_ReadOnlyAccess ポリシーを使用してください。

AWS マネージドポリシー: **AWSCodePipeline_FullAccess**

これは、 へのフルアクセスを許可するポリシーです CodePipeline。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、「」を参照してくださいAWSCodePipeline_FullAccess。

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` – にアクセス許可を付与します CodePipeline。
- `chatbot` – 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 Chatbot。
- `cloudformation` – プリンシパルが で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` – プリンシパルが でログ記録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 CloudTrail。
- `codebuild` – プリンシパルが のビルド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 CodeBuild。
- `codecommit` – プリンシパルが のソース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 CodeCommit。
- `codedeploy` – プリンシパルが のデプロイ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 CodeDeploy。
- `codestar-notifications` – プリンシパルが AWS CodeStar 通知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` – にデプロイして Amazon EC2 のエラスティックロードバランシング CodeCatalyst を管理できるようにするアクセス許可を付与します。
- `ecr` - Amazon ECR でリソースを利用するためのアクセス許可を付与します。
- `elasticbeanstalk` - プリンシパルに、Elastic Beanstalk でリソースを利用するためのアクセス許可を付与します。
- `iam` - プリンシパルに、IAM でロールとポリシーを管理するためのアクセス許可を付与します。
- `lambda` - プリンシパルに、Lambda でリソースを管理するためのアクセス許可を付与します。
- `events` – プリンシパルが CloudWatch イベントで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。

- opsworks – プリンシパルが のリソースを管理できるようにするアクセス許可を付与します AWS OpsWorks。
- s3 - プリンシパルに、Amazon S3 でリソースを管理するためのアクセス許可を付与します。
- sns - プリンシパルに、Amazon SNS で通知リソースを管理するためのアクセス許可を付与します。
- states – プリンシパルが でステートマシンを表示できるようにするアクセス許可を付与します AWS Step Functions。ステートマシンは、状態の集まりで構成され、それぞれの状態がタスクを管理し、状態間の遷移を制御します。

```
{
    "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",
                "ec2:DescribeSecurityGroups",
                "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、 への読み取り専用アクセスを許可するポリシーです CodePipeline。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、「」を参照してくださいAWSCodePipeline_ReadOnlyAccess。

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 – のアクションにアクセス許可を付与します CodePipeline。
- codestar-notifications – プリンシパルが AWS CodeStar 通知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": [
                "codestar-notifications:DescribeNotificationRule"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "codestar-notifications:NotificationsForResource":
 "arn:aws:codepipeline:*"
                }
            }
        }
    ],
    "Version": "2012-10-17"
}
```

AWS マネージドポリシー: **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、「」を参照してくださいAWSCodePipelineApproverAccess。

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 – のアクションにアクセス許可を付与します CodePipeline。

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

```
        {
            "Action": [
                "codepipeline:GetPipeline",
                "codepipeline:GetPipelineState",
                "codepipeline:GetPipelineExecution",
                "codepipeline:ListPipelineExecutions",
                "codepipeline:ListPipelines",
                "codepipeline:PutApprovalResult"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

AWS マネージドポリシー： AWSCodePipelineCustomActionAccess

これは、ビルドアクションまたはテストアクションのために で CodePipelineカスタムアクションを作成するか、Jenkins リソースを統合するアクセス許可を に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、「」を参照してくださいAWSCodePipelineCustomActionAccess。

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` – のアクションにアクセス許可を付与します CodePipeline。

```
{
    "Statement": [
        {
            "Action": [
                "codepipeline:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
```

```
            "Effect": "Allow",
            "Resource": "*"
        }
    ],
    "Version": "2012-10-17"
}
```

CodePipeline マネージドポリシーと通知

CodePipeline は、パイプラインの重要な変更をユーザーに通知できる通知をサポートしています。の管理ポリシーには、通知機能のポリシーステートメント CodePipeline 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、 CodePipeline および関連サービス CodeCommit、 CodeBuild CodeDeploy、AWS CodeStar 通知のアクセス許可を付与します。このポリシーは、Amazon S3、Elastic Beanstalk、Amazon CloudTrail Amazon EC2、 など、パイプラインと統合する他の のサービスを操作するために必要なアクセス許可も付与します AWS CloudFormation。これらのマネージドポリシーのいずれかが適用されたユーザーは、通知の Amazon SNS トピックの作成と管理、トピックに対するユーザーのサブスクライブとサブスクライブ解除、通知ルールのターゲットとして選択するトピックの一覧表示、Slack 用に設定された AWS Chatbot クライアントの一覧表示を行うこともできます。

AWSCodePipeline_FullAccess マネージドポリシーには、通知へのフルアクセスを許可する次のステートメントが含まれています。

```
    {
        "Sid": "CodeStarNotificationsReadWriteAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:CreateNotificationRule",
            "codestar-notifications:DescribeNotificationRule",
            "codestar-notifications:UpdateNotificationRule",
            "codestar-notifications:DeleteNotificationRule",
            "codestar-notifications:Subscribe",
            "codestar-notifications:Unsubscribe"
        ],
        "Resource": "*",
        "Condition" : {
            "StringLike" : {"codestar-notifications:NotificationsForResource" :
 "arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
```

```
        }
    },
    {
        "Sid": "CodeStarNotificationsListAccess",
        "Effect": "Allow",
        "Action": [
            "codestar-notifications:ListNotificationRules",
            "codestar-notifications:ListTargets",
            "codestar-notifications:ListTagsforResource",
            "codestar-notifications:ListEventTypes"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsSNSTopicCreateAccess",
        "Effect": "Allow",
        "Action": [
            "sns:CreateTopic",
            "sns:SetTopicAttributes"
        ],
        "Resource": "arn:aws:sns:*:*:codestar-notifications*"
    },
    {
        "Sid": "SNSTopicListAccess",
        "Effect": "Allow",
        "Action": [
            "sns:ListTopics"
        ],
        "Resource": "*"
    },
    {
        "Sid": "CodeStarNotificationsChatbotAccess",
        "Effect": "Allow",
        "Action": [
            "chatbot:DescribeSlackChannelConfigurations",
            "chatbot:ListMicrosoftTeamsChannelConfigurations"
          ],
       "Resource": "*"
    }
```

読み取り専用マネージドポリシーの通知に関連するアクセス許可

AWSCodePipeline_ReadOnlyAccess マネージドポリシーには、通知への読み取り専用アクセスを許可する以下のステートメントが含まれています。このポリシーを適用したユーザーは、リソースの通知を表示できますが、リソースを作成、管理、サブスクライブすることはできません。

```
  {
       "Sid": "CodeStarNotificationsPowerUserAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:DescribeNotificationRule"
       ],
       "Resource": "*",
       "Condition" : {
           "StringLike" : {"codestar-notifications:NotificationsForResource" :
"arn:aws:codepipeline:us-west-2:111222333444:MyFirstPipeline"}
       }
   },
   {
       "Sid": "CodeStarNotificationsListAccess",
       "Effect": "Allow",
       "Action": [
           "codestar-notifications:ListNotificationRules",
           "codestar-notifications:ListEventTypes",
           "codestar-notifications:ListTargets"
       ],
       "Resource": "*"
   }
```

IAM と通知の詳細については、「AWS CodeStar Notifications の Identity and Access Management」を参照してください。

AWS 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た CodePipeline 以降の 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については、 CodePipeline ドキュメント履歴ページの RSS フィードをサブスクライブしてください。

変更	説明	日付
AWSCodePipeline_FullAccess – 既存のポリシーの更新	CodePipeline は、 で をサポートするアクセス許可をこのポリシーListStacks に追加しました AWS CloudFormation。	2024 年 3 月 15 日
AWSCodePipeline_FullAccess – 既存のポリシーの更新	このポリシーが更新され、 のアクセス許可が追加されました AWS Chatbot。詳細については、「CodePipeline マネージドポリシーと通知」を参照してください。	2023 年 6 月 21 日
AWSCodePipeline_FullAccess および AWSCodePipeline_ReadOnlyAccess管理ポリシー – 既存のポリシーの更新	CodePipeline は、、 を使用した追加の通知タイプをサポートするために AWS Chatbot、これらのポリシーにアクセス許可を追加しましたchatbot:ListMicrosoftTeamsChannelConfigurations 。	2023 年 5 月 16 日
AWSCodePipelineFullAccess – 非推奨	このポリシーは AWSCodePipeline_FullAccess に置き換えられました。 2022 年 11 月 17 日以降、このポリシーは新しいユーザー、グループ、またはロールにアタッチできなくなりました。詳細については、「AWS の マネージドポリシー AWS CodePipeline」を参照してください。	2022 年 11 月 17 日

変更	説明	日付
AWSCodePipelineReadOnlyAccess – 非推奨	このポリシーは `AWSCodePipeline_ReadOnlyAccess` に置き換えられました。 2022 年 11 月 17 日以降、このポリシーは新しいユーザー、グループ、またはロールにアタッチできなくなりました。詳細については、「AWS の マネージドポリシー AWS CodePipeline」を参照してください。	2022 年 11 月 17 日
CodePipeline が変更の追跡を開始しました	CodePipeline が AWS マネージド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許可を付与するユーザーポリシーの例を示します。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、「 CodePipeline コンソールの使用に必要な許可」を参照してください。

> **Note**
>
> 各例は全て、米国西部 (オレゴン) リージョン (`us-west-2`) を使用し、架空のアカウント ID を使用しています。

例

- 例 1: パイプラインの状態を取得するアクセス許可を付与する
- 例 2: ステージ間の移行を有効または無効にするアクセス許可を付与する

- 例 3: 使用可能なすべてのアクションタイプのリストを取得するアクセス許可を付与する
- 例 4: 手動の承認アクションを承認または拒否するアクセス許可を付与する
- 例 5: カスタムアクションのジョブをポーリングするアクセス許可を付与する
- 例 6: Jenkins と の統合に関するポリシーをアタッチまたは編集する AWS CodePipeline
- 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/*"
```

```
        }
    ]
}
```

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 の統合に関するポリシーをアタッチまたは編集する AWS CodePipeline

Jenkins をビルドまたはテストに使用するようにパイプラインを設定する場合は、その統合用に別の ID を作成し、Jenkins と の統合に必要な最小限のアクセス許可を持つ IAM ポリシーをアタッチします CodePipeline。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:AcknowledgeJob",
                "codepipeline:GetJobDetails",
                "codepipeline:PollForJobs",
                "codepipeline:PutJobFailureResult",
                "codepipeline:PutJobSuccessResult"
            ],
            "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、「ウォークスルー: ロールを使用したクロスアカウントアクセス」を参照してください。

次の例は、MyFirstPipeline CodePipeline 80398EXAMPLE アカウントのポリシーを示しています。このポリシーでは、コンソールで という名前のパイプラインを表示することはできますが、変更することはできません。このポリシーは、AWSCodePipeline_ReadOnlyAccess マネージドポリシーに基づいていますが、パイプライン MyFirstPipeline に固有であるため、このマネージドポリシーを直接使用することはできません。ポリシーを特定のパイプラインに制限しない場合は、 によって作成および保守されている管理ポリシーのいずれかを使用することを検討してください CodePipeline。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",
```

```
                "codepipeline:ListPipelines",
                "iam:ListRoles",
                "s3:GetBucketPolicy",
                "s3:GetObject",
                "s3:ListAllMyBuckets",
                "s3:ListBucket",
                "codedeploy:GetApplication",
                "codedeploy:GetDeploymentGroup",
                "codedeploy:ListApplications",
                "codedeploy:ListDeploymentGroups",
                "elasticbeanstalk:DescribeApplications",
                "elasticbeanstalk:DescribeEnvironments",
                "lambda:GetFunctionConfiguration",
                "lambda:ListFunctions"
            ],
            "Resource": "arn:aws:codepipeline:us-east-2:80398EXAMPLE:MyFirstPipeline"
        }
    ],
    "Version": "2012-10-17"
}
```

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、*111111111111* 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::111111111111:root"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

次の例は、*111111111111* AWS アカウントで作成されたポリシーで、ユーザーが 80398EXAMPLE アカウントCrossAccountPipelineViewersで という名前の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",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::80398EXAMPLE:role/CrossAccountPipelineViewers"
        }
    ]
}
```

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。詳細と step-by-step 例については、別の AWS アカウントのリソース CodePipeline を使用するパイプラインを に作成する「」および「」を参照してくださいの Amazon S3 に保存されているアーティファクトのサーバー側の暗号化を設定する CodePipeline。

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。

```
{
    "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
                }
            }
        },
        {
            "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 () のサービスロールに適用する必要がありますCodePipeline_Service_Role。IAM のロールにポリシーを適用する方法について

は、「ロールの修正」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/*"
        ]
    }
}
```

次の例は、AccountB によって設定され、 の EC2 インスタンスロールに適用されるポリシーを示しています CodeDeploy。このポリシーは、AccountA がパイプラインアーティファクト (-) を保存するために使用する S3 バケットへのアクセスを許可します。*codepipeline-us-east2-1234567890*

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

次の例は、 が AccountA で作成され、AccountB AWS KMS がそのキーを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 ***arn:aws:kms:us-east-1:012ID_ACCOUNT_A:key/2222222-3333333-4444-556677EXAMPLE***である のポリシーを示しています。

```
{
    "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/2222222-3333333-4444-556677EXAMPLE"
            ]
        }
    ]
}
```

次の例は、AccountA のパイプラインで必要なアクションへのアクセスを許可する 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。CodeDeploy AccountA

```
{
    "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、「ステップ 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 のアーティファクトストアとして使用する S3 バケットのポリシーを作成するには CodePipeline

のアーティファクトストアとして、バージョニングされた任意の S3 バケットを使用できます CodePipeline。[パイプラインの作成] ウィザードを使用して最初のパイプラインを作成した場合、この S3 バケットは、アーティファクトストアにアップロードされているすべてのオブジェクトが暗

号化されており、バケットへの接続が安全であることを保証するために作成されます。ベストプラクティスとして、独自の S3 バケットを作成する場合は、以下のポリシーまたはその要素をバケットに追加することを検討してください。このポリシーでは、S3 バケットの ARN は codepipeline-us-east-2-1234567890 です。この ARN を S3 バケットの ARN に置き換えます。

```
{
    "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、 および IAM の使用時に発生する可能性がある一般的な問題の診断 CodePipeline と修正に役立ちます。

トピック

- でアクションを実行する権限がない CodePipeline
- iam を実行する権限がありません。PassRole
- 管理者として他のユーザーにアクセスを許可したい CodePipeline
- 自分の AWS アカウント以外のユーザーに自分の CodePipelineリソースへのアクセスを許可したい

でアクションを実行する権限がない CodePipeline

から、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 リソースへのアクセスが許可されるように、管理者にポリシーの更新を依頼します。

iam を実行する権限がありません。PassRole

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。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 CodePipeline。

一部の AWS のサービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次の例のエラーは、 という IAM ユーザーがコンソールを使用して marymajor でアクションを実行しようする場合に発生します CodePipeline。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。

管理者として他のユーザーにアクセスを許可したい CodePipeline

他のユーザーが にアクセスできるようにするには CodePipeline、アクセスを必要とする人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。ユーザーまたはアプリケーションは、そのエンティティの認証情報を使用して AWSにアクセスします。次に、 の適切なアクセス許可を付与するポリシーをエンティティにアタッチする必要があります CodePipeline。

すぐにスタートするには、「IAM ユーザーガイド」の「IAM が委任した初期のユーザーおよびグループの作成」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自分の 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:

- がこれらの機能 CodePipeline をサポートしているかどうかを確認するには、「」を参照してくださいが IAM と AWS CodePipeline 連携する方法。
- 所有 AWS アカウント している のリソースへのアクセスを提供する方法については、IAM ユーザーガイドの「所有 AWS アカウント している別の の IAM ユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティー に提供する方法については AWS アカウント、IAM ユーザーガイドの「サードパーティー AWS アカウント が所有する へのアクセスを提供する」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、『IAM ユーザーガイド』の「外部で認証されたユーザー (ID フェデレーション) へのアクセス権限」を参照してください。

- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いの詳細については、「IAM ユーザーガイド」の「IAM ロールとリソースベースのポリシーとの相違点」を参照してください。

CodePipeline アクセス許可リファレンス

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 オペレーションと、アクションを実行するためのアクセス許可を付与できる対応するアクションが一覧表示されています。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 オペレーションとアクションに必要なアクセス許可

AcknowledgeJob

アクション: `codepipeline:AcknowledgeJob`

特定のジョブに関する情報や、そのジョブがジョブワーカーで受け取られたかどうかについて表示するために必要です。カスタムアクションにのみ使用されます。

リソース: ポリシーの `Resource` 要素ではワイルドカード (*) のみがサポートされます。

AcknowledgeThirdPartyJob

アクション: codepipeline:AcknowledgeThirdPartyJob

ジョブワーカーが特定のジョブを受け取ったことを確認するために必要です。パートナーアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

CreateCustomActionType

アクション: codepipeline:CreateCustomActionType

AWS アカウントに関連付けられたすべてのパイプラインで使用できる新しいカスタムアクションを作成するために必要です。カスタムアクションにのみ使用されます。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

CreatePipeline

アクション: codepipeline:CreatePipeline

パイプラインを作成するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

DeleteCustomActionType

アクション: codepipeline:DeleteCustomActionType

カスタムアクションを削除済みとしてマークするために必要です。カスタムアクションの PollForJobs は、アクションが削除対象としてマークされた後は失敗します。カスタムアクションにのみ使用されます。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

DeletePipeline

アクション: codepipeline:DeletePipeline

パイプラインを削除するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

DeleteWebhook

アクション: codepipeline:DeleteWebhook

Webhook を削除するために必要です。

リソース:

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

DeregisterWebhookWithThirdParty

アクション: codepipeline:DeregisterWebhookWithThirdParty

ウェブフックを削除する前に、 は、 によって作成されたウェブフック CodePipeline と、検出されるイベントを含む外部ツール間の接続を削除する必要があります。現在、 のアクションタイプ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 GitHub。

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ているすべての CodePipeline アクションタイプの概要を生成するために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

ListPipelineExecutions

アクション: codepipeline:ListPipelineExecutions

パイプラインの最新の実行の概要を生成するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ListPipelines

アクション: codepipeline:ListPipelines

アカウントに関連付けられたすべてのパイプラインの概要を生成するために必要です。

リソース:

ワイルドカードを使用したパイプライン ARN (パイプライン名レベルのリソースレベルの許可はサポートされていません)

arn:aws:codepipeline:*region*:*account*:*

ListTagsForResource

アクション: codepipeline:ListTagsForResource

指定したリソースのタグを一覧表示するために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

ListWebhooks

アクション: codepipeline:ListWebhooks

そのリージョンのアカウントの全ウェブフックの一覧表示に必要です。

リソース:

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

PollForJobs

アクション:codepipeline:PollForJobs

がアクションを実行 CodePipeline 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

ソースへの新しいリビジョン CodePipeline に関する情報を に報告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

への手動承認リクエストへの応答を報告するために必要です CodePipeline。有効な応答は Approved および Rejectedです。

リソース:

アクション

arn:aws:codepipeline:*region*:*account*:*pipeline-name*/*stage-name*/*action-name*

> **Note**
>
> この API コールはリソースレベルのアクセス権限をサポートします。ただし、IAM コンソールまたは Policy Generator を使用して、リソース ARN を指定するポリシーを

"codepipeline:PutApprovalResult" で作成すると、エラーが発生する場合があります。エラーが発生した場合は IAM コンソールの [JSON] タブまたは CLI を使用してポリシーを作成することができます。

PutJobFailureResult

アクション: codepipeline:PutJobFailureResult

ジョブワーカーによってパイプラインに返されたジョブの失敗をレポートするために必要です。カスタムアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutJobSuccessResult

アクション: codepipeline:PutJobSuccessResult

ジョブワーカーによってパイプラインに返されたジョブの成功をレポートするために必要です。カスタムアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutThirdPartyJobFailureResult

アクション: codepipeline:PutThirdPartyJobFailureResult

ジョブワーカーによってパイプラインに返されるサードパーティージョブの失敗をレポートするために必要です。パートナーアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutThirdPartyJobSuccessResult

アクション: codepipeline:PutThirdPartyJobSuccessResult

ジョブワーカーによってパイプラインに返されるサードパーティージョブの成功をレポートするために必要です。パートナーアクションにのみ使用されます。

リソース: ポリシーの Resource 要素ではワイルドカード (*) のみがサポートされます。

PutWebhook

アクション: codepipeline:PutWebhook

ウェブフックを作成するために必要です。

リソース:

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

RegisterWebhookWithThirdParty

アクション: codepipeline:RegisterWebhookWithThirdParty

リソース:

ウェブフックの作成後、ウェブフック URL を生成するために、サポートされるサードパーティーを設定することが必要です。

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

RetryStageExecution

アクション: codepipeline:RetryStageExecution

ステージで最後に失敗したアクションを再試行することでパイプラインの実行を再開するために必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

StartPipelineExecution

アクション: codepipeline:StartPipelineExecution

指定されたパイプラインを開始するため (具体的には、パイプラインの一部として指定されたソース場所への最新のコミットの処理を開始するため) に必要です。

リソース:

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

TagResource

アクション: codepipeline:TagResource

指定されたリソースにタグを付けるために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

UntagResource

アクション: codepipeline:UntagResource

指定されたリソースにタグを付けるために必要です。

リソース:

アクションの種類

arn:aws:codepipeline:*region*:*account*:actiontype:*owner*/*category*/*provider*/*versio*

パイプライン

arn:aws:codepipeline:*region*:*account*:*pipeline-name*

ウェブフック

arn:aws:codepipeline:*region*:*account*:webhook:*webhook-name*

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。のサービスロールを変更する前に、 CodePipeline 必ず影響を理解してください。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、パイプラインに が含まれていない場合は CodeDeploy、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

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ポリシーステートメントにまだ含まれ AWS のサービス ていない のアクセス許可でサービスロールポリシーステートメントを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 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 サポート日
AWS CloudFormation StackSets アクション	2020 年 12 月 30 日
CodeCommit フルクローン出力アーティファクト形式	2020 年 11 月 11 日
CodeBuild バッチビルド	2020 年 7 月 30 日
AWS AppConfig	2020年6月22日
AWS Step Functions	2020 年 5 月 27 日
AWS CodeStar 接続	2019 年 12 月 18 日
`CodeDeployToECS` アクション	2018 年 11 月 27 日
Amazon ECR	2018 年 11 月 27 日
Service Catalog	2018 年 10 月 16 日

AWS のサービス	CodePipeline サポート日
AWS Device Farm	2018 年 7 月 19 日
Amazon ECS	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
CodeCommit	2016 年 4 月 18 日
AWS OpsWorks	2016 年 6 月 2 日
AWS CloudFormation	2016 年 11 月 3 日
AWS CodeBuild	2016 年 12 月 1 日
Elastic Beanstalk	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、https://console.aws.amazon.com/iam/ で IAM コンソールを開きます。
2. IAM コンソールのナビゲーションペインで、[ロール] を選択し、ロールのリストから `AWS-CodePipeline-Service` ロール を選択します。
3. [アクセス権限] タブの [インラインポリシー] で、サービスロールポリシーの列の [ポリシーの編集] を選択します。
4. [Policy Document] ボックスに必要なアクセス許可を追加します。

> **Note**
>
> IAM ポリシーを作成するとき、最小限の特権を認めるという標準的なセキュリティアドバイスに従いましょう。そうすれば、タスクを実行するというリクエストのアクセス許可のみを認めることができます。一部の API コールはリソースベースのアクセス許可をサポートしており、アクセスを制限できます。たとえば、この場合、`DescribeTasks` および `ListTasks` を呼び出す際のアクセス許可を制限するために、ワイルドカード文字 (*) をリソース ARN またはワイルドカード文字 (*) を含むリソース ARN に置き換えることができます。最小権限アクセスを付与するポリシーの作成の詳細については、https://

docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege を参照してください。

例えば、 CodeCommit サポートのために、ポリシーステートメントに以下を追加します。

```
{
  "Effect": "Allow",
  "Action": [
      "codecommit:GetBranch",
      "codecommit:GetCommit",
      "codecommit:UploadArchive",
      "codecommit:GetUploadArchiveStatus",
      "codecommit:CancelUploadArchive"
           ],
  "Resource": "resource_ARN"
},
```

AWS OpsWorks サポートのために、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"
},
```

AWS CloudFormation サポートのために、ポリシーステートメントに以下を追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStack",
```

```
            "cloudformation:DeleteStack",
            "cloudformation:DescribeStackEvents",
            "cloudformation:DescribeStacks",
            "cloudformation:UpdateStack",
            "cloudformation:CreateChangeSet",
            "cloudformation:DeleteChangeSet",
            "cloudformation:DescribeChangeSet",
            "cloudformation:ExecuteChangeSet",
            "cloudformation:SetStackPolicy",
            "cloudformation:ValidateTemplate",
            "iam:PassRole"
        ],
        "Resource": "resource_ARN"
    },
```

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。詳細については、「AWS CloudFormation」を参照してください。

CodeBuild サポートのために、ポリシーステートメントに以下を追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> バッチビルド対応が後日追加されました。バッチビルドのサービスロールに追加する権限については、手順 11 を参照してください。

AWS Device Farm サポートのために、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"
},
```

Service Catalog がサポートされるように、以下をポリシーステートメントに追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "servicecatalog:ListProvisioningArtifacts",
        "servicecatalog:CreateProvisioningArtifact",
        "servicecatalog:DescribeProvisioningArtifact",
        "servicecatalog:DeleteProvisioningArtifact",
        "servicecatalog:UpdateProduct"
    ],
    "Resource": "resource_ARN"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:ValidateTemplate"
    ],
    "Resource": "resource_ARN"
}
```

5. 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"
```

```
},
```

6. 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。

```
{
    "Effect": "Allow",
    "Action": [
        "ecs:DescribeServices",
        "ecs:DescribeTaskDefinition",
        "ecs:DescribeTasks",
        "ecs:ListTasks",
        "ecs:RegisterTaskDefinition",
        "ecs:TagResource",
        "ecs:UpdateService"
    ],
    "Resource": "resource_ARN"
},
```

Amazon ECS でタグ付け承認の使用をオプトインできます。オプトインする場合、ecs:TagResource というアクセス許可を付与する必要があります。オプトインの方法、必要になるアクセス許可、タグ付け承認の詳細については、Amazon Elastic Container Service 開発者ガイドの「タグ付け承認のタイムライン」を参照してください。

また、タスクに IAMロールを使用するための iam:PassRole アクセス許可を追加する必要があります。詳細については、「Amazon ECS タスク実行 IAM ロールそしてタスクの IAM ロール」を参照してください。以下のポリシーテキストを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [

 "arn:aws:iam::aws_account_ID:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

7. CodeDeployToECS アクション (ブルー/グリーンデプロイ) の場合、 CodeDeploy Amazon ECS ブルー/グリーンデプロイアクションへの 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": [
        "codedeploy:CreateDeployment",
        "codedeploy:GetDeployment",
        "codedeploy:GetApplication",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision",
        "codedeploy:GetDeploymentConfig",
        "ecs:RegisterTaskDefinition",
        "ecs:TagResource"
    ],
    "Resource": "resource_ARN"
},
```

Amazon ECS でタグ付け承認の使用をオプトインできます。オプトインする場合、ecs:TagResource というアクセス許可を付与する必要があります。オプトインの方法、必要になるアクセス許可、タグ付け承認の詳細については、Amazon Elastic Container Service 開発者ガイドの「タグ付け承認のタイムライン」を参照してください。

また、タスクに IAMロールを使用するための iam:PassRole アクセス許可を追加する必要があります。詳細については、「Amazon ECS タスク実行 IAM ロールそしてタスクの IAM ロール」を参照してください。以下のポリシーテキストを使用します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": [

 "arn:aws:iam::aws_account_ID:role/ecsTaskExecutionRole_or_TaskRole_name"
            ]
        }
    ]
}
```

この例に示すように、ecs-tasks.amazonaws.com を iam:PassedToService 条件下のサービスのリストへ追加することができます。

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "resource_ARN",
            "Condition": {
                "StringEqualsIfExists": {
                    "iam:PassedToService": [
                        "cloudformation.amazonaws.com",
                        "elasticbeanstalk.amazonaws.com",
                        "ec2.amazonaws.com",
                        "ecs-tasks.amazonaws.com"
                    ]
                }
            }
        },
```

8. AWS CodeStar 接続の場合、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": [
        "codestar-connections: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、「Connections アクセス許可リファレンス」を参照してください。

9. StepFunctions アクションでは、Step Functions 呼び出しアクションを使用してパイプラインを作成するために必要な最小限のアクセス許可は次のとおりです。

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "states:DescribeStateMachine",
        "states:DescribeExecution",
        "states:StartExecution"
    ],
    "Resource": "resource_ARN"
},
```

10AppConfig アクションの場合、呼び出しアクションで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 AWS AppConfig は次のとおりです。

```
{
    "Effect": "Allow",
    "Action": [
        "appconfig:StartDeployment",
        "appconfig:GetDeployment",
        "appconfig:StopDeployment"
    ],
    "Resource": "resource_ARN"
},
```

11バッチビルド CodeBuild 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": [
        "codebuild:BatchGetBuildBatches",
        "codebuild:StartBuildBatch"
    ],
    "Resource": "resource_ARN"
},
```

12AWS CloudFormation StackSets アクションには、次の最小限のアクセス許可が必要です。

- CloudFormationStackSet のアクションについては、以下をポリシーステートメントに追加します。

```
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:CreateStackSet",
        "cloudformation:UpdateStackSet",
```

```
            "cloudformation:CreateStackInstances",
            "cloudformation:DescribeStackSetOperation",
            "cloudformation:DescribeStackSet",
            "cloudformation:ListStackInstances"
        ],
        "Resource": "resource_ARN"
},
```

- 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"
},
```

13.フルクローンオプション CodeCommit 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": [
        "codecommit:GetRepository"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> CodeBuild アクションが CodeCommit ソースでフルクローンオプションを使用できるようにするには、プロジェクトの CodeBuild サービスロールのポリシーステートメントにもアクセスcodecommit:GitPull許可を追加する必要があります。

14Elastic Beanstalk では、ElasticBeanstalk デプロイアクションを使用してパイプラインを作成するために必要な最小限のアクセス許可は次のとおりです。

```
{
```

```
    "Effect": "Allow",
    "Action": [
        "elasticbeanstalk:*",
        "ec2:*",
        "elasticloadbalancing:*",
        "autoscaling:*",
        "cloudwatch:*",
        "s3:*",
        "sns:*",
        "cloudformation:*",
        "rds:*",
        "sqs:*",
        "ecs:*"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> リソースポリシーのワイルドカードは、アクセスを制限するアカウントのリソースに置き換える必要があります。最小権限アクセスを付与するポリシーの作成の詳細については、https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege を参照してください。

15. CloudWatch Logs に設定するパイプラインの場合、 CodePipeline サービスロールに追加する必要がある最小限のアクセス許可を次に示します。

```
{
    "Effect": "Allow",
    "Action": [
        "logs:DescribeLogGroups",
        "logs:PutRetentionPolicy"
    ],
    "Resource": "resource_ARN"
},
```

> **Note**
>
> リソースポリシーのワイルドカードは、アクセスを制限するアカウントのリソースに置き換える必要があります。最小権限アクセスを付与するポリシーの作成の詳細について

は、https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege を参照してください。

16[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。

でのログ記録とモニタリング CodePipeline

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、または に代わって行われた AWS API コールおよび関連イベントのログ記録に使用できます AWS アカウント。詳細については、「を使用した CodePipeline API コールのログ記録 AWS CloudTrail」を参照してください。
- Amazon CloudWatch Events を使用して、 で実行する AWS クラウド リソースと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、コンプライアンスプログラムAWS のサービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS 「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」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 を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、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に重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスにおける HIPAA セキュリティとコンプライアンスのアーキテクチャ — このホワイトペーパーでは、企業が AWS を使用して HIPAA 対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべて AWS のサービス HIPAA の対象となるわけではありません。詳細については、「HIPAA 対応サービスのリファレンス」を参照してください。

- AWS コンプライアンスリソース – 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。このガイドでは、ガイダンスを保護し AWS のサービス 、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にマッピングするためのベストプラクティスをまとめています。
- 「 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、リソース設定が社内プラクティス、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
- AWS Security Hub – 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確認できます 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リストについては、「Security Hub のコントロールリファレンス」を参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしい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監視することで、、、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 は、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への対応に役立ちます。

- AWS Audit Manager – 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、および高度に冗長なネットワークで接続された、物理的に分離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、AWS 「 グローバルインフラストラクチャ」を参照してください。

のインフラストラクチャセキュリティ AWS CodePipeline

マネージドサービスである AWS CodePipeline は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して 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。

が AWS 公開した API コールを使用して、ネットワーク CodePipeline 経由で 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 は必須で TLS 1.3 がお勧めです。
- 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、AWS Security Token Service (AWS STS) を使用して、一時的なセキュリティ認証情報を生成し、リクエストに署名することもできます。

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べき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必ずしも適切または十分でない可能性があるので、処方箋ではなく、あくまで有用な考慮事項とお考えください。

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。セキュリティの CodePipeline ベストプラクティスは次のとおりです。

- 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、「」で説明されているように、 を管理 CodePipeline して AWS KMS keys、Amazon S3 に保存されているアーティファクトのサーバー側の暗号化を設定しますの Amazon S3 に保存されているアーティファクトのサーバー側の暗号化を設定する CodePipeline。
- 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。 Amazon S3 Jenkins インスタンスプロファイルのロールを作成する方法については、「Jenkins 統合に使用する IAM ロールを作成する」のステップを参照してください。

AWS CodePipeline コマンドラインリファレンス

AWS CodePipeline コマンドを使用する際、および「 AWS CLI ユーザーガイド」と「 リファレンス」に記載されている情報の補足として、このAWS CLI リファレンスを使用します。

を使用する前に AWS CLI、「」の前提条件を満たしていることを確認してくださいの開始方法 CodePipeline。

使用可能なすべての CodePipeline コマンドのリストを表示するには、次のコマンドを実行します。

```
aws codepipeline help
```

特定の CodePipeline コマンドに関する情報を表示するには、次のコマンドを実行します。ここで、*command-name* は以下に示すコマンドのいずれかの名前です (例: create-pipeline) 。

```
aws codepipeline command-name help
```

の拡張機能でコマンドを使用する方法を学ぶには AWS CLI、以下のセクションの 1 CodePipeline つ以上にアクセスしてください。

- カスタムアクションを作成する
- パイプラインを作成する (CLI)
- パイプラインを削除する (CLI)
- 移行を無効化または有効化する (CLI)
- パイプラインの詳細と履歴を表示する (CLI)
- 失敗したアクションを再試行する (CLI)
- パイプラインを手動で開始する (CLI)
- パイプラインを編集する (AWS CLI)

また、「CodePipeline チュートリアル」では、これらのコマンドの基本的な使用例について確認できます。

CodePipeline パイプライン構造リファレンス

デフォルトでは、 で正常に作成したパイプラインは有効な構造 AWS CodePipeline になっています。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 のクォータ AWS CodePipeline の制約を参照してください。

トピック

- の有効なアクションタイプとプロバイダー CodePipeline
- のパイプラインとステージ構造の要件 CodePipeline
- のアクション構造の要件 CodePipeline

の有効なアクションタイプとプロバイダー CodePipeline

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。

以下は、 の有効なアクションカテゴリです CodePipeline。

- ソース
- ビルド
- テスト
- デプロイ
- 承認
- Invoke

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を検索する方法については、「アクション構造リファレンス」または「各アクションタイプの入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 Cloud、 GitHub、 GitHub Enterprise Server、または GitLab.com アクションについては、CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、GitLab.com、および GitLab セルフマネージドアクション用アクション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	アクションリファレンス
ソース	Amazon S3	Amazon S3 ソースアクション
	Amazon ECR	Amazon ECR
	CodeCommit	CodeCommit
	CodeStarSourceConnection (Bitbucket Cloud、GitHub、 GitHub Enterprise Server、または GitLab.com アクションの場合)	CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、GitLab.com、および GitLab セルフマネージドアクション用
ビルド	CodeBuild	AWS CodeBuild

アクションカテゴリ	有効なアクションプロバイダー	アクションリファレンス
	カスタム CloudBees	各アクションタイプの入出力アーティファクトの数
	カスタム Jenkins	各アクションタイプの入出力アーティファクトの数
	カスタム TeamCity	各アクションタイプの入出力アーティファクトの数
テスト	CodeBuild	AWS CodeBuild
	AWS Device Farm	各アクションタイプの入出力アーティファクトの数
	ThirdParty GhostInspector	各アクションタイプの入出力アーティファクトの数
	カスタム Jenkins	各アクションタイプの入出力アーティファクトの数

アクションカテゴリ	有効なアクションプロバイダー	アクションリファレンス
	ThirdParty Micro フォーカス StormRunner ロード	各アクションタイプの入出力アーティファクトの数
	ThirdParty ヌーボラ	各アクションタイプの入出力アーティファクトの数
デプロイ	Amazon S3	Amazon S3 デプロイアクション
	AWS CloudFormation	AWS CloudFormation
	AWS CloudFormation StackSets (`CloudFormationStackSet` および `CloudFormationStackInstances` アクションを含む)	AWS CloudFormation StackSets
	CodeDeploy	各アクションタイプの入出力アーティファクトの数
	Amazon ECS	各アクションタイプの入出力アーティファクトの数
	Amazon ECS (Blue/Green) (これは `CodeDeployToECS` アクションです)	各アクションタイプの入出力アーティファクトの数

アクションカテゴリ	有効なアクションプロバイダー	アクションリファレンス
	Elastic Beanstalk	各アクションタイプの入出力アーティファクトの数
	AWS AppConfig	AWS AppConfig
	AWS OpsWorks	各アクションタイプの入出力アーティファクトの数
	Service Catalog	各アクションタイプの入出力アーティファクトの数
	Amazon Alexa	各アクションタイプの入出力アーティファクトの数
	カスタム XebiaLabs	各アクションタイプの入出力アーティファクトの数
承認	手動	各アクションタイプの入出力アーティファクトの数
Invoke	AWS Lambda	AWS Lambda
	AWS Step Functions	AWS Step Functions

の一部のアクションタイプ CodePipeline 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 プロバイダーは使用できません。

各アクションプロバイダーの詳細については、「 CodePipeline アクションタイプとの統合」を参照してください。

以下のセクションでは、アクションタイプ別のプロバイダー情報と設定プロパティの例を示します。

のパイプラインとステージ構造の要件 CodePipeline

2 ステージパイプラインの基本構造は次のとおりです。

```
{
    "roleArn": "An IAM ARN for a service role, such as arn:aws:iam::80398EXAMPLE:role/
CodePipeline_Service_Role",
    "stages": [
        {
            "name": "SourceStageName",
            "actions": [
            ... See ########### CodePipeline ...
            ]
        },
        {
            "name": "NextStageName",
            "actions": [
            ... See ########### CodePipeline ...
            ]
        }
    ],
    "artifactStore": {
        "type": "S3",
        "location": "The name of the Amazon S3 bucket automatically generated for you
 the first time you create a pipeline
            using the console, such as codepipeline-us-east-2-1234567890, or any Amazon
 S3 bucket you provision for this purpose"
    },
    "name": "YourPipelineName",
    "version": 1
}
```

パイプライン構造には、次の要件があります。

- 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 コンソールで編集できません。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。パラメータの値 (によって返される JSON の 4 つのアスタリスクでマスクされます AWS CLI) を手動で入力し、JSON 構造に含める必要があります。
- `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。

  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 codepipeline-us-
east-1-1234567890"
            },
            "us-west-2": {
                "type": "S3",
                "location": "S3 artifact bucket name, such as codepipeline-us-
west-2-1234567890"
            }
        },
        "stages": [
```

```
            {

...
```

- 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。
- 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。

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。

のアクション構造の要件 CodePipeline

アクション構造の概要は次のとおりです。

```
[
                {
                   "inputArtifacts": [
                        An input artifact structure, if supported for the action
 category
                    ],
                   "name": "ActionName",
                   "region": "Region",
                   "namespace": "source_namespace",
                   "actionTypeId": {
                        "category": "An action category",
                        "owner": "AWS",
                        "version": "1"
                        "provider": "A provider type for the action category",
                   },
                   "outputArtifacts": [
                        An output artifact structure, if supported for the action
 category
                   ],
                   "configuration": {
```

```
                    Configuration details appropriate to the provider type
                },
                "runOrder": A positive integer that indicates the run order within
the stage,
            }
        ]
```

このプロバイダータイプに該当する configuration の例の詳細一覧については、「プロバイダータイプ別の設定の詳細」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
- 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"
      }
  ],
  ```

  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。

- 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。

- クロスリージョンアクションでは、Region フィールドを使用して、アクションが作成される 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
- アクションは変数で設定できます。namespace フィールドを使用して、実行変数の名前空間と変数の情報を設定します。実行変数とアクション出力変数のリファレンス情報については、「変数」を参照してください。
- 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。詳細については、CodePipeline 「 API リファレンス」を参照してください。
- アクションの runOrder のデフォルト値は 1 です。値は、正の整数 (自然数) にする必要があります。分数、10 進数、負の数値、ゼロを使用することはできません。アクションのシリアルシーケンスを指定するには、最初のアクションに最小値を使用し、シーケンスの残りの各アクションにそれより大きい数値を使用します。並列アクションを指定するには、並列に実行する各アクションに同一の整数を使用します。コンソールで、実行するステージのレベルで [アクショングループを追加する] を選択して、アクションのシリアルシーケンスを指定できます。[アクションを追加する] を選択して、並列シーケンスを指定することもできます。[アクショングループ] は、同じレベルにある 1 つ以上のアクションの実行順序を指します。

  例えば、3 つのアクションをステージのシーケンスで実行する場合、最初のアクションの runOrder 値には 1 を、2 番目のアクションの runOrder 値には 2 を、3 番目のアクションの runOrder 値には 3 を指定します。ただし、2 番目と 3 番目のアクションを並列に実行する場合、最初のアクションの runOrder 値には 1 を、2 番目と 3 番目のアクションの runOrder 値にはいずれも 2 を指定します。

> **Note**
>
> シリアルアクションの番号付けは、厳密な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。

- 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	出力
datetime	js-application/{datetime}.zip	この形式の UTC タイムスタンプ: <YYYY>-<MM>-DD>_<HH>-<MM>-<SS> 例： js-application/2019-01-10_07-39-57.zip
uuid	js-application/{uuid}.zip	UUID は、他のすべての識別子と異なることが保証されるグローバル一意識別子です。この形式の UUID (すべての桁は 16 進形式): <8 桁>-<4 桁>-4 桁>-<4 桁>-<12 桁> 例： js-application/54a60075-b96a-4bf3-9013-db3a9EXAMPLE.zip

- の有効な`actionTypeId`カテゴリは次のとおりです CodePipeline。
  - `Source`
  - `Build`
  - `Approval`
  - `Deploy`
  - `Test`

- Invoke

一部のプロバイダータイプと設定オプションを以下に示します。

- 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、ソースアクションタイプの場合、有効なプロバイダータイプは S3、GitHub、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"},
```

- 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	必須プロパティかどうか
Amazon S3 (デプロイアクションプロバイダー)	Amazon S3 デプロイアクションパラメータに関連する例などの詳細については、「Amazon S3 デプロイアクション」を参照してください。		
Amazon S3 (ソースアクションプロバイダー)	Amazon S3 ソースアクションパラメータに関連する例などの詳細については、「Amazon S3 ソースアクション」を参照してください。		
Amazon ECR	Amazon ECR パラメータに関連する例などの詳細については、「Amazon ECR」を参照してください。		
CodeCommit	CodeCommit 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CodeCommit。		

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	必須プロパティかどうか
GitHub	GitHub 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GitHub バージョン 1 ソースアクション構造リファレンス。		
AWS CloudFormation	AWS CloudFormation パラメータに関連する例を含む詳細については、「」を参照してくださいAWS CloudFormation。		
CodeBuild	CodeBuild 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AWS CodeBuild。		
CodeDeploy	CodeDeploy 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AWS CodeDeploy。		
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AWS Device Farm。		
AWS Elastic Beanstalk	`ElasticBeanstalk`	`ApplicationName`	必須
		`EnvironmentName`	必須
AWS Lambda	AWS Lambda パラメータに関連する例を含む詳細については、「」を参照してくださいAWS Lambda。		
AWS OpsWorks Stacks	`OpsWorks`	`Stack`	必須
		`Layer`	オプションです。
		`App`	必須
Amazon ECS	Amazon ECS パラメータに関連する詳細な説明と例については、「Amazon Elastic Container Service」を参照してください。		

プロバイダー名	アクションタイプでのプロバイダー名	設定プロパティ	必須プロパティかどうか
Amazon ECS および CodeDeploy (青/緑)	Amazon ECS および CodeDeploy Blue/Green 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Amazon Elastic Container Service と CodeDeploy Blue-Green。		
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	Plugin for Jenkins CodePipeline で指定したアクションの名前 (例: *`MyJenkinsProviderName`*)	`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。

トピック

各アクションタイプの入出力アーティファクトの数

アクションタイプによっては、入出力アーティファクトの数は以下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	Amazon S3	0	1
AWS	ソース	CodeCommit	0	1
AWS	ソース	Amazon ECR	0	1
ThirdParty	ソース	GitHub	0	1
AWS	ビルド	CodeBuild	1 ~ 5	0 ~ 5
AWS	テスト	CodeBuild	1 ~ 5	0 ~ 5
AWS	テスト	AWS Device Farm	1	0
AWS	承認	手動	0	0
AWS	デプロイ	Amazon S3	1	0
AWS	デプロイ	AWS CloudFormation	0 ~ 10	0 ~ 1
AWS	デプロイ	CodeDeploy	1	0

[所有者]	アクションのタイプ	プロバイダー	入力アーティファクト有効な数	出力アーティファクトの有効な数
AWS	デプロイ	AWS Elastic Beanstalk	1	0
AWS	デプロイ	AWS OpsWorks Stacks	1	0
AWS	デプロイ	Amazon ECS	1	0
AWS	デプロイ	Service Catalog	1	0
AWS	Invoke	AWS Lambda	0 ~ 5	0〜5
ThirdParty	デプロイ	Alexa Skills Kit	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

PollForSourceChanges パラメータのデフォルト設定

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 (該当する場合、CloudWatch イベントルール) を作成します。
- PollForSourceChanges パラメータを false に設定します。

> **Note**
>
> Events CloudWatch ルールまたはウェブフックを作成する場合は、パイプラインが複数回トリガーされないように、 パラメータを false に設定する必要があります。

`PollForSourceChanges` パラメータは、Amazon ECR ソースアクションには使用されません。

- PollForSourceChanges パラメータのデフォルト

ソース	作成方法	JSON 構造の出力の設定例
CodeCommit	パイプラインはコンソールで作成されます (変更検出リソースもコンソールで作成されます)。パラメータは、パイプライン構造の出力に表示され、デフォルトで `false` になります。	`BranchName": "main", "PollForSourceChanges": "false", "RepositoryName": "my-repo"`
	パイプラインは CLI または で作成され AWS CloudFormation、 `PollForSourceChanges` 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。	`"S3Bucket": "my-bucket", "S3ObjectKey": "object.zip", "PollForSourceChanges": "false"`
	パイプラインは CLI または で作成され AWS CloudFormation、 `PollForSourceChanges` 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-bucket", "S3ObjectKey": "object.zip"`

ソース	作成方法	JSON 構造の出力の設定例
GitHub	パイプラインはコンソールで作成されます (変更検出リソースもコンソールで作成されます)。パラメータは、パイプライン構造の出力に表示され、デフォルトで false になります。	"Owner": "*MyGitHubAccountName* ", "Repo": " *MyGitHubRepositoryName* " "PollForSourceChanges": "false", "Branch": " *main*" "OAuthToken": " ****"
	パイプラインは CLI または で作成され AWS CloudFormation、 PollForSourceChanges 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。	"Owner": "*MyGitHubAccountName* ", "Repo": "*MyGitHubRepositoryName* ", "Branch": " *main*", "OAuthToken": " ****"
	[2] PollForSourceChanges は、特定のポイントで JSON 構造または AWS CloudFormation テンプレートに追加されると、次のように表示されます。 "PollForSourceChanges": "true",	
	[3] 各ソースプロバイダーに適用される変更検出リソースの詳細については、「変更検出方法」を参照してください。	

プロバイダータイプ別の設定の詳細

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。

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"
}
```

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。

の各アクションプロバイダーは、パイプライン構造で必須およびオプションの一連の設定フィールド CodePipeline 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 ECR
- Amazon Elastic Container Service と CodeDeploy Blue-Green
- Amazon Elastic Container Service
- Amazon S3 デプロイアクション
- Amazon S3 ソースアクション
- AWS AppConfig
- AWS CloudFormation
- AWS CloudFormation StackSets
- AWS CodeBuild
- CodeCommit
- AWS CodeDeploy
- CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用
- AWS Device Farm

- AWS Lambda
- Snyk アクション構造リファレンス
- AWS Step Functions

Amazon ECR

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、他のすべてのソースアーティファクトのソースロケーションを許可 CodeCommitするために、 などの別のソースアクションと組み合わせてよく使用されます。詳細については、「チュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する」を参照してください。

コンソールを使用してパイプラインを作成または編集すると、 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 CloudWatch イベントルール CodePipeline を作成します。

Amazon ECR アクションを介してパイプラインを接続する前に、Amazon ECR リポジトリを作成し、イメージをプッシュしておく必要があります。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- アクションの宣言 (Amazon ECR の例)
- 以下も参照してください。

アクションタイプ

- カテゴリ:`Source`
- 所有者: `AWS`
- プロバイダー: `ECR`
- バージョン: `1`

設定パラメータ

RepositoryName

必須: はい

イメージがプッシュされた Amazon ECR リポジトリの名前。

ImageTag

必須: いいえ

イメージに使用するタグ。

> **Note**
>
> `ImageTag` の値を指定しない場合、デフォルト値は `latest` になります。

入力アーティファクト

- アーティファクトの数: `0`
- 説明: 入力アーティファクトは、このアクションタイプには適用されません。

出力アーティファクト

- アーティファクトの数: `1`
- 説明: このアクションは、パイプライン実行をトリガーしたイメージの URI を含む `imageDetail.json` ファイルがあるアーティファクトを生成します。`imageDetail.json` ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。このアクションは、アクションに名前空間がない場合でも、出力変数として表示できる変数を生成します。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

詳細については、「変数」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。

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"
        }
    ]
},
```

以下も参照してください。

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 ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する – このチュートリアルでは、サンプルアプリケーション仕様ファイルとサンプル CodeDeploy アプリケーションおよびデプロイグループを提供し、Amazon ECS インスタンスにデプロイする CodeCommit と Amazon ECR ソースを使用してパイプラインを作成します。

Amazon Elastic Container Service と CodeDeploy Blue-Green

ブルー/グリーンデプロイを使用してコンテナアプリケーションをデプロイ AWS CodePipeline するパイプラインを で設定できます。Blue/Green デプロイでは、古いバージョンと一緒に新しいバージョンのアプリケーションを起動し、トラフィックを再ルーティングする前に新しいバージョンをテストできます。また、デプロイプロセスをモニタリングし、問題がある場合は迅速にロールバックすることもできます。

完成したパイプラインはイメージまたはタスク定義ファイルへの変更を検出し、 CodeDeploy を使用してトラフィックを Amazon ECS クラスターにルーティングしてデプロイします。ロードバランサーでは CodeDeploy 、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が作成されます。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ロケーション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作成しておく必要があります。イメージにタグを付けてイメージリポジトリに保存し、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、 の Amazon ECS から CodeDeploy Blue/Green への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リファレンス情報については CodePipeline、「」を参照してくださいAmazon Elastic Container Service。

トピック

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

内のアプリケーションの名前 CodeDeploy。パイプラインを作成する前に、 でアプリケーションを作成しておく必要があります CodeDeploy。

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たデプロイグループ。パイプラインを作成する前に、 でデプロイグループを作成しておく必要があります CodeDeploy。

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TaskDefinition内の では、ここに示すように<TASK_DEFINITION>プレースホルダーテキストを保持できます。

AppSpecTemplatePath

必須: いいえ

パイプラインの CodeCommit リポジトリなど、パイプラインソース AppSpec ファイルの場所に保存されている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。ファイルの名前 AppSpec が同じで、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、パイプラインファイルのソース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 < 数値 > 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 < 数値 > 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、まずソース AppSpec ファイルリポジトリ内のタスク定義ファイルと 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、次にタスク定義の新しいリビジョンを動的に生成し、最後に AppSpec コマンドを実行してタスクセットとコンテナをクラスターにデプロイします。

  CodeDeployToECS アクションは、イメージ URI をイメージにマップする imageDetail.json ファイルを検索します。Amazon ECR イメージリポジトリへの変更をコミットすると、ECR ソースアクションのパイプラインはそのコミット用に imageDetail.json ファイルを作成します。アクションが自動化されていないパイプラインの場合、手動で imageDetail.json ファイルを追加することもできます。imageDetail.json ファイルの詳細については、「Amazon ECS Blue/Green デプロイアクション用の imageDetail.json ファイル」を参照してください。

  CodeDeployToECS アクションは、タスク定義の新しいリビジョンを動的に生成します。このフェーズでは、このアクションがタスク定義ファイル内のプレースホルダーを、imageDetail.json ファイルから取得したイメージ URI に置き換えます。例えば、IMAGE1_NAME を Image1ContainerName パラメータとして設定する場合は、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 Image1 として指定したアーティファクトの imageDetail.json から取得した実際のイメージ URI Image1ArtifactNameに置き換えます。

  タスク定義の更新の場合 CodeDeploy AppSpec.yaml、 ファイルには TaskDefinitionプロパティが含まれます。

  ```
  TaskDefinition: <TASK_DEFINITION>
  ```

  このプロパティは、新しいタスク定義が作成された後、CodeDeployToECS アクションによって更新されます。

  TaskDefinition フィールドの値として、プレースホルダーテキストは <TASK_DEFINITION> である必要があります。CodeDeployToECS アクションは、このプレースホルダーを、動的に生成されたタスク定義の実際の ARN に置き換えます。

出力アーティファクト

- アーティファクトの数: 0
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

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"
        }
    ]
}
```

以下も参照してください。

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 ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する – このチュートリアルでは、ブルー/グリーンデプロイに必要な CodeDeploy および Amazon ECS リソースの作成手順を説明します。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。次に、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を作成する手順を説明します。

> **Note**
>
> このトピックとチュートリアルでは、 の CodeDeploy/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

CodePipeline、「チュートリアル: を使用した継続的デプロイ CodePipeline」を参照してください。

- AWS CodeDeploy ユーザーガイド – ブルー/グリーン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 サービスのデプロイ」を参照してください。AWS CodeDeploy ユーザーガイドのこのリファレンス情報は、Amazon ECS および を使用した Blue/Green デプロイの概要を示しています AWS CodeDeploy。
- Amazon Elastic Container Service デベロッパーガイド - Docker イメージとコンテナ、ECS サービスとクラスター、および ECS タスクセットの操作については、「Amazon ECS とは」を参照してください。

Amazon Elastic Container Service

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 で作成するタスク定義と、 がイメージのデプロイ CodePipeline に使用するイメージ定義ファイルが必要です。

Important

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 CodePipeline 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

このリファレンストピックでは、 の Amazon ECS 標準デプロイアクションについて説明します CodePipeline。Amazon ECS から の CodeDeploy ブルー/グリーンデプロイアクション

へのリファレンス情報については CodePipeline、「」を参照してくださいAmazon Elastic Container Service と CodeDeploy Blue-Green。

トピック

アクションタイプ

- カテゴリ:`Deploy`
- 所有者: `AWS`
- プロバイダー: `ECS`
- バージョン: `1`

設定パラメータ

ClusterName

必須: はい

Amazon ECS 内の Amazon ECS クラスター。

ServiceName

必須: はい

Amazon ECS で作成した Amazon ECS サービス。

FileName

必須: いいえ

イメージ定義ファイルの名前は、サービスのコンテナ名およびイメージとタグを説明する JSON ファイルです。このファイルは ECS 標準デプロイに使用します。詳細については、「入力アーティファクト」および「Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル」を参照してください。

DeploymentTimeout

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 コンテナ名と image および tag. CodePipeline uses ファイルを説明する JSON ファイルで、Amazon ECR などのイメージリポジトリからイメージを取得します。アクションが自動化されていないパイプラインの場合、手動で `imagedefinitions.json` ファイルを追加することもできます。`imagedefinitions.json` ファイルの詳細については、「Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル」を参照してください。

  アクションには、イメージリポジトリにすでにプッシュされている既存のイメージが必要です。イメージマッピングは `imagedefinitions.json` ファイルの場合、アクションで Amazon ECR ソースをソースアクションとしてパイプラインに含める必要はありません。

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

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"
        }
    ]
```

```
},
```

以下も参照してください。

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: を使用した継続的デプロイ CodePipeline — このチュートリアルでは、 などのソースファイルリポジトリに保存する Dockerfile を作成する方法を示します CodeCommit。次に、チュートリアルでは、Docker イメージ CodeBuild BuildSpec を構築して Amazon ECR にプッシュし、ImageDefinitions.json ファイルを作成するファイルを組み込む方法を示します。最後に、Amazon ECS サービスとタスク定義を作成し、Amazon ECS デプロイアクションを使用してパイプラインを作成します。

  > **Note**
  >
  > このトピックとチュートリアルでは、 の Amazon ECS 標準デプロイアクションについて説明します CodePipeline。での Amazon ECS から CodeDeploy Blue/Green へのデプロイアクションについては CodePipeline、「」を参照してくださいチュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する。

- Amazon Elastic Container Service デベロッパーガイド - Docker イメージとコンテナ、Amazon ECS サービスとクラスター、および Amazon ECS タスクセットの操作については、「Amazon ECS とは」を参照してください。

Amazon S3 デプロイアクション

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、デプロイプラットフォーム CodePipeline がホスティング用に設定された Amazon S3 バケットである の Amazon S3 デプロイアクションについて説明します。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 CodePipeline、「」を参照してくださいAmazon S3 ソースアクション。

トピック

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。

KMS EncryptionKeyARN

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ます。非対称キーを使用して S3 bucket のデータを暗号化しないでください。

CannedACL

必須: いいえ

CannedACL パラメータは、指定された 既定 ACL を、Amazon S3 にデプロイされたオブジェクトに適用します。このオブジェクトに適用された既存の ACL が上書きされます。

CacheControl

必須: いいえ

CacheControl パラメータは、バケットのオブジェクトのリクエストやレスポンスのキャッシュ動作を制御します。有効な値のリストについては、HTTP オペレーションの Cache-Control ヘッダーフィールドを参照してください。CacheControl に複数の値を入力するには、各値の間にカンマを使用します。CLI の例に示すように、各カンマの後にスペースを追加できます (オプション)。

```
"CacheControl": "public, max-age=0, no-transform"
```

入力アーティファクト

- アーティファクトの数: 1
- 説明： デプロイまたはアーカイブ用のファイルは、ソースリポジトリから取得され、圧縮されて、 によってアップロードされます CodePipeline。

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'
    OutputArtifacts: []
    InputArtifacts:
```

```
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "false"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

Extract を **true** に設定する場合のアクションの設定例

以下の例で示しているのは、Extract フィールドを true に設定してアクションを作成した場合のアクションのデフォルト設定です。

YAML

```
Name: Deploy
```

```
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: S3
      Version: '1'
    RunOrder: 1
    Configuration:
      BucketName: website-bucket
      Extract: 'true'
      ObjectKey: MyWebsite
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "S3",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "BucketName": "website-bucket",
                "Extract": "true",
                "ObjectKey": "MyWebsite"
                },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
```

```
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

以下も参照してください。

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。サンプルファイルのダウンロード、 CodeCommit リポジトリへのファイルのアップロード、S3 バケットの作成、ホスティング用のバケットの設定を行います。次に、 CodePipeline コンソールを使用してパイプラインを作成し、Amazon S3 デプロイ設定を指定します。
- Amazon S3 ソースアクション – このアクションリファレンスは、 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ます CodePipeline。

Amazon S3 ソースアクション

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、ソースロケーション CodePipeline がバージョニング用に設定された Amazon S3 バケットである の Amazon S3 ソースアクションについて説明します。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 CodePipeline、「」を参照してくださいAmazon S3 デプロイアクション。

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、「バージョニングの使用」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、 は S3 ソースバケットで変更が発生したときにパイプラインを開始する CloudWatch イベントルール CodePipeline を作成します。

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

アクションタイプ

- カテゴリ:`Source`
- 所有者: `AWS`
- プロバイダー: `S3`

- バージョン: 1

設定パラメータ

S3 バケット

必須: はい

ソースの変更が検出される Amazon S3 バケットの名前。

S3ObjectKey

必須: はい

ソースの変更が検出される Amazon S3 オブジェクトキーの名前。

AllowOverrideForS3ObjectKey

必須: いいえ

`AllowOverrideForS3ObjectKey` は、 からのソースオーバーライドが、ソースアクションS3ObjectKeyで既に設定されている をオーバーライドStartPipelineExecutionできるかどうかを制御します。S3 オブジェクトキーによるソースオーバーライドの詳細については、「」を参照してください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。

> **Important**
>
> を省略すると`AllowOverrideForS3ObjectKey`、 は、このパラメータを に設定して、ソースアクションの S3 ObjectKey を上書きする機能を CodePipeline デフォルトにしますfalse。

このパラメータの有効な値:

- `true`: 設定されている場合、パイプラインの実行中に、事前設定された S3 オブジェクトキーをソースリビジョンの上書きで上書きできます。

> **Note**
>
> 新しいパイプラインの実行を開始するときに、すべての CodePipeline ユーザーが事前設定された S3 オブジェクトキーを上書きできるようにする場合は、`AllowOverrideForS3ObjectKey`を に設定する必要がありますtrue。

- `false`:

  設定されている場合、ソースリビジョンオーバーライドを使用して S3 オブジェクトキーを上書き CodePipeline することはできません。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 Amazon S3 ソースバケットを CodePipeline ポーリングするかどうかを制御します。代わりに CloudWatch 、イベント 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 イベントの設定の詳細については、S3 ソースと CloudTrail 証跡を使用してポーリングパイプラインを移行する (CLI)「」または「」を参照してください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 (AWS CloudFormation テンプレート) 。

> **Important**
>
> Events を設定する場合は CloudWatch 、パイプラインの実行が重複しないように `false` `PollForSourceChanges`を 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、ソースの変更についてソースロケーションを CodePipeline ポーリングします。

  > **Note**
  >
  > を省略すると`PollForSourceChanges`、 CodePipeline デフォルトでソースの変更のソースロケーションがポーリングされます。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、ソースの変更についてソースロケーションをポーリング CodePipeline しません。ソースの変更を検出するように CloudWatch イベントルールを設定する場合は、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) は、トリガーされたパイプライン実行のソースリビジョン CodePipeline として 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。

の変数の詳細については、 CodePipeline「」を参照してください変数。

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。

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: my-bucket-oregon
      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": "my-bucket-oregon",
                "S3ObjectKey": "my-application.zip",
                "PollForSourceChanges": "false"
            },
            "InputArtifacts": []
        }
    ]
},
```

以下も参照してください。

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とサンプル CodeDeployアプリケーションおよびデプロイグループを提供します。このチュートリアルを参照して、Amazon EC 2 インスタンスにデプロイする Amazon S3 ソースを持つパイプラインを作成します。

AWS AppConfig

AWS AppConfig は の一機能です AWS Systems Manager。 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 AppConfig をサポートし、検証チェックとモニタリングが組み込まれています。Amazon EC2 インスタンス AppConfig 、コンテナ、モバイルアプリケーション AWS Lambda、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 を使用できます。

AppConfig デプロイアクションは、パイプラインのソースロケーション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 で定義された設定を使用します。

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。

ConfigurationProfile

必須: はい

デプロイする AWS AppConfig 設定プロファイルの ID。

InputArtifactConfigurationPath

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。

DeploymentStrategy

必須: いいえ

AWS AppConfig デプロイに使用するデプロイ戦略。

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

該当しません。

アクション設定の例

YAML

```
name: Deploy
actions:
  - name: Deploy
    actionTypeId:
      category: Deploy
      owner: AWS
      provider: AppConfig
      version: '1'
    runOrder: 1
    configuration:
      Application: 2s2qv57
      ConfigurationProfile: PvjrpU
      DeploymentStrategy: frqt7ir
      Environment: 9tm27yd
      InputArtifactConfigurationPath: /
    outputArtifacts: []
    inputArtifacts:
      - name: SourceArtifact
    region: us-west-2
    namespace: DeployVariables
```

JSON

```
{
    "name": "Deploy",
    "actions": [
        {
            "name": "Deploy",
            "actionTypeId": {
                "category": "Deploy",
                "owner": "AWS",
                "provider": "AppConfig",
                "version": "1"
            },
            "runOrder": 1,
            "configuration": {
                "Application": "2s2qv57",
                "ConfigurationProfile": "PvjrpU",
                "DeploymentStrategy": "frqt7ir",
```

```
                "Environment": "9tm27yd",
                "InputArtifactConfigurationPath": "/"
            },
            "outputArtifacts": [],
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "namespace": "DeployVariables"
        }
    ]
}
```

以下も参照してください。

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS AppConfig – AWS AppConfig デプロイの詳細については、AWS Systems Manager 「 ユーザーガイド」を参照してください。
- チュートリアル: をデプロイプロバイダー AWS AppConfig として使用するパイプラインを作成する – このチュートリアルでは、シンプルなデプロイ設定ファイルと AppConfig リソースの設定を開始し、コンソールを使用して AWS AppConfig デプロイアクションでパイプラインを作成する方法を示します。

AWS CloudFormation

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。スタックと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、 は API を CodePipeline 呼び出します AWS CloudFormation `DescribeStackEvents`。アクション IAM ロールにその API へのアクセス許可がある場合、最初に失敗したリソースに関する詳細が CodePipeline エラーメッセージに含まれます。それ以外の場合、ロールポリシーに適切な

アクセス許可がない場合、 CodePipeline は API へのアクセスを無視し、代わりに一般的なエラーメッセージを表示します。そのためには、パイプラインのサービスロールまたは他の IAM ロールに cloudformation:DescribeStackEvents アクセス許可を追加する必要があります。

リソースの詳細がパイプラインのエラーメッセージに表示されないようにするには、cloudformation:DescribeStackEvents アクセス許可を削除することによって、アクション IAM ロールに対するこのアクセス許可を取り消すことができます。

トピック

アクションタイプ

- 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に対して 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。とは異なりREPLACE_ON_FAILURE、スタックが存在し、障害状態にある場合、スタックを削除して置き換え CodePipeline 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ことを明示的に承認する必要があります。

- CAPABILITY_IAM
- CAPABILITY_NAMED_IAM
- CAPABILITY_AUTO_EXPAND

機能間にカンマ (スペースなし) を使用して、複数の機能を指定できます。アクションの宣言 の例は、CAPABILITY_IAM プロパティと CAPABILITY_AUTO_EXPAND プロパティの両方を持つエントリを示しています。

の詳細についてはCapabilities、AWS CloudFormation 「 API リファレンス」のUpdateStack「」のプロパティを参照してください。

ChangeSetName

必須: 条件による

ChangeSetName は、既存の変更セットの名前、または指定されたスタック用に作成する新しい変更セットの名前です。

このプロパティは、次のアクションモードに必要です。CHANGE_SET_REPLACE および CHANGE_SET_EXECUTE。他のすべてのアクションモードでは、このプロパティは無視されます。

RoleArn

必須: 条件による

RoleArn は、指定されたスタックのリソースを操作するときに 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。 CodePipeline を使用して変更セットを作成しない場合は、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てください。

> **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 が与えられているため、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、 に表示される入力アーティファクト名です CodePipeline。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 1 MB です。での AWS CloudFormation デプロイアクションの場合 CodePipeline、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、のクォータ AWS CodePipeline 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、このアクションのパイプライン出力アーティファクト CodePipeline 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、出力ファイルやアーティファクトは生成 CodePipeline されません。

ParameterOverrides

必須: いいえ

パラメータはスタックテンプレートで定義され、スタックの作成時または更新時にそれらの値を指定できます。JSON オブジェクトを使用して、テンプレートにパラメータ値を設定できます。（これらの値は、テンプレート設定ファイルに設定された値を上書きします。）パラメータオーバーライドの使用の詳細については、設定プロパティ（JSON オブジェクト）を参照してください。

パラメータ値のほとんどは、テンプレート設定ファイルを使用して指定することをお勧めします。パラメータの上書きは、パイプラインが実行されるまで不明な値にのみ使用します。詳細については、「 AWS CloudFormation ユーザーガイド」の CodePipeline 「パイプラインでのパラメータオーバーライド関数の使用」を参照してください。

> **Note**
>
> すべてのパラメータ名がスタックテンプレートに存在する必要があります。

TemplateConfiguration

必須: いいえ

TemplateConfiguration はテンプレート構成ファイルです。このアクションの入力アーティファクトにファイルを含めます。テンプレート設定ファイルには、テンプレートのパラメータ値およびスタックポリシーを含めることができます。テンプレート設定ファイル形式の詳細については、「AWS CloudFormation アーティファクト」を参照してください。

テンプレート構成ファイル名は以下の形式に従います。

Artifactname::*TemplateConfigurationFileName*

Artifactname は、 に表示される入力アーティファクト名です CodePipeline。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次の目的でアーティファクトを受け入れます。
  - 実行するスタックテンプレートファイルを提供するため。(TemplatePath パラメータを参照。)
  - 使用するテンプレート設定ファイルを提供するため。(TemplateConfiguration パラメータを参照。) テンプレート設定ファイル形式の詳細については、「AWS CloudFormation アーティファクト」を参照してください。
  - AWS CloudFormation スタックの一部としてデプロイされる 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、出力を参照してください。

出力変数

このアクションを設定すると、パイプライン内のダウンストリームアクションのアクション設定によって参照できる変数が生成されます。名前空間を使用してアクションを設定し、これらの変数をダウンストリームアクションの設定で使用できるようにします。

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。出力を生成する唯一の CloudFormationアクションモードは、スタックの作成、スタックの更新、変更セットの実行など、スタックの作成または更新につながるアクションモードであることに注意してください。変数を生成するアクションモードは次のとおりです。

- `CHANGE_SET_EXECUTE`
- `CHANGE_SET_REPLACE`
- `CREATE_UPDATE`
- `REPLACE_ON_FAILURE`

詳細については、「変数」を参照してください。出力変数を使用する CloudFormationパイプラインに CloudFormation デプロイアクションを持つパイプラインを作成する方法を示すチュートリアルについては、「」を参照してくださいチュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。

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"
        }
    ]
},
```

以下も参照してください。

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 ユーザーガイドのこのリファレンスの章では、これらの CodePipeline パラメータの詳細と例を示します。
- AWS CloudFormation API リファレンス – AWS CloudFormation API リファレンスの CreateStackパラメータは、 テンプレートのスタックパラメータを記述します AWS CloudFormation 。

AWS CloudFormation StackSets

CodePipeline では、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できます。スタックセットを使用して、単一の 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。

の概念の詳細については AWS CloudFormation StackSets、「 ユーザーガイド」の「 のStackSets 概念AWS CloudFormation 」を参照してください。

パイプラインは、一緒に使用する 2 つの異なるアクションタイプ AWS CloudFormation StackSets を通じて と統合します。

- `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に、CloudFormation スタックセットアクションプロバイダーを選択できます。

- `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、既存のパイプラインを編集するときに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できます。

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 Organizations AWS 組織単位 IDsにデプロイできます。

> **Note**
>
> AWS Organizations アカウントまたは組織単位 IDs をターゲットにデプロイし、サービス管理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、AWS CloudFormation 「Stacksets での信頼されたアクセスの有効化」を参照してください。

トピック

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、CloudFormationStackSet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: 既に作成されたスタックセットに対して CloudFormationStackSet アクションが実行されると、 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。

  CloudFormationStackSet 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できます。

  - 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 の場合、メンバーアカウントの委任管理者として機能しますService Managed。
  - 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。

パイプラインでアクションを構築する StackSets方法

ベストプラクティスとして、スタックセットが作成され、最初にサブセットまたは単一のインスタンスにデプロイされるようにパイプラインを構築する必要があります。デプロイをテストし、生成されたスタックセットを表示したら、CloudFormationStackInstances アクションを追加し、残りのインスタンスが作成および更新されるようにします。

コンソールまたは CLI を使用して、次のように推奨されるパイプライン構造を作成します。

1. ソースアクションを持つパイプラインを作成し (必須)、CloudFormationStackSet アクションをデプロイアクションとして指定します。パイプラインを実行します。
2. パイプラインが最初に実行されると、CloudFormationStackSet アクションがスタックセットと少なくとも 1 つの初期インスタンスを作成します。スタックセットの作成を確認し、初期インスタンスへのデプロイを確認します。例えば、us-east-1 が指定されたリージョンであるアカウント Account-A のスタックセットの初期作成では、スタックインスタンスは次のスタックセットで作成されます。

スタックインスタンス	リージョン	ステータス
StackInstanceID-1	us-east-1	CURRENT

3. パイプラインを編集して、指定したターゲットのスタックインスタンスを作成または更新する 2 番目のデプロイアクションとして CloudFormationStackInstances を追加します。例えば、us-east-2 およびeu-central-1 リージョンが指定されるアカウント Account-A のスタックインスタンスの作成の場合、残りのスタックインスタンスが作成され、初期インスタンスは次のように更新されます。

スタックインスタンス	リージョン	ステータス
StackInstanceID-1	us-east-1	CURRENT
StackInstanceID-2	us-east-2	CURRENT
StackInstanceID-3	eu-central-1	CURRENT

4. 必要に応じてパイプラインを実行して、スタックセットを更新し、スタックインスタンスを更新または作成します。

アクション設定からデプロイターゲットを削除したスタックの更新を開始すると、更新用に指定されていなかったスタックインスタンスがデプロイから削除され、OUTDATED ステータスに移行します。たとえば、Account-A リージョンがアクション設定から削除されたアカウント us-east-2 のスタックインスタンスの更新の場合、残りのスタックインスタンスが作成され、削除されたインスタンスは OUTDATED に設定されます。

スタックインスタンス	リージョン	ステータス
StackInstanceID-1	us-east-1	CURRENT
StackInstanceID-2	us-east-2	OUTDATED
StackInstanceID-3	eu-central-1	CURRENT

スタックセットをデプロイするためのベストプラクティスの詳細については、「 ユーザーガイド StackSets 」の「 のベストプラクティスAWS CloudFormation 」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。ある AWS リージョンでスタックセットを作成する場合、他のリージョンからスタックセットにアクセスすることはできません。

このアクションをスタックセットに対する更新アクションとして使用する場合、少なくとも 1 つのスタック インスタンスにデプロイがないと、スタックの更新は許可されません。

トピック

アクションタイプ

- カテゴリ:Deploy
- 所有者: AWS
- プロバイダー: CloudFormationStackSet
- バージョン: 1

設定パラメータ

StackSetName

必須: はい

スタックセットに関連付ける名前。この名前は作成されるリージョンで一意であることが必要です。

名前には、英数字とハイフンのみを使用することができます。アルファベットで始まり、また 128 文字以下である必要があります。

説明

必須: いいえ

スタックセットの説明。これを使用して、スタックセットの目的やその他の関連情報を記述できます。

TemplatePath

必須: はい

スタックセット内のリソースを定義するテンプレートの口ケーション。これは、最大サイズ 460,800 byte のテンプレートを指定する必要があります。

フォーマット "InputArtifactName::TemplateFileName" で、ソースアーティファクト名とテンプレートファイルへのパスを次の例に示すように入力します。

```
SourceArtifact::template.txt
```

パラメータ

必須: いいえ

デプロイ中に更新されるスタックセットのテンプレートパラメータのリスト。

パラメータはリテラルリストまたはファイルパスとして提供できます。

- パラメータは、次の ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV のような省略構文のフォーマットで入力できます。これらのデータタイプの詳細については、「テンプレートパラメータのデータタイプ」を参照してください。

次の例は、my- BucketName の値を持つ bucket という名前のパラメータを示しています。

```
ParameterKey=BucketName,ParameterValue=my-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=my-bucket
  ParameterKey=Asset1,ParameterValue=true
  ParameterKey=Asset2,ParameterValue=true
```

- 次の例で示すように、フォーマット "InputArtifactName::ParametersFileName" で入力されたテンプレートパラメータのオーバーライドのリストを含むファイルのロケーションを入力できます。

```
SourceArtifact::parameters.txt
```

次の例では、parameters.txt のファイルの内容を示します。

```
[
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    }
]
```

機能

必須: いいえ

テンプレート内のリソースのタイプに応じて、テンプレートがリソースを作成および更新できることを示します。

このプロパティは、スタックテンプレートに IAM リソースがある場合、またはマクロを含むテンプレートから直接スタックを作成する場合に使用する必要があります。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。CloudFormationStackSet アクション設定の例 の例は、複数の機能を持つエントリを示しています。

PermissionModel

必須: いいえ

IAM ロールの作成および管理方法を決定します。フィールドを指定しない場合、デフォルトが使用されます。詳細については、「スタックセットオペレーションのアクセス許可モデル」を参照してください。

有効な値は次のとおりです。

- 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 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 AWS CloudFormation StackSets 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、 に設定されます AWSCloudFormationStackSetAdministrationRole。を指定する場合は ServiceManaged、ロール名を定義しないでください。

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 されるかどうかを示します。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。

有効な値は次のとおりです。

- Enabled。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す。アカウントがターゲット組織または OU から削除されると、 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 AWS CloudFormation StackSets を削除します。

- Disabled。必須: いいえ

  StackSets は、指定されたリージョン内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/更新し、 形式で入力する AWS アカウントのリストを含むファイルの場所`InputArtifactName::AccountsFileName`。ファイルパスを使用してアカウントまたは を指定する場合OrganizationalUnitIds、次の例に示すように、ファイル形式は JSON である必要があります。

    ```
    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。を選択した場合は、これを使用しないでくださいOrganizationsAutoDeployment。

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 OrganizationalUnitIds する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アカウントまたは を指定する場合OrganizationalUnitIds、次の例に示すように、ファイル形式は JSON である必要があります。

  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と、 も選択する必要がありますDeploymentTargets。

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、後続 AWS CloudFormation のリージョンでオペレーションを試みないでください。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 切り下げます。切り捨てるとゼロになる場合、 は代わりに数値を 1 として AWS CloudFormation 設定します。この設定で 最大値を指定する場合でも、大規模なデプロイでは、同時に処理される実際のアカウント数はサービスのスロットリングのために低くなる可能性があります。

RegionConcurrencyType

必須: いいえ

リージョン同時デプロイパラメータを設定することで、スタックセットを AWS リージョン 全体に順次デプロイするか、並列デプロイするかを指定できます。複数の にスタックを並行 AWS リージョン してデプロイするようにリージョンの同時実行を指定すると、全体的なデプロイ時間が短縮される可能性があります。

- 並列: スタックセットのデプロイは、指定された失敗許容回数をリージョンのデプロイ失敗が超えない限り、同時に行われます。
- 順次: スタックセットのデプロイは、リージョンのデプロイ失敗が指定された失敗許容回数を超えない限り、一度に 1 つずつ行われます。デフォルトでは、順次デプロイが選択されています。

ConcurrencyMode

必須: いいえ

同時実行モードでは、スタックセットオペレーション時の同時実行レベルの動作を、厳密な耐障害性またはソフトな耐障害性のいずれかを選択できます。厳密な障害耐性では、障害が発生するたびに同時実行性が低下するため、スタックセットの操作に障害が発生するため、デプロイ速度が低下します。ソフト障害耐性は、 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**
>
> このパラメータはアクセスSERVICE_MANAGED許可モデルではオプションであり、アクセスSELF_MANAGED許可モデルでは使用されません。

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の委任管理者として行動するかを指定します。

> **Note**
>
> このパラメータが に設定されている場合はDELEGATED_ADMIN、パイプラインの IAM ロールに アクセスorganizations:ListDelegatedAdministrators許可があることを確認してください。そうしないと、 の実行中にアクションが失敗し、次のようなエラーが発生します。 `Account used is not a delegated administrator`

- 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にサービスマネージド型のアクセス許可が使用されます。
- DELEGATED_ADMIN: スタックセットのデプロイでは、委任された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型のアクセス許可が使用されます。

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: 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」を参照してください。

CloudFormationStackSet アクション設定の例

次の例は、 アクションのCloudFormationStackSetアクション設定を示しています。

自己管理型のアクセス許可モデルの例

次の例は、入力されたデプロイターゲットが AWS アカウント ID であるCloudFormationStackSetアクションを示しています。

YAML

```
Name: CreateStackSet
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackSet
  Version: '1'
RunOrder: 1
Configuration:
  DeploymentTargets: '111111222222'
  FailureTolerancePercentage: '20'
  MaxConcurrentPercentage: '25'
  PermissionModel: SELF_MANAGED
  Regions: us-east-1
  StackSetName: my-stackset
  TemplatePath: 'SourceArtifact::template.json'
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
Namespace: DeployVariables
```

JSON

```
{
    "Name": "CreateStackSet",
    "ActionTypeId": {
```

```
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackSet",
        "Version": "1"
    },
    "RunOrder": 1,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "FailureTolerancePercentage": "20",
        "MaxConcurrentPercentage": "25",
        "PermissionModel": "SELF_MANAGED",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset",
        "TemplatePath": "SourceArtifact::template.json"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2",
    "Namespace": "DeployVariables"
}
```

サービス管理型のアクセス許可モデルの例

次の例は、 AWS Organizations への自動デプロイのオプションがスタック保持で有効になっている、サービスマネージド型の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

- 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/更新し、 形式で入力する AWS アカウントのリストを含むファイルの場所`InputArtifactName::AccountsFileName`。ファイル

パスを使用してアカウントまたは を指定する場合OrganizationalUnitIds、次の例に示すように、ファイル形式は JSON である必要があります。

```
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。を選択した場合は、これを使用しないでくださいOrganizationsAutoDeployment。

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 OrganizationalUnitIds する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アカウントまたは を指定する場合OrganizationalUnitIds、次の例に示すように、ファイル形式は JSON である必要があります。

ファイルのパスをフォーマット InputArtifactName::OrganizationalUnitIdsFileName で入力します。

```
SourceArtifact::OU-IDs.txt
```

次の例では、OU-IDs.txt のファイルの内容を示します。

```
[
    "ou-examplerootid111-exampleouid111","ou-examplerootid222-exampleouid222"
]
```

リージョン

必須: はい

> **Note**
> このパラメータを選択すると、 も選択する必要がありますDeploymentTargets。

スタックセットインスタンスが作成または更新される AWS リージョンのリスト。リージョンは、入力された順序で更新されます。

次の例に示すようにRegion1,Region2、有効な AWS リージョンのリストを の形式で入力します。

```
us-west-2,us-east-1
```

ParameterOverrides

必須: いいえ

選択したスタックインスタンスでオーバーライドしたいスタックセットパラメータのリスト。オーバーライドされたパラメータ値は、指定されたアカウントおよびリージョン内のすべてのスタックインスタンスに適用されます。

パラメータはリテラルリストまたはファイルパスとして提供できます。

- パラメータは、次の ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV ParameterKey=string,ParameterValue=string,UsePreviousValue=boolean,ResolvedV

のような省略構文のフォーマットで入力できます。これらのデータタイプの詳細については、「テンプレートパラメータのデータタイプ」を参照してください。

次の例は、my- BucketName の値を持つ bucket という名前のパラメータを示しています。

```
ParameterKey=BucketName,ParameterValue=my-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=my-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。

```
[
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    },
    {
        "ParameterKey": "KeyName",
        "ParameterValue": "true"
    }
]
```

FailureTolerancePercentage

必須: いいえ

このスタックオペレーションが失敗する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、後続 AWS CloudFormation のリージョンでオペレーションを試み

ないでください。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 切り下げます。切り捨てるとゼロになる場合、 は代わりに数値を 1 として AWS CloudFormation 設定します。最大値 を指定する場合でも、大規模なデプロイでは、同時に処理される実際のアカウント数はサービスのスロットリングのために低くなる可能性があります。

RegionConcurrencyType

必須: いいえ

リージョン同時デプロイパラメータを設定することで、スタックセットを AWS リージョン 全体に順次デプロイするか、並列デプロイするかを指定できます。複数の にスタックを並行 AWS リージョン してデプロイするようにリージョンの同時実行を指定すると、全体的なデプロイ時間が短縮される可能性があります。

- 並列: スタックセットのデプロイは、指定された失敗許容回数をリージョンのデプロイ失敗が超えない限り、同時に行われます。
- 順次: スタックセットのデプロイは、リージョンのデプロイ失敗が指定された失敗許容回数を超えない限り、一度に 1 つずつ行われます。デフォルトでは、順次デプロイが選択されています。

ConcurrencyMode

必須: いいえ

同時実行モードでは、スタックセットオペレーション時の同時実行レベルの動作を、厳密な耐障害性またはソフトな耐障害性のいずれかを選択できます。厳密な障害耐性では、障害が発生するたびに同時実行性が低下するため、スタックセットの操作に障害が発生するため、デプロイ速度が低下します。ソフト障害耐性は、 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**
>
> このパラメータはアクセスSERVICE_MANAGED許可モデルではオプションであり、アクセスSELF_MANAGED許可モデルでは使用されません。

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の委任管理者として行動するかを指定します。

> **Note**
>
> このパラメータが に設定されている場合はDELEGATED_ADMIN、パイプラインの IAM ロールに アクセスorganizations:ListDelegatedAdministrators許可があることを確認してください。そうしないと、 の実行中にアクションが失敗し、次のような エラーが発生します。 `Account used is not a delegated administrator`

- 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にサービスマネージド型のアクセス許可が使用されます。
- DELEGATED_ADMIN: スタックセットのデプロイでは、委任された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型のアクセス許可が使用されます。

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: 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」を参照してください。

アクション設定の例

次の例は、 アクションのCloudFormationStackInstancesアクション設定を示しています。

自己管理型のアクセス許可モデルの例

次の例は、入力されたデプロイターゲットが AWS アカウント ID であるCloudFormationStackInstancesアクションを示しています111111222222。

YAML

```
Name: my-instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
RunOrder: 2
Configuration:
  DeploymentTargets: '111111222222'
  Regions: 'us-east-1,us-east-2,us-west-1,us-west-2'
  StackSetName: my-stackset
OutputArtifacts: []
```

```
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "my-instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "111111222222",
        "Regions": "us-east-1,us-east-2,us-west-1,us-west-2",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
}
```

サービス管理型のアクセス許可モデルの例

次の例は、デプロイターゲットが AWS Organizations 組織単位 ID であるサービスマネージドアクセス許可モデルのCloudFormationStackInstancesアクションを示していますou-1111-1example。

YAML

```
Name: Instances
ActionTypeId:
  Category: Deploy
  Owner: AWS
  Provider: CloudFormationStackInstances
  Version: '1'
```

```
RunOrder: 2
Configuration:
  DeploymentTargets: ou-1111-1example
  Regions: us-east-1
  StackSetName: my-stackset
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: eu-central-1
```

JSON

```
{
    "Name": "Instances",
    "ActionTypeId": {
        "Category": "Deploy",
        "Owner": "AWS",
        "Provider": "CloudFormationStackInstances",
        "Version": "1"
    },
    "RunOrder": 2,
    "Configuration": {
        "DeploymentTargets": "ou-1111-1example",
        "Regions": "us-east-1",
        "StackSetName": "my-stackset"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "eu-central-1"
}
```

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。

セルフマネージド型のアクセス許可では、 に必要な 2 つの IAM ロールを作成します。 StackSets スタックセットインスタンスをデプロイする AWSCloudFormationStackSetExecutionRole 各アカウント AWSCloudFormationStackSetAdministrationRole で、スタックセットを定義するアカウントのなどの管理者ロールと 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、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 をデプロイ StackSets 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。

サービス管理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、 がユーザーに代わって IAM ロールを作成するため、必要な IAM ロール StackSets を作成する必要はありません。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。「 ユーザーガイド」の AWS 「Organizations で信頼されたアクセスを有効にする」を参照してください。 AWS CloudFormation

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、「」を参照してくださいDescribeStackSet。

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=my-bucket"
```

ParameterValue

- 説明: パラメータに関連付けられた入力値。
- 例 :

```
"ParameterKey=BucketName,ParameterValue=my-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。

以下も参照してください。

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 ユーザーガイドのこのリファレンスの章では、CloudFormation テンプレートパラメータの詳細と例を示します。
- ベストプラクティス - スタックセットのデプロイのベストプラクティスの詳細については、AWS CloudFormation ユーザーガイドの「https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-bestpractices.html」を参照してください。
- AWS CloudFormation API リファレンス – スタックセットオペレーションで使用されるパラメータの詳細については、AWS CloudFormation 「 API リファレンス」の以下の CloudFormation アクションを参照してください。

  - CreateStackSet アクションはスタックセットを作成します。
  - UpdateStackSet アクションは、指定されたアカウントおよびリージョン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 アクションは、セルフマネージド型の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、またはサービスマネージド型の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、 は同じ入力パラメータ UpdateStackInstances で を CreateStackInstances 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 アクションは、セルフマネージド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、またはサービスマネージド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を使用してスタックインスタンスを最新の状態にします。このアクションによって更新されたインスタンスのパラメータをオーバーライドできます。このアクションを使用してインスタンスのサブセットを更新しても、他のスタックインスタンスのステータスは変更されません。
- DescribeStackSetOperation アクションは、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 アクションは、指定されたスタックセットの説明を返します。

AWS CodeBuild

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、ビルド仕様で指定されたコマンドが CodeBuild コンテナ内で実行されます。CodeBuild アクションへの入力アーティファクトとして指定されている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。 は、ビルドアクションまたはテストアクションを提供 CodeBuild できます。詳細については、『AWS CodeBuild ユーザーガイド』を参照してください。

コンソールで 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 ビルドプロジェクトはソースプロバイダーが であることを示しています CodePipeline。コンソールで CodeBuildビルドプロジェクトを作成する場合、 をソースプロバイダー CodePipeline として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、 API リファレンスProjectSourceの「」を参照してください。 AWS CodeBuild

トピック

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、 のビルドプロジェクトの名前です CodeBuild。

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 1 つの名前である必要があります。 はビルド仕様ファイル CodeBuild を検索し、このアーティファクトの解凍されたバージョンを含むディレクトリでビルド仕様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です。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、CodePipeline 「 との統合 CodeBuild 」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、パイプライン内の 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。「アクションの宣言 (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 Systems AWS Manager パラメータストアに既に保存されているパラメータの名前、または Secrets Manager に AWS 既に保存されているシークレットの名前である必要があります。

  > **Note**
  >
  > 機密情報 (特に AWS 認証情報) を保存する場合は、環境変数を使用しないことを強くお勧めします。 CodeBuild コンソールまたは AWS CLI を使用すると、環境変数がプレーンテキストで表示されます。機密の値の場合は、代わりに SECRETS_MANAGER タイプを使用することをお勧めします。

- type: (任意) 環境変数の型。有効な値は PARAMETER_STORE、SECRETS_MANAGER、または PLAINTEXT です。指定しない場合、この値はデフォルトで PLAINTEXT になります。

> **Note**
>
> type 環境変数設定に name、value、および を入力する場合、特に環境変数に CodePipeline 出力変数構文が含まれている場合は、設定の値フィールドの 1000 文字の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、 API リファレンス EnvironmentVariableの AWS CodeBuild 「」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使用した CodeBuild アクションの例については、「」を参照してください例: CodeBuild 環境変数で変数を使用する BranchName 。

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 はビルド仕様ファイルを検索し、プライマリソースアーティファクトの ディレクトリからビルド仕様コマンドを実行します。 CodeBuild アクションに複数の入力ソースを指定する場合、このアーティファクトは 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 CodePipeline。

  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 ビルド仕様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。出力アーティファクトが 1 つだけ定義されている場合、このアーティファクトはビルド仕様ファイルの artifacts セ

クションの直下で定義できます。複数の出力アーティファクトを指定する場合、参照されるすべてのアーティファクトは、ビルド仕様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。の出力アーティファクトの名前は、ビルド仕様ファイルのアーティファクト識別子と一致する CodePipeline 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 CodePipeline の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 EnvironmentVariable」の「」を参照してください。

での CodeBuild 環境変数の使用の詳細については CodePipeline、「」の例を参照してくださいCodeBuild アクション出力変数。で使用できる環境変数のリストについては CodeBuild、「 ユーザーガイド」の「ビルド環境の環境変数AWS CodeBuild 」を参照してください。

アクションの宣言 (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": [
                {
                    "Name": "MyPipeline-BuildArtifact"
                }
            ],
            "InputArtifacts": [
```

```
                {
                    "Name": "MyApplicationSource1"
                },
                {
                    "Name": "MyApplicationSource2"
                }
            ]
        }
    ]
}
```

以下も参照してください。

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS CodeBuild ユーザーガイド – CodeBuild アクションを使用したパイプラインの例については、「 CodePipeline で CodeBuild を使用してコードをテストし、ビルドを実行する」を参照してください。複数の入力アー CodeBuild ティファクトと出力アーティファクトを含むプロジェクトの例については、CodePipeline「 CodeBuild と複数の入力ソースと出力アーティファクトのサンプル」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: で Android アプリを構築して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、 と を使用して Android CodeBuild アプリを構築およびテストする GitHub ソースを持つパイプラインを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 AWS Device Farm。
- のビルド仕様リファレンス CodeBuild — このリファレンストピックでは、 CodeBuild ビルド仕様ファイルを理解するための定義と例を示します。で使用できる環境変数のリストについては CodeBuild、「 ユーザーガイド」の「ビルド環境の環境変数AWS CodeBuild 」を参照してください。

CodeCommit

設定された CodeCommit リポジトリとブランチで新しいコミットが行われたときにパイプラインを開始します。

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、 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 CodeCommit CloudWatch イベントルール CodePipeline を作成します。

CodeCommit アクションを通じ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 — 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 CodeCommit ソースアクションを設定します。
- [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、リポジトリおよび関連する Git メタデータのクローンを作成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 参照を 以外の CodeBuild アクションに渡すと、エラーが発生します。

トピック

アクションタイプ

- カテゴリ:`Source`
- 所有者: `AWS`
- プロバイダー: `CodeCommit`
- バージョン: `1`

設定パラメータ

RepositoryName

必須: はい

ソースの変更が検出されるリポジトリの名前。

BranchName

必須: はい

ソースの変更が検出されるブランチの名前。

PollForSourceChanges

必須: いいえ

PollForSourceChanges は、ソースの変更についてリポジトリを CodePipeline CodeCommit ポーリングするかどうかを制御します。代わりに CloudWatch 、イベント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 イベントの設定の詳細については、ポーリングパイプラインの移行 (CodeCommit ソース) (CLI)「」または「」を参照してください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) 。

> **Important**
>
> CloudWatch イベントルールを設定する場合は、パイプラインの実行が重複しないように false PollForSourceChangesを 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、 はリポジトリにソースの変更を CodePipeline ポーリングします。

> **Note**
>
> を省略するとPollForSourceChanges、 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、ソースの変更についてリポジトリをポーリング CodePipeline しません。ソースの変更を検出するように CloudWatch イベントルールを設定する場合は、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` オプションはダウンストリーム CodeBuild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、「」に示すように、 CodeBuild サービスロールにアクセス`codecommit:GitPull`許可を追加する必要がありますソースアクションの CodeBuild GitClone CodeCommitアクセス許可を追加する。また、 CodePipeline「」に示すように、サービスロールにアクセス`codecommit:GetRepository`許可を追加する必要があります CodePipeline サービスロールにアクセス許可を追加する。[フルクローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: CodeCommit パイプラインソースでフルクローンを使用する を参照してください。

入力アーティファクト

- アーティファクトの数: `0`
- 説明: 入力アーティファクトは、このアクションタイプには適用されません。

出力アーティファクト

- アーティファクトの数: `1`
- 説明: このアクションの出力アーティファクトは、パイプライン実行のソースリビジョンとして指定されたコミットで設定されたリポジトリとブランチの内容を含む ZIP ファイルです。リポジトリから生成された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。ソースコードコミット ID は、トリガーされたパイプライン実行のソースリビジョン CodePipeline として 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」を参照してください。

CommitId

パイプラインの実行をトリガーした CodeCommit 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

パイプラインをトリガーしたコミットが行われた CodeCommit リポジトリの名前。

BranchName

ソースの変更が行われた CodeCommit 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)。

Git での著者とコミッターの違いに関する詳細については、Scott Chacon と Ben Straub による Pro Git の「コミット履歴の表示」を参照してください。

CommitterDate

コミットがコミットされた日付 (タイムスタンプ形式)。

Git での著者とコミッターの違いに関する詳細については、Scott Chacon と Ben Straub による Pro Git の「コミット履歴の表示」を参照してください。

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

```
Actions:
  - OutputArtifacts:
      - Name: Artifact_MyWebsiteStack
    InputArtifacts: []
    Name: source
    Configuration:
      RepositoryName: MyWebsite
      BranchName: main
      PollForSourceChanges: 'false'
    RunOrder: 1
    ActionTypeId:
      Version: '1'
      Provider: CodeCommit
      Category: Source
      Owner: AWS
  Name: Source
```

JSON

```
{
    "Actions": [
        {
            "OutputArtifacts": [
                {
                    "Name": "Artifact_MyWebsiteStack"
                }
            ],
            "InputArtifacts": [],
            "Name": "source",
            "Configuration": {
                "RepositoryName": "MyWebsite",
                "BranchName": "main",
                "PollForSourceChanges": "false"
            },
            "RunOrder": 1,
            "ActionTypeId": {
                "Version": "1",
```

```
                "Provider": "CodeCommit",
                "Category": "Source",
                "Owner": "AWS"
            }
        }
    ],
    "Name": "Source"
},
```

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"
}
```

以下も参照してください。

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とサンプル CodeDeploy アプリケーションおよびデプロイグループを提供します。このチュートリアルを使用して、Amazon EC2 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

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、 CodeDeploy デプロイプラットフォーム CodePipeline が Amazon EC2 である のデプロイアクションについて説明します。Amazon Elastic Container Service の Blue/Green デプロイアクションに関する CodeDeployリファレンス情報については CodePipeline、「」を参照してくださいAmazon Elastic Container Service と CodeDeploy Blue-Green。

トピック

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

で作成したアプリケーションの名前 CodeDeploy。

DeploymentGroupName

必須: はい

で作成したデプロイグループ CodeDeploy。

入力アーティファクト

- アーティファクトの数: 1
- 説明： が以下を決定するために CodeDeploy 使用する AppSpec ファイル。
  - Amazon S3 または のアプリケーションリビジョンからインスタンスにインストールする内容 GitHub。

- 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、「ファイルCodeDeploy AppSpec リファレンス」を参照してください。

出力アーティファクト

- アーティファクトの数: `0`
- 説明: 出力アーティファクトは、このアクションタイプには適用されません。

アクションの宣言

YAML

```
Name: Deploy
Actions:
  - Name: Deploy
    ActionTypeId:
      Category: Deploy
      Owner: AWS
      Provider: CodeDeploy
      Version: '1'
    RunOrder: 1
    Configuration:
      ApplicationName: my-application
      DeploymentGroupName: my-deployment-group
    OutputArtifacts: []
    InputArtifacts:
      - Name: SourceArtifact
    Region: us-west-2
    Namespace: DeployVariables
```

JSON

```
{
    "Name": "Deploy",
    "Actions": [
        {
            "Name": "Deploy",
```

```
            "ActionTypeId": {
                "Category": "Deploy",
                "Owner": "AWS",
                "Provider": "CodeDeploy",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "ApplicationName": "my-application",
                "DeploymentGroupName": "my-deployment-group"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "Region": "us-west-2",
            "Namespace": "DeployVariables"
        }
    ]
},
```

以下も参照してください。

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をデプロイするためのソースバケット、EC2 インスタンス、および CodeDeploy リソースの作成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 CodeDeploy デプロイする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、 CodeDeploy および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 CodeDeploy デプロイする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を示します。

CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用

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 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 GitHub。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は、トリガーでパイプラインを開始できる Git 設定を使用するように設定できます。パイプラインのトリガー設定をトリガーでフィルタリングするように設定するには、「」で詳細を参照してくださいコードプッシュまたはプルリクエストでトリガーをフィルタリングする。

> **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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」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プロバイダー経由)

  > **Note**
  >
  > Bitbucket Cloudリポジトリへの接続を作成できます。Bitbucket サーバーなど、インストールされている Bitbucket プロバイダーのタイプはサポートされていません。

- > **Note**
  >
  > Bitbucket ワークスペースを使用している場合、接続を作成するには管理者アクセス権が必要です。

- GitHub および GitHub Enterprise Cloud (CodePipeline コンソールの GitHub （ バージョン 2) 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 組織内にある場合は、接続を作成する組織の所有者である必要があります。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`プロバイダー経由)
- GitLab.com (コンソールのGitLabプロバイダーオプション CodePipelineまたは CLI の`GitLab`プロバイダー経由)

  > **Note**
  >
  > で所有者ロールを持つリポジトリへの接続を作成し GitLab、その接続を などのリソースを持つリポジトリで使用できます CodePipeline。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 (Enterprise Edition または Community Edition) のセルフマネージドインストール (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 CodePipelineまたは CLI の`GitLabSelfManaged`プロバイダーを使用)

> **Note**
>
> 各接続は、そのプロバイダーのすべてのリポジトリをサポートします。プロバイダーの種類ごとに新しい接続を作成する必要があります。

Connections により、パイプラインはサードパーティープロバイダーのインストールアプリを通じてソースの変更を検出することができます。例えば、ウェブフックは 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。接続は、GitHub プッシュタイプイベントにサブスクライブするリポジトリウェブフックを GitHub アプリにインストールします。

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、リポジトリおよび関連する Git メタデータのクローンを作成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 参照を 以外の CodeBuild アクションに渡すと、エラーが発生します。

CodePipeline 接続を作成するときに、 AWS コネクタインストールアプリをサードパーティーアカウントに追加するように求められます。`CodeStarSourceConnection` アクションを介して接続する前に、サードパーティープロバイダのアカウントとリポジトリを作成しておく必要があります。

> **Note**
>
> AWS CodeStar 接続を使用するために必要なアクセス許可を持つロールにポリシーを作成またはアタッチするには、Connections アクセス許可リファレンスを参照してください。CodePipeline サービスロールが作成された時期によっては、 AWS CodeStar 接続をサポートするためにアクセス許可を更新する必要がある場合があります。手順については、「CodePipeline サービスロールにアクセス許可を追加する」を参照してください。

> **Note**
>
> 欧州 (ミラノ) で接続を使用するには AWS リージョン、以下を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

- カテゴリ:Source
- 所有者: AWS
- プロバイダー: CodeStarSourceConnection
- バージョン: 1

設定パラメータ

ConnectionArn

必須: はい

ソースプロバイダに対して設定および認証された接続 ARN。

FullRepositoryId

必須: はい

ソースの変更が検出される所有者とリポジトリの名前。

例: `some-user/my-repo`

> **Important**
>
> FullRepositoryId 値に対して正しい大文字と小文字を維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 の場合`My-Repo`、 の推奨値は FullRepositoryIdです`some-user/My-Repo`。

BranchName

必須: はい

ソースの変更が検出されるブランチの名前。

OutputArtifactFormat

必須: いいえ

出力アーティファクト形式を指定します。`CODEBUILD_CLONE_REF` または `CODE_ZIP` のいずれかになります。指定しない場合、デフォルトの `CODE_ZIP` が使用されます。

> **Important**
>
> `CODEBUILD_CLONE_REF` オプションはダウンストリームアクションでのみ使用できます CodeBuild。
> このオプションを選択した場合は、「」に示すように、 CodeBuild プロジェクトサービスロールのアクセス許可を更新する必要がありますBitbucket、、 GitHub Enterprise

Server GitHub、または GitLab.com への接続の CodeBuild GitClone アクセス許可を追加する。[完全クローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: GitHub パイプラインソースでフルクローンを使用する を参照してください。

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` : 新しいコミットでパイプライン CodePipeline を自動的に開始します。
- `false`: CodePipeline 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。ソースコードコミット ID は、トリガーされたパイプライン実行のソースリビジョン CodePipeline として 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、「」に示すように、 CodeBuild プロジェクトサービスロールのアクセス許可を更新する必要がありますトラブルシューティング CodePipeline。[フルクローン] オプションを使用する方法を示すチュートリアルについては、チュートリアル: GitHub パイプラインソースでフルクローンを使用する を参照してください。

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」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。

アクションの宣言

次の例では、CODE_ZIP ARN との接続で出力アーティファクトが arn:aws:codestar-connections:region:*account-id*:connection/*connection-id* のデフォルト ZIP 形式に設定されています。

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: AWS
      Category: Source
      Provider: CodeStarSourceConnection
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      ConnectionArn: "arn:aws:codestar-connections:region:account-
id:connection/connection-id"
      FullRepositoryId: "some-user/my-repo"
      BranchName: "main"
      OutputArtifactFormat: "CODE_ZIP"
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
                "Owner": "AWS",
                "Category": "Source",
                "Provider": "CodeStarSourceConnection"
            },
            "OutputArtifacts": [
                {
```

```
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "ConnectionArn": "arn:aws:codestar-connections:region:account-
id:connection/connection-id",
                "FullRepositoryId": "some-user/my-repo",
                "BranchName": "main",
                "OutputArtifactFormat": "CODE_ZIP"
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

インストールアプリケーションのインストールと接続の作成

コンソールを使用してサードパーティーのリポジトリに新しい接続を初めて追加する場合は、リポジトリ CodePipeline へのアクセスを許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。

以下も参照してください。

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS::CodeStarConnections::Connection – AWS CodeStar Connections リソースの 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。

- AWS CodeStar Connections API リファレンス — AWS CodeStar Connections API リファレンスには、使用可能な接続アクションのリファレンス情報が記載されています。
- 接続でサポートされているソースアクションでパイプラインを作成するステップについては、以下を参照してください。
  - Bitbucket Cloud の場合は、コンソールの [Bitbucket] オプションまたは CLI の `CodestarSourceConnection` アクションを使用します。Bitbucket Cloud への接続 を参照してください。
  - GitHub および GitHub Enterprise Cloud の場合は、コンソールのGitHub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。GitHub 接続 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 プロバイダーオプションまたは CLI の `CodestarSourceConnection`アクションを使用します。GitHub Enterprise Server 接続 を参照してください。
  - GitLab.com の場合は、 コンソールの GitLab provider オプションを使用するか、 CLI の `GitLab` provider で `CodestarSourceConnection`アクションを使用します。GitLab.com 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入門チュートリアルを表示するには、「接続の開始方法」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示すチュートリアルについては、「」を参照してくださいチュートリアル: GitHub パイプラインソースでフルクローンを使用する。

AWS Device Farm

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、「Device Farm での AWS Device Farm」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
- アクションの宣言
- 以下も参照してください。

アクションタイプ

- カテゴリ:Test
- 所有者: AWS
- プロバイダー: DeviceFarm
- バージョン: 1

設定パラメータ

AppType

必須: はい

テストする OS とアプリケーションのタイプ。有効な値のリストを次に示します。

- iOS
- Android
- Web

ProjectId

必須: はい

Device Farm プロジェクト ID。

プロジェクト ID を見つけるには、Device Farm コンソールでプロジェクトを選択します。ブラウザで、新しいプロジェクトの URL をコピーします。URL には、プロジェクト ID が含まれます。プロジェクト ID は、projects/ 以降の URL の値です。次の例で、プロジェクト ID は eec4905f-98f8-40aa-9afc-4c1cfexample です。

```
https://<region-URL>/devicefarm/home?region=us-west-2#/projects/
eec4905f-98f8-40aa-9afc-4c1cfexample/runs
```

アプリケーション

必須: はい

入力アーティファクト内のアプリケーションファイルの名前と場所。例: `s3-ios-test-1.ipa`

TestSpec

条件付き: はい

入力アーティファクト内のテストスペック定義ファイルの場所。これはカスタムモードのテストに必要です。

DevicePoolArn

必須: はい

Device Farm デバイスプールの ARN。

上位デバイスの ARNs など、プロジェクトで使用できるデバイスプール ARN を取得するには、AWS CLI を使用して次のコマンドを入力します。

```
aws devicefarm list-device-pools --arn arn:aws:devicefarm:us-
west-2:account_ID:project:project_ID
```

TestType

必須: はい

テストでサポートされるテストフレームワークを指定します。`TestType` の有効な値のリストを次に示します。

- APPIUM_JAVA_JUNIT
- APPIUM_JAVA_TESTNG
- APPIUM_NODE
- APPIUM_RUBY
- APPIUM_PYTHON
- APPIUM_WEB_JAVA_JUNIT
- APPIUM_WEB_JAVA_TESTNG
- APPIUM_WEB_NODE

- APPIUM_WEB_RUBY
- APPIUM_WEB_PYTHON
- BUILTIN_FUZZZ
- INSTRUMENTATION
- XCTEST
- XCTEST_UI

> **Note**
>
> の アクションでは、、CodePipelineWEB_PERFORMANCE_PROFILE、REMOTE_ACCESS_RECORDおよび のテストタイプはサポートされていませんREMOTE_ACCESS_REPLAY。

Device Farm のテストタイプついては、[AWS Device Farm でのテストタイプの操作] を参照してください。

RadioBluetoothEnabled

必須: いいえ

テストの開始時に Bluetooth を有効にするかどうかを示すブール値。

RecordAppPerformanceData

必須: いいえ

テスト中に CPU、FPS、メモリパフォーマンスなどのデバイスパフォーマンスデータを記録するかどうかを示すブール値。

RecordVideo

必須: いいえ

テスト中にビデオを記録するかどうかを示すブール値。

RadioWifiEnabled

必須: いいえ

テストの開始時に Wi-Fi を有効にするかどうかを示すブール値。

RadioNfcEnabled

必須: いいえ

テストの開始時に NFC を有効にするかどうかを示すブール値。

RadioGpsEnabled

必須: いいえ

テストの開始時に GPS を有効にするかどうかを示すブール値。

テスト

必須: いいえ

ソースの場所にあるテスト定義ファイルの名前とパス。パスは、テストの入力アーティファクトのルートに関連します。

FuzzEventCount

必須: いいえ

ファズテストが実行するユーザーインターフェイスイベントの数で、1 から 10000 の間で指定します。

FuzzEventThrottle

必須: いいえ

ファズテストが次のユーザーインターフェイスイベントを実行する前に待機するミリ秒数で、1 から 1000 の間で指定します。

FuzzRandomizerSeed

必須: いいえ

ユーザインタフェースイベントをランダム化するために使用するファズテストのシード。後続のファズテストに同じ番号を使用すると、同じイベントシーケンスになります。

CustomHostMachineArtifacts

必須: いいえ

ホストマシン上でカスタムアーティファクトが格納される場所。

CustomDeviceArtifacts

必須: いいえ

カスタムアーティファクトが保存されるデバイス上の場所。

UnmeteredDevicesOnly

必須: いいえ

この手順でテストを実行するときに、測定されていないデバイスのみを使用するかどうかを示すブール値。

JobTimeoutMinutes

必須: いいえ

テスト実行がタイムアウトになるまでにデバイスごとに実行される分数。

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。

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。

アクションの宣言

YAML

```
Name: Test
Actions:
  - Name: TestDeviceFarm
    ActionTypeId: null
    category: Test
    owner: AWS
    provider: DeviceFarm
    version: '1'
RunOrder: 1
Configuration:
  App: s3-ios-test-1.ipa
  AppType: iOS
  DevicePoolArn: >-
    arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-d7d7-48a5-ba5c-b33d66efa1f5
  ProjectId: eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE
  TestType: APPIUM_PYTHON
  TestSpec: example-spec.yml
OutputArtifacts: []
InputArtifacts:
  - Name: SourceArtifact
Region: us-west-2
```

JSON

```
{
    "Name": "Test",
    "Actions": [
        {
            "Name": "TestDeviceFarm",
            "ActionTypeId": null,
            "category": "Test",
            "owner": "AWS",
            "provider": "DeviceFarm",
            "version": "1"
        }
    ],
    "RunOrder": 1,
    "Configuration": {
        "App": "s3-ios-test-1.ipa",
```

```
        "AppType": "iOS",
        "DevicePoolArn": "arn:aws:devicefarm:us-west-2::devicepool:0EXAMPLE-
d7d7-48a5-ba5c-b33d66efa1f5",
        "ProjectId": "eec4905f-98f8-40aa-9afc-4c1cfEXAMPLE",
        "TestType": "APPIUM_PYTHON",
        "TestSpec": "example-spec.yml"
    },
    "OutputArtifacts": [],
    "InputArtifacts": [
        {
            "Name": "SourceArtifact"
        }
    ],
    "Region": "us-west-2"
},
```

以下も参照してください。

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: で Android アプリを構築して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、 と Device Farm を使用して Android アプリを構築およびテストする GitHub ソースを持つパイプラインを作成するためのサンプルビルド仕様ファイル CodeBuild とサンプルアプリケーションを提供します。
- チュートリアル: で iOS アプリケーションをテストするパイプラインを作成する AWS Device Farm — このチュートリアルでは、Device Farm でビルドされた iOS アプリケーションをテストする Amazon S3 ソースでパイプラインを作成するためのサンプルアプリケーションを提供します。

AWS Lambda

パイプラインのアクションとして Lambda 関数を実行できます。この関数への入力であるイベントオブジェクトを使用して、関数はアクション設定、入力アーティファクトの場所、出力アー

ティファクトの場所、およびアーティファクトへのアクセスに必要なその他の情報にアクセスできます。Lambda 呼び出し関数に渡されるイベントの例については、JSON イベントの例 を参照してください。Lambda 関数の実装の一部として、PutJobSuccessResult API または PutJobFailureResult API への呼び出しが必要です。それ以外の場合、このアクションの実行は、アクションがタイムアウトするまでハングします。アクションの出力アーティファクトを指定する場合、関数の実装の一部として S3 バケットにアップロードする必要があります。

> **Important**
>
> が Lambda CodePipeline に送信する JSON イベントはログに記録しないでください。ログにユーザー認証情報が記録される可能性があるため CloudWatch です。 CodePipeline ロールは JSON イベントを使用して、 artifactCredentialsフィールドの Lambda に一時的な認証情報を渡します。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、PutJobSuccessResult API リクエストの `outputVariables`セクションに含まれるすべてのキーと値のペアを変数として生成します。

の変数の詳細については、 CodePipeline「」を参照してください変数。

アクション設定の例

YAML

```
Name: Lambda
Actions:
  - Name: Lambda
    ActionTypeId:
      Category: Invoke
      Owner: AWS
      Provider: Lambda
      Version: '1'
    RunOrder: 1
    Configuration:
      FunctionName: myLambdaFunction
      UserParameters: 'http://192.0.2.4'
    OutputArtifacts: []
    InputArtifacts: []
    Region: us-west-2
```

JSON

```
{
    "Name": "Lambda",
    "Actions": [
        {
            "Name": "Lambda",
            "ActionTypeId": {
                "Category": "Invoke",
                "Owner": "AWS",
                "Provider": "Lambda",
                "Version": "1"
            },
            "RunOrder": 1,
            "Configuration": {
                "FunctionName": "myLambdaFunction",
                "UserParameters": "http://192.0.2.4"
            },
            "OutputArtifacts": [],
            "InputArtifacts": [],
            "Region": "us-west-2"
        }
    ]
},
```

JSON イベントの例

Lambda アクションは、ジョブ ID、パイプラインアクション設定、入力および出力アーティファクトの場所、およびアーティファクトの暗号化情報を含む JSON イベントを送信します。ジョブワーカーは、これらの詳細にアクセスして Lambda アクションを完了します。詳細については、ジョブの詳細を参照してください。以下に示しているのは、イベントの例です。

```
{
    "CodePipeline.job": {
        "id": "11111111-abcd-1111-abcd-111111abcdef",
        "accountId": "111111111111",
        "data": {
            "actionConfiguration": {
                "configuration": {
                    "FunctionName": "MyLambdaFunction",
                    "UserParameters": "input_parameter"
```

```
                }
            },
            "inputArtifacts": [
                {
                    "location": {
                        "s3Location": {
                            "bucketName": "bucket_name",
                            "objectKey": "filename"
                        },
                        "type": "S3"
                    },
                    "revision": null,
                    "name": "ArtifactName"
                }
            ],
            "outputArtifacts": [],
            "artifactCredentials": {
                "secretAccessKey": "secret_key",
                "sessionToken": "session_token",
                "accessKeyId": "access_key_ID"
            },
            "continuationToken": "token_ID",
            "encryptionKey": {
              "id": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
              "type": "KMS"
            }
        }
    }
}
```

JSON イベントは、 の Lambda アクションに次のジョブの詳細を提供します CodePipeline。

- `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 (例えば、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) を指定できます。詳細については、「」を参照してくださいArtifactRevision。
    - 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 (例えば、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) を指定できます。詳細については、「」を参照してくださいArtifactRevision。
    - 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。

以下も参照してください。

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS CloudFormation ユーザーガイド – パイプラインの Lambda アクションと AWS CloudFormation アーティファクトの詳細については、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」、「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。
- のパイプラインで AWS Lambda 関数を呼び出す CodePipeline - 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。

Snyk アクション構造リファレンス

の Snyk 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 CodePipeline を自動化します。Snyk 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。

> **Note**

トピック

アクションタイプ ID

- カテゴリ:Invoke
- 所有者: ThirdParty
- プロバイダー: Snyk
- バージョン: 1

例：

```
{
    "Category": "Invoke",
    "Owner": "ThirdParty",
    "Provider": "Snyk",
    "Version": "1"
},
```

入力アーティファクト

- アーティファクトの数: 1
- 説明: Invoke アクションの入力アーティファクトを構成するファイル。

出力アーティファクト

- アーティファクトの数: 1
- 説明: Invoke アクションの入力アーティファクトを構成するファイル。

以下も参照してください。

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。

- で Snyk アクションを使用する方法の詳細については CodePipeline、「Snyk CodePipeline を使用した での脆弱性スキャンの自動化」を参照してください。

AWS Step Functions

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 で実行されるため、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あります。詳細については、「Lambda デベロッパーガイド」の「Lambda クォータ」を参照してください。

アクションタイプ

- カテゴリ:`Invoke`
- 所有者: `AWS`
- プロバイダー: `StepFunctions`
- バージョン: `1`

設定パラメータ

StateMachineArn

必須: はい

呼び出されるステートマシンの Amazon リソースネーム (ARN)。

ExecutionNamePrefix

必須: いいえ

デフォルトでは、アクション実行 ID がステートマシンの実行名として使用されます。プレフィックスが指定されている場合は、アクション実行 ID の先頭にハイフンが付加され、ステートマシンの実行名として使用されます。

```
myPrefix-1624a1d1-3699-43f0-8e1e-6bafd7fde791
```

Express ステートマシンの場合、名前には 0 ~ 9、A ~ Z、a ~ z、- および _ のみを含める必要があります。

InputType

必須: いいえ

- [Literal (リテラル)] (デフォルト): 指定すると、[Input (入力)] フィールドの値がステートマシンの入力に直接渡されます。

  リテラルを選択した場合の、Input フィールドの入力例。

  ```
  {"action": "test"}
  ```

- FilePath: Input フィールドで指定された入力アーティファクト内のファイルの内容は、ステートマシンの実行の入力として使用されます。が に設定されている場合InputType、入力アーティファクトが必要ですFilePath。

  がFilePath選択されているときの入力フィールドのエントリ例 :

  ```
  assets/input.json
  ```

入力

必須: 条件による

- リテラル : InputTypeがリテラル (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 に設定されている場合FilePath、このフィールドは必須です。

  が に設定されている場合はInputType、入力アーティファクトも必要ですFilePath。

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 に設定されている場合FilePath、このアーティファクトが必要であり、ステートマシン実行の入力をソースするために使用されます。

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完了した後に、Step Functions DescribeExecution API レスポンスの `output`プロパティから取得されます。
  - 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」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。

アクション設定の例

デフォルト入力の例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix"
    }
}
```

リテラル入力の例

YAML

```
Name: ActionName
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine
  ExecutionNamePrefix: my-prefix
  Input: '{"action": "test"}'
```

JSON

```
{
    "Name": "ActionName",
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "Input": "{\"action\": \"test\"}"
    }
}
```

入力ファイルの例

YAML

```
Name: ActionName
InputArtifacts:
  - Name: myInputArtifact
ActionTypeId:
  Category: Invoke
  Owner: AWS
  Version: 1
  Provider: StepFunctions
OutputArtifacts:
  - Name: myOutputArtifact
Configuration:
  StateMachineArn: 'arn:aws:states:us-east-1:111122223333:stateMachine:HelloWorld-
StateMachine'
  ExecutionNamePrefix: my-prefix
  InputType: FilePath
  Input: assets/input.json
```

JSON

```
{
    "Name": "ActionName",
    "InputArtifacts": [
        {
            "Name": "myInputArtifact"
        }
    ],
    "ActionTypeId": {
        "Category": "Invoke",
        "Owner": "AWS",
        "Version": 1,
        "Provider": "StepFunctions"
    },
    "OutputArtifacts": [
        {
            "Name": "myOutputArtifact"
        }
    ],
    "Configuration": {
```

```
        "StateMachineArn": "arn:aws:states:us-
east-1:111122223333:stateMachine:HelloWorld-StateMachine",
        "ExecutionNamePrefix": "my-prefix",
        "InputType": "FilePath",
        "Input": "assets/input.json"
    }
}
```

Behavior

リリース中、 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 CodePipeline を実行します。

InputType がリテラル に設定されている場合、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、「Step Functions API StartExecution」を参照してください。

InputType が に設定されている場合FilePath、アクションは入力アーティファクトを解凍し、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 を指定すると、Input フィールドが必須になり、入力アーティファクトが存在する必要があります。存在しない場合、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ると、 は実行がターミナルステータスになるまで `DescribeExecution` API を CodePipeline 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完了した後、Step Functions DescribeExecution API レスポンスの `output`プロパティから取得されます。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 Step Functions アクションがタイムアウト (デフォルトは 7 日) に達する前にステートマシンの実行が終了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。
- でアクションの実行が開始されると CodePipeline、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を開始できない場合、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。アクションは、ステートマシンの実行が完了するまでにかかる時間やその結果 CodePipeline に関係なく、 で成功します。

以下も参照してください。

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS Step Functions デベロッパーガイド – ステートマシン、実行、およびステートマシンの入力については、AWS Step Functions 「 デベロッパーガイド」を参照してください。
- チュートリアル: パイプライン AWS Step Functions で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。

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サービスプロバイダーは、統合モデルを使用して、 で使用するアクションタイプを実装します CodePipeline。

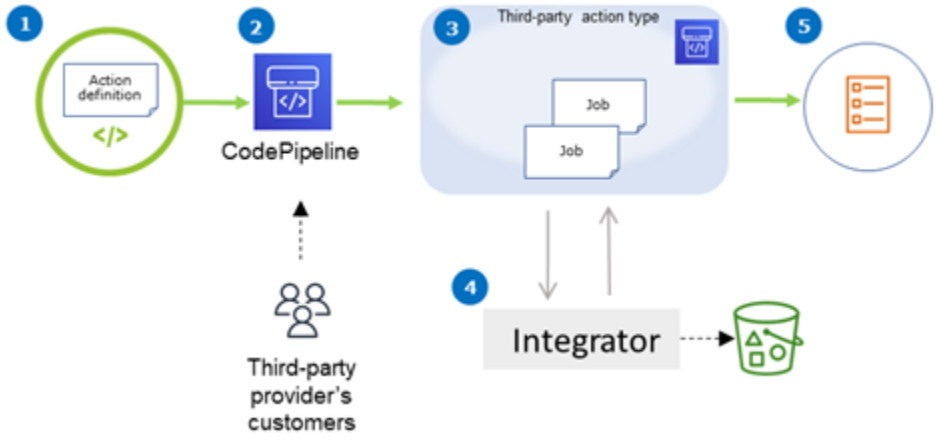
このリファレンスは、 でサポートされている統合モデルで管理されるアクションタイプを計画または操作する場合に使用します CodePipeline。

サードパーティーのアクションタイプを とのパートナー統合として認証するには CodePipeline、AWS パートナーネットワーク (APN) を参照してください。この情報は、AWS CLI リファレンス を補足するものです。

トピック

インテグレータとサードパーティーのアクションタイプがどのように機能するか

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。インテグレーターはジョブリクエストを管理し、 でアクションを実行します CodePipeline。次の図表は、顧客がパイプラインで使用するために作成されたサードパーティーのアクションタイプを示しています。顧客がアクションを設定すると、アクションが実行され、インテグレータのアクションエンジンによって処理されるジョブリクエストが作成されます。



図表に示す内容は以下のステップです。

1. アクション定義が登録され、 で利用可能になります CodePipeline。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. プロバイダーのお客様は、 で アクションを選択して設定します CodePipeline。
3. アクションが実行され、ジョブは でキューに入れられます CodePipeline。ジョブが で準備できたら CodePipeline、ジョブリクエストを送信します。
4. インテグレーター (サードパーティーポーリング API または Lambda 関数のジョブワーカー) は、ジョブリクエストを取得し、確認を返し、アクションのアーティファクトを処理します。
5. インテグレーターは、ジョブ結果と継続トークンを使用して、成功/失敗の出力 (ジョブワーカーは成功/失敗 API を使用するか、Lambda 関数が成功/失敗の出力を送信します) を返します。

アクションタイプのリクエスト、表示、および更新に使用できる手順については、 アクションタイプの使用 を参照してください。

概念

このセクションでは、サードパーティーのアクションタイプについて以下の用語を使用します。

アクションタイプ

同じ継続的デリバリーのワークロードを実行するパイプラインで再利用できる反復可能なプロセス。アクションタイプは、`Owner`、`Category`、`Provider`、および `Version` 。例:

```
{
```

```
            "Category": "Deploy",
            "Owner": "AWS",
            "Provider": "CodeDeploy",
            "Version": "1"
        },
```

同じタイプのアクションはすべて、同じ実装を共有します。

アクション

アクションタイプの単一のインスタンス。パイプラインのステージ内で発生する個別のプロセスの1つです。これには、通常、このアクションが実行されるパイプラインに固有のユーザー値が含まれます。

アクションの定義

アクションおよび入出力アーティファクトの構成に必要なプロパティを定義するアクションタイプのスキーマ。

アクションの実行

顧客のパイプラインでのアクションが成功したかどうかを判断するために実行されたジョブの集まり。

アクション実行エンジン

アクションタイプで使用される統合タイプを定義するアクション実行設定のプロパティ。有効な値は、`JobWorker` および `Lambda` です。

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`と に対応する 2 つの統合タイプをサポートします`Lambda`。

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: この統合モデルは、 でアクションタイプを操作するための推奨方法です CodePipeline。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、アクションの実行時に CodePipeline APIs に問い合わせてジョブリクエストを処理するように設定されたジョブワーカーを使用します。

比較のために、次の表に 2 つのモデルの特徴を示します。

	Lambda 統合モデル	ジョブワーカー統合モデル
説明	インテグレータは、統合を Lambda 関数として書き込みます。Lambda 関数は、アクションに使用できるジョブがある CodePipeline たびに 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。次に、ジョブワーカーはジョブを実行し、API CodePipeline を使用してジョブ結果を に送信します。CodePipeline APIs
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

	Lambda 統合モデル	ジョブワーカー統合モデル
		APIs とやり取りする必要があります。
Opsの努力	インフラストラクチャは AWS リソースにすぎないため、運用の労力が少なくなります。	ジョブワーカーはスタンドアロンのハードウェアを必要とするため、オペレーションの労力が高くなります。
最大ジョブのランタイム	統合を 15 分以上アクティブに実行する必要がある場合は、このモデルを使用できません。このアクションは、プロセスを開始する (顧客のコードアーティファクトの構築を開始するなど)、終了時に結果を返す必要があるインテグレータを対象としています。インテグレータがビルドの終了を待ち続けることはお勧めしません。代わりに、継続 を返します。 CodePipelineインテグレータのコードから継続を受信し、終了するまでジョブをチェックすると、 はさらに 30 秒で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。

からの入力を処理するように Lambda 関数を更新する CodePipeline

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。

次のパラメータを使用して、 からの入力を処理するように Lambda 関数を更新します CodePipeline。

形式:

- `jobId`:
  - システムによって生成されたジョブの固有の ID 。
  - 型: 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。
  - 型: 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。
  - 目次 : データ型のリスト Artifact、オプションで提供
- outputArtifacts:
  - アクションの出力に関する情報のリスト。
  - 目次:データ型のリスト Artifact、オプションで提供
- actionCredentials:
  - AWS セッション認証情報オブジェクトを表します。これらの認証情報は、 AWS STSによって発行される一時的な認証情報です。これらは、パイプラインのアーティファクトをに保存するために使用される S3 バケット内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 CodePipeline。

    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。
  - 型: 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。
    - 型: 文字列
  - type:
    - キーなどの暗号化 AWS KMS キーのタイプ。

    - 型: 文字列
    - 有効な値: KMS
- Artifact:
  - name:
    - アーティファクトの名前。
    - タイプ : 文字列、オプションで提供
  - revision:
    - アーティファクトのリビジョン ID 。オブジェクトのタイプに応じて、コミット ID (GitHub) またはリビジョン ID (Amazon S3) を指定できます。
    - 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。
    - 型: 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。
    - 型: 文字列
- AWSSessionCredentials:
  - accessKeyId:

- セッションのアクセスキー。
- 型: 文字列

- secretAccessKey:
  - セッションのシークレットアクセスキー。
  - 型: 文字列
- sessionToken:
  - セッションのトークン。
  - 型: 文字列

例：

```
{
    "jobId": "01234567-abcd-abcd-abcd-012345678910",
    "accountId": "012345678910",
    "data": {
        "actionConfiguration": {
            "key1": "value1",
            "key2": "value2"
        },
        "encryptionKey": {
            "id": "123-abc",
            "type": "KMS"
        },
        "inputArtifacts": [
            {
                "name": "input-art-name",
                "location": {
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "inputBucket",
                        "objectKey": "inputKey"
                    }
                }
            }
        ],
        "outputArtifacts": [
            {
                "name": "output-art-name",
                "location": {
```

```
                    "type": "S3",
                    "s3Location": {
                        "bucketName": "outputBucket",
                        "objectKey": "outputKey"
                    }
                }
            }
        ],
        "actionExecutionId": "actionExecutionId",
        "actionCredentials": {
            "accessKeyId": "access-id",
            "secretAccessKey": "secret-id",
            "sessionToken": "session-id"
        },
        "continuationToken": "continueId-xxyyzz"
    }
}
```

Lambda 関数の結果を に返す CodePipeline

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、ジョブの失敗結果を に返すときに、失敗メッセージとともに タイプを設定します CodePipeline。
  - 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。

  - 有効値:
    - ConfigurationError
    - JobFailed
    - PermissionsError
    - RevisionOutOfSync
    - RevisionUnavailable
    - SystemUnavailable
- `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 に渡し CodePipeline 、ジョブを継続する必要があることを示す方法です。例えば、インテグレーターがリソースでお客様のビルドを開始した後、ビルドが完了するまで待機しませんが、 を resultcontinueとして返し、ビルドの一意の ID をcontinuationトークン CodePipeline としてに返すことで、ターミナル結果 CodePipeline がないことを示します。

> **Note**
>
> Lambda 関数は最大 15 分まで実行できます。ジョブを長く実行する必要がある場合は、継続トークンを使用できます。

CodePipeline チームは、ペイロードで同じcontinuationトークンを使用して 30 秒後にインテグレータを呼び出し、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。

実行時にインテグレータ Lambda 関数を呼び出すアクセス許可 CodePipeline を付与する

インテグレータ Lambda 関数にアクセス許可を追加して、 CodePipeline サービスプリンシパル を使用して呼び出すアクセス許可を CodePipeline サービスに提供しますcodepipeline.amazonaws.com。 AWS CloudFormation またはコマンドラインを使用してアクセス許可を追加できます。例については、アクションタイプの使用を参照してください。

ジョブワーカー統合モデル

高レベル ワークフローを綿密に設計した後、ジョブワーカーを作成できます。最終的にはサードパーティーアクションの仕様がジョブワーカーに必要なものを決定しますが、サードパーティーアクションのジョブワーカーの多くは以下の機能を含みます:

- CodePipeline を使用したジョブのポーリング`PollForThirdPartyJobs`。
- ジョブを承認し、`AcknowledgeThirdPartyJob`、、`PutThirdPartyJobSuccessResult`および CodePipeline を使用して結果を に返します`PutThirdPartyJobFailureResult`。
- 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。には Sig V4 が必要です 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、 AWS Key Management Service （ AWS KMS) による暗号化を使用するように Amazon S3 `PutObject`リクエストも設定する必要があります。 は AWS KMS を使用します AWS KMS keys。アーティファクトをアップロードするために AWS マネージドキー またはカスタマーマネージドキーのどちらを使用するかを知るには、ジョブワーカーがジョブデータを調べ、暗号化キープロパティを確認する必要があります。プロパティが設定されている場合は、 を設定するとき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。キープロパティが null の場合は、特に設定 AWS マネージドキー されていない限り、 AWS マネージドキー. CodePipeline uses を使用します。

  Java または .NET で AWS KMS パラメータを作成する方法を示す例については、「 SDK を使用して Amazon S3 AWS Key Management Service で を指定する AWS SDKs」を参照してください。の Amazon S3 バケットの詳細については CodePipeline、「」を参照してくださいCodePipeline の概念。

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

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 CodePipeline、ユーザー管理とアクセス許可管理を統合するための戦略が必要です。

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 間のやり取りを簡素化できます CodePipeline。

Amazon EC2 の詳細を参照し、統合に適しているかどうかを判断します。詳細については、「Amazon EC2 - 仮想サーバーのホスティング」を参照してください。Amazon EC2 インスタンスのセットアップについては、「Amazon EC2 Linux インスタンスの使用開始」を参照してください。

他に考慮すべき戦略は、IAMと ID フェデレーションを使用した既存の ID プロバイダーシステムおよびリソースとの統合です。この戦略は、お客様がすでに企業 ID プロバイダーを持っているか、ウェ

ブ ID プロバイダーを使用するユーザーをサポートできるよう設定されている場合に、特に便利です。ID フェデレーションを使用すると、IAM ユーザーを作成または管理しなくても CodePipeline、を含む AWS リソースへの安全なアクセスを許可できます。パスワードのセキュリティ要件や認証情報の更新に機能やポリシーを活用できます。サンプルアプリケーションをお客様自身の設計のテンプレートとして使用できます。詳細については、「フェデレーションの管理」を参照してください。

アクセス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。

- 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「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。
- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」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する。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、「でパイプラインを作成する CodePipeline」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。
- 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。
  - Amazon ECR ソースアクションでは、ソースアクションの出力として提供される、`imageDetail.json` ファイルが生成されます。

トピック

Amazon ECS 標準デプロイアクション用の imagedefinitions.json ファイル

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、Amazon ECS コンテナと 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 CodePipeline ジョブワーカーに提供するイメージ定義ファイル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ロケーションにアップロードされているか、ビルドされた出力アーティファクトとして生成されていることを確認してください。

imagedefinitions.json ファイルはコンテナ名とイメージ URI を提供します。次のキーと値のペアのセットで構築する必要があります。

キー	値
name	*#####*
ImageURI	*imageUri*

コンテナ名が sample-app で、イメージ URI が ecs-repo、タグが latest の JSON 構造は次のとおりです。

```
[
  {
    "name": "sample-app",
    "imageUri": "11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest"
  }
]
```

複数のコンテナイメージのペアをリストするようにファイルを構築することもできます。

JSON の構造:

```
[
  {
    "name": "simple-app",
    "imageUri": "httpd:2.4"
  },
  {
    "name": "simple-app-1",
    "imageUri": "mysql"
  },
  {
    "name": "simple-app-2",
    "imageUri": "java1.8"
  }
]
```

パイプラインを作成する前に、以下の手順に従って imagedefinitions.json ファイルを設定します。

1. パイプラインのコンテナベースのアプリケーションデプロイの計画の一環として、ソースステージとビルドステージを計画します (該当する場合)。
2. 以下のうちのひとつを選択します。

   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 step-by-step チュートリアルについては、「チュートリアル: を使用した継続的デプロイ CodePipeline」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は、imageDetail.json ファイルを生成して、Amazon ECS と CodeDeployジョブワーカーに Amazon ECR などのイメージリポジトリから取得するイメージ ID を提供する必要があります。

> **Note**
>
> ファイルの名前は imageDetail.json である必要があります。

アクションとそのパラメータの説明については、「Amazon Elastic Container Service と CodeDeploy Blue-Green」を参照してください。

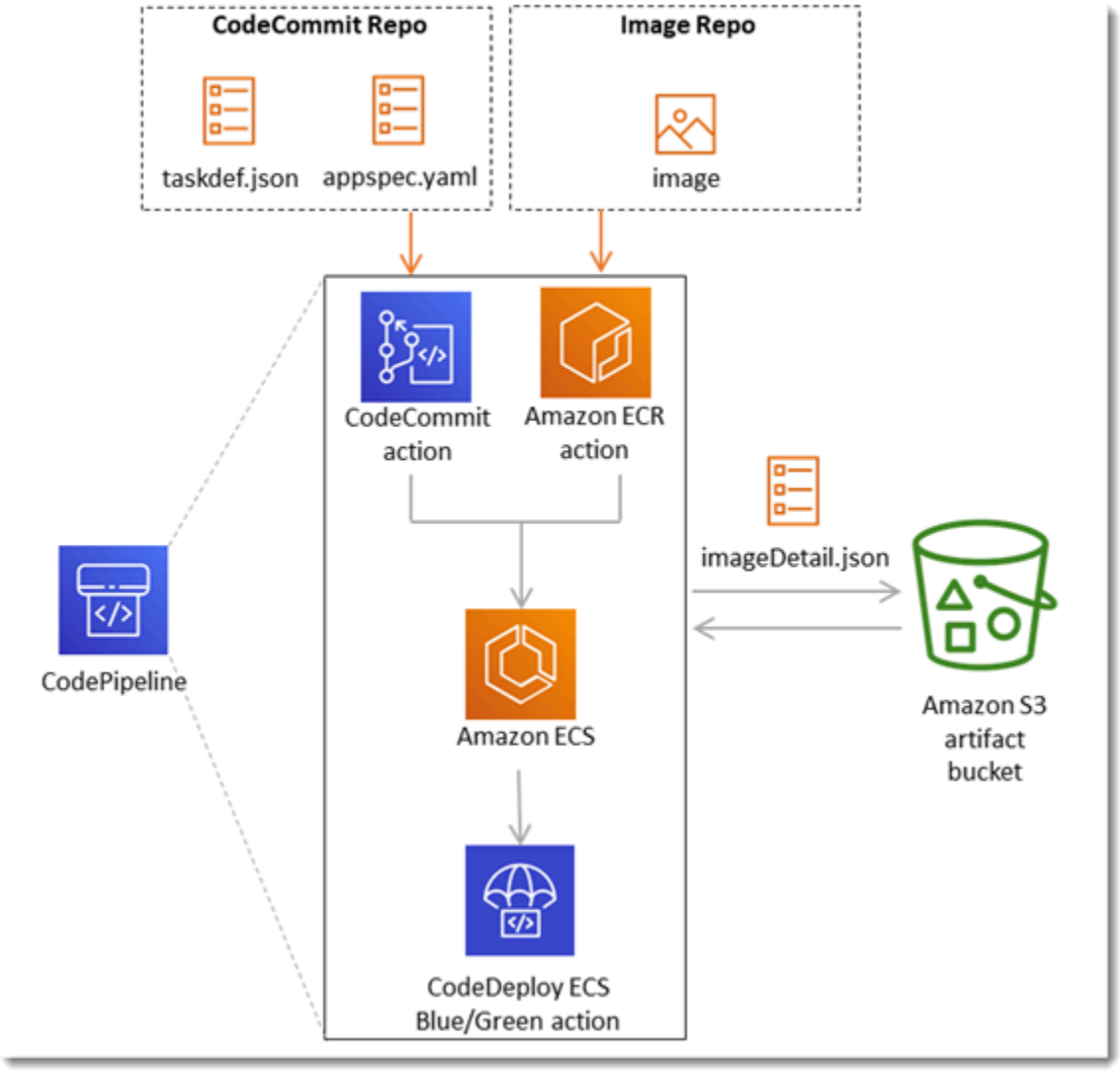
imageDetail.json ファイルは、ソースとして作成するか、デプロイアクションの入力アーティファクトになるようにアーティファクトを構築する必要があります。これらの方法のいずれかを使用して、パイプラインに imageDetail.json ファイルを提供できます。

- ソースの場所に imageDetail.json ファイルを含め、Amazon ECS Blue/Green デプロイアクションへの入力としてパイプラインで提供されるようにします。

> **Note**
>
> ソースリポジトリが Amazon S3 バケットの場合は、JSON ファイルを圧縮してください。

- Amazon ECR ソースアクションでは、次のアクションへの入力アーティファクトとして `imageDetail.json` ファイルが自動生成されます。

> **Note**
>
> Amazon ECR ソースアクションがこのファイルを作成するので、Amazon ECR ソースアクションを含むパイプラインは 手動で `imageDetail.json` ファイルを提供する必要はありません。
> Amazon ECR ソースステージを含むパイプラインの作成に関するチュートリアルについては、チュートリアル: Amazon ECR ソースと ECS-to-CodeDeploy deployment を使用してパイプラインを作成する を参照してください。



imageDetail.json ファイルは、イメージ URI を提供します。次のキーと値のペアで構築する必要があります。

キー	値
ImageURI	*image_URI*

imageDetail.json

以下に、イメージ URI が ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-repo@sha256:example3 である JSON 構造を示します。

```
{
```

```
"ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3"
}
```

imageDetail.json (generated by ECR)

変更がイメージリポジトリにプッシュされるたびに、Amazon ECR ソースアクションによって imageDetail.json ファイルが自動生成されます。Amazon ECR ソースアクションにより生成された imageDetail.json は、ソースアクションから出力アーティファクトとしてパイプラインの次のアクションに提供されます。

リポジトリ名が dk-image-repo で、イメージ URI が ecs-repo、イメージタグが latest の JSON 構造は次のとおりです。

```
{
    "ImageSizeInBytes": "44728918",
    "ImageDigest":
 "sha256:EXAMPLE1122334455667788990bfea42ea2d3b8a1ee8329ba7e68694950afd3",
    "Version": "1.0",
    "ImagePushedAt": "Mon Jan 21 20:04:00 UTC 2019",
    "RegistryId": "EXAMPLE12233",
    "RepositoryName": "dk-image-repo",
    "ImageURI": "ACCOUNTID.dkr.ecr.us-west-2.amazonaws.com/dk-image-
repo@sha256:example3",
    "ImageTags": [
        "latest"
    ]
}
```

imageDetail.json ファイルは、次のようにイメージ URI とコンテナ名を Amazon ECS タスク定義にマッピングします。

- ImageSizeInBytes: リポジトリ内のイメージのサイズ (単位: バイト)。
- ImageDigest: イメージマニフェストの sha256 ダイジェスト。
- Version: イメージバージョン。
- ImagePushedAt: 最新のイメージがリポジトリにプッシュされた日時。
- RegistryId: リポジトリを含むレジストリに関連付けられた AWS アカウント ID。
- RepositoryName: イメージがプッシュされた Amazon ECR リポジトリの名前。
- ImageURI: イメージの URI。

- ImageTags: イメージに使用されるタグ。

パイプラインを作成する前に、以下の手順に従って imageDetail.json ファイルを設定します。

1. パイプラインのコンテナベースのアプリケーション Blue/Green デプロイの計画の一環として、ソースステージとビルドステージを計画します (該当する場合)。
2. 以下のうちのひとつを選択します。

   a. パイプラインがビルドステージをスキップした場合は、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 CodeCommit、JSON ファイルを手動で作成し、 などの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 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

このセクションは参照のみを目的としています。変数の作成については、「変数の操作」を参照してください。

変数を使用すると、パイプラインの実行時またはアクションの実行時に決定される値でパイプラインアクションを設定できます。

一部のアクションプロバイダは、変数の定義されたセットを生成します。コミット ID など、そのアクションプロバイダのデフォルトの変数キーから選択します。

> **Important**
>
> シークレットパラメータを渡すときは、値を直接入力しないでください。値はプレーンテキストとしてレンダリングされるため、読み取り可能です。セキュリティ上の理由から、シークレットを含むプレーンテキストは使用しないでください。シークレットの保存 AWS Secrets Manager には を使用することを強くお勧めします。

変数の使用 step-by-step 例を表示するには：

- 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、「」を参照してくださいチュートリアル: Lambda 呼び出しアクションで変数を使用する。
- アップストリーム AWS CloudFormation アクションのスタック出力変数を参照する CloudFormation アクションのチュートリアルについては、「」を参照してくださいチュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。
- CodeCommit コミット ID とコミットメッセージに解決される出力変数を参照するメッセージテキストを含む手動承認アクションの例については、「」を参照してください例: 手動承認で変数を使用する。
- ブランチ名に解決される環境変数を使用した CodeBuild アクションの例については、 GitHub「」を参照してください例: CodeBuild 環境変数で変数を使用する BranchName 。
- 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、「のクォータ AWS CodePipeline」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

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。ListActionExecutions API の一部である `outputVariables`フィールドを調べることで、アクションによって生成された変数を確

認できます。アクションプロバイダごとに使用できるキー名のリストについては、「パイプラインアクションで使用できる変数」を参照してください。各アクションタイプが生成する変数を確認するには、「」を参照してください CodePipeline アクション構造リファレンス。

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バリエーションがあり、毎回同じパイプラインを使用する CodePipeline お客様向けです。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。パラメータ化されたパイプラインは、同じパイプラインの異なるバージョンを維持したり、複雑なパイプラインを作成 CodePipeline したりすることなく、 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 リポジトリを持つパイプラインの場合、指定されたブランチとターゲットデプロイ環境からのデプロイは、パイプラインレベルで渡された CodeBuild および CodeDeploy パラメータを使用して実行できます。

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、「でパイプラインを作成する CodePipeline」を参照してください。

パイプラインの実行時に渡されるパイプラインレベルの変数のチュートリアルについては、「チュートリアル: パイプラインレベルの変数を使用する」を参照してください。

パイプラインレベルの変数を任意の種類のソースアクションで使用することはできません。

> **Note**
>
> `variables` 名前空間がパイプライン内のいくつかのアクションで既に使用されている場合、アクション定義を更新し、競合するアクションに別の名前空間を選択する必要があります。

アクションレベルの変数の設定

アクションの名前空間を宣言して、変数を生成するようにアクションを設定します。アクションは、すでに変数を生成するアクションプロバイダの 1 つであることが必要です。それ以外の場合、使用可能な変数はパイプラインレベルの変数です。

名前空間は、次のいずれかの方法で宣言します。

- コンソールの [アクションの編集] ページで、[Variable namespace (変数の名前空間)] に名前空間を入力します。
- JSON パイプライン構造の namespace パラメータフィールドに名前空間を入力します。

この例では、 という名前の CodeCommit ソースアクションに namespaceパラメータを追加しますSourceVariables。これにより、そのアクションプロバイダ (CommitId など) で使用可能な変数を生成するようにアクションが設定されます。

```
{
    "name": "Source",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "name": "Source",
            "namespace": "SourceVariables",
            "configuration": {
                "RepositoryName": "MyRepo",
                "BranchName": "mainline",
                "PollForSourceChanges": "false"
            },
            "inputArtifacts": [],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeCommit",
                "category": "Source",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
},
```

次に、前のアクションによって生成された変数を使用するようにダウンストリームアクションを設定します。これは次の方法で行います。

- コンソールの [アクションの編集] ページで、アクション設定フィールドに変数構文 (ダウンストリームアクション用) を入力します。
- JSON パイプライン構造のアクション設定フィールドに変数構文 (ダウンストリームアクション用) を入力する

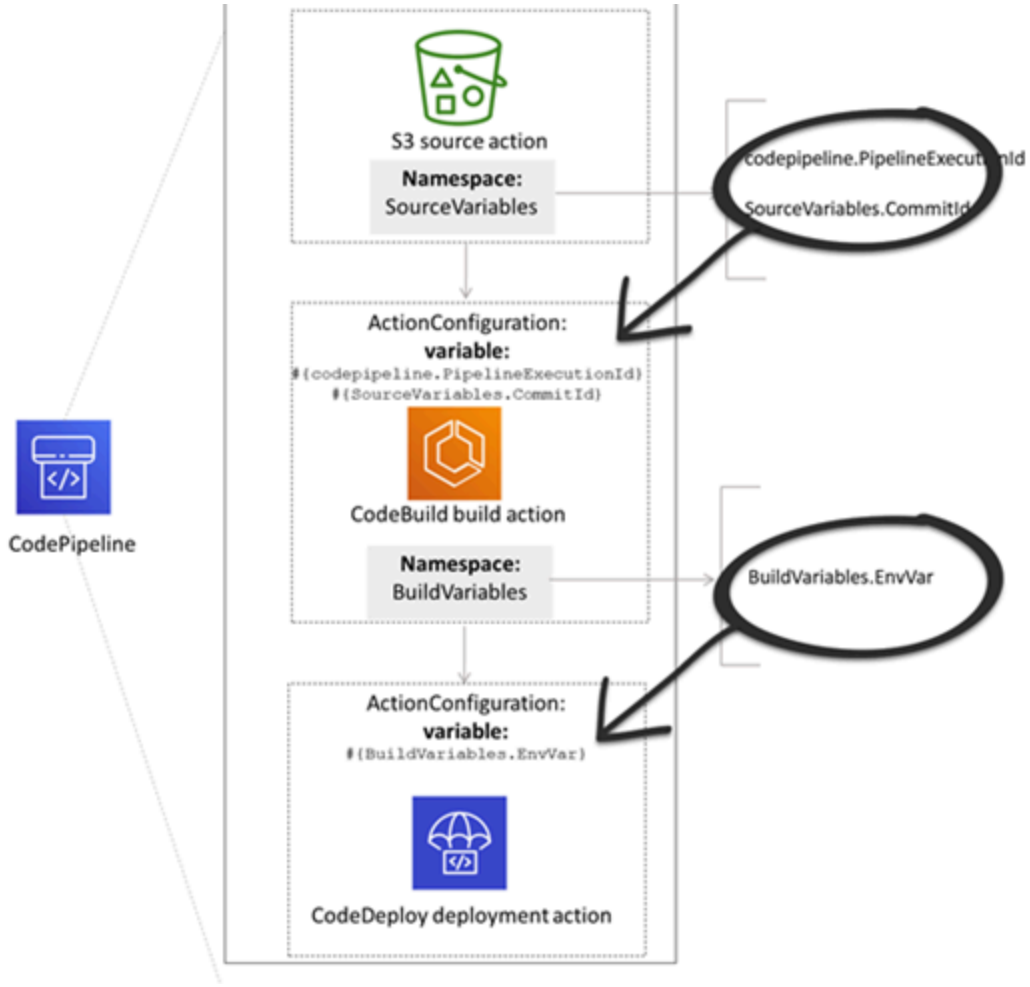
この例では、ビルドアクションの設定フィールドには、アクションの実行時に更新される環境変数が表示されます。この例では、#{codepipeline.PipelineExecutionId} で実行 ID の名前空間と変数を指定し、コミット ID には **#{SourceVariables.CommitId}** で名前空間と変数を指定します。

```
{
    "name": "Build",
    "actions": [
        {
            "outputArtifacts": [
                {
                    "name": "BuildArtifact"
                }
            ],
            "name": "Build",
            "configuration": {
                "EnvironmentVariables": "[{\"name\":\"Release_ID\",\"value\":
\"#{codepipeline.PipelineExecutionId}\",\"type\":\"PLAINTEXT\"},{\"name\":\"Commit_ID
\",\"value\":\"#{SourceVariables.CommitId}\",\"type\":\"PLAINTEXT\"}]",
                "ProjectName": "env-var-test"
            },
            "inputArtifacts": [
                {
                    "name": "SourceArtifact"
                }
            ],
            "region": "us-west-2",
            "actionTypeId": {
                "provider": "CodeBuild",
                "category": "Build",
                "version": "1",
                "owner": "AWS"
            },
            "runOrder": 1
        }
    ]
```

```
},
```

変数の解決

アクションがパイプライン実行の一部として実行されるたびに、アクションが生成する変数は、生成アクションの後に発生することが保証される任意のアクションで使用できます。これらの変数を消費アクションで使用するには、前の例で示した構文を使用して、消費アクションの設定に変数を追加します。コンシューマーアクションを実行する前に、 はアクションの実行を開始する前に、設定に存在するすべての変数参照を CodePipeline 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 step-by-step 手順については、「」を参照してください変数の操作。

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生成の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 アクション出力変数
- CodeStarSourceConnection アクション出力変数
- GitHub アクション出力変数 (GitHub アクションバージョン 1)
- S3 アクションの出力変数

CodePipeline 実行 ID 変数

CodePipeline 実行 ID 変数

プロバイダー	変数キー	値の例	変数構文例
codepipeline	`PipelineExecutionId`	8abc75f0-fbf8-4f4c-bfEXAMPLE	`#{codepipeline.PipelineExecutionId}`

Amazon ECR アクションの出力変数

Amazon ECR 変数

変数キー	値の例	変数構文例
`ImageDigest`	sha256:EXAMPLE1122334455	`#{SourceVariables.ImageDigest}`
`ImageTag`	最新	`#{SourceVariables.ImageTag}`
`ImageURI`	11111EXAMPLE.dkr.ecr.us-west-2.amazonaws.com/ecs-repo:latest	`#{SourceVariables.ImageURI}`
`RegistryId`	EXAMPLE12233	`#{SourceVariables.RegistryId}`
`RepositoryName`	my-image-repo	`#{SourceVariables.RepositoryName}`

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-1111-2222-2bbb-11111 例	`#{DeployVariables.StackSetId}`

CodeCommit アクション出力変数

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`	myCodeCommitリポジトリ	`#{SourceVariables.RepositoryName}`

CodeStarSourceConnection アクション出力変数

`CodeStarSourceConnection` 変数 (Bitbucket Cloud、 GitHub GitHubEnterprise Repository、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}`
`CommitMessage`	バグを修正 (最大サイズ 100 KB)	`#{SourceVariables.CommitMessage}`
`ConnectionArn`	arn:aws:codestar-connections:region: *`account-id`* :connection/ *`connection-id`*	`#{SourceVariables.ConnectionArn}`
`FullRepositoryName`	username/GitHubRepo	`#{SourceVariables.FullRepositoryName}`

GitHub アクション出力変数 (GitHub アクションバージョン 1)

GitHub 変数 (GitHub アクションバージョン 1)

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}`

変数キー	値の例	変数構文例
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	myGitHubリポジトリ	#{SourceVariables.RepositoryName}

S3 アクションの出力変数

S3 変数

変数キー	値の例	変数構文例
ETag	example28be1c3	#{SourceVariables.ETag}
VersionId	exampleta_IUQCv	#{SourceVariables.VersionId}

ユーザー設定の変数キーを使用したアクション

CodeBuild、 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 アクション出力変数
- CodeBuild アクション出力変数

- Lambda アクションの出力変数

CloudFormation アクション出力変数

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。出力を生成する唯一の CloudFormation アクションモードは、スタックの作成、スタックの更新、変更セットの実行など、スタックの作成または更新につながるアクションモードであることに注意してください。変数を生成するアクションモードは次のとおりです。 • CREATE_UPDATE • CHANGE_SET_EXECUTE • CHANGE_SET_REPLACE • REPLACE_ON_FAILURE これらのアクションの詳細については、「AWS CloudFormation」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チュートリアル: AWS CloudFormation デプロイアクションの変数を使用するパイプラインを作成する。	`#{DeployVariables.StackName}`

CodeBuild アクション出力変数

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。でアク	`#{BuildVariables.EnvVar}`

変数キー	変数構文例
ションを編集する CodeBuild CodePipeline か、ビルド仕様に CodeBuild 環境変数を追加して、環境変数を設定します。 CodeBuild ビルド仕様に手順を追加して、エクスポートされた変数セクションの下に環境変数を追加します。env/exported-変数 の AWS CodeBuild ユーザーガイド を参照してください。	

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、PutJobSuccessResult API リクエストの `outputVariables` セクションに含まれるすべてのキーと値のペアを変数として生成します。 アップストリームアクション (CodeCommit) の変数を使用し、出力変数を生成する Lambda アクションのチュートリアルについては、「」を参照してくださいチュートリアル: Lambda 呼び出しアクションで変数を使用する。	`#{TestVariables.testRunId}`

構文での glob パターンの使用

パイプラインのアーティファクトまたはソースロケーションで使用されるファイルまたはパスを指定する場合、アクションタイプに応じてアーティファクトを指定できます。例えば、S3 アクションでは S3 オブジェクトキーを指定します。

トリガーではフィルタを指定できます。glob パターンを使用してフィルタを指定できます。以下は例です。

構文が「glob」の場合、パスの文字列表現は正規表現に似た構文を持つ限定的なパターン言語を使用してマッチされます。例:

- `*.java` は、.java で終わるファイル名を表すパスを指定します。
- `*.*` は、ドットを含むファイル名を指定します。
- `*.{java,class}` は、.java または .class で終わるファイル名を指定します。
- `foo.?` は、foo. で始まり 1 文字の拡張子の付いたファイル名を指定します。

glob パターンの解釈には以下の規則が使用されます。

- ディレクトリ境界で 0 文字以上の名前要素を指定するには、* を使用します。
- ディレクトリ境界をまたいで 0 文字以上の名前要素を指定するには、** を使用します。
- 名前コンポーネントの 1 文字を指定するには、? を使用します。
- 特殊文字として解釈される文字をエスケープするには、バックスラッシュ文字 (\) を使用します。
- 文字セットから 1 文字を指定するには、[ ] を使用します。
- ビルドまたはソースリポジトリの場所のルートにある 1 つのファイルを指定するには、`my-file.jar` を使用します。
- サブディレクトリ内の 1 つのファイルを指定するには、`directory/my-file.jar` または `directory/subdirectory/my-file.jar` を使用します。
- すべてのファイルを指定するには、"**" を使用します。glob パターン ** は、任意の数のサブディレクトリにマッチすることを示します。
- `directory` という名前のディレクトリ内のすべてのファイルとディレクトリを指定するには、"directory/**" を使用します。glob パターン ** は、任意の数のサブディレクトリにマッチすることを示します。

- directory という名前のディレクトリ内のすべてのファイルを指定するが、そのサブディレクトリを含めないようにするには、"directory/*" を使用します。
- 角括弧式内では、*、?、および \ 文字は、文字通りの意味です。ハイフン (-) 文字は、角括弧内で最初の文字だった場合、または式を否定する ! の次の文字だった場合は、文字通りの意味です。
- 中括弧 ({ }) は、グループ内のサブパターンがマッチする場合にグループがマッチするサブパターンのグループを囲みます。カンマ (",") 文字は、サブパターンを分割するために使用します。グループはネストできません。

ポーリングパイプラインを推奨される変更検出方法に更新する

ポーリングを使用してソースの変更に対応するパイプラインがある場合は、推奨される検出方法を使用するようにパイプラインを更新できます。推奨されるイベントベースの変更検出方法を使用するようにポーリングパイプラインを更新する手順が含まれる移行ガイドについては、「ポーリングパイプラインをイベントベースの変更検出の使用に移行する」を参照してください。

GitHub バージョン 1 のソースアクションを GitHub バージョン 2 のソースアクションに更新する

では AWS CodePipeline、 GitHub ソースアクションには次の 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 バージョン 2 アクションでは、CodeStarSourceConnection Bitbucket Cloud、GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用リソースに基づく Github アプリケーションベースの認証を使用します。Connections AWS CodeStar アプリケーションを GitHub 組織にインストールして、 でアクセスを管理できるようにします GitHub。
- 非推奨： GitHub バージョン 1 アクションでは、OAuth トークンを使用して で認証 GitHub 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、「との製品とサービスの統合 CodePipeline」を参照してください。欧州 (ミラノ) リージョンでのこのアクションに関する考慮事項については、「CodeStarSourceConnection Bitbucket Cloud、 GitHub Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」の注意を参照してください。

GitHub バージョン 1 アクションの代わりに GitHub バージョン 2 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。

- 接続を使用すると、リポジトリにアクセスするために OAuth アプリケーションや個人用アクセストークンが不要 CodePipeline になります。接続を作成するときは、 GitHub リポジトリへの認証を管理し、組織レベルでアクセス許可を付与する 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に OAuth ベースの GitHub アクセスの詳細については、「」を参

照してくださいhttps://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps。

- CLI または で GitHub バージョン 2 のアクションを管理する場合 CloudFormation、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 Enterprise Server GitHub、 GitLab.com、および GitLab セルフマネージドアクション用」を参照してください。
- で GitHub バージョン 2 アクションで使用する接続リソースを作成する場合 CodePipeline、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 バージョン 2 では、後続の CodeBuild アクションで git メタデータにアクセスするためのリポジトリのクローンを作成できますが、Github バージョン 1 ではソースのみダウンロードできます。
- 管理者が Organization のリポジトリにアプリをインストールします。トークンを作成した個人に依存する OAuth トークンを追跡する必要がなくなりました。

Organization にインストールされているすべてのアプリは、同じリポジトリのセットにアクセスできます。各リポジトリにアクセスできるユーザーを変更するには、各接続の IAM ポリシーを変更します。例については、「例: 指定したリポジトリとの接続を使用するためのスコープダウンポリシー」を参照してください。

このトピックのステップを使用して、 GitHub バージョン 1 のソースアクションを削除し、CodePipeline コンソールから GitHub バージョン 2 のソースアクションを追加できます。

トピック

- ステップ 1: バージョン 1 GitHub アクションを置き換える
- ステップ 2: への接続を作成する GitHub
- ステップ 3: GitHub ソースアクションを保存する

ステップ 1: バージョン 1 GitHub アクションを置き換える

パイプラインの編集ページを使用して、バージョン 1 GitHub アクションをバージョン 2 GitHub アクションに置き換えます。

バージョン 1 GitHub アクションを置き換えるには

1. CodePipeline コンソールにサインインします。
2. パイプラインを選択し、[編集] を選択します。ソースステージで、[ステージを編集] を選択します。アクションを更新することを推奨するメッセージが表示されます。
3. アクションプロバイダー で、 GitHub （ バージョン 2) を選択します。
4. 次のいずれかを行います。
   - 「接続」で、プロバイダーへの接続をまだ作成していない場合は、「 に接続する GitHub」を選択します。ステップ 2: への接続を作成するに進みます GitHub。
   - [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: への接続を作成する GitHub

接続の作成を選択すると、Connect to GitHubページが表示されます。

への接続を作成するには GitHub

1. GitHub 接続設定 で、接続名 が接続名 に表示されます。

   GitHub アプリ で、アプリのインストールを選択するか、新しいアプリのインストールを選択して作成します。

   > **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 アプリを既にインストールしている場合は、アプリを選択してこのステップをスキップします。

2. の認証ページ GitHub が表示されたら、認証情報を使用してログインし、続行することを選択します。
3. アプリのインストールページで、 AWS CodeStar アプリが GitHub アカウントに接続しようとしていることを示すメッセージが表示されます。

> **Note**
>
> アプリは GitHub アカウントごとに 1 回だけインストールします。アプリケーションをインストール済みである場合は、[Configure] (設定) を選択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 ページに、新しいインストールの接続 ID が表示されます。[接続]を選択します。

ステップ 3: GitHub 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 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 アクションの出力アーティファクトを保存するには、CodePipeline デフォルトの を選択します。アクションは GitHub リポジトリからファイルにアクセスし、アーティファクトをパイプラインアーティファクトストアの ZIP ファイルに保存します。
   - 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、「」に示すように、 CodeBuild プロジェクトサービスロールのアクセス許可を更新する必要がありますBitbucket、、 GitHub Enterprise Server GitHub、または GitLab.com への接続の CodeBuild GitClone アクセス許可を追加する。フルクローン オプションの使い方を紹介したチュートリアルは、チュートリアル: GitHub パイプラインソースでフルクローンを使用する をご覧ください。

3. 出力アーティファクト　の場合、`SourceArtifact` のようにこのアクションの出力アーティファクトの名前を保持できます。[Done] を選択して、[アクションを編集] ページを閉じます。
4. [Done] を選択して、ステージの編集ページを閉じます。[Save] を選択して、パイプラインの編集ページを閉じます。

のクォータ AWS CodePipeline

CodePipeline には、アカウントが各 AWS リージョンで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。引き上げをリクエストするには、サポートセンターコンソールを使用してください。

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するまでの時間（これは設定可能なタイムアウトです。 設定できない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 **Note** アクションの実行中に、 は CodePipeline 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 分間に Lambda 関数がアクションがまだ進行中であると返信した場合、 CodePipeline は 20 分のタイマーを再起動して再試行します。24 時間後に成功しなかった場合、 は Lambda 呼び出しアクションの状態を失敗 CodePipeline に設定します。

リソース	デフォルト値
	Lambda には、 CodePipeline アクションのタイムアウトとは関係のない Lambda 関数の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、パイプライン内の特定のアクションに対して上書きでき、最小値を 5 分として最大 86400 分 (60 日) まで設定できます。詳細については、 API リファレンスActionDeclarationの「」を参照してください。 CodePipeline 設定すると、このタイムアウトが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 **Note** Amazon ECS デプロイアクションのタイムアウトは最大 1 時間 (デフォルトのタイムアウト) まで設定できます。
AWS アカウントのリージョンあたりのパイプラインの最大合計数	1,000 **Note** ポーリングまたはイベントベースの変更検出用に設定されたパイプラインは、このクォータにカウントされます。
AWS リージョンごとのソース変更のポーリングに設定するパイプラインの最大数	300 **Note** このクォータは固定されており、変更できません。ポーリングパイプラインの制限に達しても、まだイベントベースの変更検出を使用する追加のパイプラインを構成できます。詳細については、「ソースアクションと変更検出方法」を参照してください。[1]
AWS アカウントのリージョンあたりのウェブフックの最大数	300

リソース	デフォルト値
AWS アカウントのリージョンあたりのカスタムアクションの数	50

[1]ソースプロバイダーに基づき、以下の手順に従って、ポーリングパイプラインをイベントベースの変更検出の使用に更新します。

- CodeCommit ソースアクションを更新するには、「」を参照してくださいポーリングパイプライン (CodeCommit または Amazon S3 ソース) を移行する (コンソール) 。
- Amazon S3 のソースアクションを更新するには、ポーリングパイプライン (CodeCommit または Amazon S3 ソース) を移行する (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には、「」を参照してくださいポーリングパイプラインをウェブフックに移行する (GitHub バージョン 1 のソースアクション) (コンソール) 。

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 CodePipeline のサービスエンドポイントのリストについては、「 AWS 全般のリファレンス」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 (アラブ首長国連邦)
	AWS GovCloud （米国西部）
	AWS GovCloud （米国東部）

アクション名に使用できる文字	アクション名は 100 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 0 ~ 9 の数字。 特殊文字の . (ピリオド)、@ (アットマーク)、- (マイナス記号)、および _ (下線)。 スペースなどの他の文字は使用できません。
アクションの種類で使用される文字	アクションの種類名は 25 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 数値 (0 ~ 9)。 特殊文字の . (ピリオド)、@ (アットマーク)、- (マイナス記号)、_ (下線)。 スペースなどの他の文字は使用できません。

アーティファクト名に使用できる文字	アーティクラフト名は 100 文字を超えることはできません。使用できる文字は次のとおりです。 小文字 (a ~ z)。 大文字 (A ~ Z)。 0 ~ 9 の数字。 特殊文字の - (マイナス記号)、および _(下線) スペースなどの他の文字は使用できません。
パートナーのアクション名に使用できる文字	パートナーアクション名は、 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す CodePipeline。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 ビルドアクションとテストアクション: 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
アクション設定キーの最大長 (設定キーは 、 CodeBuildProjectName PrimarySource 、 などEnvironmentVariables)	50 文字

アクション設定値の最大長 (例えば、CodeCommit アクションRepositoryName 設定の設定値は 1000 文字未満にする必要があります。 `"RepositoryName": "my-repo-name-less-than-1000-characters" )`	1000 文字
パイプラインあたりのアクションの最大数	500
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アクション実行の最大数	5
Amazon S3 オブジェクトの最大ファイル数	100,000
パイプラインの実行履歴情報を保持できる最大の月数	12
ステージで同時に実行されるアクションの最大数	50
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	50

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 で実行されるため、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あります。詳細については、「Lambda デベロッパーガイド」の「Lambda クォータ」を参照してください。

`ParameterOverrides` プロパティに保存できる JSON オブジェクトの最大サイズ	をプロバイダー 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、「各アクションタイプの入出力アーティファクトの数」を参照してください。
パイプラインのステージの数	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。 • で始まる AWS : • 空白文字のみで構成されている • 末尾にスペースを使用する • 絵文字、または以下の文字を含める: ? ^ * [\ ~ ! # $ % & * ( ) > < | " '

パイプラインのタグ値	任意の組み合わせで使用できる文字は、Unicode 文字、数字、スペース、および許可されている UTF-8 文字 (1 ~ 256 文字) です。使用できる文字は、+ - = . _ : / @ です。 使用できるキーの値は 1 つのみですが、多数のキー値と同じ値を含めることができます。タグは以下のようにはできません。 • で始まる AWS : • 空白文字のみで構成されている • 末尾にスペースを使用する • 絵文字、または以下の文字を含める: ? ^ * [\ ~ ! # $ % & * ( ) > < | " '

トリガー	パイプライン定義には、 `push`および `pull request`設定全体で最大 50 個のトリガーがあります。 プッシュトリガーとプルリクエストトリガーごとに最大 3 つのフィルターがあります。 **Note** 同じイベントタイプ配列内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 つのインクルードパターン、ブランチ、およびファイルパスを追加できます。 パターン値で使用できる文字には、すべての文字タイプが含まれます。 包含パターンと除外パターンの場合、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 を超えないようにしてください

トリガーフィルター	ファイルパス： • パターン数: 最大 8 つのインクルードパターンと 8 つの除外パターンを追加できます。 • パターンのサイズ: 各包含または除外パターンのサイズは最大 255 文字です。 ブランチ： • パターン数: 最大 8 つのインクルードパターンと 8 つの除外パターンを追加できます。 • パターンのサイズ: 各包含または除外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： • パターン数: 最大 8 つのインクルードパターンと 8 つの除外パターンを追加できます。 • パターンのサイズ: 各包含または除外パターンのサイズは最大 255 文字です。
名前の一意性	1 つの AWS アカウント内では、 AWS リージョンで作成する各パイプラインには一意の名前が必要です。パイプラインの名前は、別の AWS リージョンで再利用できます。 ステージ名は、パイプライン内で一意である必要があります。 アクション名は、ステージ内で一意である必要があります。

出力変数と名前空間のクォータ	特定のアクションに対して結合されたすべての出力変数には、122880 バイトという最大サイズ制限があります。 特定のアクションの解決済みアクション設定の合計には、100 KB という最大サイズ制限があります。 出力変数名では、大文字と小文字が区別されます。 名前空間では、大文字と小文字が区別されます。 使用できる文字は次のとおりです。 • 小文字 (a ~ z)。 • 大文字 (A ~ Z)。 • 数値 (0 ~ 9)。 • 特殊文字の ^ (キャレット)、@ (アットマーク)、- (マイナス記号)、_ (下線)、[(左角カッコ)、] (右角カッコ)、* (アスタリスク)、$ (ドル記号)。 スペースなどの他の文字は使用できません。

パイプラインレベルの変数のクォータ	パイプラインレベルの変数は 1 パイプラインあたり最大 50 個です。 パイプラインレベルの変数の変数名は以下の要件を満たす必要があります。 • 最大文字数は 128 文字 • 小文字 (a ~ z)。 • 大文字 (A ~ Z)。 • 数値 (0 ~ 9)。 • 特殊文字 @\-_]+ スペースなどの他の文字は使用できません。 変数値の最大長は 1000 文字です。 変数の値には、すべての文字を使用できます。 変数の説明の最大長は 200 文字です。

*使用する前に、このリージョンを有効にする必要があります。

付録 A: GitHub バージョン 1 のソースアクション

この付録では、 の GitHub アクションのバージョン 1 に関する情報を提供します CodePipeline。

> **Note**
>
> GitHub バージョン 1 アクションを使用することはお勧めしませんが、 GitHub バージョン 1 アクションを含む既存のパイプラインは、影響を受けずに引き続き機能します。バージョン 1 アクションの GitHub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 (バージョン 2) は接続リソースを使用してリソースを GitHub リポジトリに関連付け AWS 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。パイプラインを接続を使用する推奨 GitHub アクションに更新する方法の詳細については、「」を参照してください GitHub バージョン 1 のソースアクションを GitHub バージョン 2 のソースアクションに更新する。アプリベースの GitHub アクセスとは対照的に OAuth ベースの GitHub アクセスの詳細については、「」を参照してくださいhttps://docs.github.com/en/developers/apps/differences-between-github-apps-and-oauth-apps。

と統合するために GitHub、 はパイプラインに GitHub OAuth アプリケーション CodePipeline を使用します。 CodePipeline はウェブフックを使用して、 GitHub バージョン 1 のソースアクションでパイプラインの変更検出を管理します。

> **Note**
>
> で GitHub バージョン 2 の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AWS::CodeStarConnections::Connection「」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 バージョン 1 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 バージョン 1 の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 バージョン 1 のソースアクションの追加。

- GitHub バージョン 1 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 バージョン 1 ソースアクション構造リファレンス。

トピック

GitHub バージョン 1 のソースアクションの追加

GitHub バージョン 1 のソースアクションは、 CodePipeline 次の方法で に追加します。

- CodePipeline コンソールでパイプラインの作成ウィザード (パイプラインを作成する (コンソール)) またはアクションの編集ページを使用して、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 バージョン 1 ソースアクション構造リファレンス 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。 GitHub バージョン 1 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します。

GitHub バージョン 1 ソースアクション構造リファレンス

> **Note**
>
> GitHub バージョン 1 アクションを使用することはお勧めしませんが、 GitHub バージョン 1 アクションを含む既存のパイプラインは、影響を受けずに引き続き機能します。 GitHub GitHub バージョン 1 のソースアクションを持つパイプラインの場合、 CodePipeline は OAuth ベースのトークンを使用して GitHub リポジトリに接続します。対照的に、新しい GitHub アクション (バージョン 2) では、接続リソースを使用して AWS リソースを GitHub リポジトリに関連付けます。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。パイプラインを接続を使用する推奨 GitHub アクションに更新する方法の詳細について

は、「」を参照してください GitHub バージョン 1 のソースアクションを GitHub バージョン 2 のソースアクションに更新する。

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ると、パイプラインをトリガーします。

と統合するには GitHub、パイプラインに OAuth アプリケーションまたは個人用アクセストークン CodePipeline 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、 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 GitHub ウェブフック CodePipeline を作成します。

GitHub アクションを通じ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。

リポジトリ CodePipeline へのアクセスを制限する場合は、 GitHub アカウントを作成し、 と統合するリポジトリにのみアクセス許可をアカウントに付与します CodePipeline。パイプラインのソースステージに GitHub リポジトリを使用する CodePipeline ように を設定するときに、そのアカウントを使用します。

詳細については、 GitHub ウェブサイトのGitHub デベロッパー向けドキュメントを参照してください。

トピック

- アクションタイプ
- 設定パラメータ
- 入力アーティファクト
- 出力アーティファクト
- 出力変数
- アクションの宣言 (GitHub の例)
- への接続 GitHub (OAuth)
- 以下も参照してください。

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

リポジトリを所有 GitHub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

が GitHub リポジトリでオペレーションを実行 CodePipeline できるようにする 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、このフィールドに 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、 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 GitHub。`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 スコープの詳細については、 GitHub ウェブサイトのGitHub 「 デベロッパー API リファレンス」を参照してください。

PollForSourceChanges

必須: いいえ

PollForSourceChanges は、ソースの変更についてリポジトリを CodePipeline GitHubポーリングするかどうかを制御します。この方法の代わりに、ウェブフックを使用してソースの変更を検出することをお勧めします。ウェブフックの設定に関する詳細は、「ポーリングパイプラインをウェブフックに移行する (GitHub バージョン 1 のソースアクション) (CLI)」または「プッシュイベント (GitHub バージョン 1 ソースアクション) のパイプラインを更新する (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、 はリポジトリにソースの変更を CodePipeline ポーリングします。

  > **Note**
  >
  > を省略するとPollForSourceChanges、 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 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、ソースの変更についてリポジトリをポーリング CodePipeline 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。ソースコードコミット ID は、トリガーされたパイプライン実行のソースリビジョン CodePipeline として 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。

の変数の詳細については、 CodePipeline「」を参照してください変数。

CommitId

パイプラインの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行われた 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)。

Git での著者とコミッターの違いに関する詳細については、Scott Chacon と Ben Straub による Pro Git の「コミット履歴の表示」を参照してください。

CommitterDate

コミットがコミットされた日付 (タイムスタンプ形式)。

Git での著者とコミッターの違いに関する詳細については、Scott Chacon と Ben Straub による Pro Git の「コミット履歴の表示」を参照してください。

アクションの宣言 (GitHub の例)

YAML

```
Name: Source
Actions:
  - InputArtifacts: []
    ActionTypeId:
      Version: '1'
      Owner: ThirdParty
      Category: Source
      Provider: GitHub
    OutputArtifacts:
      - Name: SourceArtifact
    RunOrder: 1
    Configuration:
      Owner: MyGitHubAccountName
      Repo: MyGitHubRepositoryName
      PollForSourceChanges: 'false'
      Branch: main
      OAuthToken: '{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}'
    Name: ApplicationSource
```

JSON

```
{
    "Name": "Source",
    "Actions": [
        {
            "InputArtifacts": [],
            "ActionTypeId": {
                "Version": "1",
```

```
                "Owner": "ThirdParty",
                "Category": "Source",
                "Provider": "GitHub"
            },
            "OutputArtifacts": [
                {
                    "Name": "SourceArtifact"
                }
            ],
            "RunOrder": 1,
            "Configuration": {
                "Owner": "MyGitHubAccountName",
                "Repo": "MyGitHubRepositoryName",
                "PollForSourceChanges": "false",
                "Branch": "main",
                "OAuthToken":
 "{{resolve:secretsmanager:MyGitHubSecret:SecretString:token}}"
            },
            "Name": "ApplicationSource"
        }
    ]
},
```

への接続 GitHub (OAuth)

コンソールを使用してパイプラインに GitHub リポジトリを初めて追加するときは、リポジトリ CodePipeline へのアクセスを許可するように求められます。トークンには以下の 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、 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 GitHub。

以下も参照してください。

このアクションを利用する際に役立つ関連リソースは以下の通りです。

- AWS CloudFormation ユーザーガイド AWS::CodePipeline::Webhookのリソースリファレンス – これには、 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。
- AWS CloudFormation ユーザーガイドリポジトリのリソースリファレンス – これには、 AWS::CodeStar::GitHub 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。
- チュートリアル: で Android アプリを構築してテストするパイプラインを作成する AWS Device Farm – このチュートリアルでは、 GitHub ソースを使用してパイプラインを作成するためのサンプル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。および を使用して Android アプリを構築 CodeBuild してテストします AWS Device Farm。

AWS CodePipeline ユーザーガイドのドキュメント履歴

以下の表は、 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ています。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： 2024 年 5 月 7 日

変更	説明	日付
S3ソースアクションを更新して、ソースオーバーライドの新しいオプションを追加	ソースアクションには、 という名前のS3ソースオーバーライドの新しいオプションS3_OBJECT_KEY を使用できます。S3 ソースアクションに新しいAllowOverrideForS3ObjectKey パラメータが追加されました。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。	2024 年 6 月 7 日
S3ソースアクションを更新して新しい出力変数を追加	S3 ソースアクションObjectKey には、BucketName および という名前の新しい出力変数を使用できます。Amazon S3 ソースアクションのリファレンスページを参照してください。	2024 年 6 月 5 日
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	このPipelineCostAnalyzer.py スクリプトは、既存の V1 タイプのパイプラインを	2024 年 5 月 30 日

動するためのコスト分析のサポート	V2 タイプのパイプラインに移動するコスト分析を実行するために追加されました。「自分に適したパイプラインのタイプ」を参照してください。	
CloudFormationStackSet および CloudFormationStackInstances アクションの更新	CloudFormationStackSet および CloudFormationStackInstances アクションに CallAs パラメータが追加されました。「アクションリファレンスページ」を参照してください。	2024 年 5 月 2 日
ステージレベルのロールバックのサポート	ステージを、ステージの前回正常に実行されたパイプラインに手動または自動的にロールバックできます。「ステージのロールバックの設定」と「概念」を参照してください。	2024 年 4 月 26 日
StackSets および Step Functions アクションのリージョン可用性の更新	StackSets および Step Functions アクションが、CodePipeline が利用可能なすべてのリージョンで利用可能になりました。AWS CloudFormation StackSets「アクションリファレンス」およびAWS「Step Functions アクションリファレンス」を参照してください。	2024 年 3 月 27 日

マネージドポリシーの更新	AWS 管理ポリシーが更新されAWSCodePipeline_FullAccess ました。AWS CodePipelineのAWS マネージドポリシーを参照してください。	2024 年 3 月 15 日
手動承認アクションの設定可能なタイムアウトのサポート	手動承認アクションの新しい設定可能なタイムアウトフィールドに追加されたクォータ情報。詳細については、「クォータ」を参照してください。	2024 年 2 月 15 日
ブランチとファイルパスによるトリガーフィルタリングのサポート	V2 タイプのパイプラインのプルリクエストのステータス、ブランチ、ファイルパスをフィルタリングできるトリガー設定のサポートが追加されました。詳細については、「コードのプッシュまたはプルリクエストでのトリガーのフィルタリング」、「トリガー」、「特徴ブランチでのフィルタリングによるパイプラインの起動」、および「クォータ」を参照してください。	2024 年 2 月 8 日

新しいパイプライン実行モードのサポート	PARALLEL および QUEUED パイプライン実行モードのサポートが追加されました。詳細については、「パイプライン実行モードの設定」、「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」、および「クォータ」を参照してください。	2024 年 2 月 8 日
アクションの詳細を表示したり、手動承認アクションを確認したり、パイプラインのリストページを表示したりするためのコンソールページの更新	新しい [詳細を表示] ボタンとダイアログボックス、新しい手動承認ダイアログ、およびパイプラインのリストページの最近の実行に関する新しい列について、コンソールの更新内容が文書化されました。詳細については、「パイプラインの表示 (コンソール)」、「パイプラインのアクションの詳細の表示」、および「パイプラインの承認アクションの管理」を参照してください。	2024 年 1 月 10 日
GitLab セルフマネージドのサポート	セルフマネージドと GitLab やり取りする AWS リソースの接続設定のサポートが追加されました。詳細については、「セルフマネージド の接続 GitLab」を参照してください。	2023 年 12 月 28 日

CloudFormationStackSet および CloudFormationStackInstances アクションの更新	CloudFormationStackSet および CloudFormationStackInstances アクションに ConcurrencyMode パラメータが追加されました。「アクションリファレンスページ」を参照してください。	2023 年 12 月 19 日
での AWS Device Farm アクションパラメータの更新 CodePipeline	の AWS Device Farm アクションのパラメータが更新され CodePipeline ました。詳細については、「AWS Device Farm アクションリファレンス」を参照してください。	2023 年 12 月 18 日
で AWS CloudFormation アクションの詳細なエラーメッセージのサポートが追加されました CodePipeline	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、「AWS CloudFormation アクションリファレンス」を参照してください。	2023 年 12 月 15 日
でソースリビジョンオーバーライドを使用してパイプラインを開始するための更新 CodePipeline	指定したソースリビジョンでパイプラインを起動できるようになりました。詳細については、「ソースリビジョンオーバーライドでパイプラインを開始する」を参照してください。	2023 年 11 月 17 日

「サポートされているリージョン」を参照してください。	CodePipeline が、アジアパシフィック (ハイデラバード)、アジアパシフィック (ジャカルタ)、アジアパシフィック (メルボルン)、アジアパシフィック (大阪)、中東 (アラブ首長国連邦)、欧州 (スペイン)、イスラエル (テルアビブ) の各リージョンで利用可能になりました。「イベントプレースホルダーバケットリファレンス」トピックと「AWS のサービス エンドポイント」トピックが更新されました。	2023 年 11 月 13 日
Amazon のイベントフィールドの更新 EventBridge	Amazon で更新されたイベントフィールドを表示できるようになりました EventBridge。詳細については、「イベントのモニタリング CodePipeline」を参照してください。	2023 年 11 月 9 日

の新しいパイプラインタイプ V2 パイプライン、Git タグのトリガー、および のパイプライン変数の更新 CodePipeline	でパイプラインタイプを選択できるようになりました CodePipeline。V2 タイプのパイプラインで、Git タグでパイプラインを開始するためのトリガー設定を使用できるようになりました。V2 タイプのパイプラインでは、パイプラインレベルの変数を使用して、パイプライン実行の入力パラメータを渡すこともできます。詳細については、「変数」、「チュートリアル: パイプラインレベルの変数を使用する」、「チュートリアル: Git タグを使用してパイプラインを開始する」を参照してください。パイプラインのタイプの詳細については、「パイプラインのタイプ」を参照してください。	2023 年 10 月 24 日
CodePipeline は、失敗したステージのすべてのアクションを再試行できます	で失敗したステージの場合 CodePipeline、パイプラインを再実行せずに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。詳細については、「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」を参照してください。	2023 年 10 月 17 日

GitLab グループのサポート	グループと GitLabやり取りする AWS リソースの接続の設定のサポートが追加されました。詳細については、「GitLab 接続」を参照してください。	2023 年 9 月 15 日
CodePipeline が GitLab.com への接続をサポート	接続を使用して、 GitLab.com とやり取りする 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、「GitLab 接続」およびCodeStarSourceConnection「アクション構造リファレンス」トピックを参照してください。	2023 年 8 月 10 日
CloudFormationStackInstances アクションの更新	CloudFormationStackInstances アクションに RegionConcurrencyType パラメータが追加されました。CloudFormationStackInstances アクションについては、アクションリファレンスページを参照してください。	2023 年 8 月 8 日

CloudFormationStackSet アクションの更新	CloudFormationStackSet アクションに RegionConcurrencyType パラメータが追加されました。CloudFormationStackSet アクションについては、アクションリファレンスページを参照してください。	2023 年 7 月 24 日
マネージドポリシーの更新	AWS 管理ポリシーが更新されAWSCodePipeline_FullAccess ました。「AWS CodePipelineのAWS マネージドポリシー」を参照してください。	2023 年 6 月 21 日
ポーリングパイプラインの移行手順の更新	イベントベースの変更検出を使用するようにポーリングパイプラインを移行 (更新) する手順は、 への通知が有効になっている 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 EventBridge。詳細については、「ポーリングパイプラインをイベントベースの変更検出の使用に移行する」を参照してください。	2023 年 6 月 12 日

マネージドポリシーの更新	AWS 管理ポリシー AWSCodePipeline_FullAccess と AWSCodePipeline_ReadOnlyAccess が追加のアクセス許可で更新されました。詳細については、「AWS CodePipelineAWS 管理ポリシーの更新」を参照してください。	2023 年 5 月 16 日
マネージドポリシーの更新	AWS 管理ポリシー AWSCodePipelineFullAccess および AWSCodePipelineReadOnlyAccess は廃止されました。AWSCodePipeline_FullAccess および AWSCodePipeline_ReadOnlyAccess ポリシーを使用してください。「AWS CodePipeline での AWS マネージドポリシーの更新」を参照してください。	2022 年 11 月 17 日

を使用する手順の更新 CloudTrail	S3 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 AWS CloudFormation スニペットとテンプレートが更新され、 の管理イベントで書き込みと false を選択するオプションが追加されました CloudTrail。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: を使用してパイプラインを作成する AWS CloudFormation」、「プッシュイベントを使用するようにパイプラインを編集する」、および「ポーリングパイプラインを更新する」を参照してください。	2022 年 4 月 27 日
新しくサポートされた Snyk との統合	で Snyk 呼び出しアクションを使用して CodePipeline 、オープンソースコードのセキュリティスキャンを自動化できます。詳細は [Snyk アクションリファレンス] と [統合] を参照してください。	2021 年 6 月 10 日
新しくサポートされる欧州 (ミラノ) リージョン	CodePipeline が欧州 (ミラノ) で利用可能になりました。「制限」および「AWS のサービス のエンドポイント」トピックが更新されました。	2021 年 1 月 27 日

変更の検出は、接続ソースアクションでオフにできます	ソースリポジトリの自動変更検出をオフにするためには、CLI または SDK を使って CodeStarSourceConnection ソースアクションを更新します。CodeStarSourceConnection アクション構造のリファレンストピックが、 DetectChanges パラメータの説明で更新されました。	2021 年 1 月 8 日
CodePipeline で AWS CloudFormation StackSets デプロイアクションがサポート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: をデプロイプロバイダー AWS CloudFormation StackSets として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。AWS CloudFormation StackSets アクション構造リファレンストピックも追加されました。	2020 年 12 月 30 日
新しくサポートされるアジアパシフィック (香港) リージョン	CodePipeline がアジアパシフィック (香港) で利用可能になりました。「制限」および「AWS のサービス のエンドポイント」トピックが更新されました。	2020 年 12 月 22 日

で更新された EventBridge イベントパターンを表示する CodePipeline	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のイベントパターンとステータスがモニタリング CodePipeline イベント に追加されました。	2020 年 12 月 21 日
でインバウンドパイプラインの実行を表示する CodePipeline	コンソールまたは CLI を使ってインバウンド実行を表示できます。詳細については、[インバウンド実行の表示 (コンソール)] と [インバウンド実行のステータス表示 (CLI)] を参照してください。	2020 年 11 月 16 日
の CodeCommit ソースアクションは、完全なクローンオプション CodePipeline をサポートします。	CodeCommit ソースアクションを使用する場合、ダウンストリーム CodeBuild アクションに Git コマンドとメタデータを使用するための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」および「チュートリアル: CodeCommit パイプラインソース でフルクローンを使用する」を参照してください。	2020 年 11 月 11 日

CodePipeline が GitHub および GitHub Enterprise Server への接続をサポート	接続を使用して、、Enterprise Cloud GitHub、 GitHubおよび GitHub Enterprise Server とやり取りする 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 接続」、GitHub 「エンタープライズサーバー接続」、および「チュートリアル: GitHub パイプラインソース でフルクローンを使用する」を参照してください。ソースアクションを持つ GitHub既存のパイプラインがある場合は、 GitHub 「バージョン 1 ソースアクションを GitHub バージョン 2 ソースアクション に更新する」を参照してください。	2020 年 9 月 30 日
CodeBuild アクションは、でのバッチビルドの有効化をサポートします。 AWS CodePipeline	パイプラインの CodeBuild アクションでは、バッチビルドを有効にして、1 回の実行で複数のビルドを実行できます。詳細については、CodeBuild 「アクション構造リファレンス」および「パイプラインの作成 (コンソール) 」を参照してください。	2020 年 7 月 30 日

AWS CodePipeline で AWS AppConfig デプロイアクションがサポート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: をデプロイプロバイダー AWS AppConfig として使用するパイプラインを作成する」では、 AWS AppConfig を使用してパイプラインで設定ファイルをデプロイする手順を説明します。AWS AppConfig 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 6 月 25 日
AWS CodePipeline が AWS GovCloud （米国西部) で Amazon VPC をサポートするようになりました	AWS GovCloud （米国西部) のプライベート Amazon VPC エンドポイント AWS CodePipeline を介して に直接接続できるようになりました。詳細については、Amazon Virtual Private Cloud CodePipeline で を使用する」を参照してください。	2020 年 6 月 2 日

AWS CodePipelineAWS Step Functions が呼び出しアクションをサポートするようになりました	を呼び出しアクションプロバイダー CodePipeline AWS Step Functions として使用するパイプラインを で作成できるようになりました。新しいチュートリアル「チュートリアル: パイプライン AWS Step Functions で 呼び出しアクションを使用する」では、パイプラインからステートマシンの実行を開始する手順について説明します。「AWS Step Functions アクション構造参照」トピックも追加されました。	2020 年 5 月 28 日
接続の表示、一覧表示、更新	コンソールでは、接続を一覧表示、削除、更新できます。「 の接続のリスト CodePipeline」を参照してください。	2020 年 5 月 21 日
Connections が CLI での接続リソースのタグ付けをサポート	接続リソースが AWS CLI でのタグ付けをサポートするようになりました。接続が と統合されるようになりました AWS CodeGuru。Connections IAM アクセス許可リファレンスを参照してください。	2020 年 5 月 6 日
CodePipeline が AWS GovCloud （米国西部) で利用可能になりました	AWS GovCloud （米国西部) CodePipeline で を使用できるようになりました。詳細については、「クォータ」を参照してください。	2020 年 4 月 8 日

クォータのトピックでは、設定可能な CodePipeline サービスクォータを示します。	CodePipeline クォータトピックが再フォーマットされました。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。「」の「クォータ AWS CodePipeline」を参照してください。	2020 年 3 月 12 日
Amazon ECS のデプロイアクションのタイムアウトは設定可能です	Amazon ECS のデプロイアクションのタイムアウトは、最大 1 時間 (デフォルトのタイムアウト) まで設定できます。「AWS のクォータCode Pipeline」を参照してください。	2020 年 2 月 5 日
新しいトピックで、パイプライン実行を停止する方法について説明	パイプラインの実行は で停止できます CodePipeline。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。「」の「パイプラインの実行を停止する方法」および「パイプラインの実行を停止する CodePipeline」を参照してください。	2020 年 1 月 21 日

CodePipeline が接続をサポート	接続を使用して、外部コードリポジトリとやり取りするように AWS リソースを設定できます。各接続は、Bitbucket Cloud などのサードパーティーリポジトリに接続 CodePipeline するためになどのサービスで使用できるリソースです。詳細については、「 での接続の使用 CodePipeline」を参照してください。	2019 年 12 月 18 日
セキュリティ、認証、アクセスコントロールのトピックを更新	のセキュリティ、認証、アクセスコントロール情報は、新しいセキュリティの章にまとめ CodePipeline られました。詳細については、「セキュリティ」を参照してください。	2019 年 12 月 17 日
新しいトピックで、パイプラインで変数を使用する方法について説明	アクションの名前空間を設定し、アクションの実行が完了するたびに変数を生成できるようになりました。これらの名前空間と変数を参照するようにダウンストリームアクションを設定できます。「変数の操作」および「変数」を参照してください。	2019 年 11 月 14 日

新しいトピックで、パイプライン実行のしくみ、実行中にステージがロックされる理由、パイプライン実行が優先されるタイミングについて説明	「ようこそ」セクションには、実行中にステージがロックされる理由や、パイプライン実行が優先される場合の処理など、パイプライン実行がどのように機能するかを説明するトピックが数多く追加されています。これらのトピックには、概念のリスト、DevOpsワークフローの例、パイプラインの構造化方法に関する推奨事項が含まれています。次のトピックが追加されました: 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。	2019 年 11 月 11 日
CodePipeline が通知ルールをサポート	通知ルールを使用して、パイプラインの重要な変更をユーザーに通知できるようになりました。詳細については、「通知ルールを作成する」を参照してください。	2019 年 11 月 5 日

CodeBuild で使用できる環境変数 CodePipeline	パイプラインの CodeBuild ビルドアクションで CodeBuild 環境変数を設定できます。コンソールまたは CLI を使用して、パイプライン構造に `EnvironmentVariables` パラメータを追加できます。「パイプラインを作成する (コンソール)」トピックが更新されています。のアクションリファレンスのアクション設定例CodeBuildも更新されました。	2019 年 10 月 14 日
新しいリージョン	CodePipeline が欧州 (ストックホルム) で利用可能になりました。「制限」および「AWS のサービス のエンドポイント」トピックが更新されました。	2019年9月5日
Amazon S3 のデプロイアクションの既定 ACL とキャッシュコントロールを指定	で Amazon S3 デプロイアクションを作成するときに、既定 ACL とキャッシュ制御オプションを指定できるようになりました CodePipeline。次のトピックが更新されました: パイプラインの作成 (コンソール) 、CodePipeline パイプライン構造リファレンス 、およびチュートリアル: Amazon S3 をデプロイプロバイダー として使用するパイプラインの作成。	2019 年 6 月 27 日

でリソースにタグを追加できるようになりました AWS CodePipeline	タグ付けを使用して、パイプライン、カスタムアクション、ウェブフックなどの AWS CodePipeline リソースを追跡および管理できるようになりました。次の新しいトピックが追加されました: リソースのタグ付け、 CodePipeline リソースへのアクセスを制御するためのタグの使用、でのパイプラインのタグ付け CodePipeline、 でのカスタムアクションのタグ付け CodePipeline、 でのウェブフックのタグ付け CodePipeline。CLI を使用してリソースにタグを付ける方法を示すために、次のトピックが更新されました。パイプラインの作成 (CLI)、カスタムアクションの作成 (CLI)、およびソース の GitHubウェブフックの作成。	2019 年 5 月 15 日

でアクション実行履歴を表示できるようになりました AWS CodePipeline	パイプラインにおけるすべてのアクションについて、過去の実行に関する詳細を表示できるようになりました。これらの詳細には、開始時刻と終了時刻、継続時間、アクションの実行 ID、ステータス、入力および出力アーティファクトの場所の詳細、外部リソースの詳細などが含まれます。「パイプラインの詳細と履歴を表示する」トピックが更新されて、このサポートが反映されています。	2019 年 3 月 20 日
AWS CodePipeline が へのアプリケーションの公開をサポートするようになりました AWS Serverless Application Repository	サーバーレスアプリケーションを に発行 CodePipeline するパイプラインを で作成できるようになりました AWS Serverless Application Repository。新しいチュートリアル「チュートリアル: アプリケーションを に公開する」では AWS Serverless Application Repository、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および設定する手順について説明します AWS Serverless Application Repository。	2019 年 3 月 8 日

AWS CodePipeline がコンソールでクロスリージョンアクションをサポートするようになりました	AWS CodePipeline コンソールでクロスリージョンアクションを管理できるようになりました。[クロスリージョンアクションを追加する] が更新されて、パイプラインとは異なる AWS リージョンでアクションを追加、編集、または削除する手順が示されています。「パイプラインの作成」、「パイプラインの編集」、およびCodePipeline「パイプライン構造リファレンス」のトピックが更新されました。	2019 年 2 月 14 日
AWS CodePipeline が Amazon S3 デプロイをサポート	Amazon S3 CodePipeline をデプロイアクションプロバイダーとして使用するパイプラインを で作成できるようになりました。 Amazon S3 新しいチュートリアル「チュートリアル: Amazon S3 をデプロイプロバイダー として使用するパイプラインを作成する」では、 を使用して Amazon S3 バケットにサンプルファイルをデプロイする手順について説明します CodePipeline。CodePipeline パイプライン構造リファレンストピックも更新されました。	2019 年 1 月 16 日

AWS CodePipeline で Alexa Skills Kit のデプロイがサポートされるようになりました	CodePipeline と Alexa Skills Kit を使用して、Alexa スキル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「チュートリアル: Amazon Alexa スキル をデプロイするパイプラインを作成する」には、 AWS CodePipeline が Alexa Skills Kit デベロッパーアカウントに接続し、サンプルスキルをデプロイするパイプラインを作成するための認証情報を作成する手順が含まれています。CodePipeline パイプライン構造リファレンストピックが更新されました。	2018 年 12 月 19 日
AWS CodePipeline が 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 AWS PrivateLink	VPC 内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 CodePipeline でを使用する」を参照してください。	2018 年 12 月 6 日

AWS CodePipeline で Amazon ECR ソースアクションと ECS からCodeDeploy デプロイへのアクションがサポートされるようになりました	コンテナベースのアプリケーションの継続的なデプロイのために、Amazon ECR および Amazon ECS CodeDeploy で CodePipeline および を使用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-to-CodeDeploy deployment を使用してパイプライン」を作成する」には、コンソールを使用して、イメージリポジトリに保存されているコンテナアプリケーションを CodeDeploy トラフィックルーティング付きの Amazon ECS クラスターに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。「パイプライン」と「CodePipeline パイプライン構造の作成」リファレンストピックが更新されました。	2018 年 11 月 27 日
AWS CodePipeline がパイプラインでクロスリージョンアクションをサポートするようになりました	新しいトピック「クロスリージョンアクションの追加」には、 AWS CLI または 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 AWS CloudFormation を追加する手順が含まれています。「パイプラインの作成」、「パイプラインの編集」、および「CodePipeline パイプライン構造リファレンス」のトピックが更新されました。	2018 年 11 月12 日

AWS CodePipeline が Service Catalog と統合されるようになりました	パイプラインへのデプロイアクションとして Service Catalog を追加できるようになりました。これにより、ソースリポジトリを変更したときに製品の更新を Service Catalog に発行するパイプラインを設定できます。「統合」トピックは更新され、Service Catalog のサポートが反映されています。2 つの Service Catalog チュートリアルが「AWS CodePipeline チュートリアル」セクションに追加されました。	2018 年 10 月 16 日
AWS CodePipeline が と統合されるようになりました AWS Device Farm	をテストアクション AWS Device Farm としてパイプラインに追加できるようになりました。これにより、パイプラインを設定してモバイル アプリケーションをテストすることができます。の統合トピックが更新され、 に対するこのサポートが反映されました AWS Device Farm。AWS Device Farm チュートリアル のセクションには、2 つの AWS CodePipeline チュートリアルが追加されました。	2018 年 7 月 19 日

AWS CodePipeline ユーザーガイドの更新通知が RSS から利用可能に	CodePipeline ユーザーガイドの HTML バージョンでは、ドキュメントの更新履歴ページに記載されている更新の RSS フィードがサポートされるようになりました。RSS フィードには、2018 年 6 月 30 日以降に行われた更新が含まれています。以前に発表された更新は、「ドキュメントの更新履歴」ページで引き続き利用できます。このフィードをサブスクライブするには、トップメニューパネルの RSS ボタンを使用します。	2018 年 6 月 30 日

以前の更新

次の表は、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ています。

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変更を検出する	コンソールでパイプラインを作成または編集すると、CodePipeline は GitHub ソースリポジトリへの変更を検出するウェブフックを作成し、パイプラインを開始します。パイプラインの移行については、「変更検出にウェブフックを使用するようにパイプラインを設定する GitHub」を参照してください。詳細については、「 でパイプライン実行を開始する CodePipeline」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、 は Amazon CloudWatch Events ルールと、Amazon S3 ソースバケットへ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 CodePipeline ようになり	2018 年 3 月 22 日

変更	説明	変更日
	ました。パイプラインの移行については、「ソースアクションと変更検出方法」を参照してください。 チュートリアル: シンプルなパイプラインを作成する (S3 バケット) が更新され、Amazon S3 ソースを選択したときに Amazon CloudWatch Events ルールと証跡がどのように作成されるかが示されます。 でパイプラインを作成する CodePipelineおよび でパイプラインを編集する CodePipelineも更新されました。 詳細については、「でパイプラインを開始する CodePipeline」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。「のクォータ AWS CodePipeline」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	コンテナベースのアプリケーションの継続的なデプロイに CodePipeline と Amazon ECS を使用できるようになりました。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、との製品とサービスの統合 CodePipeline 、でパイプラインを作成する CodePipeline、CodePipeline パイプライン構造リファレンス のトピックを更新しました。	2017 年 12 月 12 日

変更	説明	変更日
トピックの更新	コンソールでパイプラインを作成または編集すると、 は CodeCommit リポジトリへの変更を検出し、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 CodePipeline ようになりました。既存のパイプラインの移行については、「ソースアクションと変更検出方法」を参照してください。 チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ) は、 CodeCommit リポジトリとブランチを選択したときに Amazon CloudWatch Events ルールとロールがどのように作成されるかを示すように更新されました。 でパイプラインを作成する CodePipeline と でパイプラインを編集する CodePipeline も更新されました。 詳細については、「でパイプラインを開始する CodePipeline」を参照してください。	2017 年 10 月 11 日
新しく更新されたトピック	CodePipeline では、Amazon CloudWatch Events および Amazon Simple Notification Service (Amazon SNS) によるパイプライン状態変更通知のサポートが組み込まれました。新しいチュートリアル「チュートリアル: パイプラインの状態変更に関する E メール通知を受信するように CloudWatch イベントルールを設定する」が追加されました。詳細については、「 CodePipeline イベントのモニタリング」を参照してください。	2017 年 9 月 8 日

変更	説明	変更日
新しく更新されたトピック	Amazon CloudWatch Events アクションのターゲット CodePipeline として を追加できるようになりました。Amazon CloudWatch Events ルールは、ソースの変更を検出して、変更が発生した直後にパイプラインを起動す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実行を実行するように設定することもできます。PollForSourceChanges ソースアクション設定オプションに関する情報が追加されました。詳細については、「でパイプラインを開始する CodePipeline」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。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2017 年 7 月 27 日
新しいリージョン	CodePipeline が、米国西部 (北カリフォルニア)、カナダ (中部)、欧州 (ロンドン) で利用可能になりました。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2017 年 6 月 29 日
トピックの更新	パイプラインの最新の実行だけでなく過去の実行についても、詳細を表示できるようになりました。これらの詳細には、開始時刻と終了時刻、継続時間、実行 ID が含まれます。最新の 12 か月間の最大 100 のパイプラインの実行について、詳細を表示できるようになりました。このサポートを反映するように、「でパイプラインと詳細を表示する CodePipeline」、「CodePipeline アクセス許可リファレンス」、「のクォータ AWS CodePipeline」のトピックを更新しました。	2017 年 6 月 22 日
トピックの更新	「Nouvola」が「テストアクションの統合」の利用可能なアクションのリストに追加されました。	2017 年 5 月 18 日

変更	説明	変更日
トピックの更新	AWS CodePipeline ウィザードで、ステップ 4: ベータページの名前がステップ 4: をデプロイ 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。 との製品とサービスの統合 CodePipeline、、および のトピックCodePipeline パイプライン構造リファレンスが更新されでパイプラインを編集する CodePipeline、 に対するこのサポートが反映されました AWS CodeBuild。	2017 年 3 月 8 日

変更	説明	変更日
新しく更新されたトピック	コンテンツのテーブルは再編成され、パイプライン、アクション、ステージ移行のセクションが追加されています。CodePipeline チュートリアル用の新しいセクションが追加されました。より使いやすくするため、「との製品とサービスの統合 CodePipeline」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」では、AWS Identity and Access Management (IAM) と を使用して、認証情報を使用して リソースに安全にアクセスする CodePipeline のに役立つ包括的な情報を提供します。これらの認証情報は、Amazon S3 バケットからのアーティファクトの配置と取得、パイプラインへの 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。	2017 年 2 月 8 日
新しいリージョン	CodePipeline がアジアパシフィック (東京) で利用可能になりました。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 14 月 12 日
新しいリージョン	CodePipeline が南米 (サンパウロ) で利用可能になりました。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 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、すぐにデプロイできる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 コンソールで作成できます。その後、パイプラインの一部として、ビルドプロジェクトの出力をデプロイできます。 トピック との製品とサービスの統合 CodePipeline、でパイプラインを作成する CodePipeline、認証とアクセスコントロール、および CodePipeline パイプライン構造リファレンスが更新され、 に対するこのサポートが反映されました AWS CodeBuild。 AWS CloudFormation と AWS サーバーレスアプリケーションモデル CodePipeline で 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「との製品とサービスの統合 CodePipeline」は更新され、この新しいサポートが反映されています。 との製品とサービスの統合 CodePipeline は、アクションタイプ別にグループ AWS およびパートナーサービスに再編成されました。	2016 年 12 月 1 日
新しいリージョン	CodePipeline が欧州 (フランクフルト) で利用可能になりました。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 11 月 16 日

変更	説明	変更日
トピックの更新	AWS CloudFormation を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 との製品とサービスの統合 CodePipeline、でパイプラインを作成する CodePipeline、認証とアクセスコントロール、および が更新され、 に対するこのサポートが反映されCodePipeline パイプライン構造リファレンスました 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。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 10 月 26 日
新しいリージョン	CodePipeline がアジアパシフィック (シンガポール) で利用可能になりました。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 10 月 20 日
新しいリージョン	CodePipeline が米国東部 (オハイオ) リージョンで利用可能になりました。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 10 月 17 日
トピックの更新	「でパイプラインを作成する CodePipeline」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	「での承認アクションの管理 CodePipeline」セクションは更新され、承認アクションのレビュー担当者が E メール通知から直接 [Approve or reject the revision] フォームを開くことができるように機能強化されています。	2016 年 9 月 14 日

変更	説明	変更日
新しく更新されたトピック	新しいトピックで、ソフトウェアリリースのパイプラインで現在流れているコードの変更の詳細を表示する方法について説明しています。手動の承認アクションやパイプラインのトラブルシューティングの失敗を確認する場合、この情報にすばやくアクセスできると便利です。 新しいセクション「パイプラインのモニタリング」は、パイプラインのステータスおよび進行状況のモニタリングに関するすべてのトピックの中心となる場所です。	2016 年 9 月 08 日
新しく更新されたトピック	新しいセクション「での承認アクションの管理 CodePipeline」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。「のクォータ AWS CodePipeline」トピックおよび「リージョンおよびエンドポイント」トピックは更新されています。	2016 年 6 月 23 日
新しいトピック	ステージ内で失敗したアクション、または同時に失敗したアクションのグループを再試行する方法について説明した、新しいトピック「ステージ内の失敗したアクションを再試行する」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	でパイプラインを作成する CodePipeline、Authentication and Access Control、、および を含む多くのトピックが更新されCodePipeline パイプライン構造リファレンス、で作成されたカスタム 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するパイプラインの設定のサポートが反映されとの製品とサービスの統合 CodePipelineました AWS OpsWorks。 CodePipeline のサポート 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「チュートリアル: シンプルなパイプラインを作成する (CodeCommit リポジトリ)」が追加されました。このトピックでは、パイプライン内のソースアクションのソースロケーションとして CodeCommit リポジトリとブランチを使用する方法を示すサンプルチュートリアルを提供します。認証とアクセスコントロール CodeCommit、、、 など、 との統合を反映するために、他のいくつかのトピックが更新されましたとの製品とサービスの統合 CodePipelineチュートリアル: 4 ステージのパイプラインを作成するトラブルシューティング CodePipeline。	2016 年 4 月 18 日
新しいトピック	新しいトピック「のパイプラインで AWS Lambda 関数を呼び出す CodePipeline」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
新しいトピック	新しいトピック「との製品とサービスの統合 CodePipeline」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されました。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

変更	説明	変更日
トピックの更新	Solano CI との統合の詳細を「との製品とサービスの統合 CodePipeline」に追加しました。	2015 年 11 月 17 日
トピックの更新	Jenkins 用 CodePipeline プラグインは、Jenkins 用プラグインのライブラリの一部として Jenkins Plugin Manager から利用できるようになりました。プラグインのインストール手順は、「チュートリアル: 4 ステージのパイプラインを作成する」で更新されています。	2015 年 9 月 11 日
新しいリージョン	CodePipeline が米国西部 (オレゴン) リージョンで利用可能になりました。「のクォータ AWS CodePipeline」トピックが更新されました。リンクが「リージョンおよびエンドポイント」に追加されています。	2015 年 10 月 22 日
新しいトピック	新しいトピック「の Amazon S3 に保存されているアーティファクトのサーバー側の暗号化を設定する CodePipeline」および「別の AWS アカウントのリソース CodePipeline を使用するパイプラインを に作成する」が追加されました。「Authentication and Access Control」、「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」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	「でカスタムアクションを作成して追加する CodePipeline」トピックは更新され、`inputArtifactDetails` および `outputArtifactDetails` など、構造の変更が反映されています。	2015 年 8 月 17 日
トピックの更新	サービスロールおよび Elastic Beanstalk に関する問題のトラブルシューティング手順の変更を反映するために、トラブルシューティング CodePipeline 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	認証とアクセスコントロールのトピックが、 のサービスロール CodePipelineに対する最新の変更で更新されました。	2015 年 8 月 6 日

変更	説明	変更日
新しいトピック	「トラブルシューティング CodePipeline」トピックが追加されました。「チュートリアル: 4 ステージのパイプラインを作成する」の IAM ロールおよび Jenkins について更新された手順が追加されました。	2015 年 7 月 24 日
トピックの更新	「チュートリアル: シンプルなパイプラインを作成する (S3 バケット)」および「チュートリアル: 4 ステージのパイプラインを作成する」のサンプルファイルをダウンロードするための更新後のステップが追加されています。	2015 年 7 月 22 日
トピックの更新	サンプルファイルに関するダウンロードの問題の一時回避策が「チュートリアル: シンプルなパイプラインを作成する (S3 バケット)」に追加されました。	2015 年 7 月 17 日
トピックの更新	制限の変更に関する情報を示すリンクが「のクォータ AWS CodePipeline」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 ユーザーガイドの最初のパブリックリリースです。	2015 年 7 月 9 日

AWS 用語集

最新の AWS 用語については、「 AWS の用語集 リファレンス」のAWS 「 用語集」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。