AWS CodeBuild

ユーザーガイド

API Version 2016-10-06

AWS CodeBuild: ユーザーガイド

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

AWS CodeBuild とは

AWS CodeBuild はクラウドで動作する、完全マネージド型のビルドサービスです。AWS CodeBuild はソースコードをコンパイルし、単体テストを実行して、すぐにデプロイできるアーティファクトを生成します。AWS CodeBuild により、独自のビルドサーバーのプロビジョニング、管理、スケーリングが不要になります。Apache Maven、Gradle などの最も一般的なプログラミング言語とビルドツール用のパッケージ済みのビルド環境を提供します。AWS CodeBuild のビルド環境をカスタマイズして、独自のビルドツールを使用することもできます。AWS CodeBuild はピーク時のビルドリクエストに合わせて自動的にスケーリングします。

AWS CodeBuild には、以下のような利点があります。

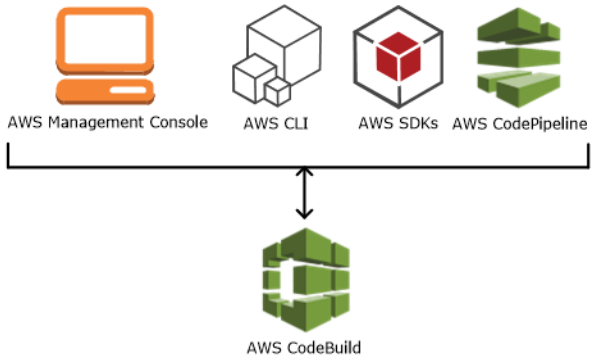
- 完全マネージド型 – AWS CodeBuild では、お客様独自のビルドサーバーをセットアップ、パッチ適用、更新、管理する必要がありません。
- オンデマンド – AWS CodeBuild はビルドのニーズに合わせてオンデマンドでスケーリングされます。料金は、使用したビルド分数に対してのみ発生します。
- すぐに使える – AWS CodeBuild は、事前設定された最も一般的なプログラミング言語でのビルド環境を提供します。最初のビルドを開始するには、ビルドスクリプトを指すだけです。

詳細については、AWS CodeBuild を参照してください。

トピック

AWS CodeBuild を実行する方法

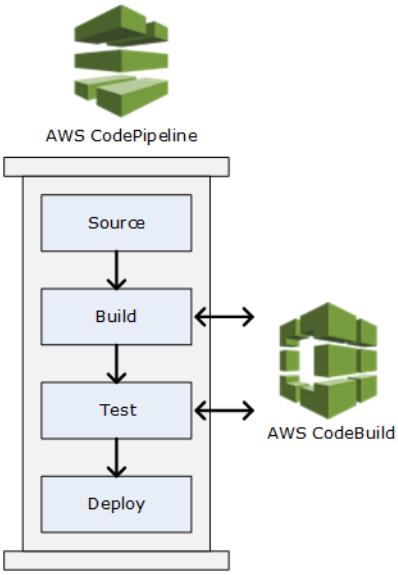
AWS CodeBuild を実行するには、AWS CodeBuild または AWS CodePipeline コンソールを使用します。また、AWS Command Line Interface(AWS CLI) または AWS SDK を使用して、AWS CodeBuild の実行を自動化することもできます。



AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用して AWS CodeBuild を実行するには、「AWS CodeBuild を直接実行する (p. 114)」を参照してください。

次の図に示すように、AWS CodePipeline のパイプラインのビルドまたはテストステージに、ビルドまたはテストアクションとして AWS CodeBuild を追加できます。AWS CodePipeline は、コードをリリース

するために必要な手順のモデル化、視覚化、および自動化ができる継続的なデリバリサービスです。これには、コードの構築が含まれます。パイプラインは、リリースプロセスを通したコードの変更を説明したワークフロー構造です。



AWS CodePipeline を使用してパイプラインを作成し、AWS CodeBuild ビルドまたはテストアクションを追加するには、「AWS CodeBuild で AWS CodePipeline を使用する (p. 126)」を参照してください。AWS CodePipeline の詳細については、AWS CodePipeline ユーザーガイドを参照してください。

AWS CodeBuild の料金表

詳細については、「AWS CodeBuild の料金表」を参照してください。

AWS CodeBuild の使用を開始するには

次の手順を実行することをお勧めします。

1. AWS CodeBuild の詳細については、「概念 (p. 2)」の情報を参照してください。
2. 「ご利用開始にあたって (p. 4)」の手順に従って、サンプルのシナリオで AWS CodeBuild を試してみてください。
3. 自分のシナリオで AWS CodeBuild を使用するには、「ビルドを計画する (p. 94)」の手順に従います。

AWS CodeBuild の概念

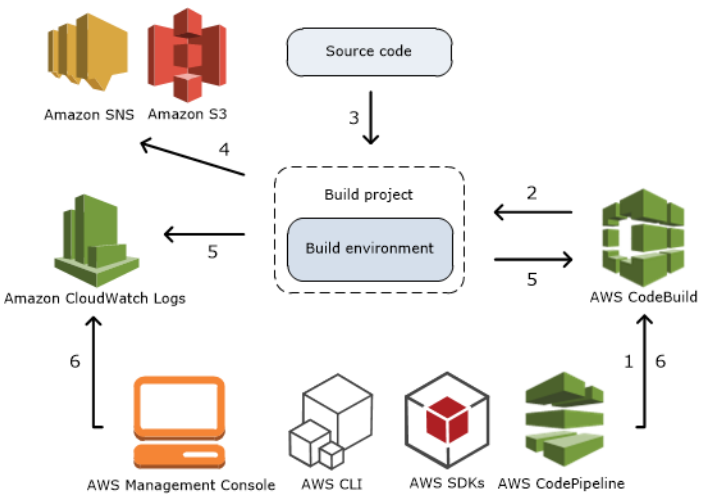
以下の概念は、AWS CodeBuild の仕組みを理解するうえで重要です。

トピック

- AWS CodeBuild の詳細 (p. 3)
- 次のステップ (p. 3)

AWS CodeBuild の詳細

次の図は、AWS CodeBuild でビルドを実行するとどうなるかを示しています。



1. 入力として、AWS CodeBuild にビルドプロジェクトを指定する必要があります。ビルドプロジェクトは、AWS CodeBuild がビルドを実行する方法を定義します。これには、ソースコード、使用するビルド環境、実行するビルドの入手先、ビルド出力の保存先などの情報が含まれます。 ビルド環境は、ビルドを実行するために AWS CodeBuild が使用するオペレーティングシステム、プログラミング言語ランタイム、およびツールの組み合わせを表します。 詳細については、以下を参照してください。

   - ビルドプロジェクトを作成する (p. 146)
   - ビルド環境リファレンス (p. 102)
2. AWS CodeBuild は、ビルドプロジェクトを使用して、ビルド環境を作成します。
3. AWS CodeBuild は、ビルド環境にソースコードをダウンロードし、ビルドプロジェクトで定義されている、または、ソースコードに直接含まれているビルド仕様 (ビルドスペック) を使用します。ビルド仕様は、AWS CodeBuild がビルドを実行するために使用する YAML 形式のビルドコマンドと関連設定のコレクションです。 詳細については、「ビルドスペックリファレンス (p. 95)」を参照してください。
4. ビルド出力がある場合、ビルド環境はその出力を Amazon S3 バケットにアップロードします。ビルド環境では、ビルド仕様で指定したタスク (たとえば、ビルド通知を Amazon SNS トピックに送信するなど) を実行することもできます。例については、「ビルド通知サンプル (p. 42)」を参照してください。
5. ビルドが実行されている間に、ビルド環境は AWS CodeBuild および Amazon CloudWatch Logs に情報を送信します。
6. ビルドが実行されている間は、AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用して、AWS CodeBuild の要約されたビルド情報および Amazon CloudWatch Logs の詳細なビルド情報を取得できます。AWS CodePipeline を使用して、ビルドを実行する場合は、AWS CodePipeline から制限されたビルド情報を取得できます。

次のステップ

AWS CodeBuild の詳細について説明したので、以下の手順を完了することをお勧めします。

1. 「ご利用開始にあたって (p. 4)」の手順に従って、サンプルのシナリオで AWS CodeBuild を試してみてください。
2. 自分のシナリオで AWS CodeBuild を使用するには、「ビルドを計画する (p. 94)」の手順に従います。

AWS CodeBuild の使用開始

このチュートリアルでは、AWS CodeBuild を使用して、サンプルのソースコード入力ファイル (ビルド入力アーティファクトまたはビルド入力と呼ばれる) のコレクションをソースコードのデプロイ可能なバージョン (ビルド出力アーティファクトまたは ビルド出力と呼ばれる) に組み込みます。具体的には、一般的なビルドツールである Apache Maven を使用して Java クラスファイルのセットを Java アーカイブ (JAR) ファイルにビルドするように AWS CodeBuild に指示します。このチュートリアルを完了するために、Apache Maven または Java に精通している必要はありません。

> Important
>
> このチュートリアルを完了すると、AWS アカウントに料金が発生する可能性があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

トピック

ステップ 1: Amazon S3 バケットを作成または使用してビルド入力と出力を保存する

このチュートリアルを完了するには、2 つの Amazon S3 バケットが必要です。

- これらのバケットの 1 つにビルド入力 (入力バケットと呼ばれる) が格納されます。 このチュートリアルでは、この入力バケットを `codebuild-`*`region-ID`*`-`*`account-ID`*`-input-bucket` という名前にし、*`region-ID`* はバケットの AWS リージョンを、*`account-ID`* は、AWS アカウント ID を表します。
- 別のバケットにはビルド出力 (出力バケットと呼ばれる) が格納されます。このチュートリアルでは、この出力バケットを `codebuild-`*`region-ID`*`-`*`account-ID`*`-output-bucket` という名前にします。

これらのバケットのいずれかに別の名前を選択した場合は、このチュートリアル全体で置き換えてください。

これらの 2 つのバケットは、ビルドと同じ AWS リージョン内にある必要があります。たとえば、ビルドの実行先として 米国東部 (オハイオ) リージョンを AWS CodeBuild に指示する場合、これらのバケットも米国東部 (オハイオ) リージョン内にあることが必要です。

バケットを作成するには、Amazon Simple Storage Service ユーザーガイドの「バケットの作成」を参照してください。

Note

このチュートリアルでは、単一のバケットを使用できます。ただし、2 つのバケットを使用すると、ビルド入力の送信元、ビルド出力の送信先を簡単に確認できます。
AWS CodeBuild は、AWS CodeCommit、GitHub、および Bitbucket の各リポジトリに保存されているビルド入力もサポートしますが、このチュートリアルではこれらの使用方法については説明しません。詳細については、「ビルドを計画する (p. 94)」を参照してください。

ステップ 2: ビルドするソースコードを作成する

このステップでは、AWS CodeBuild が出力バケットにビルドするソースコードを作成します。このソースコードは 2 つの Java クラスファイルと Apache Maven プロジェクトオブジェクトモデル (POM) ファイルで構成されています。

1. ローカルコンピュータまたはインスタンスの空のディレクトリに、このディレクトリ構造を作成します。

```
(root directory name)
    `-- src
         |-- main
         |     `-- java
         `-- test
               `-- java
```

2. 任意のテキストエディタを使用して、このファイルを作成し、`MessageUtil.java` という名前を付けて、`src/main/java` ディレクトリに保存します。

```
public class MessageUtil {
  private String message;

  public MessageUtil(String message) {
    this.message = message;
  }

  public String printMessage() {
    System.out.println(message);
    return message;
  }

  public String salutationMessage() {
    message = "Hi!" + message;
    System.out.println(message);
    return message;
  }
}
```

このクラスファイルは、渡された文字列を出力として作成します。`MessageUtil` コンストラクタは、文字列を設定します。`printMessage` メソッドは出力を作成します。`salutationMessage` メソッドが `Hi!` を出力した後に文字列が続きます。

3. このファイルを作成し、`TestMessageUtil.java` という名前を付けて、`/src/test/java` ディレクトリに保存します。

```
import org.junit.Test;
import org.junit.Ignore;
```

```
import static org.junit.Assert.assertEquals;

public class TestMessageUtil {

  String message = "Robert";
  MessageUtil messageUtil = new MessageUtil(message);

  @Test
  public void testPrintMessage() {
    System.out.println("Inside testPrintMessage()");
    assertEquals(message,messageUtil.printMessage());
  }

  @Test
  public void testSalutationMessage() {
    System.out.println("Inside testSalutationMessage()");
    message = "Hi!" + "Robert";
    assertEquals(message,messageUtil.salutationMessage());
  }
}
```

このクラスファイルは `MessageUtil` クラスの `message` 変数を `Robert` に設定します。その後、文字列 `Robert` および `Hi!Robert` が出力に表示されているかどうかを調べることによって、`message` 変数が正常に設定されたかどうかを調べます。

4. このファイルを作成し、`pom.xml` という名前を付けて、ルート (最上位) ディレクトリに保存します。

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example</groupId>
  <artifactId>messageUtil</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
  <name>Message Utility Java Sample App</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Apache Maven はこのファイルの指示に従って、`MessageUtil.java` および `TestMessageUtil.java` ファイルを `messageUtil-1.0.jar` という名前のファイルに変換し、指定されたテストを実行します。

この時点で、ディレクトリ構造は次のようになります。

```
(root directory name)
    |-- pom.xml
    `-- src
         |-- main
         |     `-- java
         |           `-- MessageUtil.java
         `-- test
               `-- java
```

```
                `-- TestMessageUtil.java
```

ステップ 3: ビルドスペックを作成する

このステップでは、ビルド仕様 (ビルドスペック) ファイルを作成します。ビルド仕様は、AWS CodeBuild がビルドを実行するために使用する YAML 形式のビルドコマンドと関連設定のコレクションです。 ビルドスペックがないと、AWS CodeBuild はビルド入力をビルド出力に正常に変換できません。また、ビルド環境でビルド出力アーティファクトを特定して出力バケットにアップロードすることもできません。

このファイルを作成し、`buildspec.yml` という名前を付けて、ルート (最上位) ディレクトリに保存します。

```
version: 0.2

phases:
  install:
    commands:
      - echo Nothing to do in the install phase...
  pre_build:
    commands:
      - echo Nothing to do in the pre_build phase...
  build:
    commands:
      - echo Build started on `date`
      - mvn install
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - target/messageUtil-1.0.jar
```

> **Important**
>
> ビルドスペック宣言は有効な YAML である必要があるため、ビルドスペック宣言のスペースは重要です。ビルドスペック宣言のスペース数が YAML と一致しない場合、ビルドは即座に失敗する場合があります。YAML validator を使用して、ビルドスペック宣言が有効な YAML かどうかをテストできます。

> **Note**
>
> ソースコードにビルド仕様ファイルを含める代わりに、ビルドプロジェクトを作成するときに個別にビルドコマンドを宣言することができます。 これは、毎回ソースコードのリポジトリを更新せずに、異なるビルドコマンドでソースコードをビルドする場合に役立ちます。詳細については、「ビルドスペックの構文 (p. 96)」を参照してください。

このビルドスペック宣言では。

- `version` は、使用されているビルドスペックスタンダードのバージョンを表します。このビルドスペック宣言では、最新バージョン `0.2` が使用されます。
- `phases` は、AWS CodeBuild にコマンドの実行を指示するビルドフェーズを表します。これらのビルドフェーズは `install`、`pre_build`、`build`、`post_build` として、ここにリストされています。これらのビルドフェーズ名のスペルを変更することはできず、追加のビルドフェーズ名を作成することもできません。

  この例では、`build` フェーズ中に、AWS CodeBuild が `mvn install` コマンドを実行します。このコマンドは、コンパイルされた Java クラスファイルをビルド出力アーティファクトにコンパイル、テスト、パッケージ化するように Apache Maven に指示します。完全にするために、この例では、いくつかの `echo` コマンドが各ビルドフェーズに配置されています。このチュートリアルの後半で詳細なビルド

情報を表示すると、これらの `echo` コマンドの出力は、AWS CodeBuild がコマンドを実行する方法とその順序を理解するのに役立ちます。(この例にはすべてのビルドフェーズが含まれていますが、そのフェーズの間にコマンドを実行しない場合は、ビルドフェーズを含める必要はありません。)各ビルドフェーズについて、AWS CodeBuild は最初から最後まで、各コマンドを一度に 1 つずつ、指定された順序で実行します。

- `artifacts` は、AWS CodeBuild が出力バケットにアップロードする一連のビルド出力アーティファクトを表します。`files` は、ビルド出力に含めるファイルを表します。AWS CodeBuild は、ビルド環境の `messageUtil-1.0.jar` 相対ディレクトリにある、単一の `target` ファイルをアップロードします。ファイル名 `messageUtil-1.0.jar` およびディレクトリ名 `target` は、この例でのみ Apache Maven がビルド出力アーティファクトを作成して格納する方法に基づいています。独自のビルドでは、これらのファイル名とディレクトリは異なります。

詳細については、「ビルドスペックリファレンス (p. 95)」を参照してください。

この時点で、ディレクトリ構造は次のようになります。

```
(root directory name)
    |-- pom.xml
    |-- buildspec.yml
    `-- src
         |-- main
         |     `-- java
         |           `-- MessageUtil.java
         `-- test
               `-- java
                     `-- TestMessageUtil.java
```

ステップ 4: ソースコードとビルドスペックを入力バケットに追加する

このステップでは、入力バケットにソースコードとビルドスペックファイルを追加します。

オペレーティングシステムの zip ユーティリティを使用して、`MessageUtil.java`、`TestMessageUtil.java`、`pom.xml`、および `buildspec.yml` を含む `MessageUtil.zip` という名前のファイルを作成します。

`MessageUtil.zip` ファイルのディレクトリ構造は、次のようになっている必要があります。

```
MessageUtil.zip
    |-- pom.xml
    |-- buildspec.yml
    `-- src
         |-- main
         |     `-- java
         |           `-- MessageUtil.java
         `-- test
               `-- java
                     `-- TestMessageUtil.java
```

Important

(root directory name) ディレクトリを含めないでください。*(root directory name)* ディレクトリ内のディレクトリとファイルのみを含めます。

`MessageUtil.zip` ファイルを `codebuild-`*region-ID*`-`*account-ID*`-input-bucket` という名前の入力バケットにアップロードします。

Important

AWS CodeCommit、GitHub、および Bitbucket の各リポジトリでは、規約に従って、`buildspec.yml` というビルドスペックファイルを各リポジトリのルート (最上位) に保存するか、ビルドスペック宣言をビルドプロジェクト定義の一部として含める必要があります。リポジトリのソースコードとビルドスペックファイルを含む ZIP ファイルを作成しないでください。Amazon S3 バケットに保存されたビルド入力に限り、ソースコードおよび規約に基づく `buildspec.yml` というビルドスペックファイルを圧縮した ZIP ファイルをルート (最上位) に作成するか、ビルドスペック宣言をビルドプロジェクト定義の一部として含める必要があります。ビルドスペックファイルに別の名前を使用するか、ルート以外の場所でビルドスペックを参照する場合は、ビルドプロジェクト定義の一部としてビルドスペックの上書きを指定できます。詳細については、「ビルドスペックのファイル名とストレージの場所 (p. 95)」を参照してください。

ステップ 5: ビルドプロジェクトを作成する

このステップでは、AWS CodeBuild がビルドの実行に使用するビルドプロジェクトを作成します。ビルドプロジェクトは、AWS CodeBuild がビルドを実行する方法を定義します。これには、ソースコード、使用するビルド環境、実行するビルドの入手先、ビルド出力の保存先などの情報が含まれます。 ビルド環境は、ビルドを実行するために AWS CodeBuild が使用するオペレーティングシステム、プログラミング言語ランタイム、およびツールの組み合わせを表します。 ビルド環境は Docker イメージとして表されます。(詳細については、Docker Docs ウェブサイトの「Docker Overview」を参照してください。) このビルド環境では、AWS CodeBuild に、Java 開発キット (JDK) のバージョンおよび Apache Maven が含まれる Docker イメージを使用するように指示します。

AWS CodeBuild コンソール (p. 9)または AWS CLI (p. 10) を使用して、このステップを完了できます。

Note

AWS CodeBuild は、AWS CodeBuild コンソール、AWS CodePipeline、AWS CLI、または AWS SDK を通じて、いくつかの方法で操作できます。このチュートリアルでは、AWS CodeBuild コンソールと AWS CLI を使用する方法を示しています。AWS CodePipeline を使用する方法については、「AWS CodeBuild で AWS CodePipeline を使用する (p. 126)」を参照してください。AWS SDK の使用方法については、「AWS CodeBuild を直接実行する (p. 114)」を参照してください。

ビルドプロジェクトを作成するには (コンソール)

1. Sign in to the AWS マネジメントコンソール and open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. AWS リージョンセレクタで、AWS CodeBuild をサポートするリージョンを選択します。詳細については、アマゾン ウェブ サービス全般のリファレンスの「リージョンとエンドポイント」トピックにある「AWS CodeBuild」を参照してください。
3. ウェルカムページが表示された場合は、[Get started] を選択します。

   ウェルカムページが表示されない場合は、ナビゲーションペインで [Build projects] を選択し、次に [Create project] を選択します。
4. [Configure your project] ページで、[Project name] にこのビルドプロジェクトの名前を入力します (この例では `codebuild-demo-project`)。ビルドプロジェクトの名前は、各 AWS アカウントで一意である必要があります。別の名前を使用する場合は、このチュートリアル全体でそれを置き換えてください。

   Note

   [Configure project] ページで、次のようなエラーメッセージが表示されることがあります: "ユーザー: ***user-ARN*** は次のことを実行する権限がありません: codebuild:ListProjects"。 こ

れは、多くの場合、コンソールで AWS CodeBuild を使用するための十分な権限を持たない IAM ユーザーとして AWS マネジメントコンソール にサインインしたためです。これを修正するには、AWS マネジメントコンソール, からサインアウトし、次の IAM エンティティのいずれかに属する認証情報でサインインし直します。

- AWS ルートアカウント。これは推奨されません。詳細については、IAM ユーザーガイドの「アカウント root ユーザー」を参照してください。
- AWS アカウントの管理者 IAM ユーザー。詳細については、IAM ユーザーガイドの「最初の IAM 管理者ユーザーおよびグループの作成」を参照してください。
- AWSCodeBuildAdminAccess、AmazonS3ReadOnlyAccess、IAMFullAccess という名前の AWS 管理ポリシーが IAM ユーザーまたは IAM ユーザーが属する IAM グループにアタッチされている AWS アカウントの IAM ユーザー。これらのアクセス許可を持つ IAM ユーザーやグループが AWS アカウントになく、これらのアクセス許可を IAM ユーザやグループに追加できない場合は、AWS アカウント管理者にお問い合わせください。詳細については、「AWS CodeBuild の AWS 管理 (定義済み) ポリシー (p. 198)」を参照してください。

5. [Source: What to build] で、[Source provider] として [Amazon S3] を選択します。
6. [Bucket] で、[codebuild-***region-ID-account-ID***-input-bucket] を選択します。
7. [S3 object key] に「`MessageUtil.zip`」と入力します。
8. [Environment: How to build] で、[Environment image] として [Use an image managed by AWS CodeBuild] を選択したままにします。
9. [Operating system] で、[Ubuntu] を選択します。
10. [Runtime] で、[Java] を選択します。
11. [Version] で、[aws/codebuild/java:openjdk-8] を選択します。
12. [Build specification] で [Use the buildspec.yml in the source code root directory] を選択したままにします。
13. [Artifacts: Where to put the artifacts from this build project] で、[Artifacts type] として [Amazon S3] を選択します。
14. [Artifacts name] は空白のままにします。
15. [Bucket name] で、[codebuild-***region-ID-account-ID***-output-bucket] を選択します。
16. [Service role] で、[Create a service role in your account] を選択したままにして、[Role name] を未変更のままにします。
17. [Continue] を選択します。
18. [Review] ページで、[Save] を選択します。

[ステップ 6: ビルドを実行する (p. 13)] に進んでください。

ビルドプロジェクト (AWS CLI) を作成するには

1. 次のように、AWS CLI を使用して `create-project` コマンドを実行します。

```
aws codebuild create-project --generate-cli-skeleton
```

JSON 形式のデータが出力に表示されます。AWS CLI がインストールされているローカルコンピュータまたはインスタンス上の場所にある `create-project.json` というファイルにデータをコピーします。別のファイル名を使用する場合は、このチュートリアル全体で `create-project.json` をそのファイル名に置き換えてください。

コピーされたデータをこの形式に従って変更して、結果を保存します。

```
{
  "name": "codebuild-demo-project",
  "source": {
```

```
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/java:openjdk-8",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "serviceIAMRole"
}
```

serviceIAMRole を、AWS CodeBuild サービスロールの Amazon リソースネーム (ARN) (例: arn:aws:iam::*account-ID*:role/*role-name*) に置き換えます。サービスロールを作成する場合は、「AWS CodeBuild サービスロールの作成 (p. 185)」を参照してください。

このデータの各要素は以下のとおりです。

- name は、このビルドプロジェクト (この例では codebuild-demo-project) に必要な識別子を表します。別の名前を使用する場合は、この手順全体でそれを置き換えてください。ビルドプロジェクト名は、アカウント内のすべてのビルドプロジェクト間で一意である必要があります。
- source の場合、type はソースコードのリポジトリのタイプを表す必須の値です (この例では、Amazon S3 バケットの場合は S3)。
- source の場合、location は、ソースコードへのパスを表します (この例では、入力バケット名の後に ZIP ファイル名が続きます)。
- artifacts の場合、type は、ビルド出力アーティファクトのリポジトリのタイプを表す必須の値です (この例では、Amazon S3 バケットの場合は S3)。
- artifacts の場合、location は、以前に作成または識別した出力バケットの名前を表します (この例では、codebuild-*region-ID*-*account-ID*-output-bucket)。
- environment の場合、type はビルド環境のタイプを表す必須の値です (現在許容されている値は LINUX_CONTAINER のみです)。
- environment の場合、image は、Docker イメージリポジトリタイプで指定された、このビルドプロジェクトが使用する Docker イメージ名とタグの組み合わせを表す必須の値です (この例では、AWS CodeBuild Docker イメージリポジトリ内の Docker イメージの場合は aws/codebuild/java:openjdk-8)。aws/codebuild/java は、Docker イメージの名前です。openjdk-8 は、Docker イメージのタグです。

  シナリオで使用できる Docker イメージをさらに見つけるには、「ビルド環境リファレンス (p. 102)」を参照してください。
- environment の場合、computeType は、AWS CodeBuild が使用するコンピューティングリソース (この例では BUILD_GENERAL1_SMALL) を表す必須の値です。

  Note

  description、buildspec、auth (type と resource を含む)、path、namespaceType、name (artifacts)、packaging、environmentVariables (name と value を含む)、timeoutInMinutes、encryptionKey、tags (key と value を含む) などの元の JSON 形式のデータで使用可能なその他の値はオプションです。これらは、このチュートリアルで使用されていないため、ここには表示されません。詳細については、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」を参照してください。

2. 保存したばかりのファイルがあるディレクトリに移動してから、create-project コマンドをもう一度実行します。

```
aws codebuild create-project --cli-input-json file://create-project.json
```

成功した場合、次のようなデータが出力に表示されます。

```
{
  "project": {
    "name": "codebuild-demo-project",
    "serviceRole": "serviceIAMRole",
    "tags": [],
    "artifacts": {
      "packaging": "NONE",
      "type": "S3",
      "location": "codebuild-region-ID-account-ID-output-bucket",
      "name": "message-util.zip"
    },
    "lastModified": 1472661575.244,
    "timeoutInMinutes": 60,
    "created": 1472661575.244,
    "environment": {
      "computeType": "BUILD_GENERAL1_SMALL",
      "image": "aws/codebuild/java:openjdk-8",
      "type": "LINUX_CONTAINER",
      "environmentVariables": []
    },
    "source": {
      "type": "S3",
      "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
    },
    "encryptionKey": "arn:aws:kms:region-ID:account-ID:alias/aws/s3",
    "arn": "arn:aws:codebuild:region-ID:account-ID:project/codebuild-demo-project"
  }
}
```

- `project` は、このビルドプロジェクトに関する情報を表します。
  - `tags` は、宣言されたタグを表します。
  - `packaging` は、ビルド出力アーティファクトが出力バケットに保存される方法を表します。`NONE` は、出力バケット内にフォルダが作成され、ビルド出力アーティファクトがそのフォルダ内に格納されることを意味します。
  - `lastModified` は、ビルドプロジェクトに関する情報が最後に変更された時刻 (Unix の時間形式) を表します。
  - `timeoutInMinutes` は、ビルドが完了していない場合、AWS CodeBuild がビルドを停止するまでの時間 (分) を表します。(デフォルトは 60 分です。)
  - `created` は、ビルドプロジェクトが作成された時刻 (Unix の時間形式) を表します。
  - `environmentVariables` は、AWS CodeBuild がビルド中に使用するために宣言されて、使用可能な環境変数を表します。
  - `encryptionKey` は、AWS CodeBuild がビルド出力アーティファクトの暗号化に使用した AWS KMS カスタマーマスターキー (CMK) の Amazon リソースネーム (ARN) を表します。
  - `arn` は、ビルドプロジェクトの ARN を表します。

Note

create-project コマンドの実行後に、次のようなメッセージが出力される場合があります: "ユーザー: ***user-ARN*** は次のことを実行する権限がありません: codebuild:CreateProject"。これは、多くの場合、ビルドプロジェクトの作成に AWS CodeBuild を使用するための十分な権限を持たない IAM ユーザーの認証情報で AWS CLI を設定したためです。これを修正するには、次の IAM エンティティのいずれかに属する認証情報を使用して AWS CLI を設定します。

- AWS ルートアカウント。これは推奨されません。詳細については、IAM ユーザーガイドの「アカウント root ユーザー」を参照してください。
- AWS アカウントの管理者 IAM ユーザー。詳細については、IAM ユーザーガイドの「最初の IAM 管理者ユーザーおよびグループの作成」を参照してください。
- AWSCodeBuildAdminAccess、AmazonS3ReadOnlyAccess、IAMFullAccess という名前の AWS 管理ポリシーが IAM ユーザーまたは IAM ユーザーが属する IAM グループにアタッチされている AWS アカウントの IAM ユーザー。これらのアクセス許可を持つ IAM ユーザーやグループが AWS アカウントになく、これらのアクセス許可を IAM ユーザやグループに追加できない場合は、AWS アカウント管理者にお問い合わせください。詳細については、「AWS CodeBuild の AWS 管理 (定義済み) ポリシー (p. 198)」を参照してください。

ステップ 6: ビルドを実行する

このステップでは、ビルドプロジェクトの設定でビルドを実行するように AWS CodeBuild に指示します。

AWS CodeBuild コンソール (p. 13)または AWS CLI (p. 13) を使用して、このステップを完了できます。

ビルドを実行するには (コンソール)

1. [Build projects] ページが表示されない場合は、ナビゲーションペインで [Build projects] を選択します。
2. ビルドプロジェクトのリストで、[codebuild-demo-project]、[Start build] の順に選択します。
3. [Start new build] ページで、[Start build] を選択します。
4. [ステップ 7: 要約されたビルド情報を表示する (p. 14)] に進んでください。

ビルドを実行するには (AWS CLI)

1. AWS CLI を使用して `start-build` コマンドを実行します。

```
aws codebuild start-build --project-name project-name
```

   project-name を、前の手順のビルドプロジェクト名 (例: `codebuild-demo-project`) に置き換えます。
2. 成功すると、次のようなデータが出力に表示されます。

```
{
  "build": {
    "buildComplete": false,
    "initiator": "user-name",
    "artifacts": {
      "location": "arn:aws:s3:::codebuild-region-ID-account-ID-output-bucket/message-
util.zip"
    },
    "projectName": "codebuild-demo-project",
    "timeoutInMinutes": 60,
    "buildStatus": "IN_PROGRESS",
    "environment": {
      "computeType": "BUILD_GENERAL1_SMALL",
      "image": "aws/codebuild/java:openjdk-8",
      "type": "LINUX_CONTAINER",
      "environmentVariables": []
    },
```

```
    "source": {
      "type": "S3",
      "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
    },
    "currentPhase": "SUBMITTED",
    "startTime": 1472848787.882,
    "id": "codebuild-demo-project:0cfbb6ec-3db9-4e8c-992b-1ab28EXAMPLE",
    "arn": "arn:aws:codebuild:region-ID:account-ID:build/codebuild-demo-
project:0cfbb6ec-3db9-4e8c-992b-1ab28EXAMPLE"
  }
}
```

- `build` は、このビルドに関する情報を表します。
  - `buildComplete` は、ビルドの完了 (`true`) または未完了 (`false`) を表します。
  - `initiator` は、ビルドを開始したエンティティを表します。
  - `artifacts` は、場所を含む、ビルド出力に関する情報を表します。
  - `projectName` は、ビルドプロジェクトの名前を表します。
  - `buildStatus` は、`start-build` コマンドが実行されたときの現在のビルドのステータスを表します。
  - `currentPhase` は、`start-build` コマンドが実行されたときの現在のビルドフェーズを表します。
  - `startTime` は、ビルドプロセスが開始された時刻 (Unix の時間形式) を表します。
  - `id` は、ビルドの ID を表します。
  - `arn`は、ビルドの Amazon リソースネーム (ARN) を表します。

[`id`] の値を書き留めておきます。これは次のステップで必要になります。

ステップ 7: 要約されたビルド情報を表示する

このステップでは、ビルドのステータスに関する要約情報を表示します。

AWS CodeBuild コンソール (p. 14)または AWS CLI (p. 15) を使用して、このステップを完了できます。

要約されたビルド情報を表示するには (コンソール)

1. [codebuild-demo-project:*build-ID*] ページが表示されない場合は、ナビゲーションバーで [Build history] を選択します。次にビルドプロジェクトのリストで、[Project] の [codebuild-demo-project] に対応する [Build run] リンクを選択します。一致するリンクは 1 つだけです。(このチュートリアルを以前に完了している場合は、[Completed] 列の最新の値に対応するリンクを選択してください。)
2. ビルドの詳細ページの [Phase details] に、以下のビルドフェーズのリストが表示され、[Status] 列に [Succeeded] と表示されます。

   - SUBMITTED
   - PROVISIONING
   - DOWNLOAD_SOURCE
   - INSTALL
   - PRE_BUILD
   - BUILD
   - POST_BUILD
   - UPLOAD_ARTIFACTS

- FINALIZING
- COMPLETED

ページのタイトル領域で、緑のボックスに [Succeeded] と表示されます。

代わりに青色のボックスに [In Progress] と表示された場合は、更新ボタンを選択して最新の進行状況を確認します。

3. 各ビルドフェーズの横にある [Duration] 値は、ビルドフェーズの所要時間を示します。[Completed] 値は、ビルドフェーズの完了日時を示します。

   ビルドフェーズを展開すると、フェーズの開始時刻と終了時刻が表示されます。

   [ステップ 8: 詳細なビルド情報を表示する (p. 16)] に進んでください。

要約されたビルド情報を表示するには (AWS CLI)

AWS CLI を使用して `batch-get-builds` コマンドを実行します。

```
aws codebuild batch-get-builds --ids id
```

id を、前のステップの出力に表示された `id` 値に置き換えます。

成功した場合、次のようなデータが出力に表示されます。

```
{
  "buildsNotFound": [],
  "builds": [
    {
      "buildComplete": true,
      "phases": [
        {
          "phaseStatus": "SUCCEEDED",
          "endTime": 1472848788.525,
          "phaseType": "SUBMITTED",
          "durationInSeconds": 0,
          "startTime": 1472848787.882
        },
        ... The full list of build phases has been omitted for brevity ...
        {
          "phaseType": "COMPLETED",
          "startTime": 1472848878.079
        }
      ],
      "logs": {
        "groupName": "/aws/codebuild/codebuild-demo-project",
        "deepLink": "https://console.aws.amazon.com/cloudwatch/home?region=region-
ID#logEvent:group=/aws/codebuild/codebuild-demo-project;stream=38ca1c4a-e9ca-4dbc-bef1-
d52bfEXAMPLE",
        "streamName": "38ca1c4a-e9ca-4dbc-bef1-d52bfEXAMPLE"
      },
      "artifacts": {
        "md5sum": "MD5-hash",
        "location": "arn:aws:s3:::codebuild-region-ID-account-ID-output-bucket/message-
util.zip",
        "sha256sum": "SHA-256-hash"
      },
      "projectName": "codebuild-demo-project",
      "timeoutInMinutes": 60,
      "initiator": "user-name",
```

```
      "buildStatus": "SUCCEEDED",
      "environment": {
        "computeType": "BUILD_GENERAL1_SMALL",
        "image": "aws/codebuild/java:openjdk-8",
        "type": "LINUX_CONTAINER",
        "environmentVariables": []
      },
      "source": {
        "type": "S3",
        "location": "codebuild-region-ID-account-ID-input-bucket/MessageUtil.zip"
      },
      "currentPhase": "COMPLETED",
      "startTime": 1472848787.882,
      "endTime": 1472848878.079,
      "id": "codebuild-demo-project:38ca1c4a-e9ca-4dbc-bef1-d52bfEXAMPLE",
      "arn": "arn:aws:codebuild:region-ID:account-ID:build/codebuild-demo-project:38ca1c4a-
e9ca-4dbc-bef1-d52bfEXAMPLE"
    }
  ]
}
```

- `buildsNotFound` は、情報が利用できないビルドのビルド ID を表します。この例では、空である必要があります。
- `builds` は、利用可能な各ビルドに関する情報を表します。この例では、出力に 1 つのビルドのみに関する情報が表示されます。
  - `phases` は、AWS CodeBuild がビルドプロセス中に実行する一連のビルドフェーズを表します。各ビルドフェーズに関する情報が `startTime`、`endTime`、`durationInSeconds` (フェーズの開始時と終了時、Unix 時間形式で表示、持続時間 (秒))、`phaseType` (`SUBMITTED`、`PROVISIONING`、`DOWNLOAD_SOURCE`、`INSTALL`、`PRE_BUILD`、`BUILD`、`POST_BUILD`、`UPLOAD_` など)、`phaseStatus` (`SUCCEEDED`、`FAILED`、`FAULT`、`TIMED_OUT`、`IN_PROGRESS`、`STOPPED` など) として個別にリストされます。`batch-get-builds` コマンドを初めて実行するときは、多くの (またはまったく) フェーズが存在しない可能性があります。 同じビルド ID を持つ `batch-get-builds` コマンドを続けて実行すると、より多くのビルドフェーズが出力に表示されます。
  - `logs` は、Amazon CloudWatch Logs のビルドのログに関する情報を表します。
  - `md5sum` および `sha256sum` は、ビルドの出力アーティファクトの MD5 と SHA-256 ハッシュを表します。これらは、関連するビルドプロジェクトの `packaging` 値が `ZIP` (このチュートリアルでは設定していない) に設定されている場合にのみ出力に表示されます。これらのハッシュとチェックサムツールを使用して、ファイルの完全性と信頼性の両方を確認することができます。

    Note

    Amazon S3 コンソールを使用して、これらのハッシュを表示することもできます。ビルド出力アーティファクトの横にあるボックスを選択し、[Actions]、[Properties] の順に選択します。[Properties] ペインで [Metadata] を展開し、[x-amz-meta-codebuild-content-md5] と [x-amz-meta-codebuild-content-sha256] の値を確認します。(Amazon S3 コンソールでは、ビルド出力アーティファクトの [ETag] 値を MD5 または SHA-256 ハッシュのいずれかに解釈することはできません。)
    AWS SDK を使用して、これらのハッシュを取得する場合、値の名前は `codebuild-content-md5` および `codebuild-content-sha256` です。
  - `endTime` は、ビルドプロセスが終了した時刻 (Unix の時間形式) を表します。

ステップ 8: 詳細なビルド情報を表示する

このステップでは、CloudWatch Logs のビルドに関する詳細情報を表示します。

AWS CodeBuild コンソール (p. 17)または AWS CLI (p. 17) を使用して、このステップを完了できます。

詳細なビルド情報を表示するには (コンソール)

1. ビルドの詳細ページが前のステップから引き続き表示された状態で、ビルドログの最後の 20 行が [Build logs] に表示されます。CloudWatch Logs でビルドログ全体を表示するには、[View entire log] リンクを選択します。
2. CloudWatch Logs ログストリームでは、ログイベントを参照できます。デフォルトでは、ログイベントの最後のセットだけが表示されます。以前のログイベントを表示するには、リストの先頭にスクロールします。
3. このチュートリアルでは、ほとんどのログイベントに、AWS CodeBuild でビルド環境にダウンロードしてインストールするビルド依存ファイルの詳細情報が含まれますが、これらの情報は不要な場合があります。[Filter events] ボックスを使用すると、表示する情報量を減らすことができます。たとえば、[Filter events] ボックスに `"[INFO]"` と入力して Enter キーを押すと、`[INFO]` の文字を含むイベントのみが表示されます。詳細については、Amazon CloudWatch ユーザーガイドの「フィルタとパターンの構文」を参照してください。

[ステップ 9: ビルド出力アーティファクトを取得する (p. 18)] に進んでください。

詳細なビルド情報を表示するには (AWS CLI)

1. ウェブブラウザを使用して、前の手順の出力に表示された `deepLink` の場所に移動します (例: `https://console.aws.amazon.com/cloudwatch/home?region=region-ID#logEvent:group=/aws/codebuild/codebuild-demo-project;stream=38ca1c4a-e9ca-4dbc-bef1-d52bfEXAMPLE`)。
2. CloudWatch Logs ログストリームでは、ログイベントを参照できます。デフォルトでは、ログイベントの最後のセットだけが表示されます。以前のログイベントを表示するには、リストの先頭にスクロールします。
3. このチュートリアルでは、ほとんどのログイベントに、AWS CodeBuild でビルド環境にダウンロードしてインストールするビルド依存ファイルの詳細情報が含まれますが、これらの情報は不要な場合があります。[Filter events] ボックスを使用すると、表示する情報量を減らすことができます。たとえば、[Filter events] ボックスに `"[INFO]"` と入力して Enter キーを押すと、`[INFO]` の文字を含むイベントのみが表示されます。詳細については、Amazon CloudWatch ユーザーガイドの「フィルタとパターンの構文」を参照してください。

CloudWatch Logs のログストリームの以下の部分は、このチュートリアルに関係します。

```
...
[Container] 2016/04/15 17:49:42 Entering phase PRE_BUILD
[Container] 2016/04/15 17:49:42 Running command echo Entering pre_build phase...
[Container] 2016/04/15 17:49:42 Entering pre_build phase...
[Container] 2016/04/15 17:49:42 Phase complete: PRE_BUILD Success: true
[Container] 2016/04/15 17:49:42 Entering phase BUILD
[Container] 2016/04/15 17:49:42 Running command echo Entering build phase...
[Container] 2016/04/15 17:49:42 Entering build phase...
[Container] 2016/04/15 17:49:42 Running command mvn install
[Container] 2016/04/15 17:49:44 [INFO] Scanning for projects...
[Container] 2016/04/15 17:49:44 [INFO]
[Container] 2016/04/15 17:49:44 [INFO]
 ------------------------------------------------------------------------
[Container] 2016/04/15 17:49:44 [INFO] Building Message Utility Java Sample App 1.0
[Container] 2016/04/15 17:49:44 [INFO]
 ------------------------------------------------------------------------
...
[Container] 2016/04/15 17:49:55 -------------------------------------------------------
[Container] 2016/04/15 17:49:55  T E S T S
[Container] 2016/04/15 17:49:55 -------------------------------------------------------
[Container] 2016/04/15 17:49:55 Running TestMessageUtil
[Container] 2016/04/15 17:49:55 Inside testSalutationMessage()
[Container] 2016/04/15 17:49:55 Hi!Robert
```

```
[Container] 2016/04/15 17:49:55 Inside testPrintMessage()
[Container] 2016/04/15 17:49:55 Robert
[Container] 2016/04/15 17:49:55 Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time
 elapsed: 0.018 sec
[Container] 2016/04/15 17:49:55
[Container] 2016/04/15 17:49:55 Results :
[Container] 2016/04/15 17:49:55
[Container] 2016/04/15 17:49:55 Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
...
[Container] 2016/04/15 17:49:56 [INFO]
 ------------------------------------------------------------------------
[Container] 2016/04/15 17:49:56 [INFO] BUILD SUCCESS
[Container] 2016/04/15 17:49:56 [INFO]
 ------------------------------------------------------------------------
[Container] 2016/04/15 17:49:56 [INFO] Total time: 11.845 s
[Container] 2016/04/15 17:49:56 [INFO] Finished at: 2016-04-15T17:49:56+00:00
[Container] 2016/04/15 17:49:56 [INFO] Final Memory: 18M/216M
[Container] 2016/04/15 17:49:56 [INFO]
 ------------------------------------------------------------------------
[Container] 2016/04/15 17:49:56 Phase complete: BUILD Success: true
[Container] 2016/04/15 17:49:56 Entering phase POST_BUILD
[Container] 2016/04/15 17:49:56 Running command echo Entering post_build phase...
[Container] 2016/04/15 17:49:56 Entering post_build phase...
[Container] 2016/04/15 17:49:56 Phase complete: POST_BUILD Success: true
[Container] 2016/04/15 17:49:57 Preparing to copy artifacts
[Container] 2016/04/15 17:49:57 Assembling file list
[Container] 2016/04/15 17:49:57 Expanding target/messageUtil-1.0.jar
[Container] 2016/04/15 17:49:57 Found target/messageUtil-1.0.jar
[Container] 2016/04/15 17:49:57 Creating zip artifact
```

この例では、AWS CodeBuild はビルド前、ビルド、およびビルド後のビルドフェーズを正常に完了しました。ユニットテストを実行し、`messageUtil-1.0.jar` ファイルは正常に構築されました。

ステップ 9: ビルド出力アーティファクトを取得する

このステップでは、AWS CodeBuild が構築して出力バケットにアップロードした `messageUtil-1.0.jar` ファイルを取得します。

AWS CodeBuild コンソール (p. 18)または Amazon S3 コンソール (p. 18)を使用して、このステップを完了できます。

ビルド出力アーティファクトを取得するには (AWS CodeBuild コンソール)

1. AWS CodeBuild コンソールが開いていて、ビルドの詳細ページが前のステップから引き続き表示されている状態で、[Build details] を展開し、[Build artifacts] リンクを選択します。これにより、ビルド出力アーティファクトのフォルダが Amazon S3 で開きます。(ビルドの詳細ページが表示されていない場合は、ナビゲーションバーで [Build history] を選択し、次に [Build run] リンクを選択します。)
2. `target` という名前のフォルダを開き、`messageUtil-1.0.jar` という名前のビルド出力アーティファクトを見つけます。

   [ステップ 10: クリーンアップ (p. 19)] に進んでください。

ビルド出力アーティファクトを取得するには (Amazon S3 コンソール)

1. https://console.aws.amazon.com/s3/ にある Amazon S3 コンソールを開きます。
2. `codebuild-`*`region-ID`*`-`*`account-ID`*`-output-bucket` という名前のバケットを開きます。

3. `codebuild-demo-project` という名前のフォルダを開きます。
4. `target` という名前のフォルダを開き、`messageUtil-1.0.jar` という名前のビルド出力アーティファクトを見つけます。

ステップ 10: クリーンアップ

AWS アカウントで継続的に料金が発生するのを防ぐために、このチュートリアルで使用されている入力バケットを削除できます。手順については、Amazon Simple Storage Service 開発者ガイドの「バケットを削除する、または空にする」を参照してください。

AWS ルートアカウントまたは管理者 IAM ユーザーではなく IAM ユーザーを使用してこのバケットを削除する場合は、追加のアクセス許可が必要です。(AWS ルートアカウントを使用することは推奨されません。)マーカー間でステートメント (*###BEGIN ADDING STATEMENT HERE###* と *###END ADDING STATEMENTS HERE###*) を既存のアクセスポリシーに追加します。省略記号 (`...`) は、簡潔にするために使用され、ステートメントを追加する場所の特定に役立ちます。既存のアクセスポリシーのステートメントを削除しないでください。また、既存のポリシーにこれらの省略記号を入力しないでください。

```
{
  "Version": "2012-10-17",
  "Id": "...",
  "Statement": [
    ### BEGIN ADDING STATEMENT HERE ###
    {
      "Effect": "Allow",
      "Action": [
        "s3:DeleteBucket",
        "s3:DeleteObject"
      ],
      "Resource": "*"
    }
    ### END ADDING STATEMENT HERE ###
  ]
}
```

次のステップ

このチュートリアルでは、AWS CodeBuild を使用して、一連の Java クラスファイルを JAR ファイルに構築しました。次に、ビルドの結果を表示しました。

今後は ビルドを計画する (p. 94) の手順に従って独自のシナリオで AWS CodeBuild を使用できます。もう少し準備が必要な場合は、用意されているサンプルでいくつか構築を試すことができます。詳細については、「サンプル (p. 20)」を参照してください。

AWS CodeBuild サンプル

これらのユースケースベースのサンプルは、AWS CodeBuild を試用するために使用できます。

名前	説明
Amazon ECR サンプル (p. 22)	Amazon ECR リポジトリの Docker イメージを使用して、Apache Maven を使用して単一の JAR ファイルを生成します。
Docker サンプル (p. 25)	Docker をサポートする AWS CodeBuild に用意されているビルドイメージを使用して Apache Maven の Docker イメージを生成します。Docker イメージを Amazon ECR のリポジトリにプッシュします。このサンプルを適応させて、Docker イメージを Docker Hub にプッシュすることができます。
??? (p. 30)	コード変更がリポジトリにプッシュされるたびにソースコードを自動的に再構築するには、証明書がインストールされてウェブフックが有効化された状態で、GitHub Enterprise をソースリポジトリとする AWS CodeBuild を使用します。
GitHub プルリクエストサンプル (p. 36)	GitHub をソースリポジトリとする AWS CodeBuild を使用し、ウェブフックを有効にして、コード変更がリポジトリにプッシュされるたびにソースコードを再構築します。
AWS CodeBuild サンプルを使用した AWS Config (p. 38)	AWS Config を設定する方法を示します。追跡対象の AWS CodeBuild リソースを一覧表示し、AWS Config で AWS CodeBuild プロジェクトを検索する方法について説明します。
ビルドバッジサンプル (p. 39)	ビルドバッジを使用して AWS CodeBuild を設定する方法を示します。
ビルド通知サンプル (p. 42)	Apache Maven を使用して単一の JAR ファイルを生成します。Amazon SNS トピックのサブスクライバーにビルド通知を送信します。
カスタム Docker イメージのサンプル (p. 59)	カスタム Docker イメージを使用して Docker イメージを生成します。
AWS CodeDeploy サンプル (p. 61)	Apache Maven を使用して単一の JAR ファイルを生成します。AWS CodeDeploy を使用して、Amazon Linux インスタンスに JAR ファイルをデプロイします。AWS CodePipeline を使用して、サンプルをビルドおよびデプロイすることもできます。
AWS Lambda サンプル (p. 64)	AWS CodeBuild を Lambda、AWS CloudFormation、および AWS CodePipeline と共に使用して、AWS サーバーレスアプリケーションモデル (AWS SAM) 標準に準拠したサーバーレ

名前	説明
	スアプリケーションをビルドおよびデプロイします。
Elastic Beanstalk サンプル (p. 65)	Apache Maven を使用して単一の WAR ファイルを生成します。Elastic Beanstalk を使用して Elastic Beanstalk インスタンスに WAR ファイルをデプロイします。

AWS CodeBuild を試用するには、これらのコードベースのサンプルを使用できます。

名前	説明
C++ サンプル (p. 73)	C++ を使用して、単一の .out ファイルを出力します。
Go サンプル (p. 75)	Go を使用して、単一のバイナリファイルを出力します。
Maven サンプル (p. 77)	Apache Maven を使用して単一の JAR ファイルを生成します。
Node.js サンプル (p. 79)	Mocha を使用して、コード内の内部変数に特定の文字列値が含まれているかどうかをテストします。単一の .js ファイルを生成します。
Python サンプル (p. 81)	Python を使用して、コード内の内部変数が特定の文字列値に設定されているかどうかをテストします。単一の .py ファイルを生成します。
Ruby サンプル (p. 83)	RSpec を使用して、コード内の内部変数が特定の文字列値に設定されているかどうかをテストします。単一の .rb ファイルを生成します。
Scala サンプル (p. 86)	sbt を使用して、単一の JAR ファイルを生成します。
Java サンプル (p. 89)	Apache Maven を使用して単一の WAR ファイルを生成します。
Linux における .NET Core のサンプル (p. 91)	.NET Core を使用して C# で書かれたコードから実行可能ファイルをビルドします。

AWS CodeBuild ユースケースベースのサンプル

AWS CodeBuild を試用するために、これらのユースケースベースのサンプルを使用できます。

名前	説明
Amazon ECR サンプル (p. 22)	Amazon ECR リポジトリの Docker イメージを使用して、Apache Maven を使用して単一の JAR ファイルを生成します。
Docker サンプル (p. 25)	Docker をサポートする AWS CodeBuild に用意されているビルドイメージを使用して Apache

名前	説明
	Maven の Docker イメージを生成します。Docker イメージを Amazon ECR のリポジトリにプッシュします。このサンプルを適応させて、Docker イメージを Docker Hub にプッシュすることもできます。
GitHub Enterprise のサンプル (p. 30)	コード変更がリポジトリにプッシュされるたびにソースコードを自動的に再構築するには、証明書がインストールされてウェブフックが有効化された状態で、GitHub Enterprise をソースリポジトリとする AWS CodeBuild を使用します。
GitHub プルリクエストサンプル (p. 36)	GitHub をソースリポジトリとする AWS CodeBuild を使用し、ウェブフックを有効にして、コード変更がリポジトリにプッシュされるたびにソースコードを再構築します。
AWS CodeBuild サンプルを使用した AWS Config (p. 38)	AWS Config を設定する方法を示します。追跡対象の AWS CodeBuild リソースを一覧表示し、AWS Config で AWS CodeBuild プロジェクトを検索する方法について説明します。
ビルドバッジサンプル (p. 39)	ビルドバッジを使用して AWS CodeBuild を設定する方法を示します。
ビルド通知サンプル (p. 42)	Apache Maven を使用して単一の JAR ファイルを生成します。Amazon SNS トピックのサブスクライバーにビルド通知を送信します。
カスタム Docker イメージのサンプル (p. 59)	カスタム Docker イメージを使用して Docker イメージを生成します。
AWS CodeDeploy サンプル (p. 61)	Apache Maven を使用して単一の JAR ファイルを生成します。AWS CodeDeploy を使用して、Amazon Linux インスタンスに JAR ファイルをデプロイします。AWS CodePipeline を使用して、サンプルをビルドおよびデプロイすることもできます。
AWS Lambda サンプル (p. 64)	AWS CodeBuild を Lambda、AWS CloudFormation、および AWS CodePipeline と共に使用して、AWS サーバーレスアプリケーションモデル (AWS SAM) 標準に準拠したサーバーレスアプリケーションをビルドおよびデプロイします。
Elastic Beanstalk サンプル (p. 65)	Apache Maven を使用して単一の WAR ファイルを生成します。Elastic Beanstalk を使用して Elastic Beanstalk インスタンスに WAR ファイルをデプロイします。

AWS CodeBuild の Amazon ECR サンプル

このサンプルでは、Amazon Elastic Container Registry (Amazon ECR) イメージレポジトリの Docker イメージを使用して、AWS CodeBuild の「Maven サンプル (p. 77)」をビルドします。

Important

このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild の料金、および Amazon S3、AWS KMS、CloudWatch Logs、Amazon ECR に関連した AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」、「Amazon Elastic Container Registry 料金表」を参照してください。

サンプルの実行

このサンプルを実行するには。

1. Amazon ECR でイメージレポジトリに Docker イメージを作成してプッシュするには、「Docker サンプル (p. 25)」の「サンプルの実行」セクションにある手順を完了します。
2. ビルドするソースコードを作成しアップロードするには、「Maven サンプル (p. 77)」の「サンプルの実行」セクションの 1 から 4 の手順を完了します。
3. AWS CodeBuild がリポジトリの Docker イメージをビルド環境にプルできるように、Amazon ECR のイメージリポジトリにアクセス許可を割り当てます。

   1. AWS ルートアカウントまたは管理者 IAM ユーザーの代わりに IAM ユーザーを使用して Amazon ECR を操作する場合は、ユーザー (またはユーザーが関連付けられている IAM グループ) にステートメントを (*### BEGIN ADDING STATEMENT HERE ###* と *### END ADDING STATEMENT HERE ###* の間に) 追加します。(AWS ルートアカウントを使用することは推奨されません。)このステートメントは、Amazon ECR リポジトリの管理権限にアクセスするためのものです。省略記号 (. . .) は、簡潔にするために使用され、ステートメントを追加する場所の特定に役立ちます。ステートメントを削除しないでください、また、これらの省略記号をポリシーに入力しないでください。詳細については、IAM ユーザーガイドの「AWS マネジメントコンソール でのインラインポリシーの使用」を参照してください。

```
{
  "Statement": [
    ### BEGIN ADDING STATEMENT HERE ###
    {
      "Action": [
        "ecr:GetRepositoryPolicy",
        "ecr:SetRepositoryPolicy"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    ### END ADDING STATEMENT HERE ###
    ...
  ],
  "Version": "2012-10-17"
}
```

      Note

      このポリシーを変更する IAM エンティティは、ポリシーを変更するために IAM のアクセス権限を持っている必要があります。

   2. https://console.aws.amazon.com/ecs/ にある Amazon ECS コンソールを開きます。
   3. [Repositories] を選択します。
   4. リポジトリ名のリストで、作成または選択したリポジトリの名前を選択します。
   5. [アクセス許可] タブ、[Add] を順に選択して、ステートメントを作成します。
   6. [Sid] で、識別子を入力します (**`CodeBuildAccess`**、など)。
   7. [Effect] では、AWS CodeBuild へのアクセスを許可するため、[Allow] を選択したままにしておきます。

8. [プリンシパル] には **codebuild.amazonaws.com** と入力します。. AWS CodeBuild のみへアクセスを許可するため、[Everybody] はオフのままにしておきます。
9. [すべての IAM エンティティ] リストをスキップします。
10. [Action] で、[プル専用アクション] を選択します。

    すべてのプル専用アクション ([ecr:GetDownloadUrlForLayer]、[ecr:BatchGetImage]、[ecr:BatchCheckLayerAvailability]) が選択されます。
11. [すべて保存] を選択します。

    このポリシーは [ポリシードキュメント] に表示されます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CodeBuildAccess",
      "Effect": "Allow",
      "Principal": {
        "Service": "codebuild.amazonaws.com"
      },
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability"
      ]
    }
  ]
}
```

4. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルドプロジェクトを作成し、ビルドを実行し、ビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力はこれに似ています。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "amazon-ecr-sample-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/MavenIn5MinutesSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "MavenIn5MinutesOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "account-ID.dkr.ecr.us-east-2.amazonaws.com/your-Amazon-ECR-repo-
name:latest",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

5. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
6. *MavenIn5MinutesOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、*MavenIn5MinutesOutputArtifact*.zip ファイルの内容を抽出します。抽出された内容で [target] フォルダを開き、my-app-1.0-SNAPSHOT.jar ファイルを取得します。

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の Docker サンプル

このサンプルでは、Docker イメージをビルド出力として生成し、Docker イメージを Amazon Elastic Container Registry(Amazon ECR) イメージリポジトリにプッシュします。このサンプルを適応させて、Docker イメージを Docker Hub にプッシュすることができます。詳細については、「イメージを Docker Hub にプッシュするためのサンプルの適応 (p. 28)」を参照してください。

代わりに、カスタム Docker ビルドイメージ (Docker Hub の `docker:dind`) を使用して Docker イメージをビルドする方法については、「カスタム Docker イメージのサンプル (p. 59)」を参照してください。

このサンプルは、`golang:1.9` を参照してテストされています。

このサンプルでは、新しいマルチステージの Docker ビルド機能を使用しています。この機能により、Docker イメージがビルド出力として生成されます。次に、Docker イメージが Amazon ECR イメージリポジトリにプッシュされます。マルチステージの Docker イメージビルドは、最終的な Docker イメージのサイズを縮小するのに役立ちます。詳細については、「Docker でのマルチステージビルドの使用」を参照してください。

> Important
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild の料金、および Amazon S3、AWS KMS、CloudWatch Logs、Amazon ECR に関連した AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」、「Amazon Elastic Container Registry 料金表」を参照してください。

トピック

サンプルの実行

このサンプルを実行するには

1. 使用する Amazon ECR にイメージリポジトリがすでにある場合は、ステップ 3 に進みます。それ以外の場合は、AWS ルートアカウントまたは管理者 IAM ユーザーの代わりに IAM ユーザーを使用して Amazon ECR を操作する場合は、ユーザー (またはユーザーが関連付けられている IAM グループ) にこのステートメントを (*`### BEGIN ADDING STATEMENT HERE ###`* と *`### END ADDING STATEMENT HERE ###`* の間に) 追加します。(AWS ルートアカウントを使用することは推奨されません。)このステートメントにより、Docker イメージを保存するための Amazon ECR リポジトリを作成できます。省略記号 (`...`) は、簡潔にするために使用され、ステートメントを追加する場所の特定に

役立ちます。ステートメントを削除しないでください、また、これらの省略記号をポリシーに入力しないでください。詳細については、IAM ユーザーガイドの「AWS マネジメントコンソール でのインラインポリシーの使用」を参照してください。

```
{
  "Statement": [
    ### BEGIN ADDING STATEMENT HERE ###
    {
      "Action": [
        "ecr:CreateRepository"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    ### END ADDING STATEMENT HERE ###
    ...
  ],
  "Version": "2012-10-17"
}
```

Note

このポリシーを変更する IAM エンティティは、ポリシーを変更するために IAM のアクセス許可を持っている必要があります。

2. Amazon ECR にイメージリポジトリを作成します。必ず、ビルド環境を作成してビルドを実行するのと同じ AWS リージョンにリポジトリを作成してください。詳細については、Amazon ECR ユーザーガイドの「リポジトリの作成」を参照してください。このリポジトリの名前は、`IMAGE_REPO_NAME` 環境変数で表され、この手順の後の方で指定するリポジトリ名と一致する必要があります。
3. このステートメント (*###BEGIN ADDING STATEMENT HERE###* と *###END ADDING STATEMENT HERE###* の間) を AWS CodeBuild サービスロールにアタッチしたポリシーに追加します。このステートメントにより、AWS CodeBuild は Amazon ECR リポジトリに Docker イメージをアップロードできます。省略記号 (`...`) は、簡潔にするために使用され、ステートメントを追加する場所の特定に役立ちます。ステートメントを削除しないでください、また、これらの省略記号をポリシーに入力しないでください。

```
{
  "Statement": [
    ### BEGIN ADDING STATEMENT HERE ###
    {
      "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:CompleteLayerUpload",
        "ecr:GetAuthorizationToken",
        "ecr:InitiateLayerUpload",
        "ecr:PutImage",
        "ecr:UploadLayerPart"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    ### END ADDING STATEMENT HERE ###
    ...
  ],
  "Version": "2012-10-17"
}
```

Note

このポリシーを変更する IAM エンティティは、ポリシーを変更するために IAM のアクセス許可を持っている必要があります。

4. このトピックの「ディレクトリ構造」セクションと「ファイル」セクションの説明に従ってファイルを作成し、Amazon S3 入力バケットにアップロードするか、AWS CodeCommit、GitHub、または Bitbucket の各リポジトリにアップロードします。

   Important

   (root directory name) をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
   Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。

5. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従ってビルドプロジェクトを作成し、ビルドを実行して、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-docker-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/DockerSample.zip"
  },
  "artifacts": {
    "type": "NO_ARTIFACTS"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/docker:17.09.0",
    "computeType": "BUILD_GENERAL1_SMALL",
    "environmentVariables": [
      {
        "name": "AWS_DEFAULT_REGION",
        "value": "region-ID"
      },
      {
        "name": "AWS_ACCOUNT_ID",
        "value": "account-ID"
      },
      {
        "name": "IMAGE_REPO_NAME",
        "value": "Amazon-ECR-repo-name"
      },
      {
        "name": "IMAGE_TAG",
        "value": "latest"
      }
    ]
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

6. AWS CodeBuild が Docker イメージをリポジトリに正常にプッシュしたことを確認します。

   1. https://console.aws.amazon.com/ecs/ にある Amazon ECS コンソールを開きます。
   2. [Repositories] を選択します。
   3. リポジトリ名を選択します。イメージは、[Images] タブに表示されます。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
    |-- buildspec.yml
    `-- Dockerfile
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

buildspec.yml (*(root directory name)* 内)

> **Note**
>
> バージョン 17.06 より前の Docker を使用している場合は、`--no-include-email` オプションを削除します。

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - $(aws ecr get-login --no-include-email --region $AWS_DEFAULT_REGION)
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/
$IMAGE_REPO_NAME:$IMAGE_TAG
```

Dockerfile (*(root directory name)* 内)

```
FROM golang:1.9 as builder
RUN go get -d -v golang.org/x/net/html
RUN go get -d -v github.com/alexellis/href-counter/
WORKDIR /go/src/github.com/alexellis/href-counter/.
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
CMD ["./app"]
```

イメージを Docker Hub にプッシュするためのサンプルの適応

Docker イメージを Amazon ECR の代わりに Docker Hub にプッシュするには、このサンプルのコードを変更します。

1. `buildspec.yml` ファイルで、以下の Amazon ECR 固有のコード行を置き換えます。

Note

バージョン 17.06 より前の Docker を使用している場合は、`--no-include-email` オプションを削除します。

```
...
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - $(aws ecr get-login --no-include-email --region $AWS_DEFAULT_REGION)
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $AWS_ACCOUNT_ID.dkr.ecr.
$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/
$IMAGE_REPO_NAME:$IMAGE_TAG
...
```

代わりに、以下の Docker Hub 固有のコードの行に置き換えます。

```
...
  pre_build:
    commands:
      - echo Logging in to Docker Hub...
      # Type the command to log in to your Docker Hub account here.
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $IMAGE_REPO_NAME:$IMAGE_TAG .
      - docker tag $IMAGE_REPO_NAME:$IMAGE_TAG $IMAGE_REPO_NAME:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      - docker push $IMAGE_REPO_NAME:$IMAGE_TAG
...
```

2. 変更済みのコードを Amazon S3 入力バケットにアップロードするか、AWS CodeCommit、GitHub、Bitbucket のいずれかのリポジトリにアップロードします。

   Important

   (root directory name) をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
   Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。

3. `create-project` コマンドに対する JSON 形式の入力で、以下のコード行が置き換えの対象です。

```
...
    "environmentVariables": [
      {
```

```
        "name": "AWS_DEFAULT_REGION",
        "value": "region-ID"
      },
      {
        "name": "AWS_ACCOUNT_ID",
        "value": "account-ID"
      },
      {
        "name": "IMAGE_REPO_NAME",
        "value": "Amazon-ECR-repo-name"
      },
      {
        "name": "IMAGE_TAG",
        "value": "latest"
      }
    ]
...
```

以下のコード行に置き換えます。

```
...
    "environmentVariables": [
      {
        "name": "IMAGE_REPO_NAME",
        "value": "your-Docker-Hub-repo-name"
      },
      {
        "name": "IMAGE_TAG",
        "value": "latest"
      }
    ]
...
```

4. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルド環境を作成して、ビルドを実行し、関連するビルド情報を表示します。
5. AWS CodeBuild が Docker イメージをリポジトリに正常にプッシュしたことを確認します。Docker Hub にサインインし、リポジトリに進み、[Tags] タブを選択します。`latest` タグには、ごく最近の [Last Updated] の値が含まれています。

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

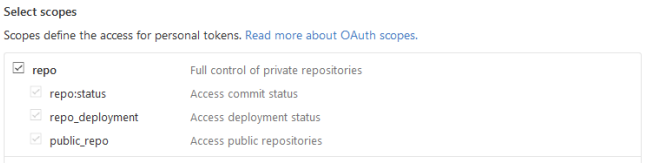
AWS CodeBuild 向けの GitHub Enterprise のサンプル

AWS CodeBuild は GitHub Enterprise をソースリポジトリとしてサポートするようになりました。このサンプルは、GitHub Enterprise リポジトリが証明書をインストールしている場合に、AWS CodeBuild をセットアップする方法を説明しています。また、プライベート GitHub Enterprise リポジトリにコード変更がプッシュされるたびに AWS CodeBuild がソースコードを再構築できるように、ウェブフックを有効化する方法も説明します。

前提条件

1. AWS CodeBuild プロジェクトに入る個人用のアクセストークンを生成します。新規の GitHub Enterprise ユーザーを作成し、このユーザーに個人用のアクセストークンを生成することが推奨されます。AWS CodeBuild プロジェクトを作成する際に使用できるように、クリップボードにこれをコピーします。詳細については、GitHub ヘルプウェブサイトの「GitHub Enterprise で個人用アクセストークンを作成する」を参照してください。

   個人用アクセストークンを作成するときには、定義にリポジトリスコープを含めてください。

2. GitHub Enterprise から証明書をダウンロードします。AWS CodeBuild は、証明書を使用して信頼された SSL 接続をリポジトリに作成します。

   Linux/macOS クライアント:

   のターミナルウィンドウから、以下のコマンドを実行します。

   ```
   echo -n | openssl s_client -connect HOST:PORTNUMBER \
       | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > /folder/filename.pem
   ```

   コマンドのプレースホルダを次の値で置き換えます。

   HOST. GitHub Enterprise リポジトリの IP アドレス。

   PORTNUMBER. 接続するときに使用するポート番号 (たとえば、443)。

   ####. 証明書をダウンロードしたフォルダ。

   filename. 証明書ファイルのファイル名。

   > **Important**
   >
   > 証明書を .pem ファイルとして保存します。

   Windows クライアント:

   ブラウザを使用して GitHub Enterprise から証明書をダウンロードします。サイトの証明書の詳細を表示するには、南京錠アイコンを選択します。証明書をエクスポートする方法についての詳細は、ブラウザのドキュメントを参照してください。

   > **Important**
   >
   > 証明書を .pem ファイルとして保存します。

3. 証明書ファイルを Amazon S3 バケットにアップロードします。Amazon S3 バケットを作成する方法については、「Amazon S3 バケットを作成する方法」を参照してください。Amazon S3 バケットにオブジェクトをアップロードする方法については、「バケットにファイルとフォルダをアップロードする方法」を参照してください。

   > **Note**
   >
   > このバケットは、ビルドと同じ AWS リージョン内にある必要があります。たとえば、ビルドの実行先として米国東部 (オハイオ) リージョンを AWS CodeBuild に指示する場合、バケットは米国東部 (オハイオ) リージョン内にあることが必要です。

GitHub Enterprise をソースリポジトリとして使用してビルドプロジェクトを作成し、ウェブフックを有効にする (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ウェルカムページが表示された場合は、[Get started] を選択します。ウェルカムページが表示されない場合は、ナビゲーションペインで [Build projects] を選択し、次に [Create project] を選択します。
3. [Configure your project] ページの [Project name] に、このビルドプロジェクトの名前を入力します。ビルドプロジェクトの名前は、各 AWS アカウントで一意である必要があります。
4. [Source: What to build] で、[Source provider] として [GitHub Enterprise] を選択します。

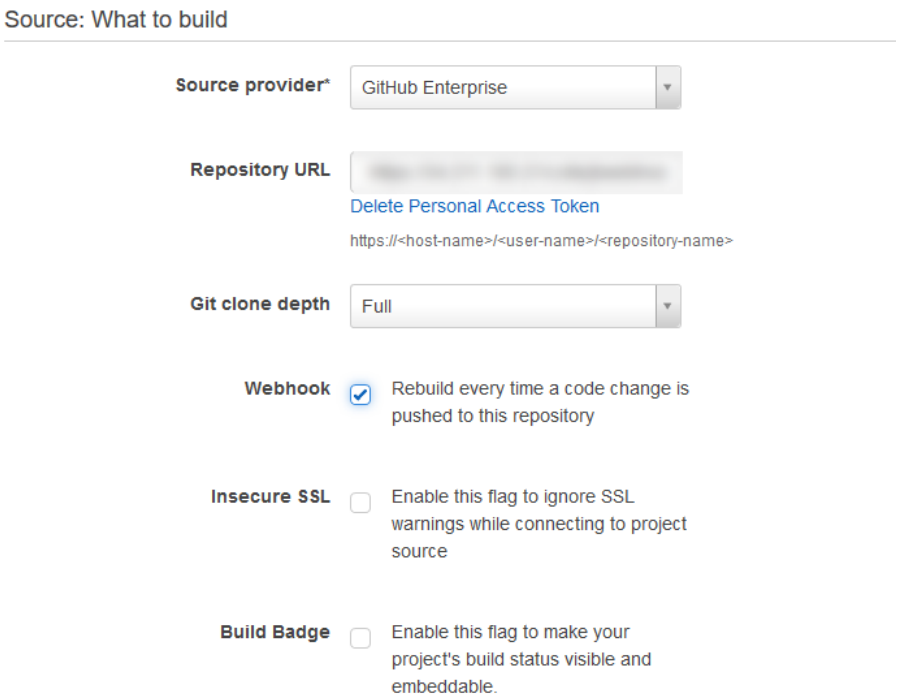
   - [個人用アクセストークン] には、クリップボードにコピーしたトークンを貼り付け、[トークンの保存] を選択します。[リポジトリ URL] には、GitHub Enterprise リポジトリの URL を入力します。

     Note

     個人用アクセストークンは、一回のみ入力して保存することが必要となります。次からのすべての AWS CodeBuild プロジェクトには、このトークンが使用されます。
   - [Webhook] を選択して、コード変更がこのリポジトリにプッシュされるたびに再構築を行います。
   - GitHub Enterprise プロジェクトリポジトリに接続するときに SSL 警告を無視するには、[Insecure SSL] を選択します。

     Note

     [Insecure SSL] はテストのみに使用することが推奨されます。本番環境では使用しないでください。



5. [Environment: How to build] で以下の操作を行います。

   [Environment image] で、次のいずれかの操作を行います。

- AWS CodeBuild で管理されている Docker イメージを使用するには、[Use an image managed by AWS CodeBuild] を選択し、次に [Operating system]、[Runtime]、および [Version] で選択を行います。
- 別の Docker イメージを使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Other] または [Amazon ECR] を選択します。[Other] を選択した場合は、[Custom image ID] に Docker Hub の Docker イメージの名前とタグを *repository-name*/*image-name*:*image-tag* の形式で入力します。[Amazon ECR] を選択した場合は、[Amazon ECR repository] および [Amazon ECR image] を使用して AWS アカウントの Docker イメージを選択します。

[Build specification] で、次のいずれかの操作を行います。

- ソースコードのルートディレクトリの buildspec.yml ファイルを使用します。
- ビルドコマンドを挿入してビルド仕様をオーバーライドします。

詳細については、「ビルドスペックリファレンス (p. 95)」を参照してください。

[証明書] では、[S3 から証明書をインストールする] を選択します。[証明書のバケット] では、SSL 証明書が保存されている S3 バケットを選択します。[証明書のオブジェクトキー] に S3 オブジェクトキーの名前を入力します。

6. [Artifacts: Where to put the artifacts from this build project] で、[Artifacts type] として次のいずれかの操作を行います。

   - ビルド出力アーティファクトを作成しない場合は、[No artifacts] を選択します。
   - ビルド出力を Amazon S3 バケットに保存する場合は、[Amazon S3] を選択して次のいずれかの操作を行います。
     - ビルド出力 ZIP ファイルまたはフォルダにプロジェクト名を使用する場合は、[Artifacts name] を空白のままにします。それ以外の場合は、[Artifacts name] ボックスに名前を入力します。デフォルトでは、アーティファクト名はプロジェクト名です。別の名前を指定する場合は、アーティファクト名ボックスに名前を入力します。ZIP ファイルを出力する場合は、zip 拡張子を含めます。
     - [Bucket name] で、出力バケットの名前を選択します。
     - この手順の前の方で [Insert build commands] を選択した場合は、[Output files] に、ビルド出力 ZIP ファイルまたはフォルダに格納する、ビルドからのファイルの場所を入力します。複数の場所の場合は、各場所をコンマで区切ります (例: `appspec.yml, target/my-app.jar`)。詳細については、「ビルドスペックの構文 (p. 96)」の `files` の説明を参照してください。
7. [Cache] で、次のいずれかの操作を行います。

   - キャッシュを使用しない場合は、[No cache] を選択します。
   - キャッシュを使用するには、[Amazon S3] を選択し、次の操作を行います。
     - [バケット] では、キャッシュが保存される Amazon S3 バケットの名前を選択します。
     - (オプション) [Path prefix] に、Amazon S3 パスプレフィックスを入力します。[Path prefix] 値は、バケット内の同じディレクトリにキャッシュを保存できるディレクトリ名に似ています。

       Important

       [Path prefix] の末尾に「/」を追加しないでください。

   キャッシュを使用すると、再利用可能なビルド環境がキャッシュに保存され、ビルド全体で使用されるため、かなりのビルド時間が節約されます。
8. [Service role] で、次のいずれかの操作を行います。

- AWS CodeBuild サービスロールがない場合は、[Create a service role in your account] を選択します。[Role name] で、デフォルト名を受け入れるか、独自の名前を入力します。
- AWS CodeBuild サービスロールがある場合は、[Choose an service existing role from your account] を選択します。[Role name] で、サービスロールを選択します。

  Note

  コンソールでは、ビルドプロジェクトの作成時や更新時に AWS CodeBuild サービスロールも作成できます。デフォルトでは、ロールはそのビルドプロジェクトでのみ使用できます。コンソールでは、このサービスロールを別のビルドプロジェクトと関連付けると、この別のビルドプロジェクトで使用できるようにロールが更新されます。サービスロールは最大 10 個のビルドプロジェクトで使用できます。

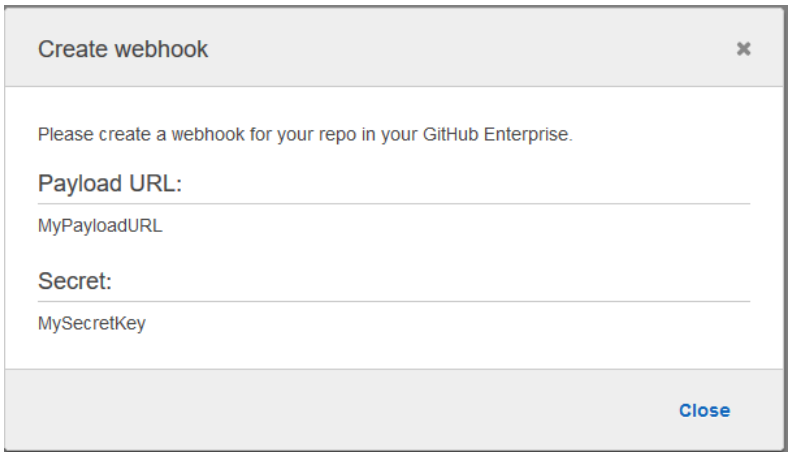
9. [VPC] で、次のいずれかの操作を行います。

   - プロジェクトに VPC を使用していない場合は、[No VPC] を選択します。
   - AWS CodeBuild を使用して VPC で作業する場合は、次のようにします。
     - [VPC] で、AWS CodeBuild が使用する VPC ID を選択します。
     - [Subnets] で、AWS CodeBuild が使用するリソースを含むサブネットを選択します。
     - [Security Groups] で、AWS CodeBuild が VPC 内のリソースへのアクセスを許可するために使用するセキュリティグループを選択します。

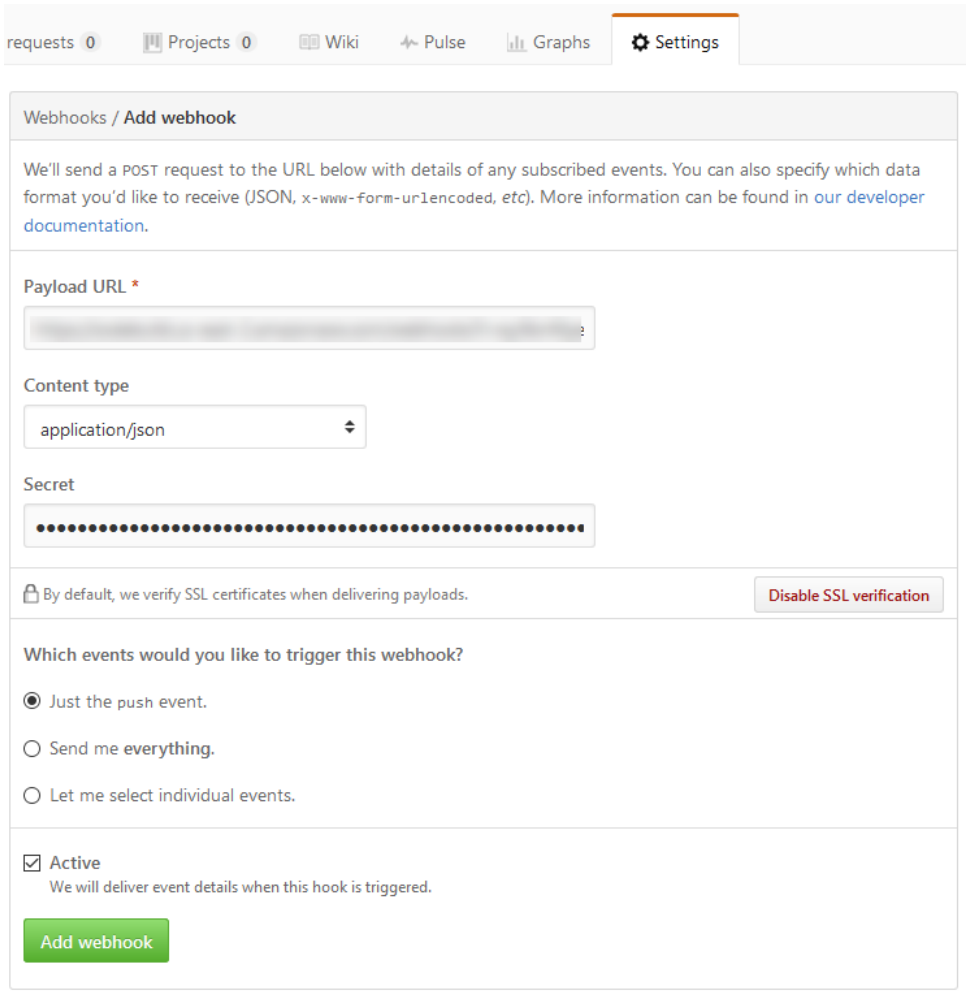
   詳細については、「Amazon Virtual Private Cloud で AWS CodeBuild を使用する (p. 115)」を参照してください。
10. [詳細設定の表示] を展開し、必要に応じて設定します。
11. [Continue] を選択します。[Review] ページで、[Save and build] を選択します。後でビルドを実行する場合は、[Save] を選択します。
12. [ソース: 何をビルドするか] でウェブフックを有効にすると、[ペイロードの URL] の値および [シークレット] を示した [ウェブフックの作成] ダイアログボックスが表示されます。

    Important

    [ウェブフックの作成] ダイアログボックスが表示されるのは 1 回だけです。ペイロード URL とシークレットキーをコピーします。GitHub Enterprise にウェブフックを追加するときにこれが必要となります。
    ペイロード URL およびシークレットキーの作成が必要な場合には、まず GitHub Enterprise リポジトリからウェブフックを削除してください。AWS CodeBuild プロジェクトで [ウェブフック] チェックボックスをオフにし、[保存] を選択します。そして、チェックボックスで選択した [ウェブフック] で AWS CodeBuild プロジェクトを作成あるいは更新できます。[ウェブフックの作成] ダイアログボックスが再度表示されます。



13. GitHub Enterprise で AWS CodeBuild プロジェクトが保存されているリポジトリを選択し、[設定]、[Hooks & services]、[Add webhook] の順に選択します。ペイロード URL とシークレットキーを入力し、その他のフィールドにはデフォルト値を選択して、[Add webhook] を選択します。



14. AWS CodeBuild プロジェクトに戻ります。[ウェブフックの作成] ダイアログボックスを閉じ、[ビルドの開始] を選択します。

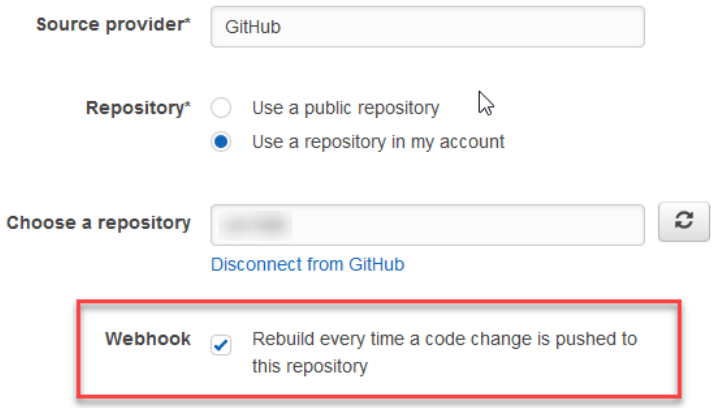
AWS CodeBuild の GitHub プルリクエストサンプル

AWS CodeBuild は、ソースリポジトリが GitHub の場合、ウェブフックをサポートするようになりました。つまり、ソースコードがプライベート GitHub リポジトリに保存されている AWS CodeBuild ビルドプロジェクトでは、ウェブフックにより、コード変更がプライベートリポジトリにプッシュされるたびに AWS CodeBuild がソースコードを自動的に再構築できるようになります。

GitHub をソースリポジトリとして使用してビルドプロジェクトを作成し、ウェブフックを有効にする (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ウェルカムページが表示された場合は、[Get started] を選択します。ウェルカムページが表示されない場合は、ナビゲーションペインで [Build projects] を選択し、次に [Create project] を選択します。
3. [Configure your project] ページの [Project name] に、このビルドプロジェクトの名前を入力します。ビルドプロジェクトの名前は、各 AWS アカウントで一意である必要があります。
4. [Source: What to build] で、[Source provider] として [GitHub] を選択します。手順に従って GitHub に接続 (または再接続) し、[Authorize] を選択します。

   [Webhook] では、[コードの変更がこのレポジトリにプッシュされるたびに再構築する] チェックボックスをオンにします。このチェックボックスを選択できるのは、[Repository] で [自分のアカウントでリポジトリを使用する] を選択した場合のみです。



5. [Environment: How to build] で以下の操作を行います。

   [Environment image] で、次のいずれかの操作を行います。

   - AWS CodeBuild で管理されている Docker イメージを使用するには、[Use an image managed by AWS CodeBuild] を選択し、次に [Operating system]、[Runtime]、および [Version] で選択を行います。
   - 別の Docker イメージを使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Other] または [Amazon ECR] を選択します。[Other] を選択した場合は、[Custom image ID] に Docker Hub の Docker イメージの名前とタグを *repository-name*/*image-name*:*image-tag* の形式で入力します。[Amazon ECR] を選択した場合は、[Amazon ECR repository] および [Amazon ECR image] を使用して AWS アカウントの Docker イメージを選択します。

   [Build specification] で、次のいずれかの操作を行います。

   - ソースコードのルートディレクトリの buildspec.yml ファイルを使用します。
   - ビルドコマンドを挿入してビルド仕様をオーバーライドします。

詳細については、「ビルドスペックリファレンス (p. 95)」を参照してください。

6. [Artifacts: Where to put the artifacts from this build project] で、[Artifacts type] として次のいずれかの操作を行います。

   - ビルド出力アーティファクトを作成しない場合は、[No artifacts] を選択します。
   - ビルド出力を Amazon S3 バケットに保存する場合は、[Amazon S3] を選択して次のいずれかの操作を行います。
     - ビルド出力 ZIP ファイルまたはフォルダにプロジェクト名を使用する場合は、[Artifacts name] を空白のままにします。それ以外の場合は、[Artifacts name] ボックスに名前を入力します。(デフォルトでは、アーティファクト名はプロジェクト名です。別の名前を指定する場合は、アーティファクト名ボックスに名前を入力します。ZIP ファイルを出力する場合は、zip 拡張子を含めます。
     - [Bucket name] で、出力バケットの名前を選択します。
     - この手順の前の方で [Insert build commands] を選択した場合は、[Output files] に、ビルド出力 ZIP ファイルまたはフォルダに格納する、ビルドからのファイルの場所を入力します。複数の場所の場合は、各場所をコンマで区切ります (例: `appspec.yml, target/my-app.jar`)。詳細については、「ビルドスペックの構文 (p. 96)」の `files` の説明を参照してください。

7. [Service role] で、次のいずれかの操作を行います。

   - AWS CodeBuild サービスロールがない場合は、[Create a service role in your account] を選択します。[Role name] で、デフォルト名を受け入れるか、独自の名前を入力します。
   - AWS CodeBuild サービスロールがある場合は、[Choose an service existing role from your account] を選択します。[Role name] で、サービスロールを選択します。

     Note

     コンソールでは、ビルドプロジェクトの作成時や更新時に AWS CodeBuild サービスロールも作成できます。デフォルトでは、ロールはそのビルドプロジェクトでのみ使用できます。コンソールでは、このサービスロールを別のビルドプロジェクトと関連付けると、この別のビルドプロジェクトで使用できるようにロールが更新されます。サービスロールは最大 10 個のビルドプロジェクトで使用できます。

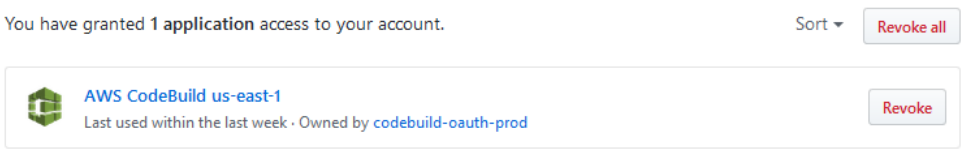
8. [詳細設定の表示] を展開し、必要に応じて設定します。
9. [Continue] を選択します。[Review] ページで、[Save and build] を選択します。後でビルドを実行する場合は、[Save] を選択します。

検証チェック

1. [AWS CodeBuild プロジェクト] ページで以下の操作を実行します。

   - [プロジェクト詳細] を選択してから [Webhook] URL リンクを選択します。GitHub リポジトリの [Settings] ページの [Webhooks] で、[Pull Request] と [Push] の両方が選択されていることを確認します。

2. GitHub の [アカウント]、[設定]、[Authorized OAuth Apps] に、承認された AWS CodeBuild リージョンが表示されます。



AWS CodeBuild サンプルを使用した AWS Config

AWS Config は AWS リソースのインベントリと、それらのリソースへの設定変更の履歴を提供します。AWS Config で AWS CodeBuild が AWS リソースとしてサポートされるようになりました。これにより、サービスは AWS CodeBuild のプロジェクトを追跡できます。AWS Config の詳細については、「AWS Config とは」を AWS Config 開発者ガイドで参照してください。

AWS Config コンソールの [リソースインベントリ] ページには、AWS CodeBuild のリソースに関する次の情報が表示されます。

- AWS CodeBuild 設定変更のタイムライン。
- 各 AWS CodeBuild プロジェクトの設定詳細。
- 他の AWS リソースとの関係。
- AWS CodeBuild プロジェクトの変更のリスト。

このトピックの手順では、AWS Config をセットアップし、AWS CodeBuild プロジェクトを検索および表示する方法を説明します。

トピック

前提条件

AWS CodeBuild プロジェクトを作成します。詳細については、「ビルドプロジェクトを作成する (p. 146)」を参照してください。

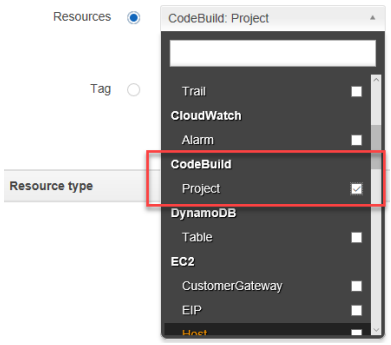
AWS Config を設定する

- AWS Config をセットアップする (コンソール)
- AWS Config をセットアップする (AWS CLI)

> **Note**
>
> ユーザーが AWS Config コンソールで AWS CodeBuild プロジェクトを見ることができるようになるまでに、最大で 10 分かかることがあります。

AWS CodeBuild プロジェクトの検索

1. AWS Management Console にサインインして、https://console.aws.amazon.com/config で AWS Config コンソールを開きます。
2. [リソースのインベントリ] ページで、[リソース] を選択します。下方にスクロールして [CodeBuild プロジェクト] チェックボックスをオンにします。



3. [検索] を選択します。
4. AWS CodeBuild プロジェクトのリストが追加されたら、[設定のタイムライン] 列で AWS CodeBuild プロジェクト名のリンクを選択します。

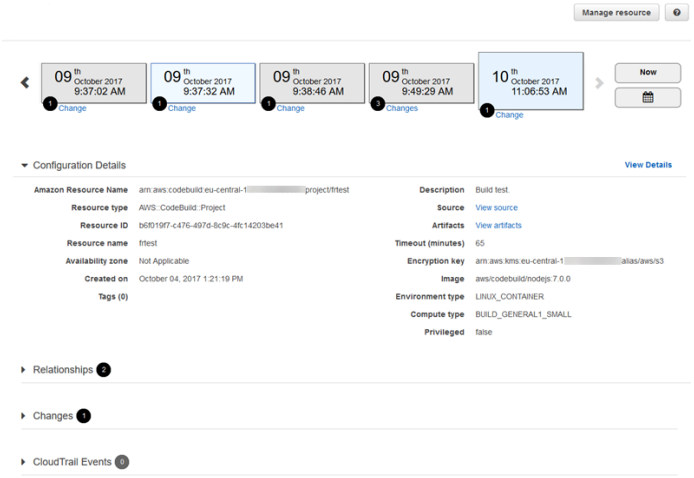
AWS Config コンソールでの AWS CodeBuild 設定詳細の表示

[リソースのインベントリ] ページでリソースを検索する場合、AWS Config タイムラインを選択して、AWS CodeBuild プロジェクトに関する詳細を表示できます。リソースの詳細ページは、リソースの設定、関係、および変更回数の情報を提供します。

ページの上部にあるブロックは、まとめてタイムラインと呼ばれます。タイムラインは、記録を取った日付と時刻を示します。

詳細については、AWS Config Developer Guide の Viewing Configuration Details in the AWS Config Console を参照してください。

AWS Config での AWS CodeBuild プロジェクトの例



AWS CodeBuild のビルドバッジサンプル

AWS CodeBuild でビルドバッジが使用可能になりました。ビルドバッジは、埋め込み可能なイメージ (バッジ) として動的に生成され、プロジェクトの最新ビルドのステータスを示します。このイメージにアクセスするには、AWS CodeBuild プロジェクトに対して生成されるパブリックアクセス可能な URL を使用できます。このため、誰でも AWS CodeBuild プロジェクトのステータスを確認できます。ビルドバッジにはセキュリティ情報が含まれないため、認証は不要です。

ビルドバッジが有効化されたビルドプロジェクトの作成 (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ウェルカムページが表示された場合は、[Get started] を選択します。ウェルカムページが表示されない場合は、ナビゲーションペインで [Build projects] を選択し、次に [Create project] を選択します。
3. [Configure your project] ページの [Project name] に、このビルドプロジェクトの名前を入力します。ビルドプロジェクトの名前は、各 AWS アカウントで一意である必要があります。
4. [Source: What to build] で、[Source provider] のソースコードプロバイダタイプを選択し、次のいずれかの操作を行います。

   - [Amazon S3] を選択した場合は、[Bucket] でソースコードが含まれている入力バケットの名前を選択します。[S3 object key] に、ソースコードが含まれている ZIP ファイルの名前を入力します。
   - [AWS CodeCommit] を選択した場合は、[Repository] でリポジトリの名前を選択します。[Build Badge] チェックボックスをオンにして、プロジェクトのビルドステータスを表示可能および埋め込み可能にします。
   - [GitHub] を選択した場合は、手順に従って GitHub に接続 (または再接続) します。GitHub の [Authorize application] ページの [Organization access] で、AWS CodeBuild からのアクセスを許可するリポジトリごとに横にある [Request access] を選択します。[Authorize application] を選択した後で AWS CodeBuild コンソールに戻り、[Repository] でソースコードが含まれているリポジトリの名前を選択します。[Build Badge] チェックボックスをオンにして、プロジェクトのビルドステータスを表示可能および埋め込み可能にします。
   - [Bitbucket] を選択した場合は、手順に従って Bitbucket に接続 (または再接続) します。Bitbucket の [Confirm access to your account] ページで、[Organization access] の [Grant access] を選択します。[Grant access] を選択した後で AWS CodeBuild コンソールに戻り、[Repository] でソースコードが含まれているリポジトリの名前を選択します。[Build Badge] チェックボックスをオンにして、プロジェクトのビルドステータスを表示可能および埋め込み可能にします。

     Important

     プロジェクトソースを更新すると、プロジェクトのビルドバッジの正確性に影響する場合があります。
5. [Environment: How to build] で以下の操作を行います。

   [Environment image] で、次のいずれかの操作を行います。

   - AWS CodeBuild で管理されている Docker イメージを使用するには、[Use an image managed by AWS CodeBuild] を選択し、次に [Operating system]、[Runtime]、および [Version] で選択を行います。
   - 別の Docker イメージを使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Other] または [Amazon ECR] を選択します。[Other] を選択した場合は、[Custom image ID] に Docker Hub の Docker イメージの名前とタグを *repository-name*/*image-name*:*image-tag* の形式で入力します。[Amazon ECR] を選択した場合は、[Amazon ECR repository] および [Amazon ECR image] を使用して AWS アカウントの Docker イメージを選択します。

   [Build specification] で、次のいずれかの操作を行います。

   - ソースコードのルートディレクトリの buildspec.yml ファイルを使用します。
   - ビルドコマンドを挿入してビルド仕様をオーバーライドします。

   詳細については、「ビルドスペックリファレンス (p. 95)」を参照してください。
6. [Artifacts: Where to put the artifacts from this build project] で、[Artifacts type] として次のいずれかの操作を行います。

- ビルド出力アーティファクトを作成しない場合は、[No artifacts] を選択します。
- ビルド出力を Amazon S3 バケットに保存する場合は、[Amazon S3] を選択して次のいずれかの操作を行います。
  - ビルド出力 ZIP ファイルまたはフォルダにプロジェクト名を使用する場合は、[Artifacts name] を空白のままにします。それ以外の場合は、[Artifacts name] ボックスに名前を入力します。(デフォルトでは、アーティファクト名はプロジェクト名です。別の名前を指定する場合は、アーティファクト名ボックスに名前を入力します。ZIP ファイルを出力する場合は、zip 拡張子を含めます。
  - [Bucket name] で、出力バケットの名前を選択します。
  - この手順の前の方で [Insert build commands] を選択した場合は、[Output files] に、ビルド出力 ZIP ファイルまたはフォルダに格納する、ビルドからのファイルの場所を入力します。複数の場所の場合は、各場所をコンマで区切ります (例: `appspec.yml, target/my-app.jar`)。詳細については、「ビルドスペックの構文 (p. 96)」の `files` の説明を参照してください。

7. [Service role] で、次のいずれかの操作を行います。
   - AWS CodeBuild サービスロールがない場合は、[Create a service role in your account] を選択します。[Role name] で、デフォルト名を受け入れるか、独自の名前を入力します。
   - AWS CodeBuild サービスロールがある場合は、[Choose an service existing role from your account] を選択します。[Role name] で、サービスロールを選択します。

     Note

     コンソールでは、ビルドプロジェクトの作成時や更新時に AWS CodeBuild サービスロールも作成できます。デフォルトでは、ロールはそのビルドプロジェクトでのみ使用できます。コンソールでは、このサービスロールを別のビルドプロジェクトと関連付けると、この別のビルドプロジェクトで使用できるようにロールが更新されます。サービスロールは最大 10 個のビルドプロジェクトで使用できます。
8. [Show advanced settings] を展開し、必要に応じて他の詳細な設定を行います。
9. [Continue] を選択します。[Review] ページで、[Save and build] を選択します。後でビルドを実行する場合は、[Save] を選択します。

ビルドバッジが有効化されたビルドプロジェクトの作成 (CLI)

ビルドプロジェクトの作成の詳細については、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」を参照してください。ビルドバッジを AWS CodeBuild プロジェクトに含めるには、*`badgeEnabled`* を `true` 値に設定する必要があります。

AWS CodeBuild のビルドバッジへのアクセス

ビルドバッジにアクセスするには、AWS CodeBuild コンソールまたは AWS CLI を使用できます。

- AWS CodeBuild コンソールでは、ビルドプロジェクトのリストの [Project] 列で、ビルドプロジェクトに対応するリンクを選択します。[Build project: ***`project-name`***] ページで、[Project details] を展開します。ビルドバッジの URL が [Advanced] の下に表示されます。詳細については、「ビルドプロジェクトの詳細を表示する (コンソール) (p. 160)」を参照してください。
- AWS CLI で、`batch-get-projects` コマンドを実行します。ビルドバッジの URL は出力のプロジェクト環境の詳細セクションを含まれています。詳細については、「ビルドプロジェクトの詳細を表示する (AWS CLI) (p. 161)」を参照してください。

Important

表示されるビルドバッジのリクエスト URL はマスターブランチのものですが、ビルドを実行したソースリポジトリの任意のブランチを指定できます。

AWS CodeBuild のビルドバッジの発行

ビルドバッジのリクエスト URL は、任意のリポジトリ (GitHub や AWS CodeCommit など) のマークダウンファイルに含め、最新ビルドのステータスを表示できます。

マークダウンのコード例:

```
![Build Status](https://codebuild.us-east-1.amazon.com/badges?uuid=...&branch=master)
```

AWS CodeBuild バッジのステータス

- PASSING 該当するブランチで最新ビルドが成功しました。
- FAILING 該当するブランチで最新ビルドがタイムアウト、失敗、途中終了、または停止しました。
- IN_PROGRESS 該当するブランチで最新ビルドが進行中です。
- UNKNOWN 該当するブランチでプロジェクトがビルドをまだ実行していないか、まったく実行したことがありません。また、ビルドバッジ機能が無効になっている可能性もあります。

AWS CodeBuild のビルド通知サンプル

Amazon CloudWatch Events には AWS CodeBuild のサポートが組み込まれています。CloudWatch イベント は、AWS リソースの変更を説明するシステムイベントのストリームです。CloudWatch イベント では、宣言型のルールを書き込んで、目的のイベントを自動アクションに関連付けます。このサンプルでは、Amazon CloudWatch Events と Amazon Simple Notification Service (Amazon SNS) を使用して、ビルドの成功、失敗、各ビルドフェーズへの移行、またはこれらのイベントの組み合わせを行うたびに、ビルド通知をサブスクライバーに送信します。

Important

このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon CloudWatch と Amazon SNS に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon CloudWatch 料金表」、および「Amazon SNS 料金表」を参照してください。

サンプルの実行

このサンプルを実行するには。

1. このサンプルで使用するトピックをすでに設定して Amazon SNS で購読している場合は、ステップ 4 に進みます。それ以外の場合は、AWS ルートアカウントまたは管理者 IAM ユーザーの代わりに IAM ユーザーを使用して Amazon SNS を操作する場合は、ユーザー (またはユーザーが関連付けられている IAM グループ) に以下のステートメントを (*`### BEGIN ADDING STATEMENT HERE ###`* と *`### END ADDING STATEMENT HERE ###`* の間に) 追加します。(AWS ルートアカウントを使用することは推奨されません。)このステートメントにより、Amazon SNS のトピックへの通知の表示、作成、サブスクライブ、および送信テストができます。省略記号 (`...`) は、簡潔にするために使用され、ステートメントを追加する場所の特定に役立ちます。ステートメントを削除しないでください、また、これらの省略記号を既存のポリシーに入力しないでください。

```
{
```

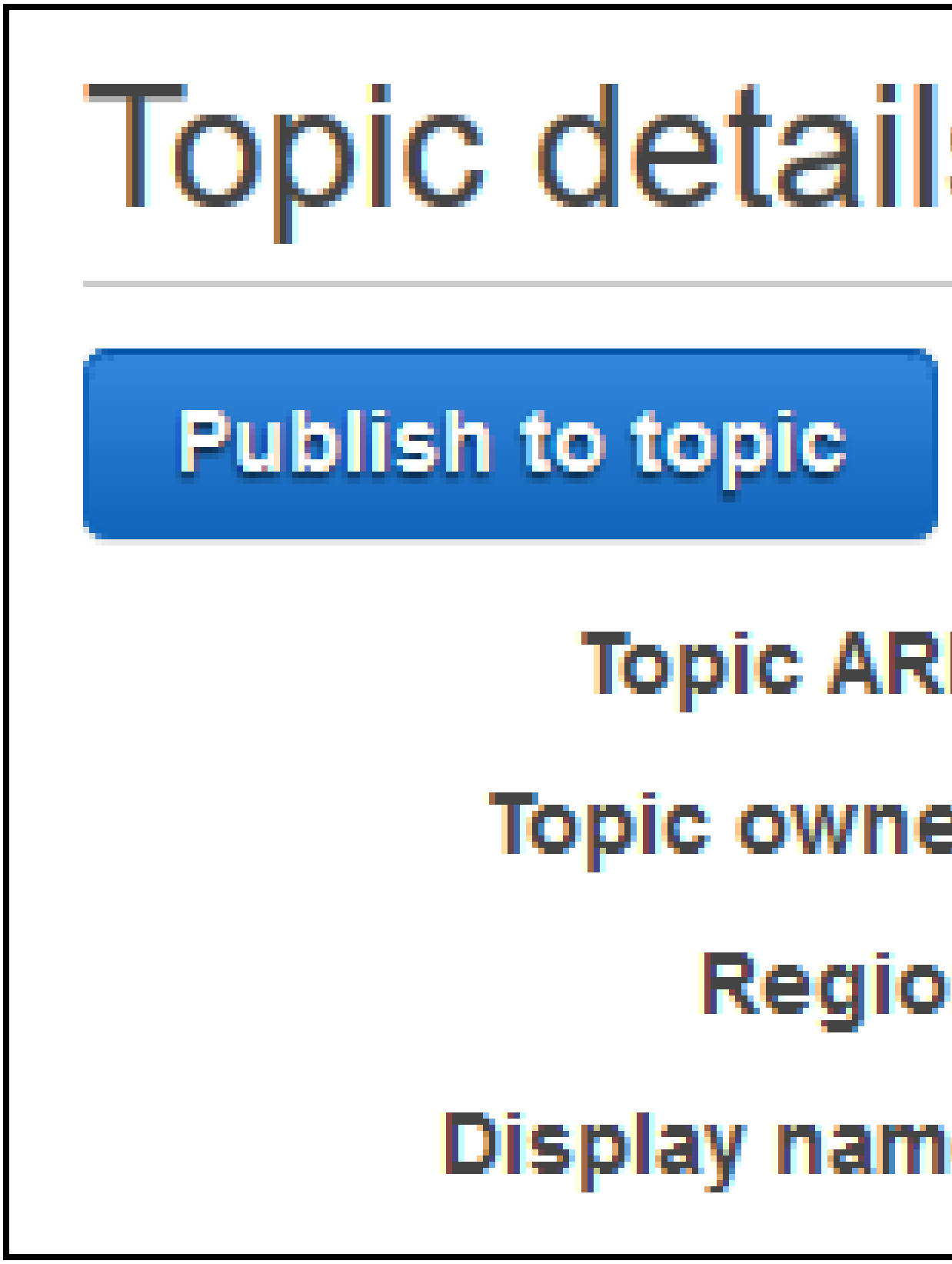
```
  "Statement": [
    ### BEGIN ADDING STATEMENT HERE ###
    {
      "Action": [
        "sns:CreateTopic",
        "sns:GetTopicAttributes",
        "sns:List*",
        "sns:Publish",
        "sns:SetTopicAttributes",
        "sns:Subscribe"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    ### END ADDING STATEMENT HERE ###
    ...
  ],
  "Version": "2012-10-17"
}
```

Note

このポリシーを変更する IAM エンティティは、ポリシーを変更するために IAM のアクセス権限を持っている必要があります。
詳細については、「カスタマー管理ポリシーの編集」または、「IAM ユーザーガイド」の「インラインポリシーの使用 (コンソール)」の「グループ、ユーザー、ロールのインラインポリシーを編集または削除するには」セクションを参照してください。

2. Amazon SNS でトピックを作成または識別します。AWS CodeBuild は、Amazon SNS を介してこのトピックにビルド通知を送信するために CloudWatch イベント を使用します。トピックを作成するには:

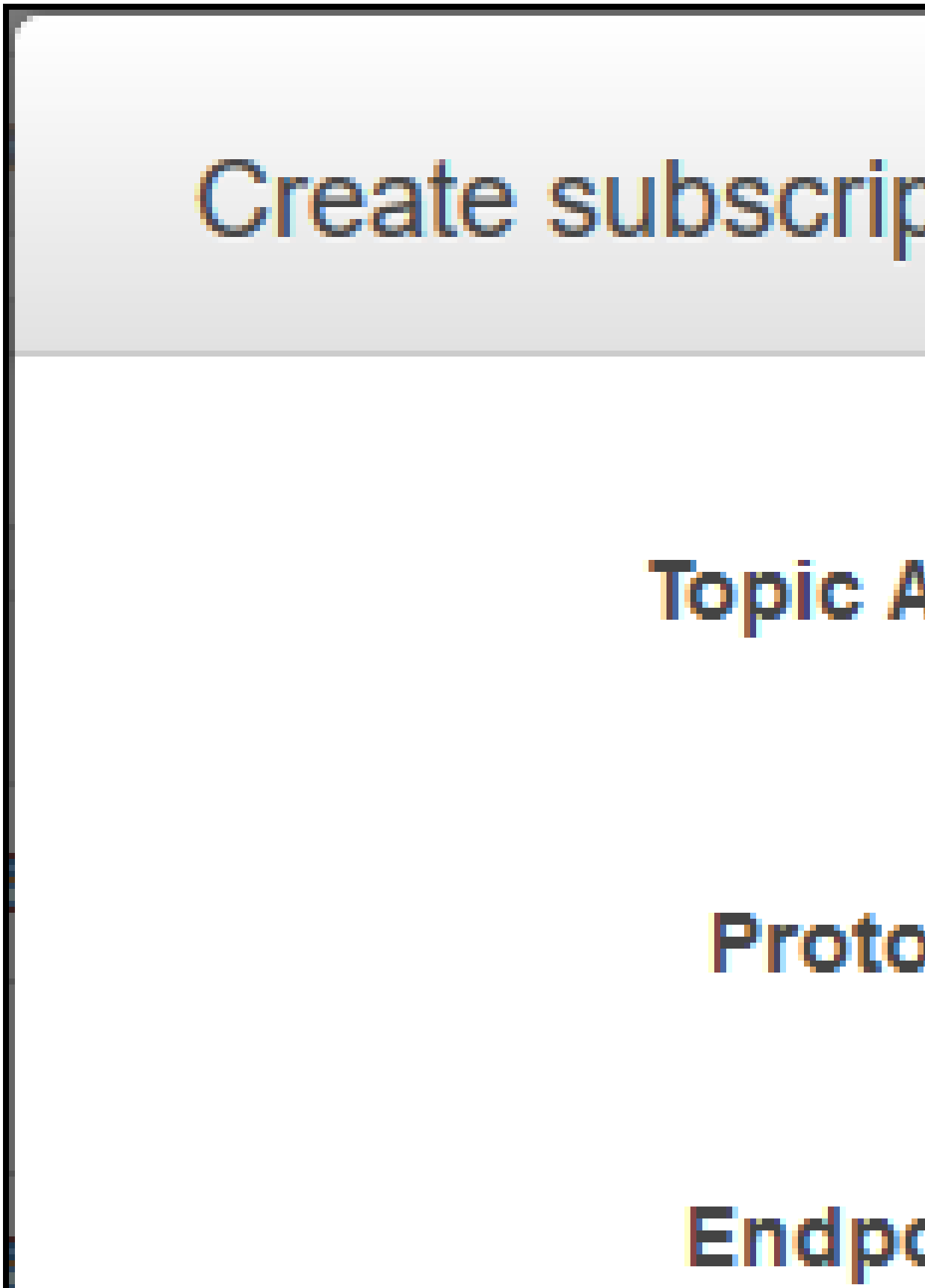
   1. https://console.aws.amazon.com/sns で Amazon SNS コンソールを開きます。
   2. [Create topic] を選択します。
   3. [Create new topic] ダイアログボックスの [Topic name] に、トピックの名前 (例: **`CodeBuildDemoTopic`**) を入力します。(別の名前を選択する場合は、このサンプル全体でそれを置き換えてください。)
   4. [Create topic] を選択します。
   5. 次のスクリーンショットに示すように、[Topic details: CodeBuildDemoTopic] ページで、[Topic ARN] の値をコピーします。この値は次のステップで必要になります。


Topic detail
Publish to topic
Topic AR
Topic owne
Regio
Display nam

詳細については、『Amazon SNS 開発者ガイド』の「トピックの作成」を参照してください。

3. 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. 前の手順で Amazon SNS コンソールを開き、ナビゲーションペインで、[Subscriptions] を選択してから、[Create subscription] を選択します。
   2. [Create subscription] ダイアログボックスの [Topic ARN] に、前のステップからコピーしたトピック ARN を貼り付けます。
   3. [Protocol] で [Email] 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。結果を以下のスクリーンショットと比較します。


Create subscrip
Topic A
Proto
Endpo

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 メール通知の受信を開始するには、受信者は受信登録確認メールで [Confirm subscription] 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 開発者ガイド』の「トピックのサブスクライブ」を参照してください。

4. AWS ルートアカウントまたは管理者 IAM ユーザーの代わりに IAM ユーザーを使用して CloudWatch イベント を操作する場合は、ユーザー (またはユーザーが関連付けられている IAM グループ) に以下のステートメントを (*### BEGIN ADDING STATEMENT HERE ###* と *### END ADDING STATEMENT HERE ###* の間に) 追加します。(AWS ルートアカウントを使用することは推奨されません。)このステートメントにより、CloudWatch イベント を使用できます。省略記号 (`...`) は、簡潔にするために使用され、ステートメントを追加する場所の特定に役立ちます。ステートメントを削除しないでください、また、これらの省略記号を既存のポリシーに入力しないでください。

```
{
  "Statement": [
    ### BEGIN ADDING STATEMENT HERE ###
    {
      "Action": [
        "events:*",
        "iam:PassRole"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    ### END ADDING STATEMENT HERE ###
    ...
  ],
  "Version": "2012-10-17"
}
```

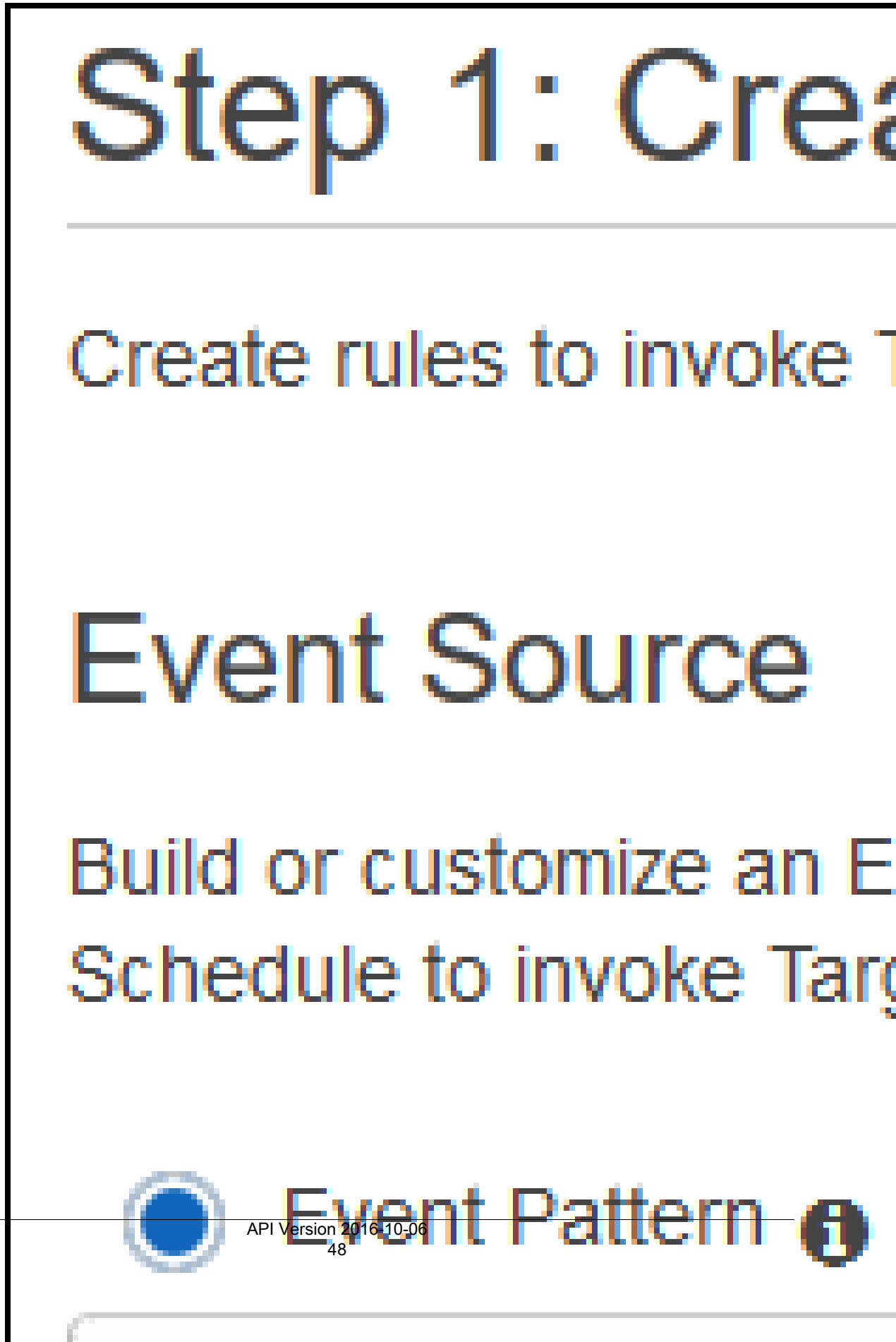
Note

このポリシーを変更する IAM エンティティは、ポリシーを変更するために IAM のアクセス権限を持っている必要があります。
詳細については、「カスタマー管理ポリシーの編集」または、「IAM ユーザーガイド」の「インラインポリシーの使用 (コンソール)」の「グループ、ユーザー、ロールのインラインポリシーを編集または削除するには」セクションを参照してください。

5. CloudWatch イベント にルールを作成するこれを行うには、https://console.aws.amazon.com/cloudwatch で、CloudWatch console を開きます。
6. ナビゲーションペインの [Events] で、[Rules] を選択してから、[Create rule] を選択します。
7. [Step 1: Create rule page] で、[Event Pattern] および [Build event pattern to match events by service] が既に選択されている必要があります。
8. [Service Name] で、[CodeBuild] を選択します。[Event Type] では、[All Events] が既に選択されている必要があります。
9. [Event Pattern Preview] では次のコードが表示されているはずです。

```
{
  "source": [
    "aws.codebuild"
  ]
}
```

これまでの結果を以下のスクリーンショットと比較します。


Step 1: Crea
Create rules to invoke T
Event Source
Build or customize an E
Schedule to invoke Tarq
Event Pattern

10. [Edit] を選択して、Event Pattern Preview のコードを、次の 2 つのルールパターンのいずれかに置き換えます。

    この最初のルールパターンは、AWS CodeBuild で指定されたビルドプロジェクトに対して、ビルドが開始または完了するたびにイベントをトリガーします。

```
{
  "source": [
    "aws.codebuild"
  ],
  "detail-type": [
    "CodeBuild Build State Change"
  ],
  "detail": {
    "build-status": [
      "IN_PROGRESS",
      "SUCCEEDED",
      "FAILED",
      "STOPPED"
    ],
    "project-name": [
      "my-demo-project-1",
      "my-demo-project-2"
    ]
  }
}
```

    前述のルールで、必要に応じて次のコードを変更します。

    - ビルドが開始または完了するたびにイベントをトリガーするには、build-status 配列に示すように、すべて値をそのままにするか、build-status 配列を完全に削除します。
    - ビルドが完了したときにのみイベントをトリガーするには、build-status 配列から IN_PROGRESS を削除します。
    - ビルドの開始時にのみイベントをトリガーするには、build-status 配列から IN_PROGRESS を除くすべての値を削除します。
    - すべてのビルドプロジェクトのイベントをトリガーするには、project-name 配列を完全に削除します。
    - 個々のビルドプロジェクトのイベントのみをトリガーするには、project-name 配列に各ビルドプロジェクトの名前を指定します。

    この 2 番目のルールパターンでは、AWS CodeBuild の指定されたビルドプロジェクトに対して、ビルドが各ビルドフェーズに移動するたびにイベントをトリガーします。

```
{
  "source": [
    "aws.codebuild"
  ],
  "detail-type": [
    "CodeBuild Build Phase Change"
  ],
  "detail": {
    "completed-phase": [
      "SUBMITTED",
      "PROVISIONING",
      "DOWNLOAD_SOURCE",
      "INSTALL",
      "PRE_BUILD",
      "BUILD",
      "POST_BUILD",
```

```
      "UPLOAD_ARTIFACTS",
      "FINALIZING"
    ],
    "completed-phase-status": [
      "TIMED_OUT",
      "STOPPED",
      "FAILED",
      "SUCCEEDED",
      "FAULT",
      "CLIENT_ERROR"
    ],
    "project-name": [
      "my-demo-project-1",
      "my-demo-project-2"
    ]
  }
}
```

前述のルールで、必要に応じて次のコードを変更します。

- 各ビルドフェーズの変更 (ビルドごとに最大 9 個の通知を送信する可能性があります) ごとにイベントをトリガーするには、`completed-phase` 配列に示されているように、すべての値をそのままにするか、`completed-phase` 配列を完全に削除します。
- 個々のビルドフェーズの変更に対してのみイベントをトリガーするには、イベントをトリガーしない `completed-phase` 配列の各ビルドフェーズの名前を削除します。
- 各ビルドフェーズステータスが変更するたびにイベントをトリガーするには、`completed-phase-status` 配列に示すように、すべて値をそのままにするか、`completed-phase-status` 配列を完全に削除します。
- 個々のビルドフェーズステータスの変更に対してのみイベントをトリガーするには、イベントをトリガーしない `completed-phase-status` 配列の各ビルドフェーズステータスの名前を削除します。
- すべてのビルドプロジェクトのイベントをトリガーするには、`project-name` 配列を削除します。
- 個々のビルドプロジェクトのイベントをトリガーするには、`project-name` 配列に各ビルドプロジェクトの名前を指定します。

> **Note**
>
> ビルド状態の変更とビルドフェーズの変更の両方のイベントをトリガーする場合は、ビルド状態の変更とビルドフェーズの変更に対して 2 つの個別のルールを作成する必要があります。両方のルールを 1 つのルールに結合しようとすると、組み合わされたルールが予期しない結果を引き起こすか、まったく動作しなくなる可能性があります。

コードの置換を完了したら、[Save] を選択します。

11. [Targets] で、[Add target] を選択します。
12. ターゲットのリストで、[SNS topic] を選択します。
13. [Topic] で、以前に指定した、または作成したトピックを選択します。
14. [Configure input] を展開して、[Input Transformer] を閉じます。
15. [Input Path] ボックスに、次のいずれかの入力パスを入力します。

    `CodeBuild Build State Change` の `detail-type` 値を持つルールの場合は、次のように入力します。

```
{"build-id":"$.detail.build-id","project-name":"$.detail.project-name","build-
status":"$.detail.build-status"}
```

`CodeBuild Build Phase Change` の `detail-type` 値を持つルールの場合は、次のように入力します。

```
{"build-id":"$.detail.build-id","project-name":"$.detail.project-name","completed-phase":"$.detail.completed-phase","completed-phase-status":"$.detail.completed-phase-status"}
```

Note

他のタイプの情報を取得するには、「ビルド通知入力形式リファレンス (p. 55)」を参照してください。

16. [Input Template] ボックスに、次のいずれかの入力テンプレートを入力します。

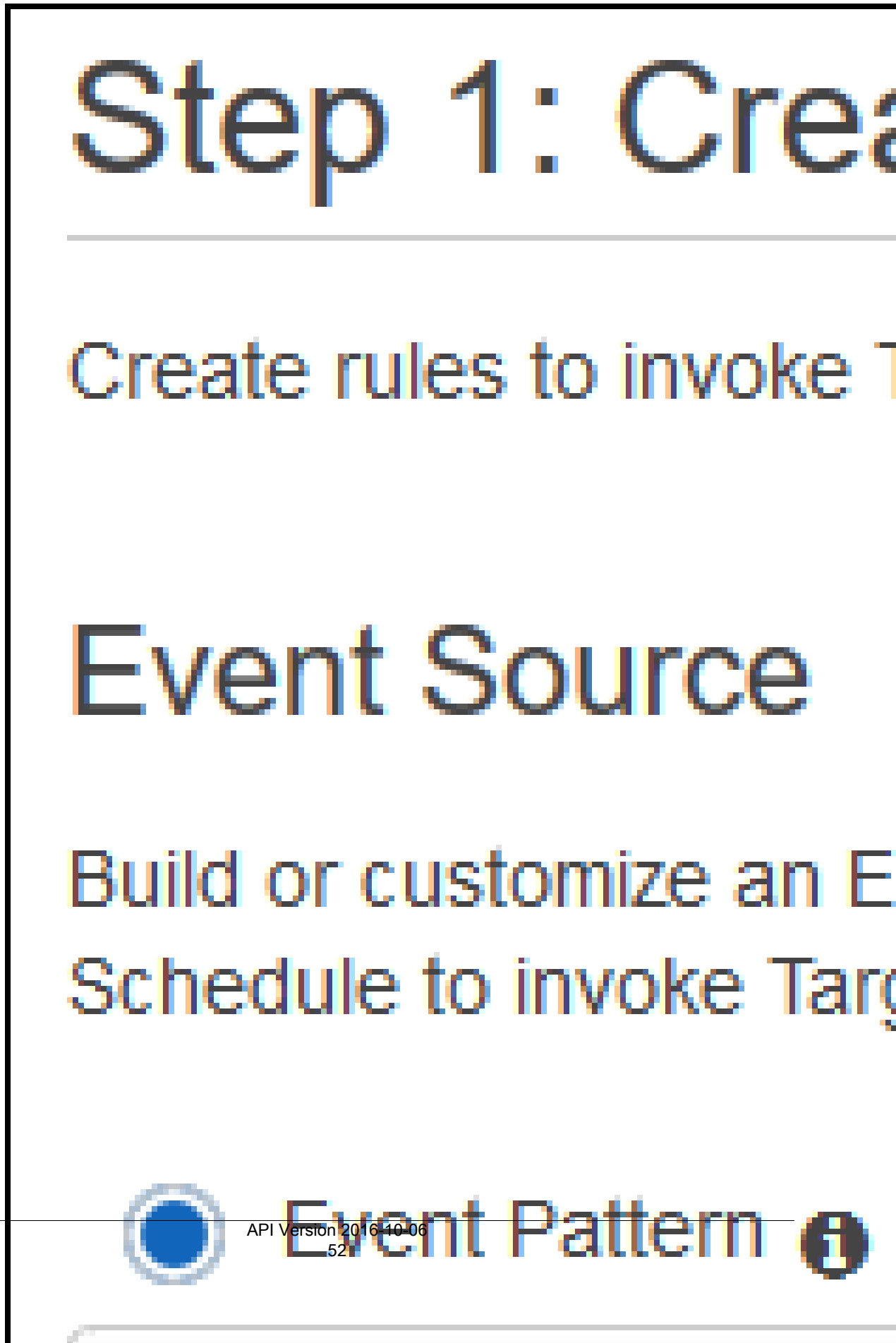
`CodeBuild Build State Change` の `detail-type` 値を持つルールの場合は、次のように入力します。

```
"Build '<build-id>' for build project '<project-name>' has reached the build status of '<build-status>'."
```

`CodeBuild Build Phase Change` の `detail-type` 値を持つルールの場合は、次のように入力します。

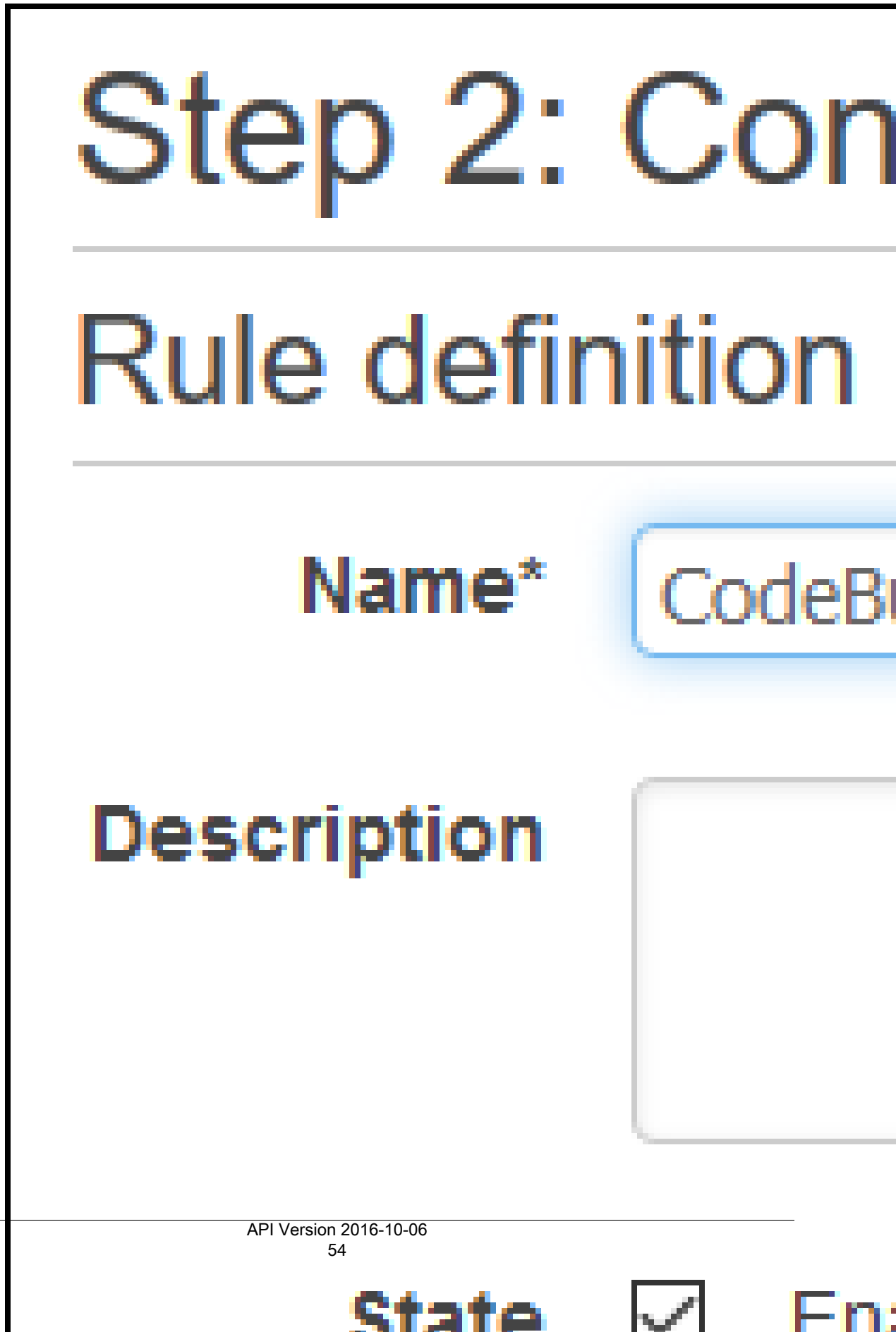
```
"Build '<build-id>' for build project '<project-name>' has completed the build phase of '<completed-phase>' with a status of '<completed-phase-status>'."
```

これまでの結果を次のスクリーンショットと比較します。このスクリーンショットでは、`CodeBuild Build State Change` の `detail-type` 値を持つルールが表示されます。


Step 1: Crea
Create rules to invoke T
Event Source
Build or customize an E
Schedule to invoke Targ
Event Pattern

17. [Configure details] を選択します。
18. [Step 2: Configure rule details] ページで、[Name] とオプションの [Description] を入力します。[State] の [Enabled] ボックスを選択したままにします。

    これまでの結果を以下のスクリーンショットと比較します。


Step 2: Con
Rule definition
Name*
CodeB
Description
State
En

19. [Create rule] を選択します。
20. たとえば、「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルドプロジェクトの作成とビルドを実行し、ビルド情報を表示します。
21. AWS CodeBuild がビルド通知を現在正常に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。

ルールの動作を変更するには、CloudWatch コンソールで、変更するルールを選択してから、[Actions]、[Edit] を選択します。ルールを変更し、[Configure details] を選択してから [Update rule] を選択します。

ルールがビルド通知を送信するのを停止するには、CloudWatch コンソールで使用を停止するルールを選択し、[Actions]、[Disable] を選択します。

ルールをまとめて削除するには、CloudWatch コンソールで、削除するルールを選択してから、[Actions]、[Delete] を選択します。

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

ビルド通知入力形式リファレンス

CloudWatch では、JSON 形式で通知が送信されます。

ビルド状態変更通知は次の形式を使用します。

```
{
  "version": "0",
  "id": "c030038d-8c4d-6141-9545-00ff7b7153EX",
  "detail-type": "CodeBuild Build State Change",
  "source": "aws.codebuild",
  "account": "123456789012",
  "time": "2017-09-01T16:14:28Z",
  "region": "us-west-2",
  "resources":[
    "arn:aws:codebuild:us-west-2:123456789012:build/my-sample-project:8745a7a9-
c340-456a-9166-edf953571bEX"
  ],
  "detail":{
    "build-status": "SUCCEEDED",
    "project-name": "my-sample-project",
    "build-id": "arn:aws:codebuild:us-west-2:123456789012:build/my-sample-project:8745a7a9-
c340-456a-9166-edf953571bEX",
    "additional-information": {
      "artifact": {
        "md5sum": "da9c44c8a9a3cd4b443126e823168fEX",
        "sha256sum": "6ccc2ae1df9d155ba83c597051611c42d60e09c6329dcb14a312cecc0a8e39EX",
        "location": "arn:aws:s3:::codebuild-123456789012-output-bucket/my-output-
artifact.zip"
      },
      "environment": {
        "image": "aws/codebuild/dot-net:1.1",
        "privileged-mode": false,
        "compute-type": "BUILD_GENERAL1_SMALL",
        "type": "LINUX_CONTAINER",
        "environment-variables": []
```

```
      },
      "timeout-in-minutes": 60,
      "build-complete": true,
      "initiator": "MyCodeBuildDemoUser",
      "build-start-time": "Sep 1, 2017 4:12:29 PM",
      "source": {
        "location": "codebuild-123456789012-input-bucket/my-input-artifact.zip",
        "type": "S3"
      },
      "logs": {
        "group-name": "/aws/codebuild/my-sample-project",
        "stream-name": "8745a7a9-c340-456a-9166-edf953571bEX",
        "deep-link": "https://console.aws.amazon.com/cloudwatch/home?region=us-
west-2#logEvent:group=/aws/codebuild/my-sample-project;stream=8745a7a9-c340-456a-9166-
edf953571bEX"
      },
      "phases": [
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:12:29 PM",
          "end-time": "Sep 1, 2017 4:12:29 PM",
          "duration-in-seconds": 0,
          "phase-type": "SUBMITTED",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:12:29 PM",
          "end-time": "Sep 1, 2017 4:13:05 PM",
          "duration-in-seconds": 36,
          "phase-type": "PROVISIONING",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:05 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 4,
          "phase-type": "DOWNLOAD_SOURCE",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 0,
          "phase-type": "INSTALL",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 0,
          "phase-type": "PRE_BUILD",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 70,
          "phase-type": "BUILD",
          "phase-status": "SUCCEEDED"
        },
        {
```

```
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 0,
          "phase-type": "POST_BUILD",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 0,
          "phase-type": "UPLOAD_ARTIFACTS",
          "phase-status": "SUCCEEDED"
        },
         {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:26 PM",
          "duration-in-seconds": 4,
          "phase-type": "FINALIZING",
          "phase-status": "SUCCEEDED"
        },
        {
          "start-time": "Sep 1, 2017 4:14:26 PM",
          "phase-type": "COMPLETED"
        }
      ]
    },
    "current-phase": "COMPLETED",
    "current-phase-context": "[]",
    "version": "1"
  }
}
```

ビルドフェーズ変更通知は次の形式を使用します。

```
{
  "version": "0",
  "id": "43ddc2bd-af76-9ca5-2dc7-b695e15adeEX",
  "detail-type": "CodeBuild Build Phase Change",
  "source": "aws.codebuild",
  "account": "123456789012",
  "time": "2017-09-01T16:14:21Z",
  "region": "us-west-2",
  "resources":[
    "arn:aws:codebuild:us-west-2:123456789012:build/my-sample-project:8745a7a9-
c340-456a-9166-edf953571bEX"
  ],
  "detail":{
    "completed-phase": "COMPLETED",
    "project-name": "my-sample-project",
    "build-id": "arn:aws:codebuild:us-west-2:123456789012:build/my-sample-project:8745a7a9-
c340-456a-9166-edf953571bEX",
    "completed-phase-context": "[]",
    "additional-information": {
      "artifact": {
        "md5sum": "da9c44c8a9a3cd4b443126e823168fEX",
        "sha256sum": "6ccc2ae1df9d155ba83c597051611c42d60e09c6329dcb14a312cecc0a8e39EX",
        "location": "arn:aws:s3:::codebuild-123456789012-output-bucket/my-output-
artifact.zip"
      },
      "environment": {
        "image": "aws/codebuild/dot-net:1.1",
        "privileged-mode": false,
```

```
        "compute-type": "BUILD_GENERAL1_SMALL",
        "type": "LINUX_CONTAINER",
        "environment-variables": []
      },
      "timeout-in-minutes": 60,
      "build-complete": true,
      "initiator": "MyCodeBuildDemoUser",
      "build-start-time": "Sep 1, 2017 4:12:29 PM",
      "source": {
        "location": "codebuild-123456789012-input-bucket/my-input-artifact.zip",
        "type": "S3"
      },
      "logs": {
        "group-name": "/aws/codebuild/my-sample-project",
        "stream-name": "8745a7a9-c340-456a-9166-edf953571bEX",
        "deep-link": "https://console.aws.amazon.com/cloudwatch/home?region=us-
west-2#logEvent:group=/aws/codebuild/my-sample-project;stream=8745a7a9-c340-456a-9166-
edf953571bEX"
      },
      "phases": [
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:12:29 PM",
          "end-time": "Sep 1, 2017 4:12:29 PM",
          "duration-in-seconds": 0,
          "phase-type": "SUBMITTED",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:12:29 PM",
          "end-time": "Sep 1, 2017 4:13:05 PM",
          "duration-in-seconds": 36,
          "phase-type": "PROVISIONING",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:05 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 4,
          "phase-type": "DOWNLOAD_SOURCE",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 0,
          "phase-type": "INSTALL",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:13:10 PM",
          "duration-in-seconds": 0,
          "phase-type": "PRE_BUILD",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:13:10 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 70,
          "phase-type": "BUILD",
```

```
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 0,
          "phase-type": "POST_BUILD",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:21 PM",
          "duration-in-seconds": 0,
          "phase-type": "UPLOAD_ARTIFACTS",
          "phase-status": "SUCCEEDED"
        },
        {
          "phase-context": [],
          "start-time": "Sep 1, 2017 4:14:21 PM",
          "end-time": "Sep 1, 2017 4:14:26 PM",
          "duration-in-seconds": 4,
          "phase-type": "FINALIZING",
          "phase-status": "SUCCEEDED"
        },
        {
          "start-time": "Sep 1, 2017 4:14:26 PM",
          "phase-type": "COMPLETED"
        }
      ]
    },
    "completed-phase-status": "SUCCEEDED",
    "completed-phase-duration-seconds": 4,
    "version": "1",
    "completed-phase-start": "Sep 1, 2017 4:14:21 PM",
    "completed-phase-end": "Sep 1, 2017 4:14:26 PM"
  }
}
```


AWS CodeBuild のカスタム Docker イメージのサンプル

このサンプルでは、AWS CodeBuild とカスタム Docker ビルドイメージ (Docker ハブの `docker:dind`) を使用して Docker イメージをビルドして実行します。

代わりに、Docker をサポートする AWS CodeBuild に用意されているビルドイメージを使用して Docker イメージをビルドする方法については、「Docker サンプル (p. 25)」を参照してください。

> **Important**
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

トピック

- 関連リソース (p. 61)

サンプルの実行

このサンプルを実行するには

1. このトピックの「ディレクトリ構造」セクションと「ファイル」セクションの説明に従ってファイルを作成し、Amazon S3 入力バケットにアップロードするか、AWS CodeCommit、GitHub、または Bitbucket の各リポジトリにアップロードします。

   Important

   (root directory name) をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
   Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。
2. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルドプロジェクトを作成して、ビルドを実行し、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-docker-custom-image-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-
bucket/DockerCustomImageSample.zip"
  },
  "artifacts": {
    "type": "NO_ARTIFACTS"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "docker:dind",
    "computeType": "BUILD_GENERAL1_SMALL",
    "privilegedMode": true
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. ビルドの結果を表示するには、ビルドの log で文字列 `Hello, World!` を探します。詳細については、「ビルドの詳細を表示する (p. 173)」を参照してください。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
    |-- buildspec.yml
    `-- Dockerfile
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

`buildspec.yml` (*(root directory name)* 内)

```
version: 0.2

phases:
  install:
    commands:
      - nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://0.0.0.0:2375 --storage-driver=overlay&
      - timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
  pre_build:
    commands:
      - docker build -t helloworld .
  build:
    commands:
      - docker images
      - docker run helloworld echo "Hello, World!"
```

`Dockerfile` (*(root directory name)* 内)

```
FROM alpine

RUN ls
```

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の AWS CodeDeploy サンプル

このサンプルでは、AWS CodeBuild に Maven を使用して、ビルド出力として `my-app-1.0-SNAPSHOT.jar` という名前の 1 つの JAR ファイルを生成するよう指示しています。このサンプルでは、AWS CodeDeploy を使用して JAR ファイルを Amazon Linux インスタンスにデプロイします。(または、AWS CodePipeline を使用して、AWS CodeDeploy の使用を自動化して JAR ファイルを Amazon Linux インスタンスにデプロイすることもできます)。このサンプルは、Apache Maven ウェブサイトの「5 分で Maven」トピックに基づいています。

> **Important**
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild の料金、および Amazon S3、AWS KMS、CloudWatch Logs、Amazon EC2 に関連した AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」、「Amazon EC2 料金表」を参照してください。

サンプルの実行

このサンプルを実行するには。

1. Maven をダウンロードし、インストールします。詳細については、Apache Maven ウェブサイトの「Apache Maven のダウンロード」および「Apache Maven のインストール」を参照してください。

2. ローカルコンピュータまたはインスタンスの空のディレクトリに切り替えて、この Maven コマンドを実行します。

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

成功すると、このディレクトリ構造とファイルが作成されます。

```
(root directory name)
     `-- my-app
           |-- pom.xml
           `-- src
                 |-- main
                 |     `-- java
                 |           `-- com
                 |                 `-- mycompany
                 |                       `-- app
                 |                             `-- App.java
                 `-- test
                       `-- java
                             `-- com
                                   `-- mycompany
                                         `-- app
                                               `-- AppTest.java
```

3. このコンテンツを含むファイルを作成します。ファイルに `buildspec.yml` という名前を付け、それを *(root directory name)*/my-app ディレクトリに追加します。

```
version: 0.2

phases:
  build:
    commands:
      - echo Build started on `date`
      - mvn test
  post_build:
    commands:
      - echo Build completed on `date`
      - mvn package
artifacts:
  files:
    - target/my-app-1.0-SNAPSHOT.jar
    - appspec.yml
  discard-paths: yes
```

4. このコンテンツを含むファイルを作成します。ファイルに `appspec.yml` という名前を付け、それを *(root directory name)*/my-app ディレクトリに追加します。

```
version: 0.0
os: linux
files:
  - source: ./my-app-1.0-SNAPSHOT.jar
    destination: /tmp
```

完了したら、ディレクトリ構造およびファイルは次のようになります。

```
(root directory name)
     `-- my-app
           |-- buildspec.yml
           |-- appspec.yml
           |-- pom.xml
```

```
          `-- src
                |-- main
                |     `-- java
                |           `-- com
                |                 `-- mycompany
                |                       `-- app
                |                             `-- App.java
                `-- test
                      `-- java
                            `-- com
                                  `-- mycompany
                                        `-- app
                                              ` -- AppTest.java
```

5. *(root directory name)*/my-app の中のディレクトリ構造とファイルを含む ZIP ファイルを作成し、AWS CodeBuild および AWS CodeDeploy でサポートされているソースコードリポジトリのタイプ (Amazon S3 入力バケットまたは GitHub または Bitbucket リポジトリなど) に ZIP ファイルをアップロードします。

   Important

   最終的なビルド出力アーティファクトを AWS CodePipeline を使用してデプロイする場合、ソースコードを Bitbucket リポジトリにアップロードすることはできません。
   (root directory name) または *(root directory name)*/my-app を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)*/my-app の中のディレクトリとファイルだけです。ZIP ファイルは、次のディレクトリとファイルが含まれている必要があります。

```
CodeDeploySample.zip
     |--buildspec.yml
     |-- appspec.yml
     |-- pom.xml
     `-- src
           |-- main
           |     `-- java
           |           `-- com
           |                 `-- mycompany
           |                       `-- app
           |                             `-- App.java
           `-- test
                 `-- java
                       `-- com
                             `-- mycompany
                                   `-- app
                                         ` -- AppTest.java
```

6. ビルドプロジェクトを作成する (p. 146) の手順に従って、ビルドプロジェクトを作成します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、create-project コマンドへの JSON 形式の入力はこれに似ています。(プレースホルダを独自の値に置き換えます。)

```
{
  "name": "sample-codedeploy-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/CodeDeploySample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "CodeDeployOutputArtifact.zip"
  },
```

```
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/java:openjdk-8",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

7. ビルド出力アーティファクトを AWS CodeDeploy でデプロイする場合は、ビルドの実行 (p. 167) の各ステップに従う必要があります。それ以外の場合は、この手順をスキップしてください。(AWS CodePipeline でビルド出力アーティファクトをデプロイする予定がある場合、AWS CodePipeline は AWS CodeBuild を使用して自動的にビルドを実行するため)。
8. 次のものを含め、AWS CodeDeploy を使用するための設定手順を完了します。

   - AWS CodeDeploy、および、AWS CodeDeploy が依存する AWS のサービスとアクションへの IAM ユーザーのアクセスを許可します。詳細については、AWS CodeDeploy ユーザーガイドの「IAM ユーザーのプロビジョニング」を参照してください。
   - AWS CodeDeploy がビルド出力アーティファクトをデプロイする場所のインスタンスを識別できるように、サービスロールを作成または指定します。詳細については、『AWS CodeDeploy ユーザーガイド』の「AWS CodeDeploy のサービスロールの作成」を参照してください。
   - IAM インスタンスプロファイルを作成または指定して、インスタンスがビルド出力アーティファクトを含む Amazon S3 入力バケットまたは GitHub リポジトリにアクセスできるようにします。詳細については、『AWS CodeDeploy ユーザーガイド』の「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。
9. ビルド出力アーティファクトがデプロイされる AWS CodeDeploy と互換性のある Amazon Linux インスタンスを作成または指定します。詳細については、『AWS CodeDeploy ユーザーガイド』の「AWS CodeDeploy のインスタンスの使用」を参照してください。
10. AWS CodeDeploy アプリケーションとデプロイグループを作成または指定します。詳細については、『AWS CodeDeploy ユーザーガイド』の「AWS CodeDeploy を使用したアプリケーションの作成」を参照してください。
11. ビルド出力アーティファクトをインスタンスにデプロイします。

    AWS CodeDeploy でデプロイするには、『AWS CodeDeploy ユーザーガイド』の「AWS CodeDeploy でのリビジョンのデプロイ」を参照してください。

    AWS CodePipeline でデプロイするには、「AWS CodeBuild で AWS CodePipeline を使用する (p. 126)」を参照してください。
12. デプロイが完了した後にビルド出力アーティファクトを見つけるには、インスタンスにサインインして、`/tmp` ディレクトリ内で `my-app-1.0-SNAPSHOT.jar` という名前のファイルを探します。

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の AWS Lambda サンプル

Lambda などのリソースを使用するサーバーレスアプリケーションの標準モデルを定義するために、AWS では AWS サーバーレスアプリケーションモデル (AWS SAM) を作成しました。詳細については、GitHub の AWS サーバーレスアプリケーションモデルリポジトリを参照してください。

AWS CodeBuild を使用して、AWS SAM 標準に準拠するサーバーレスアプリケーションをパッケージ化してデプロイすることができます。デプロイのステップで、AWS CodeBuild は AWS CloudFormation を使用できます。AWS CodeBuild と AWS CloudFormation を使用したサーバーレスアプリケーションの構築とデプロイを自動化するには、AWS CodePipeline を使用できます。

詳細については、AWS Lambda Developer Guideの「Lambda ベースのアプリケーションのデプロイ」を参照してください。Lambda、AWS CloudFormation、AWS CodePipeline とともに AWS CodeBuild を使用したサーバーレスアプリケーションのサンプルを試すには、AWS Lambda Developer Guideの「Lambda ベースのアプリケーションのデプロイの自動化」を参照してください。

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の AWS Elastic Beanstalk サンプル

このサンプルでは、AWS CodeBuild に Maven を使用して、ビルド出力として `my-web-app.war` という名前の 1 つの WAR ファイルを生成するよう指示しています。このサンプルでは、WAR ファイルを Elastic Beanstalk 環境のインスタンスにデプロイします。このサンプルは、Java サンプル (p. 89) に基づいています。

> Important
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild の料金、および Amazon S3、AWS KMS、CloudWatch Logs、Amazon EC2 に関連した AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」、「Amazon EC2 料金表」を参照してください。

ソースコードの作成

このセクションでは、ビルドするソースコードを生成するために Maven を使用します。その後、AWS CodeBuild を使用して、このソースコードに基づいて WAR ファイルを構築します。

1. Maven をダウンロードし、インストールします。詳細については、Apache Maven ウェブサイトの「Apache Maven のダウンロード」および「Apache Maven のインストール」を参照してください。
2. ローカルコンピュータまたはインスタンスの空のディレクトリに切り替えて、この Maven コマンドを実行します。

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-web-app -
DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

成功すると、このディレクトリ構造とファイルが作成されます。

```
(root directory name)
     `-- my-web-app
           |-- pom.xml
           `-- src
                 `-- main
                       |-- resources
```

```
                    `-- webapp
                          |-- WEB-INF
                          |     `-- web.xml
                          `-- index.jsp
```

Maven を実行したら、次のいずれかのシナリオに進みます。

- シナリオ A: AWS CodeBuild を手動で実行し、Elastic Beanstalk に手動でデプロイする (p. 66)
- シナリオ B: AWS CodePipeline を使用して AWS CodeBuild を実行し、Elastic Beanstalk にデプロイする (p. 68)
- シナリオ C: Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用して、AWS CodeBuild を実行し、Elastic Beanstalk 環境にデプロイする (p. 70)

シナリオ A: AWS CodeBuild を手動で実行し、Elastic Beanstalk に手動でデプロイする

このシナリオでは、作成するソースコードを手動で作成してアップロードします。次に、AWS CodeBuild と Elastic Beanstalk コンソールを使用してソースコードを作成し、Elastic Beanstalk アプリケーションと環境を作成し、ビルド出力を環境にデプロイします。

ステップ A1: ソースコードにファイルを追加する

このステップでは、Elastic Beanstalk 設定ファイルとビルドスペックファイルを ソースコードの作成 (p. 65) のコードに追加します。その後、ソースコードを Amazon S3 入力バケットまたは AWS CodeCommit または GitHub リポジトリにアップロードします。

1. `.ebextensions` という名前のサブディレクトリを、*(root directory name)*`/my-web-app` ディレクトリ内に作成します。`.ebextensions` サブディレクトリに、`fix-path.config` という名前のファイルをこのコンテンツで作成します。

```
container_commands:
  fix_path:
    command: "unzip my-web-app.war 2>&1 > /var/log/my_last_deploy.log"
```

2. 次の内容で、`buildspec.yml` というファイルを作成します ファイルを *(root directory name)*`/my-web-app` ディレクトリ内に保存します。

```
version: 0.2

phases:
  post_build:
    commands:
      - mvn package
      - mv target/my-web-app.war my-web-app.war
artifacts:
  files:
    - my-web-app.war
    - .ebextensions/**/*
```

3. ファイル構造は次のようになります。

```
(root directory name)
     `-- my-web-app
           |-- .ebextensions
           |     `-- fix-path.config
           |-- src
```

```
|     `-- main
|           |-- resources
|           `-- webapp
|                 |-- WEB-INF
|                 |     `-- web.xml
|                 `-- index.jsp
|-- buildpsec.yml
`-- pom.xml
```

4. my-web-app ディレクトリのこの内容を、Amazon S3 入力バケットまたは AWS CodeCommit、GitHub、または Bitbucket リポジトリにアップロードします。

   Important

   (root directory name) または *(root directory name)*/my-web-app をアップロードしないでください。アップロードするのは、*(root directory name)*/my-web-app の中のディレクトリとファイルだけです。
   Amazon S3 入力バケットを使用している場合は、ディレクトリ構造とファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* または *(root directory name)*/my-web-app を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)*/my-web-app の中のディレクトリとファイルだけです。

ステップ A2: ビルドプロジェクトを作成し、ビルドを実行する

このステップでは、AWS CodeBuild コンソールを使用してプロジェクトを作成し、ビルドを実行します。

1. ビルド出力を保存する Amazon S3 出力バケットを作成または識別します。Amazon S3 入力バケットにソースコードを保存する場合、出力バケットは入力バケットと同じ AWS リージョンに存在する必要があります。
2. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.

   AWS リージョンセレクタを使用して、AWS CodeBuild をサポートするリージョンを選択し、Amazon S3 出力バケットが保存されているリージョンに合わせます。
3. ビルドプロジェクトを作成し、ビルドを実行します。詳細については、ビルドプロジェクトの作成 (コンソール) (p. 146) および ビルドの実行 (コンソール) (p. 167) を参照してください。これらの設定を除いて、すべての設定をデフォルト値のままにします。

   - [環境: ビルド方法] では、
     - [環境イメージ] で、[AWS CodeBuild によって管理されたイメージの使用] を選択します。
     - [Operating system] で、[Ubuntu] を選択します。
     - [Runtime] で、[Java] を選択します。
     - [バージョン] で、aws/codebuild/java:openjdk-8 を選択します。
   - [アーティファクト: このビルドプロジェクトからアーティファクトを配置する場所] では、
     - [アーティファクト名] には、覚えやすいビルド出力ファイル名を入力します。.zip 拡張子を含めます。
   - [詳細設定の表示] では、
     - [アーティファクトのパッケージ化] では、[Zip] を選択します。

ステップ A3: アプリケーションと環境を作成してデプロイする

このステップでは、Elastic Beanstalk コンソールを使用してアプリケーションと環境を作成します。環境の作成の一環として、前の手順のビルド出力を環境にデプロイします。

1. Elastic Beanstalk コンソール (https://console.aws.amazon.com/elasticbeanstalk) を開きます。

AWS リージョンセレクタを使用して、Amazon S3 出力バケットが保存されているリージョンと一致するリージョンを選択します。

2. Elastic Beanstalk アプリケーションを作成します。詳細については、「AWS Elastic Beanstalk アプリケーションの管理と設定」を参照してください。
3. このアプリケーション用の Elastic Beanstalk 環境を作成します。詳細については、「新しい環境の作成ウィザード」を参照してください。これらの設定を除いて、すべての設定をデフォルト値のままにします。

   - [プラットフォーム] では、[Tomcat] を選択します。
   - [Application code] では、[Upload your code] を選択して、[Upload] を選択します。[ソースコード元] では、[Public S3 URL] を選択し、出力バケットのビルド出力 ZIP ファイルの完全な URL を入力します。次に、アップロードを選択します。
4. Elastic Beanstalk がビルド出力を環境にデプロイすると、ウェブブラウザに結果が表示されます。インスタンスの環境 URL (例:`http://`*`my-environment-name`*`.`*`random-string`*`.`*`region-ID`*`.elasticbeanstalk.com`) へ移動します。ウェブブラウザにテキスト `Hello World!` が表示されます。

シナリオ B: AWS CodePipeline を使用して AWS CodeBuild を実行し、Elastic Beanstalk にデプロイする

このシナリオでは、作成するソースコードを手動で準備してアップロードする作業が完了します。その後、AWS CodePipeline コンソールを使用して、パイプラインと Elastic Beanstalk アプリケーションと環境を作成します。パイプラインを作成すると、AWS CodePipeline は自動的にソースコードを構築し、ビルド出力を環境にデプロイします。

ステップ B1: ビルドスペックファイルをソースコードに追加する

このステップでは、ソースコードの作成 (p. 65) で作成したコードにビルドスペックファイルを追加します。その後、ソースコードを Amazon S3 入力バケットまたは AWS CodeCommit または GitHub リポジトリにアップロードします。

1. 次の内容で、`buildspec.yml` というファイルを作成します ファイルを *`(root directory name)`*`/my-web-app` ディレクトリの中に保存します。

```
version: 0.2

phases:
  post_build:
    commands:
      - mvn package
artifacts:
  files:
    - '**/*'
  base-directory: 'target/my-web-app'
```

2. ファイル構造は次のようになります。

```
(root directory name)
     `-- my-web-app
           |-- src
           |     `-- main
           |           |-- resources
           |           `-- webapp
           |                 |-- WEB-INF
           |                 |     `-- web.xml
           |                 `-- index.jsp
```

```
            |-- buildpsec.yml
            `-- pom.xml
```

3. my-web-app ディレクトリのこの内容を、Amazon S3 入力バケットまたは AWS CodeCommit、GitHub、または Bitbucket リポジトリにアップロードします。

   Important

   (root directory name) または *(root directory name)*/my-web-app をアップロードしないでください。アップロードするのは、*(root directory name)*/my-web-app の中のディレクトリとファイルだけです。
   Amazon S3 入力バケットを使用している場合は、ディレクトリ構造とファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* または *(root directory name)*/my-web-app を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)*/my-web-app の中のディレクトリとファイルだけです。

ステップ B2: パイプラインの作成とデプロイ

このステップでは、Elastic Beanstalk および AWS CodePipeline コンソールを使用して、パイプライン、アプリケーション、環境を作成します。パイプラインを作成して実行すると、AWS CodePipeline は AWS CodeBuild を使用してソースコードを作成し、Elastic Beanstalk を使用して環境にビルド出力をデプロイします。

1. Elastic Beanstalk、AWS CodePipeline、および AWS CodeBuild がユーザーのための作業に使用できるサービスロールを作成または識別します。詳細については、「前提条件 (p. 127)」を参照してください。
2. AWS CodePipeline コンソール (https://console.aws.amazon.com/codepipeline/) を開きます。

   AWS リージョンセレクタを使用して AWS CodeBuild をサポートするリージョンを選択し、ソースコードを Amazon S3 入力バケットに保存する場合は、入力バケットが保存されているリージョンと一致するリージョンを選択します。
3. パイプラインを作成します。詳細については、AWS CodeBuild を使用するパイプラインを作成する (AWS CodePipeline コンソール) (p. 128) を参照してください。これらの設定を除いて、すべての設定をデフォルト値のままにします。

   - [ステップ 3: ビルド] では、[プロジェクトの設定] で、[新しいビルドプロジェクトを作成] を選択します。[環境: ビルド方法] では、
     - [環境イメージ] で、[AWS CodeBuild によって管理されたイメージの使用] を選択します。
     - [Operating system] で、[Ubuntu] を選択します。
     - [Runtime] で、[Java] を選択します。
     - [バージョン] で、aws/codebuild/java:openjdk-8 を選択します。
   - [ステップ 4: ベータ] では、[デプロイプロバイダ] で [AWS Elastic Beanstalk] を選択します。
     - アプリケーションでは、[Elastic Beanstalk に新規作成] リンクを選択します。これにより、Elastic Beanstalk コンソールが開きます。詳細については、「AWS Elastic Beanstalk アプリケーションの管理と設定」を参照してください。アプリケーションを作成したら、AWS CodePipeline コンソールに戻り、作成したばかりのアプリケーションを選択します。
     - 環境では、[Elastic Beanstalk に新規作成] リンクを選択します。これにより、Elastic Beanstalk コンソールが開きます。詳細については、「新しい環境の作成ウィザード」を参照してください。1 つの設定を除くすべての設定をデフォルト値のままにしておきます。[プラットフォーム] では、[Tomcat] を選択します。環境を作成したら、AWS CodePipeline コンソールに戻り、作成したばかりの環境を選択します。
4. パイプラインが正常に実行されたら、ウェブブラウザで結果を表示できます。インスタンスの環境 URL (例:http://*my-environment-name*.*random-string*.*region-*

ID.elasticbeanstalk.com) へ移動します。ウェブブラウザにテキスト Hello World! が表示されます。

これで、ソースコードを変更して元の Amazon S3 入力バケットまたは AWS CodeCommit、GitHub、または Bitbucket リポジトリに変更をアップロードするたびに、AWS CodePipeline によって変更が検出され、パイプラインが再度実行されます。これにより、AWS CodeBuild は自動的にコードを再構築し、Elastic Beanstalk に再構築された出力を環境を自動的にデプロイさせます。

シナリオ C: Elastic Beanstalk コマンドラインインターフェイス (EB CLI) を使用して、AWS CodeBuild を実行し、Elastic Beanstalk 環境にデプロイする

このシナリオでは、作成するソースコードを手動で準備してアップロードする作業が完了します。次に、EB CLI を実行して Elastic Beanstalk アプリケーションと環境を作成し、AWS CodeBuild を使用してソースコードを作成した後、ビルド出力を環境にデプロイします。詳細については、『AWS Elastic Beanstalk 開発者ガイド』の「AWS CodeBuild での EB CLI の使用」を参照してください。

ステップ C1: ソースコードにファイルを追加する

このステップでは、Elastic Beanstalk 設定ファイルとビルドスペックファイルを ソースコードの作成 (p. 65) で作成したコードに追加します。また、ビルドスペックファイルのサービスロールを作成または識別します。

1. Elastic Beanstalk や EB CLI がユーザーのために使用できるサービスロールを作成または指定します。詳細については、AWS CodeBuild サービスロールの作成 (p. 185) を参照してください。
2. .ebextensions という名前のサブディレクトリを、*(root directory name)*/my-web-app ディレクトリ内に作成します。.ebextensions サブディレクトリに、fix-path.config という名前のファイルをこのコンテンツで作成します。

```
container_commands:
  fix_path:
    command: "unzip my-web-app.war 2>&1 > /var/log/my_last_deploy.log"
```

3. 次の内容で、buildspec.yml というファイルを作成します ファイルを *(root directory name)*/my-web-app ディレクトリの中に保存します。

```
version: 0.2

phases:
  post_build:
    commands:
      - mvn package
      - mv target/my-web-app.war my-web-app.war
artifacts:
  files:
    - my-web-app.war
    - .ebextensions/**/*
eb_codebuild_settings:
  CodeBuildServiceRole: my-service-role-name
  ComputeType: BUILD_GENERAL1_SMALL
  Image: aws/codebuild/java:openjdk-8
  Timeout: 60
```

前のコードでは、*my-service-role-name* を、以前に作成または識別したサービスロールの名前で置き換えます。

4. ファイル構造は次のようになります。

```
(root directory name)
     `-- my-web-app
           |-- .ebextensions
           |     `-- fix-path.config
           |-- src
           |     `-- main
           |           |-- resources
           |           `-- webapp
           |                 |-- WEB-INF
           |                 |     `-- web.xml
           |                 `-- index.jsp
           |-- buildpsec.yml
           `-- pom.xml
```

ステップ C2: EB CLI のインストールと実行

1. EB CLI をまだインストールしていない場合は、ソースコードを作成した同じコンピュータまたはインスタンスにインストールして設定します。詳細については、「Elastic Beanstalk コマンドラインインターフェイスのインストール (EB CLI)」および「EB CLI の設定」を参照してください。
2. コンピュータまたはインスタンスのコマンドラインまたはターミナルから、cd コマンドなどを実行して、*(root directory name)*/my-web-app ディレクトリに切り替えます。eb init コマンドを実行して、EB CLI を設定します。

   ```
   eb init
   ```

   プロンプトが表示されたら:

   - AWS CodeBuild がサポートされている AWS リージョンを選択し、Elastic Beanstalk アプリケーションおよび環境を作成する場所と一致させます。
   - Elastic Beanstalk アプリケーションを作成し、アプリケーションの名前を入力します。
   - `Tomcat` プラットフォームを選択します。
   - `Tomcat 8 Java 8` バージョンを選択します。
   - SSH を使用して環境のインスタンスへのアクセスを設定するかどうかを選択します。
3. 同じディレクトリから eb create コマンドを実行して、Elastic Beanstalk 環境を作成します。

   ```
   eb create
   ```

   プロンプトが表示されたら:

   - 新しい環境の名前を入力するか、提案された名前をそのまま使用します。
   - 環境の DNS CNAME プレフィックスを入力するか、推奨値をそのまま使用します。
   - このサンプルでは、Classic Load Balancer タイプを受け入れます。
4. eb create コマンドを実行すると、EB CLI は次のことを行います。

   1. ソースコードから ZIP ファイルを作成し、ZIP ファイルをアカウントの Amazon S3 バケットにアップロードします。
   2. Elastic Beanstalk アプリケーションとアプリケーションバージョンを作成します。
   3. AWS CodeBuild プロジェクトを作成します。
   4. 新しいプロジェクトに基づいてビルドを実行します。
   5. ビルドが完了した後にプロジェクトを削除します。
   6. Elastic Beanstalk 環境を作成します。

7. ビルドの出力を環境にデプロイします。

5. EB CLI がビルド出力を環境にデプロイすると、ウェブブラウザに結果が表示されます。インスタンスの環境 URL (例:`http://my-environment-name.random-string.region-ID.elasticbeanstalk.com`) へ移動します。ウェブブラウザにテキスト `Hello World!` が表示されます。

必要に応じて、ソースコードを変更して、同じディレクトリから eb deploy コマンドを実行できます。EB CLI は eb create コマンドと同じ手順を実行しますが、新しい環境を作成する代わりに既存の環境にビルド出力をデプロイします。

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild コードベースのサンプル

これらのコードベースのサンプルを参考にして AWS CodeBuild を試してください。

名前	説明
C++ サンプル (p. 73)	C++ を使用して、単一の .out ファイルを出力します。
Go サンプル (p. 75)	Go を使用して、単一のバイナリファイルを出力します。
Maven サンプル (p. 77)	Apache Maven を使用して単一の JAR ファイルを生成します。
Node.js サンプル (p. 79)	Mocha を使用して、コード内の内部変数に特定の文字列値が含まれているかどうかをテストします。単一の .js ファイルを生成します。
Python サンプル (p. 81)	Python を使用して、コード内の内部変数が特定の文字列値に設定されているかどうかをテストします。単一の .py ファイルを生成します。
Ruby サンプル (p. 83)	RSpec を使用して、コード内の内部変数が特定の文字列値に設定されているかどうかをテストします。単一の .rb ファイルを生成します。
Scala サンプル (p. 86)	sbt を使用して、単一の JAR ファイルを生成します。
Java サンプル (p. 89)	Apache Maven を使用して単一の WAR ファイルを生成します。
Linux における .NET Core のサンプル (p. 91)	.NET Core を使用して C# で書かれたコードから実行可能ファイルをビルドします。

AWS CodeBuild の C++ Hello World サンプル

この C++ サンプルは `hello.out` という単一のバイナリファイルをビルド出力として生成します。

> **Important**
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

トピック

サンプルの実行

このサンプルを実行するには

1. このトピックの「ディレクトリ構造」セクションと「ファイル」セクションの説明に従ってファイルを作成し、Amazon S3 入力バケットにアップロードするか、AWS CodeCommit、GitHub、または Bitbucket の各リポジトリにアップロードします。

   > **Important**
   >
   > *(root directory name)* をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
   > Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。

2. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルドプロジェクトを作成して、ビルドを実行し、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-c-plus-plus-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/CPlusPlusSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "CPlusPlusOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/ubuntu-base:14.04",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
```

```
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
4. *CPlusPlusOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、 ファイルの内容を抽出します。展開したコンテンツから、hello.out ファイルを取得します。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
    |-- buildspec.yml
    `-- hello.cpp
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

buildspec.yml (*(root directory name)* 内)

```
version: 0.2

phases:
  install:
    commands:
      - apt-get update -y
      - apt-get install -y build-essential
  build:
    commands:
      - echo Build started on `date`
      - echo Compiling the C++ code...
      - g++ hello.cpp -o hello.out
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - hello.out
```

hello.cpp (*(root directory name)* 内)

```
#include <iostream>

int main()
{
  std::cout << "Hello, World!\n";
}
```

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。

- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の Go Hello World サンプル

この Go サンプルは `hello` という単一のバイナリファイルをビルド出力として生成します。

> Important
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

トピック

サンプルの実行

このサンプルを実行するには

1. このトピックの「ディレクトリ構造」セクションと「ファイル」セクションの説明に従ってファイルを作成し、Amazon S3 入力バケットにアップロードするか、AWS CodeCommit、GitHub、または Bitbucket の各リポジトリにアップロードします。

   > Important
   >
   > *(root directory name)* をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
   > Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。

2. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルドプロジェクトを作成して、ビルドを実行し、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-go-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/GoSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "GoOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/golang:1.7.3",
```

```
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
4. *GoOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、ファイルの内容を抽出します。展開したコンテンツから、hello ファイルを取得します。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
    |-- buildspec.yml
    `-- hello.go
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

buildspec.yml (*(root directory name)* 内)

```
version: 0.2

phases:
  build:
    commands:
      - echo Build started on `date`
      - echo Compiling the Go code...
      - go build hello.go
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - hello
```

hello.go (*(root directory name)* 内)

```
package main
import "fmt"

func main() {
  fmt.Println("hello world")
  fmt.Println("1+1 =", 1+1)
  fmt.Println("7.0/3.0 =", 7.0/3.0)
  fmt.Println(true && false)
  fmt.Println(true || false)
  fmt.Println(!true)
}
```

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。

- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の「5 分で Maven」サンプル

Maven サンプルは `my-app-1.0-SNAPSHOT.jar` という単一の JAR ファイルをビルド出力として生成します。 このサンプルは、Apache Maven ウェブサイトの「5 分で Maven」トピックに基づいています。

> Important
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

サンプルの実行

このサンプルを実行するには。

1. Maven をダウンロードし、インストールします。詳細については、Apache Maven ウェブサイトの「Apache Maven のダウンロード」および「Apache Maven のインストール」を参照してください。
2. ローカルコンピュータまたはインスタンスの空のディレクトリに切り替えて、この Maven コマンドを実行します。

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

成功すると、このディレクトリ構造とファイルが作成されます。

```
(root directory name)
     `-- my-app
           |-- pom.xml
           `-- src
                 |-- main
                 |     `-- java
                 |           `-- com
                 |                 `-- mycompany
                 |                       `-- app
                 |                             `-- App.java
                 `-- test
                       `-- java
                             `-- com
                                   `-- mycompany
                                         `-- app
                                               `-- AppTest.java
```

3. このコンテンツを含むファイルを作成します。ファイルに `buildspec.yml` という名前を付け、それを *(root directory name)*/my-app ディレクトリに追加します。

```
version: 0.2

phases:
  build:
    commands:
      - echo Build started on `date`
      - mvn test
```

```
  post_build:
    commands:
      - echo Build completed on `date`
      - mvn package
artifacts:
  files:
    - target/my-app-1.0-SNAPSHOT.jar
```

完了したら、ディレクトリ構造およびファイルは次のようになります。

```
(root directory name)
     `-- my-app
           |-- buildspec.yml
           |-- pom.xml
           `-- src
                 |-- main
                 |     `-- java
                 |           `-- com
                 |                 `-- mycompany
                 |                       `-- app
                 |                             `-- App.java
                 `-- test
                       `-- java
                             `-- com
                                   `-- mycompany
                                         `-- app
                                               ` -- AppTest.java
```

4. my-app ディレクトリのこの内容を、Amazon S3 入力バケットまたは AWS CodeCommit、GitHub、または Bitbucket リポジトリにアップロードします。

   Important

   (root directory name) または *(root directory name)*/my-app をアップロードしないでください。アップロードするのは、*(root directory name)*/my-app の中のディレクトリとファイルだけです。
   Amazon S3 入力バケットを使用している場合は、ディレクトリ構造とファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* または *(root directory name)*/my-app を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)*/my-app の中のディレクトリとファイルだけです。

5. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従ってビルドプロジェクトを作成し、ビルドを実行して、関連するビルド情報を表示します。

AWS CLI を使用してビルドプロジェクトを作成する場合、create-project コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-maven-in-5-minutes-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/MavenIn5MinutesSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "MavenIn5MinutesOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/java:openjdk-8",
```

```
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

6. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
7. *MavenIn5MinutesOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、*MavenIn5MinutesOutputArtifact*.zip ファイルの内容を抽出します。抽出された内容で [target] フォルダを開き、my-app-1.0-SNAPSHOT.jar ファイルを取得します。

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の Node.js Hello World サンプル

この Node.js サンプルは、コードの内部変数が Hello という文字列で始まるかどうかをテストします。これにより、ビルド出力として HelloWorld.js という名前の 1 つのファイルが生成されます。

> Important
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

トピック

サンプルの実行

このサンプルを実行するには

1. ローカルコンピュータまたはインスタンスで、このトピックの「ディレクトリ構造」と「ファイル」の各セクションにある説明に従ってファイルを作成してから、Amazon S3 入力バケットまたは AWS CodeCommit、GitHub、または Bitbucket リポジトリにアップロードします。

   > Important
   >
   > *(root directory name)* をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
   > Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。

2. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従ってビルドプロジェクトを作成し、ビルドを実行して、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`start-build` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-nodejs-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/NodeJSSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "NodeJSOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/nodejs:6.3.1",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
4. *NodeJSOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、 ファイルの内容を抽出します。展開したコンテンツから、`HelloWorld.js` ファイルを取得します。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
    |-- buildspec.yml
    `-- HelloWorld.js
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

`buildspec.yml` (*(root directory name)* 内)

```
version: 0.2

phases:
  install:
    commands:
      - echo Installing Mocha...
      - npm install -g mocha
  pre_build:
    commands:
      - echo Installing source NPM dependencies...
      - npm install unit.js
  build:
    commands:
      - echo Build started on `date`
```

```
      - echo Compiling the Node.js code
      - mocha HelloWorld.js
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - HelloWorld.js
```

`HelloWorld.js` (*(root directory name)* 内)

```
var test = require('unit.js');
var str = 'Hello, world!';

test.string(str).startsWith('Hello');

if (test.string(str).startsWith('Hello')) {
  console.log('Passed');
}
```

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の Python Hello World サンプル

この Python サンプルテストでは、コードの内部変数が `Hello world!` という文字列を含んでいるかをテストします。これにより、ビルド出力として `HelloWorld.py` という名前の 1 つのファイルが生成されます。

> **Important**
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

トピック

サンプルの実行

このサンプルを実行するには

1. このトピックの「ディレクトリ構造」セクションと「ファイル」セクションの説明に従ってファイルを作成し、Amazon S3 入力バケットにアップロードするか、AWS CodeCommit、GitHub、または Bitbucket の各リポジトリにアップロードします。

Important

(root directory name) をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。

2. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルドプロジェクトを作成して、ビルドを実行し、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-python-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/PythonSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "PythonOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/python:3.5.2",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
4. *PythonOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、 ファイルの内容を抽出します。展開したコンテンツから、`HelloWorld.py` ファイルを取得します。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
    |-- buildspec.yml
    |-- HelloWorld.py
    `-- HelloWorld_tst.py
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

`buildspec.yml` (*(root directory name)* 内)

```
version: 0.2
```

```
phases:
  build:
    commands:
      - echo Build started on `date`
      - echo Compiling the Python code...
      - python HelloWorld_tst.py
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - HelloWorld.py
```

`HelloWorld.py` (*`(root directory name)`* 内)

```
class HelloWorld:
  def __init__(self):
    self.message = 'Hello world!'
```

`HelloWorld_tst.py` (*`(root directory name)`* 内)

```
import unittest
from HelloWorld import HelloWorld

class MyTestCase(unittest.TestCase):
  def test_default_greeting_set(self):
    hw = HelloWorld()
    self.assertEqual(hw.message, 'Hello world!')

if __name__ == '__main__':
  unittest.main()
```

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の Ruby Hello World サンプル

この Ruby サンプルテストでは、コードの内部変数が `Hello, world!` という文字列を含んでいるかをテストします。これにより、ビルド出力として `HelloWorld.rb` という名前の 1 つのファイルが生成されます。

> **Important**
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

トピック

- サンプルの実行 (p. 84)

サンプルの実行

このサンプルを実行するには

1. このトピックの「ディレクトリ構造」セクションと「ファイル」セクションの説明に従ってファイルを作成し、Amazon S3 入力バケットにアップロードするか、AWS CodeCommit、GitHub、または Bitbucket の各リポジトリにアップロードします。

   Important

   (root directory name) をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
   Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。
2. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従ってビルドプロジェクトを作成し、ビルドを実行して、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-ruby-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/RubySample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "RubyOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/ruby:2.3.1",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
4. *RubyOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、ファイルの内容を抽出します。展開したコンテンツから、`HelloWorld.rb` ファイルを取得します。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
```

```
    |-- buildspec.yml
    |-- HelloWorld.rb
    `-- HelloWorld_spec.rb
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

`buildspec.yml` (*(root directory name)* 内)

```
version: 0.2

phases:
  install:
    commands:
      - echo Installing RSpec...
      - gem install rspec
  build:
    commands:
      - echo Build started on `date`
      - echo Compiling the Ruby code...
      - rspec HelloWorld_spec.rb
  post_build:
    commands:
      - echo Build completed on `date`
artifacts:
  files:
    - HelloWorld.rb
```

`HelloWorld.rb` (*(root directory name)* 内)

```
class HelloWorld
  def say_hello()
    return 'Hello, world!'
  end
end
```

`HelloWorld_spec.rb` (*(root directory name)* 内)

```
require './HelloWorld'

describe HelloWorld do
  context "When testing the HelloWorld class" do
    it "The say_hello method should return 'Hello World'" do
      hw = HelloWorld.new
      message = hw.say_hello
      puts 'Succeed' if expect(message).to eq "Hello, world!"
    end
  end
end
```

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild のための Scala Hello World サンプル

この Scala サンプルは `core_2.11-1.0.0.jar` という単一の JAR ファイルをビルド出力として生成します。

> **Important**
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

トピック

サンプルの実行

このサンプルを実行するには

1. Scala プロジェクトのビルドツールである sbt を含む Docker イメージを特定します。 互換性のある Docker イメージを検索するには、Docker Hub で **`sbt`** を検索します。
2. このトピックの「ディレクトリ構造」セクションと「ファイル」セクションで説明しているようにディレクトリ構造とファイルを作成し、Amazon S3 入力バケット、AWS CodeCommit、GitHub、または Bitbucket リポジトリにアップロードします。

   > **Important**
   >
   > *`(root directory name)`* をアップロードしないでください。アップロードするのは、*`(root directory name)`* 内のディレクトリとファイルのみです。
   > Amazon S3 入力バケットを使用している場合は、ディレクトリ構造とファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*`(root directory name)`* を ZIP ファイルに追加しないでください。追加するのは、*`(root directory name)`* 内のディレクトリとファイルのみです。
3. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従ってビルドプロジェクトを作成し、ビルドを実行して、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-scala-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/ScalaSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "ScalaOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "scala-image-ID",
```

```
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

4. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
5. *ScalaOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、ファイルの内容を抽出します。抽出された内容で [core/target/scala-2.11] フォルダを開き、core_2.11-1.0.0.jar ファイルを取得します。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
    |-- buildspec.yml
    |-- core
    |     `-- src
    |           `-- main
    |                 `-- scala
    |                       `-- Test.scala
    |-- macros
    |     `-- src
    |           `-- main
    |                 `-- scala
    |                       `-- Macros.scala
    `-- project
          |-- Build.scala
          `-- build.properties
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

buildspec.yml (*(root directory name)* 内)

```
version: 0.2

phases:
  build:
    commands:
      - echo Build started on `date`
      - echo Run the test and package the code...
      - sbt run
  post_build:
    commands:
      - echo Build completed on `date`
      - sbt package
artifacts:
  files:
    - core/target/scala-2.11/core_2.11-1.0.0.jar
```

Test.scala (*(root directory name)*/core/src/main/scala 内)

```
object Test extends App {
  Macros.hello
}
```

Macros.scala (*(root directory name)*/macros/src/main/scala 内)

```
import scala.language.experimental.macros
import scala.reflect.macros.Context

object Macros {
  def impl(c: Context) = {
    import c.universe._
    c.Expr[Unit](q"""println("Hello World")""")
  }

  def hello: Unit = macro impl
}
```

Build.scala (*(root directory name)*/project 内)

```
import sbt._
import Keys._

object BuildSettings {
  val buildSettings = Defaults.defaultSettings ++ Seq(
    organization := "org.scalamacros",
    version := "1.0.0",
    scalaVersion := "2.11.8",
    crossScalaVersions := Seq("2.10.2", "2.10.3", "2.10.4", "2.10.5", "2.10.6", "2.11.0",
 "2.11.1", "2.11.2", "2.11.3", "2.11.4", "2.11.5", "2.11.6", "2.11.7", "2.11.8"),
    resolvers += Resolver.sonatypeRepo("snapshots"),
    resolvers += Resolver.sonatypeRepo("releases"),
    scalacOptions ++= Seq()
  )
}

object MyBuild extends Build {
  import BuildSettings._

  lazy val root: Project = Project(
    "root",
    file("."),
    settings = buildSettings ++ Seq(
      run <<= run in Compile in core)
  ) aggregate(macros, core)

  lazy val macros: Project = Project(
    "macros",
    file("macros"),
    settings = buildSettings ++ Seq(
      libraryDependencies <+= (scalaVersion)("org.scala-lang" % "scala-reflect" % _),
      libraryDependencies := {
        CrossVersion.partialVersion(scalaVersion.value) match {
          // if Scala 2.11+ is used, quasiquotes are available in the standard distribution
          case Some((2, scalaMajor)) if scalaMajor >= 11 =>
            libraryDependencies.value
          // in Scala 2.10, quasiquotes are provided by macro paradise
          case Some((2, 10)) =>
            libraryDependencies.value ++ Seq(
              compilerPlugin("org.scalamacros" % "paradise" % "2.1.0-M5" cross
 CrossVersion.full),
              "org.scalamacros" %% "quasiquotes" % "2.1.0-M5" cross CrossVersion.binary)
        }
      }
    )
  )

  lazy val core: Project = Project(
    "core",
    file("core"),
```

```
    settings = buildSettings
  ) dependsOn(macros)
}
```

build.properties (*(root directory name)*/project 内)

```
sbt.version=0.13.7
```

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の WAR Hello World サンプル

この Maven サンプルは、my-web-app.war という名前の単一のウェブアプリケーションアーカイブ (WAR) ファイルをビルド出力として生成します。

> Important
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

サンプルの実行

このサンプルを実行するには。

1. Maven をダウンロードし、インストールします。詳細については、Apache Maven ウェブサイトの「Apache Maven のダウンロード」および「Apache Maven のインストール」を参照してください。
2. ローカルコンピュータまたはインスタンスの空のディレクトリに切り替えて、この Maven コマンドを実行します。

   ```
   mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-web-app -
   DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
   ```

   成功すると、このディレクトリ構造とファイルが作成されます。

   ```
   (root directory name)
        `-- my-web-app
              |-- pom.xml
              `-- src
                    `-- main
                          |-- resources
                          `-- webapp
                                |-- WEB-INF
                                |     `-- web.xml
                                `-- index.jsp
   ```

3. このコンテンツを含むファイルを作成します。ファイルに buildspec.yml という名前を付け、それを *(root directory name)*/my-web-app ディレクトリに追加します。

```
version: 0.2

phases:
  post_build:
    commands:
      - echo Build completed on `date`
      - mvn package
artifacts:
  files:
    - target/my-web-app.war
  discard-paths: yes
```

完了したら、ディレクトリ構造およびファイルは次のようになります。

```
(root directory name)
     `-- my-web-app
           |-- buildspec.yml
           |-- pom.xml
           `-- src
                 `-- main
                       |-- resources
                       `-- webapp
                             |-- WEB-INF
                             |     `-- web.xml
                             `-- index.jsp
```

4. my-web-app ディレクトリの内容を、Amazon S3 入力バケットまたは AWS CodeCommit、GitHub、または Bitbucket リポジトリにアップロードします。

   Important

   (root directory name) または *(root directory name)*/my-web-app をアップロードしないでください。アップロードするのは、*(root directory name)*/my-web-app の中のディレクトリとファイルだけです。
   Amazon S3 入力バケットを使用している場合は、ディレクトリ構造とファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* または *(root directory name)*/my-web-app を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)*/my-web-app の中のディレクトリとファイルだけです。たとえば、ZIP ファイルには、次のディレクトリとファイルが含まれている必要があります。

   ```
   WebArchiveHelloWorldSample.zip
        |-- buildpsec.yml
        |-- pom.xml
        `-- src
              `-- main
                    |-- resources
                    `-- webapp
                          |-- WEB-INF
                          |     `-- web.xml
                          `-- index.jsp
   ```

5. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルドプロジェクトを作成し、ビルドを実行し、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、create-project コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

   ```
   {
     "name": "sample-web-archive-project",
   ```

```
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-
bucket/WebArchiveHelloWorldSample.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "WebArchiveHelloWorldOutputArtifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/java:openjdk-8",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

6. ビルド出力アーティファクトを取得するには、Amazon S3 出力バケットを開きます。
7. *WebArchiveHelloWorldOutputArtifact*.zip ファイルをローカルコンピュータまたはインスタンスへダウンロードし、 ファイルの内容を抽出します。抽出された内容で [target] フォルダを開き、my-web-app.war ファイルを取得します。

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild の Linux における .NET Core のサンプル

このサンプルでは、.NET Core が実行されている AWS CodeBuild ビルド環境を使用して、C# で書かれたコードから実行可能ファイルをビルドします。

> Important
>
> このサンプルを実行すると、AWS アカウントに課金される場合があります。これには、AWS CodeBuild および Amazon S3、AWS KMS、CloudWatch Logs に関連する AWS リソースおよびアクションの料金が含まれます。詳細については、「AWS CodeBuild 料金表」、「Amazon S3 料金表」、「AWS Key Management Service 料金表」、「Amazon CloudWatch 料金表」を参照してください。

サンプルの実行

このサンプルを実行するには

1. このトピックの「ディレクトリ構造」セクションと「ファイル」セクションの説明に従ってファイルを作成し、Amazon S3 入力バケットにアップロードするか、AWS CodeCommit、GitHub、または Bitbucket の各リポジトリにアップロードします。

Important

(root directory name) をアップロードしないでください。アップロードするのは、*(root directory name)* 内のファイルのみです。
Amazon S3 入力バケットを使用している場合は、ファイルを含む ZIP ファイルを作成してから、入力バケットにアップロードしてください。*(root directory name)* を ZIP ファイルに追加しないでください。追加するのは、*(root directory name)* 内のファイルのみです。

2. 「AWS CodeBuild を直接実行する (p. 114)」の手順に従って、ビルドプロジェクトを作成して、ビルドを実行し、関連するビルド情報を表示します。

   AWS CLI を使用してビルドプロジェクトを作成する場合、`create-project` コマンドへの JSON 形式の入力は次のようになります。(プレースホルダは独自の値に置き換えてください。)

```
{
  "name": "sample-dot-net-core-project",
  "source": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-input-bucket/windows-dotnetcore.zip"
  },
  "artifacts": {
    "type": "S3",
    "location": "codebuild-region-ID-account-ID-output-bucket",
    "packaging": "ZIP",
    "name": "dot-net-core-output-artifact.zip"
  },
  "environment": {
    "type": "LINUX_CONTAINER",
    "image": "aws/codebuild/dot-net:core-2.0",
    "computeType": "BUILD_GENERAL1_SMALL"
  },
  "serviceRole": "arn:aws:iam::account-ID:role/role-name",
  "encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. ビルドの出力アーティファクトを取得するには、Amazon S3 出力バケットで、*dot-net-core-output-artifact*`.zip` ファイルをローカルコンピュータまたはインスタンスにダウンロードします。コンテンツを抽出し、実行可能ファイル `HelloWorldSample.dll` に移動します。このファイルは `bin\Debug\netcoreapp2.0` ディレクトリにあります。

ディレクトリ構造

このサンプルのディレクトリ構造は次のとおりとします。

```
(root directory name)
    |-- buildspec.yml
    |-- HelloWorldSample.csproj
    `-- Program.cs
```

ファイル

このサンプルで使用するファイルは以下のとおりです。

`buildspec.yml` (*(root directory name)* 内)

```
version: 0.2

phases:
```

```
  build:
    commands:
      - dotnet restore
      - dotnet build
artifacts:
  files:
    - bin/Debug/netcoreapp2.0/*
```

`HelloWorldSample.csproj` (*(root directory name)* 内)

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>
</Project>
```

`Program.cs` (*(root directory name)* 内)

```
using System;

namespace HelloWorldSample {
  public static class Program {
    public static void Main() {
      Console.WriteLine("Hello, World!");
    }
  }
}
```

関連リソース

- AWS CodeBuild の開始方法の詳細については、「AWS CodeBuild の使用開始 (p. 4)」を参照してください。
- AWS CodeBuild の問題の詳しいトラブルシューティング方法については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。
- AWS CodeBuild の制限の詳細については、「AWS CodeBuild の制限 (p. 217)」を参照してください。

AWS CodeBuild のビルドを計画する

AWS CodeBuild を使用してビルドを実行する前に、次の質問に答える必要があります。

1. ソースコードはどこにありますか? AWS CodeBuild は現在、次のソースコードのリポジトリプロバイダからのビルドをサポートしています。ソースコードには、ビルド仕様ファイルが含まれている必要があります。またはビルド仕様をビルドプロジェクト定義の一部として宣言する必要があります。ビルド仕様は、AWS CodeBuild がビルドを実行するために使用する YAML 形式のビルドコマンドと関連設定のコレクションです。

リポジトリプロバイダ	必須	ドキュメント
AWS CodeCommit	リポジトリ名. (オプション) ソースコードに関連付けられているコミット ID。	「AWS CodeCommit ユーザーガイド」の以下のトピックを参照してください。 AWS CodeCommit リポジトリの作成 AWS CodeCommit でのコミットの作成
Amazon S3	バケット名を入力します。 ソースコードを含むビルド入力 ZIP ファイルに対応するオブジェクト名。 (オプション) ビルド入力 ZIP ファイルに関連付けられているバージョン ID。	「Amazon S3 入門ガイド」の以下のトピックを参照してください。 バケットの作成 バケットにオブジェクトを追加
GitHub	リポジトリ名. (オプション) ソースコードに関連付けられているコミット ID。	GitHub のヘルプウェブサイトでこのトピックを参照してください。 Repo の作成
Bitbucket	リポジトリ名. (オプション) ソースコードに関連付けられているコミット ID。	Bitbucket Cloud のドキュメントウェブサイトでこのトピックを参照してください。 リポジトリの作成

2. どのビルドコマンドを、どのような順番で実行する必要がありますか?デフォルトでは、AWS CodeBuild は指定したプロバイダーからビルド入力をダウンロードし、指定したバケットにビルド出力をアップロードします。ビルド仕様を使用して、ダウンロードされたビルド入力を予想されるビルド出力に変換する方法を指示します。詳細については、「ビルドスペックリファレンス (p. 95)」を参照してください。

3. ビルドを実行するためにどのランタイムとツールが必要ですか? たとえば、Java、Ruby、Python、Node.js を構築していますか?ビルドでは、Maven、Ant または、Java、Ruby、Python のコンパイラが必要ですか?ビルドでは、Git、AWS CLI、またはその他のツールが必要ですか?

   AWS CodeBuild は、Docker イメージを使用するビルド環境でビルドを実行します。これらの Docker イメージを AWS CodeBuild でサポートされているリポジトリタイプに保存する必要があります。これらには、AWS CodeBuild Docker イメージリポジトリ、Docker ハブ、および Amazon Elastic Container Registry (Amazon ECR) が含まれます。AWS CodeBuild Docker イメージリポジトリの詳細については、「AWS CodeBuild に用意されている Docker イメージ (p. 103)」を参照してください。
4. AWS CodeBuild によって自動的に提供されない AWS リソースが必要ですか?その場合は、それらのリソースにはどのセキュリティポリシーが必要ですか? たとえば、AWS CodeBuild サービスロールを変更して、AWS CodeBuild がそれらのリソースを操作できるようにする必要が生じることがあります。
5. AWS CodeBuild を VPC と連携させますか?その場合は、VPC ID、サブネット ID、および VPC 設定のセキュリティグループ ID が必要です。詳細については、「Amazon Virtual Private Cloud で AWS CodeBuild を使用する (p. 115)」を参照してください。

これらの質問に答えると、ビルドを正常に実行するために必要な設定とリソースがあるはずです。ビルドを実行するには、次の操作を実行できます。

- AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用します。詳細については、「AWS CodeBuild を直接実行する (p. 114)」を参照してください。
- AWS CodePipeline でパイプラインを作成または指定し、AWS CodeBuild が自動的にコードをテストし、ビルドを実行 (またはその両方) するよう指示するビルドまたはテストアクションを追加します。詳細については、「AWS CodeBuild で AWS CodePipeline を使用する (p. 126)」を参照してください。

AWS CodeBuild のビルド仕様に関するリファレンス

このトピックでは、ビルド仕様 (ビルドスペック) に関する重要なリファレンス情報を提供します。ビルド仕様は、AWS CodeBuild がビルドを実行するために使用する YAML 形式のビルドコマンドと関連設定のコレクションです。 ビルドスペックをソースコードの一部として含めることも、ビルドプロジェクトの作成時にビルドスペックを定義することもできます。ビルド仕様の仕組みについては、「AWS CodeBuild の詳細 (p. 3)」を参照してください。

トピック

ビルドスペックのファイル名とストレージの場所

ビルドスペックをソースコードの一部として含める場合、デフォルトのビルドスペックファイルの名前は `buildspec.yml` で、ソースディレクトリのルートに配置する必要があります。

また、デフォルトのビルドスペックファイルの名前と場所を変更することもできます。たとえば、以下のことが可能です。

- 同じリポジトリ内の異なるビルドに、`buildspec_debug.yml` や `buildspec_release.yml` などの異なるビルドスペックファイルを使用する。

- `config/buildspec.yml` など、ソースディレクトリのルート以外の場所にビルドスペックファイルを保存する。

ビルドスペックファイルの名前に関係なく、ビルドプロジェクトには 1 つのビルドスペックしか指定できません。

デフォルトのビルドスペックファイルの名前、場所、またはその両方をオーバーライドするには、次のいずれかを実行します。

- AWS CLI の `create-project` または `update-project` コマンドを実行し、`buildspec` の値を、組み込みの環境変数 `CODEBUILD_SRC_DIR` の値を基準にした代替ビルドスペックファイルのパスに設定します。また、AWS SDK に含まれているプロジェクトの作成オペレーションで同等の処理を行うこともできます。詳細については、「ビルドプロジェクトを作成する (p. 146)」または「ビルドプロジェクトの設定を変更する (p. 161)」を参照してください。
- AWS CLI の `start-build` コマンドを実行し、`buildspecOverride` の値を、組み込みの環境変数 `CODEBUILD_SRC_DIR` の値を基準にした代替ビルドスペックファイルのパスに設定します。また、AWS SDK に含まれているビルドの開始オペレーションで同等の処理を行うこともできます。詳細については、「ビルドの実行 (p. 167)」を参照してください。
- AWS CloudFormation テンプレートで、`AWS::CodeBuild::Project` タイプのリソース `Source` の `BuildSpec` プロパティを、組み込みの環境変数 `CODEBUILD_SRC_DIR` の値を基準にした代替ビルドスペックファイルのパスに設定します。詳細については、AWS CloudFormation ユーザーガイド の「AWS CodeBuild プロジェクトソース」の「BuildSpec」プロパティを参照してください。

ビルドスペックの構文

ビルドスペックは YAML 形式で表現する必要があります。

ビルドスペックの構文は次のとおりです。

```
version: 0.2

env:
  variables:
    key: "value"
    key: "value"
  parameter-store:
    key: "value"
    key: "value"

phases:
  install:
    commands:
      - command
      - command
  pre_build:
    commands:
      - command
      - command
  build:
    commands:
      - command
      - command
  post_build:
    commands:
      - command
      - command
artifacts:
  files:
    - location
```

```
    - location
  discard-paths: yes
  base-directory: location
```

ビルドスペックには次のものが含まれています。

- version: 必要なマッピング。ビルドスペックのバージョンを表します。0.2 を使用することをお勧めします。

  Note

  バージョン 0.1 は引き続きサポートされます。ただし、可能な場合はバージョン 0.2 を使用することをお勧めします。詳細については、「ビルドスペックのバージョン (p. 102)」を参照してください。

- env: オプションのシーケンス。1 つ以上のカスタム環境変数の情報を表します。
  - variables: env を指定し、プレーンテキストでカスタム環境変数を定義する場合は必須です。##と#のスカラーのマッピングを含み、各マッピングはプレーンテキストで 1 つのカスタム環境変数を表します。##は、カスタム環境変数の名前で、#はその変数の値です。

    Important

    機密情報、特に AWS アクセスキー ID とシークレットアクセスキーを格納する場合には、環境変数を使用しないことを強くお勧めします。環境変数は、AWS CodeBuild コンソールや AWS CLI などのツールを使用してプレーンテキストで表示できます。機密情報については、このセクションの後半で説明するように、parameter-store マッピングを代わりに使用することをお勧めします。
    既存の環境変数は、設定した環境変数により置き換えられます。たとえば、Docker イメージに MY_VAR という名前で値が my_value の環境変数がすでに含まれていて、MY_VAR という名前で値が other_value の環境変数を設定した場合は、other_value が my_value に置き換わります。同様に、Docker イメージに PATH という名前で値が /usr/local/sbin:/usr/local/bin の環境変数がすでに含まれていて、PATH という名前で値が $PATH:/usr/share/ant/bin の環境変数を設定した場合は、リテラル値 $PATH:/usr/share/ant/bin が /usr/local/sbin:/usr/local/bin に置き換わります。
    CODEBUILD_ で始まる名前の環境変数は設定しないでください。このプレフィックスは内部使用のために予約されています。
    同じ名前の環境変数が複数の場所で定義されている場合は、その値は次のように決定されます。

    - ビルド開始オペレーション呼び出しの値が最も優先順位が高くなります。
    - ビルドプロジェクト定義の値が次に優先されます。
    - ビルドスペック宣言の値の優先順位が最も低くなります。

  - parameter-store: env が指定されていて、Amazon EC2 Systems Manager パラメータストアに保存されているカスタム環境変数を取得する場合は必須です。##と#のスカラーのマッピングを含み、各マッピングは Amazon EC2 Systems Manager パラメータストアに保存されている 1 つのカスタム環境変数を表します。## は、後でビルドコマンドで使用するこのカスタム環境変数を参照する名前で、# は Amazon EC2 Systems Manager パラメータストアに保存されているカスタム環境変数の名前です。重要な値を保存するには、Amazon EC2 Systems Manager ユーザーガイド の「Systems Manager パラメータストア」および「Systems Manager パラメータストアコンソールのチュートリアル」を参照してください。

    Important

    AWS CodeBuild が Amazon EC2 Systems Manager パラメータストアに保存されているカスタム環境変数を取得するには、AWS CodeBuild サービスロールに ssm:GetParameters アクションを追加する必要があります。詳細については、「AWS CodeBuild サービスロールの作成 (p. 185)」を参照してください。
    Amazon EC2 Systems Manager パラメータストアから取得する環境変数は、既存の環境変数を置き換えます。たとえば Docker イメージに MY_VAR という名前で値が my_value の環

境変数がすでに含まれていて、MY_VAR という名前で値が other_value の環境変数を取得した場合、my_value は other_value に置き換えられます。同様に、Docker イメージに PATH という名前で値が /usr/local/sbin:/usr/local/bin の環境変数がすでに含まれていて、PATH という名前で値が $PATH:/usr/share/ant/bin の環境変数を取得した場合、/usr/local/sbin:/usr/local/bin はリテラル値 $PATH:/usr/share/ant/bin に置き換えられます。
CODEBUILD_ で始まる名前の環境変数は保存しないでください。このプレフィックスは内部使用のために予約されています。
同じ名前の環境変数が複数の場所で定義されている場合は、その値は次のように決定されます。

- ビルド開始オペレーション呼び出しの値が最も優先順位が高くなります。
- ビルドプロジェクト定義の値が次に優先されます。
- ビルドスペック宣言の値の優先順位が最も低くなります。

- phases: 必要なシーケンス。ビルドの各段階で AWS CodeBuild が実行するコマンドを表します。

  Note

  ビルドスペックバージョン 0.1 では、AWS CodeBuild はビルド環境のデフォルトシェルの各インスタンスで各コマンドを実行します。つまり、各コマンドは他のすべてのコマンドとは独立して実行されます。したがって、デフォルトでは、以前のコマンド (ディレクトリの変更や環境変数の設定など) の状態に依存する単一のコマンドを実行することはできません。この制限を回避するには、バージョン 0.2 を使用することをお勧めします。これにより、問題が解決されます。何らかの理由でビルドスペックバージョン 0.1 を使用する必要がある場合は、「ビルド環境でのシェルとコマンド (p. 111)」のアプローチをお勧めします。

  許可されるビルドフェーズ名は次のとおりです。

  - install: オプションのシーケンス。インストール時に AWS CodeBuild が実行するコマンドがある場合は、そのコマンドを表します。install フェーズは、ビルド環境でのパッケージのインストールにのみ使用することをお勧めします。たとえば、このフェーズを使用して、Mocha や RSpec などのコードテストフレームワークをインストールすることができます。
    - commands: install が指定されている場合は必須のシーケンスです。一連のスカラーが含まれ、各スカラーは、インストール中に AWS CodeBuild が実行する単一のコマンドを表します。AWS CodeBuild は、最初から最後まで、各コマンドを一度に 1 つずつ、指定された順序で実行します。
  - pre_build: オプションのシーケンス。ビルドの前に AWS CodeBuild が実行するコマンドがあれば、それを表します。たとえば、このフェーズを使用して Amazon ECR にログインするか、npm の依存関係をインストールすることができます。
    - commands: pre_build が指定されている場合は必須のシーケンスです。一連のスカラーが含まれ、各スカラーは、ビルドの前に AWS CodeBuild が実行する単一のコマンドを表します。AWS CodeBuild は、最初から最後まで、各コマンドを一度に 1 つずつ、指定された順序で実行します。
  - build: オプションのシーケンス。ビルド中に AWS CodeBuild が実行するコマンドがあれば、それを表します。たとえば、このフェーズを使用して、Mocha、RSpec、または sbt を実行できます。
    - commands: build が指定されている場合は必須です。一連のスカラーが含まれ、各スカラーは、ビルド中に AWS CodeBuild が実行する単一のコマンドを表します。AWS CodeBuild は、最初から最後まで、各コマンドを一度に 1 つずつ、指定された順序で実行します。
  - post_build: オプションのシーケンス。ビルド後に AWS CodeBuild が実行するコマンドがあれば、それを表します。たとえば、Maven を使用してビルドアーティファクトを JAR または WAR ファイルにパッケージ化するか、Docker イメージを Amazon ECR にプッシュすることができます。次に、Amazon SNS を介してビルド通知を送信できます。
    - commands: post_build が指定されている場合は必須です。一連のスカラーが含まれ、各スカラーは、ビルド後に AWS CodeBuild が実行する単一のコマンドを表します。AWS CodeBuild は、最初から最後まで、各コマンドを一度に 1 つずつ、指定された順序で実行します。

Important

以前のビルドフェーズのコマンドが失敗した場合、一部のビルドフェーズのコマンドが実行されないことがあります。たとえば、install フェーズ中にコマンドが失敗した場合は、pre_build、build、および post_build フェーズの中のコマンドのいずれも、そのビルドのライフサイクルでは実行されません。詳細については、「ビルドフェーズの移行 (p. 174)」を参照してください。

- artifacts オプションのシーケンス。AWS CodeBuild がビルド出力を見つけることができる場所、および、AWS CodeBuild が Amazon S3 出力バケットにアップロードするための準備方法についての情報を表します。たとえば、Docker イメージを作成して Amazon ECR にプッシュしている場合、または、ソースコードでユニットテストを実行していてもビルドしていない場合、このシーケンスは必要ありません。
  - files: 必要なシーケンス。ビルド環境でのビルド出力アーティファクトを含む場所を表します。スカラーのシーケンスが含まれ、各スカラーは、AWS CodeBuild が元のビルドの場所を基準に、あるいは設定されている場合はベースディレクトリを基準にして、ビルド出力アーティファクトを見つけることができる個別の場所を表します。場所には次のものが含まれます。
    - 1 つのファイル (例:my-file.jar)。
    - サブディレクトリ内の単一のファイル (*my-subdirectory*/my-file.jar や *my-parent-subdirectory*/*my-subdirectory*/my-file.jar など)。
    - '**/*' はすべてのファイルを再帰的に表します。
    - *my-subdirectory*/* は、*my-subdirectory* という名前のサブディレクトリ内のすべてのファイルを表します。
    - *my-subdirectory*/**/* は、*my-subdirectory* という名前のサブディレクトリから再帰的にすべてのファイルを表します。

    ビルド出力アーティファクトの場所を指定すると、AWS CodeBuild はビルド環境で元のビルドの場所を特定できます。ビルドアーティファクトの出力先の場所に、元のビルドの場所へのパスを追加する、または ./ などで場所を指定する必要はありません。この場所へのパスを知りたい場合は、ビルド中に echo $CODEBUILD_SRC_DIR などのコマンドを実行できます。各ビルド環境の場所は多少異なる場合があります。
  - discard-paths: オプションのマッピング。ビルド出力アーティファクト内のファイルへのパスを破棄するかどうかを表します。パスを破棄する場合は yes、それ以外の場合は、no または指定しません (デフォルト)。たとえば、ビルド出力アーティファクト内のファイルへのパスは com/mycompany/app/HelloWorld.java となり、yes を指定することで、パスは HelloWorld.java だけに短縮されます。
  - base-directory: オプションのマッピング。AWS CodeBuild がビルド出力アーティファクトに含めるファイルとサブディレクトリを決定するために使用する、元のビルドの場所を基準とした、1 つ以上の最上位ディレクトリを表します。有効な値を次に示します。
    - 単一の最上位ディレクトリ (例:my-directory)。
    - 'my-directory*' my-directory で始まる名前を持つすべての最上位ディレクトリを表します。

    一致する最上位ディレクトリはビルド出力アーティファクトに含まれず、ファイルとサブディレクトリにのみ含まれます。

    files および discard-paths を使用して、どのファイルとサブディレクトリを含めるかをさらに制限できます。たとえば、以下のようなディレクトリ構造があります。

```
|-- my-build1
|      `-- my-file1.txt
`-- my-build2
       |-- my-file2.txt
       `-- my-subdirectory
              `-- my-file3.txt
```

また、次のような `artifacts` シーケンスがあります。

```
artifacts:
  files:
    - '*/my-file3.txt'
  base-directory: my-build2
```

次のサブディレクトリとファイルが、ビルド出力アーティファクトに含まれます。

```
my-subdirectory
  `-- my-file3.txt
```

さらに、次のような `artifacts` シーケンスがあります。

```
artifacts:
  files:
    - '**/*'
  base-directory: 'my-build*'
  discard-paths: yes
```

次のファイルが、ビルド出力アーティファクトに含まれます。

```
|-- my-file1.txt
|-- my-file2.txt
`-- my-file3.txt
```

- `cache` オプションのシーケンス。AWS CodeBuild が Amazon S3 キャッシュバケットにキャッシュをアップロードするためにファイルを準備できる場所に関する情報を表します。プロジェクトのキャッシュタイプが `No Cache` の場合、このシーケンスは不要です。
  - `paths`: 必要なシーケンス。キャッシュの場所を表します。スカラーのシーケンスが含まれ、各スカラーは、AWS CodeBuild が元のビルドの場所を基準に、あるいは設定されている場合はベースディレクトリを基準にして、ビルド出力アーティファクトを見つけることができる個別の場所を表します。場所には次のものが含まれます。
    - 1 つのファイル (例:`my-file.jar`)。
    - サブディレクトリ内の単一のファイル (*`my-subdirectory`*`/my-file.jar` や *`my-parent-subdirectory`*`/`*`my-subdirectory`*`/my-file.jar` など)。
    - `'**/*'` はすべてのファイルを再帰的に表します。
    - *`my-subdirectory`*`/*` は、*`my-subdirectory`* という名前のサブディレクトリ内のすべてのファイルを表します。
    - *`my-subdirectory`*`/**/*` は、*`my-subdirectory`* という名前のサブディレクトリから再帰的にすべてのファイルを表します。

    Important

    ビルドスペック宣言は有効な YAML でなければならないので、ビルドスペック宣言のスペースは重要です。ビルドスペックの宣言にあるスペースの数が無効な場合、すぐにビルドが失敗する可能性があります。YAML validator を使用して、ビルドスペックの宣言が有効な YAML かどうかをテストできます。
    ビルドプロジェクトを作成または更新するときに、AWS CLI または AWS SDK を使用してビルドスペックを宣言する場合、ビルドスペックは、必要な空白や改行のエスケープ文字のある、YAML 形式で表される単一の文字列でなければなりません。次のセクションに例があります。
    buildspec.yml ファイルの代わりに AWS CodeBuild または AWS CodePipeline コンソールを使用する場合は、`build` フェーズのみコマンドを挿入できます。上記の構文を使用する代わりに、ビ

ルドフェーズで実行するすべてのコマンドを 1 行に記載します複数のコマンドについては、`&&` で各コマンドを区切ります (例: `mvn test && mvn package`)。
buildspec.yml ファイルの代わりに AWS CodeBuild または AWS CodePipeline コンソールを使用して、ビルド環境でビルド出力アーティファクトの場所を指定することができます。上記の構文を使用する代わりに、すべての場所を 1 行に記載します。複数の場所の場合は、各場所をコンマで区切ります (例: `appspec.yml, target/my-app.jar`)。

ビルドスペックの例

buildspec.yml ファイルの例を次に示します。

```
version: 0.2

env:
  variables:
    JAVA_HOME: "/usr/lib/jvm/java-8-openjdk-amd64"
  parameter-store:
    LOGIN_PASSWORD: "dockerLoginPassword"

phases:
  install:
    commands:
      - echo Entered the install phase...
      - apt-get update -y
      - apt-get install -y maven
  pre_build:
    commands:
      - echo Entered the pre_build phase...
      - docker login -u User -p $LOGIN_PASSWORD
  build:
    commands:
      - echo Entered the build phase...
      - echo Build started on `date`
      - mvn install
  post_build:
    commands:
      - echo Entered the post_build phase...
      - echo Build completed on `date`
artifacts:
  files:
    - target/messageUtil-1.0.jar
  discard-paths: yes
cache:
  paths:
    - '/root/.m2/**/*'
```

次に、AWS CLI または AWS SDK で使用するための、単一の文字列として表現された前述のビルドスペックの例を示します。

```
"version: 0.2\n\nenv:\n  variables:\n    JAVA_HOME: "/usr/lib/jvm/java-8-openjdk-amd64"\n
  parameter-store:\n    LOGIN_PASSWORD: "dockerLoginPassword"\n\nphases:\n  install:\n
   commands:\n      - apt-get update -y\n      - apt-get install -y maven\n  pre_build:
\n    commands:\n      - echo Nothing to do in the pre_build phase...\n  build:\n
 commands:\n      - echo Build started on `date`\n      - mvn install\n  post_build:\n
  commands:\n      - echo Build completed on `date`\nartifacts:\n  files:\n    - target/
messageUtil-1.0.jar\n  discard-paths: yes"
```

AWS CodeBuild または AWS CodePipeline コンソールで使用する `build` フェーズのコマンドの例を次に示します。

```
echo Build started on `date` && mvn install
```

これらの例では:

- `JAVA_HOME` のキーと `/usr/lib/jvm/java-8-openjdk-amd64` の値を持つプレーンテキストのカスタム環境変数が設定されます。
- Amazon EC2 Systems Manager パラメータストアに保存された `dockerLoginPassword` という名前のカスタム環境変数は、後で `LOGIN_PASSWORD` キーを使用してビルドコマンドで参照されます。
- これらのビルドフェーズ名は変更できません。この例で実行されるコマンドは、`apt-get update -y` と `apt-get install -y maven` (Apache Maven をインストールする)、`mvn install` (ソースコードをコンパイル、テストして、ビルド出力アーティファクトにパッケージ化し、ビルド出力アーティファクトを内部リポジトリにインストールするなどの他のアクションを実行する)、`docker login` (Amazon EC2 Systems Manager パラメータストアで設定したカスタム環境変数 `dockerLoginPassword` の値に対応するパスワードを使用して Docker へログインする)、およびいくつかの `echo` コマンドです。これらの `echo` コマンドは、AWS CodeBuild がコマンドを実行する方法と、コマンドの実行順序を示すためにここに含まれています。
- `files` はビルド出力場所にアップロードするファイルを表します。この例では、AWS CodeBuild が 1 つのファイル `messageUtil-1.0.jar` をアップロードします。`messageUtil-1.0.jar` ファイルは、ビルド環境の `target` という名の相対ディレクトリ内にあります。`discard-paths: yes` が指定されたため、`messageUtil-1.0.jar` は直接アップロードされます (中間の `target` ディレクトリにはアップロードされません)。ファイル名 `messageUtil-1.0.jar` および相対ディレクトリ名 `target` は、Apache Maven がこの例のビルド出力アーティファクトを作成して保存する方法にのみ基づいています。独自のシナリオでは、これらのファイル名とディレクトリは異なります。

ビルドスペックのバージョン

次の表に、ビルドスペックのバージョンとバージョン間の変更を示します。

Version	変更
0.2	`environment_variables` が `env` に名称変更されました。`plaintext` が `variables` に名称変更されました。 バージョン 0.1 では、AWS CodeBuild はビルド環境のデフォルトシェルの個別のインスタンスで各ビルドコマンドを実行します。バージョン 0.2 では、この問題に対処しています。AWS CodeBuild は、ビルド環境のデフォルトシェルと同じインスタンスですべてのビルドコマンドを実行します。
0.1	これは、ビルド仕様形式の最初の定義です。

AWS CodeBuild のビルド環境リファレンス

ビルドを実行するために AWS CodeBuild を呼び出すときは、AWS CodeBuild が使用するビルド環境に関する情報を提供する必要があります。ビルド環境は、ビルドを実行するために AWS CodeBuild が使用するオペレーティングシステム、プログラミング言語ランタイム、およびツールの組み合わせを表します。ビルド環境の仕組みについては、「AWS CodeBuild の詳細 (p. 3)」を参照してください。

ビルド環境には Docker イメージが含まれています。詳細については、Docker Docs ウェブサイトの「Docker Glossary: Image」を参照してください。

ビルド環境について AWS CodeBuild に情報を提供する場合は、サポートされているリポジトリタイプの Docker イメージの識別子を指定します。これには、AWS CodeBuild Docker イメージリポジトリ、Docker Hub に公開されているイメージ、AWS アカウントの Amazon Elastic Container Registry (Amazon ECR) リポジトリなどがあります。

- AWS CodeBuild イメージリポジトリに格納されている Docker イメージは、サービスで使用するために最適化されているため、使用することをお勧めします。詳細については、「AWS CodeBuild に用意されている Docker イメージ (p. 103)」を参照してください。
- Docker Hub に保存されて公開されている Docker イメージの識別子を取得するには、Docker Docs ウェブサイトの「イメージの検索」を参照してください。
- AWS アカウントの Amazon ECR リポジトリに保存されている Docker イメージの操作方法については、Amazon ECR サンプル (p. 22) を参照してください。

Docker イメージ識別子に加えて、ビルド環境で使用する一連のコンピューティングリソースも指定します。詳細については、「ビルド環境コンピューティングタイプ (p. 110)」を参照してください。

トピック

AWS CodeBuild に用意されている Docker イメージ

AWS CodeBuild で管理される以下の Docker イメージは、AWS CodeBuild コンソールおよび AWS CodePipeline コンソールで使用できます。

プラットフォーム	プログラミング言語またはフレームワーク	ランタイムバージョン	追加のコンポーネント	イメージ識別子	定義
Ubuntu 14.04	(ベースイメージ)		AWS CLI、Git 1.9.1	`aws/codebuild/ubuntu-base:14.04`	ubuntu/ubuntu-base/14.04
Ubuntu 14.04	Android	24.4.1	AWS CLI、Git 1.9.1、Java 8、pip 8.1.2、Python 2.7	`aws/codebuild/android-java-8:24.4.1`	ubuntu/android-java-8/24.4.1
Ubuntu 14.04	Docker	1.12.1	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7、Docker-compose 1.16.1	`aws/codebuild/docker:1.12.1`	ubuntu/docker/1.12.1
Ubuntu 14.04	Docker	17.09.0	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7、Docker-	`aws/codebuild/docker:17.09.0`	ubuntu/docker/17.09.0

プラットフォーム	プログラミング言語またはフレームワーク	ランタイムバージョン	追加のコンポーネント	イメージ識別子	定義
			compose 1.16.1		
Ubuntu 14.04	Golang	1.7.3	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7	`aws/codebuild/golang:1.7.3`	ubuntu/golang/1.7.3
Ubuntu 14.04	Golang	1.6.3	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7	`aws/codebuild/golang:1.6.3`	ubuntu/golang/1.6.3
Ubuntu 14.04	Golang	1.5.4	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7	`aws/codebuild/golang:1.5.4`	ubuntu/golang/1.5.4
Ubuntu 14.04	Java	8	Apache Ant 1.9.6、Apache Maven 3.3.3、AWS CLI、Git 1.9.1、Gradle 2.7、pip 8.1.2、Python 2.7	`aws/codebuild/java:openjdk-8`	ubuntu/java/openjdk-8
Ubuntu 14.04	Java	7	Apache Ant 1.9.6、Apache Maven 3.3.3、AWS CLI、Git 1.9.1、Gradle 2.7、pip 8.1.2、Python 2.7	`aws/codebuild/java:openjdk-7`	ubuntu/java/openjdk-7
Ubuntu 14.04	Java	6	Apache Ant 1.9.6、Apache Maven 3.2.5、AWS CLI、Git 1.9.1、Gradle 2.7、pip 8.1.2、Python 2.7	`aws/codebuild/java:openjdk-6`	ubuntu/java/openjdk-6
Ubuntu 14.04	Node.js	7.0.0	AWS CLI、Git 1.9.1、NPM、pip 8.1.2、Python 2.7	`aws/codebuild/nodejs:7.0.0`	ubuntu/nodejs/7.0.0

プラットフォーム	プログラミング言語またはフレームワーク	ランタイムバージョン	追加のコンポーネント	イメージ識別子	定義
Ubuntu 14.04	Node.js	6.3.1	AWS CLI、Git 1.9.1、NPM、pip 8.1.2、Python 2.7	`aws/codebuild/nodejs:6.3.1`	ubuntu/nodejs/6.3.1
Ubuntu 14.04	Node.js	5.12.0	AWS CLI、Git 1.9.1、NPM、pip 8.1.2、Python 2.7	`aws/codebuild/nodejs:5.12.0`	ubuntu/nodejs/5.12.0
Ubuntu 14.04	Node.js	4.4.7	AWS CLI、Git 1.9.1、NPM、pip 8.1.2、Python 2.7	`aws/codebuild/nodejs:4.4.7`	ubuntu/nodejs/4.4.7
Ubuntu 14.04	Node.js	4.3.2	AWS CLI、Git 1.9.1、NPM、pip 8.1.2、Python 2.7	`aws/codebuild/nodejs:4.3.2`	ubuntu/nodejs/4.3.2
Ubuntu 14.04	Python	3.5.2	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7	`aws/codebuild/python:3.5.2`	ubuntu/python/3.5.2
Ubuntu 14.04	Python	3.4.5	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7	`aws/codebuild/python:3.4.5`	ubuntu/python/3.4.5
Ubuntu 14.04	Python	3.3.6	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7	`aws/codebuild/python:3.3.6`	ubuntu/python/3.3.6
Ubuntu 14.04	Python	2.7.12	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7	`aws/codebuild/python:2.7.12`	ubuntu/python/2.7.12
Ubuntu 14.04	Ruby	2.3.1	AWS CLI、Bundler 1.13.5、Git 1.9.1、pip 8.1.2、Python 2.7、RubyGems 2.6.7	`aws/codebuild/ruby:2.3.1`	ubuntu/ruby/2.3.1

プラットフォーム	プログラミング言語またはフレームワーク	ランタイムバージョン	追加のコンポーネント	イメージ識別子	定義
Ubuntu 14.04	Ruby	2.2.5	AWS CLI、Bundler 1.13.5、Git 1.9.1、pip 8.1.2、Python 2.7、RubyGems 2.6.7	`aws/codebuild/ruby:2.2.5`	ubuntu/ruby/2.2.5
Ubuntu 14.04	Ruby	2.1.10	AWS CLI、Bundler 1.13.5、Git 1.9.1、pip 8.1.2、Python 2.7、RubyGems 2.6.7	`aws/codebuild/ruby:2.1.10`	ubuntu/ruby/2.1.10
Ubuntu 14.04	.NET Core	1.1	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7、.NET Core SDK 1.0.4	`aws/codebuild/dot-net:core-1`	ubuntu/dot-net/core-1
Ubuntu 14.04	.NET Core	2.0	AWS CLI、Git 1.9.1、pip 8.1.2、Python 2.7、.NET Core SDK 2.0.3	`aws/codebuild/dot-net:core-2.0`	ubuntu/dot-net/core-2

AWS CodeBuild で管理される以下の Docker イメージについては、AWS CodeBuild コンソールおよび AWS CodePipeline コンソールで使用できません。

プラットフォーム	プログラミング言語またはフレームワーク	ランタイムバージョン	追加のコンポーネント	イメージ識別子
Amazon Linux 2016.03, 64 ビット v2.3.2	Golang	1.6	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-go-1.6-amazonlinux-64:2.3.2`
Amazon Linux 2016.03, 64 ビット v2.1.6	Golang	1.5.3	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-go-1.5-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Golang	1.5.3	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-go-1.5-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.4.3	Java	1.8.0	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-java-8-amazonlinux-64:2.4.3`

プラットフォーム	プログラミング言語またはフレームワーク	ランタイムバージョン	追加のコンポーネント	イメージ識別子
Amazon Linux 2016.03, 64 ビット v2.1.6	Java	1.8.0	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-java-8-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Java	1.8.0	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-java-8-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.4.3	Java	1.7.0	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-java-7-amazonlinux-64:2.4.3`
Amazon Linux 2016.03, 64 ビット v2.1.6	Java	1.7.0	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-java-7-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Java	1.7.0	Apache Maven 3.3.3、Apache Ant 1.9.6、Gradle 2.7	`aws/codebuild/eb-java-7-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v4.0.0	Node.js	6.10.0	Git 2.7.4、npm 2.15.5	`aws/codebuild/eb-nodejs-6.10.0-amazonlinux-64:4.0.0`
Amazon Linux 2016.03、64 ビット v2.1.3	Node.js	4.4.6	Git 2.7.4、npm 2.15.5	`aws/codebuild/eb-nodejs-4.4.6-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.1.6	Python	3.4.3	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-3.4-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Python	3.4.3	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-3.4-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.3.2	Python	3.4	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-3.4-amazonlinux-64:2.3.2`
Amazon Linux 2016.03, 64 ビット v2.1.6	Python	2.7.10	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-2.7-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Python	2.7.10	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-2.7-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.3.2	Python	2.7	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-2.7-amazonlinux-64:2.3.2`

プラットフォーム	プログラミング言語またはフレームワーク	ランタイムバージョン	追加のコンポーネント	イメージ識別子
Amazon Linux 2016.03, 64 ビット v2.1.6	Python	2.6.9	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-2.6-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Python	2.6.9	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-2.6-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.3.2	Python	2.6	meld3 1.0.2、pip 7.1.2、setuptools 18.4	`aws/codebuild/eb-python-2.6-amazonlinux-64:2.3.2`
Amazon Linux 2016.03, 64 ビット v2.1.6	Ruby	2.3.1	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.3-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Ruby	2.3.1	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.3-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.3.2	Ruby	2.3	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.3-amazonlinux-64:2.3.2`
Amazon Linux 2016.03, 64 ビット v2.1.6	Ruby	2.2.5	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.2-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Ruby	2.2.5	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.2-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.3.2	Ruby	2.2	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.2-amazonlinux-64:2.3.2`
Amazon Linux 2016.03, 64 ビット v2.1.6	Ruby	2.1.9	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.1-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Ruby	2.1.9	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.1-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.3.2	Ruby	2.1	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.1-amazonlinux-64:2.3.2`
Amazon Linux 2016.03, 64 ビット v2.3.2	Ruby	2.0	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.0-amazonlinux-64:2.3.2`
Amazon Linux 2016.03, 64 ビット v2.1.6	Ruby	2.0.0	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.0-amazonlinux-64:2.1.6`

プラットフォーム	プログラミング言語またはフレームワーク	ランタイムバージョン	追加のコンポーネント	イメージ識別子
Amazon Linux 2016.03、64 ビット v2.1.3	Ruby	2.0.0	Bundler、RubyGems	`aws/codebuild/eb-ruby-2.0-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.1.6	Ruby	1.9.3	Bundler、RubyGems	`aws/codebuild/eb-ruby-1.9-amazonlinux-64:2.1.6`
Amazon Linux 2016.03、64 ビット v2.1.3	Ruby	1.9.3	Bundler、RubyGems	`aws/codebuild/eb-ruby-1.9-amazonlinux-64:2.1.3`
Amazon Linux 2016.03, 64 ビット v2.3.2	Ruby	1.9	Bundler、RubyGems	`aws/codebuild/eb-ruby-1.9-amazonlinux-64:2.3.2`
識別子に `eb-` を含む Docker イメージの詳細については、AWS Elastic Beanstalk 開発者ガイドの「サポートされているプラットフォーム」および「プラットフォームの履歴」を参照してください。識別子に `eb-` を含む Docker イメージは、Elastic Beanstalk では使用できますが、AWS CodeBuild コンソールと AWS CodePipeline コンソールでは使用できません。				

`install` ビルドフェーズでは、ビルド仕様を使用して、他のコンポーネント (AWS CLI、Apache Maven、Apache Ant、Mocha、RSpec など) をインストールできます。詳細については、「ビルドスペックの例 (p. 101)」を参照してください。

AWS CodeBuild は Docker イメージのリストを頻繁に更新します。最新のリストを取得するには、次のいずれかを実行します。

- AWS CodeBuild コンソールの [Create project] ウィザードまたは [Update project] ページで、[Environment image] として [Use an image managed by AWS CodeBuild] を選択します。[Operating system]、[Runtime]、および [Version] の各ドロップダウンリストで適切な選択を行います。詳細については、「ビルドプロジェクトの作成 (コンソール) (p. 146)」または「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」を参照してください。
- AWS CodePipeline コンソールの [Step 3: Build] ページの [Create pipeline] ウィザード、あるいは [Add action] ペインまたは [Edit action] ペインの [AWS CodeBuild] セクションで、[Create a new build project] を選択します。[Environment: How to build] で、[Environment image] として [Use an image managed by AWS CodeBuild] を選択します。[Operating system]、[Runtime]、および [Version] の各ドロップダウンリストで適切な選択を行います。詳細については、「AWS CodeBuild を使用するパイプラインを作成する (AWS CodePipeline コンソール) (p. 128)」または「AWS CodeBuild ビルドアクションをパイプラインに追加する (AWS CodePipeline コンソール) (p. 135)」を参照してください。
- AWS CLI では、`list-curated-environment-images` コマンドを実行します。

```
aws codebuild list-curated-environment-images
```

- AWS SDK では、ターゲットのプログラミング言語の `ListCuratedEnvironmentImages` オペレーションを呼び出します。詳細については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

Docker イメージにインストールされているコンポーネントのバージョンを確認するには、ビルド中にコマンドを実行します。コンポーネントのバージョン番号が出力に表示されます。たとえば、ビルド仕様に次のコマンドを 1 つ以上含めます。

- Apache Ant の場合は、`ant -version` を実行します。
- Apache Maven の場合は、`mvn -version` を実行します。
- AWS CLI の場合は、`aws --version` を実行します。
- Bundler の場合は、`bundle version` を実行します。
- Git の場合は、`git --version` を実行します。
- Gradle の場合は、`gradle --version` を実行します。
- Java の場合は、`java -version` を実行します。
- NPM の場合は、`npm --version` を実行します。
- pip の場合は、`pip --version` を実行します。
- Python の場合は、`python --version` を実行します。
- RubyGems の場合は、`gem --version` を実行します。
- setuptools の場合は、`easy_install --version` を実行します。

次のビルドコマンド (ビルドプロジェクトの設定の一部として AWS CodeBuild または AWS CodePipeline コンソールから入力) は、これらのコンポーネントがインストールされている Docker イメージ上で AWS CLI、Git、pip、および Python のバージョンを返します `aws --version && git --version && pip --version && python --version`。

ビルド環境コンピューティングタイプ

AWS CodeBuild は、以下のビルド環境に使用可能なメモリ、vCPU、および使用可能なディスクスペースを提供します。

コンピューティングタイプ	computeType 値	メモリ	vCPU	ディスク容量
build.general1.small	`BUILD_GENERAL1_SMALL`	3 GB	2	64 GB
build.general1.medium	`BUILD_GENERAL1_MEDIUM`	7 GB	4	128 GB
build.general1.large	`BUILD_GENERAL1_LARGE`	15 GB	8	128 GB

> **Note**
>
> カスタムビルド環境イメージとして、AWS CodeBuild は、コンピューティングタイプを問わず、最大 10 GB の未圧縮の Docker イメージをサポートします。ビルドイメージのサイズを確認するには、Docker を使用して `docker images` *`REPOSITORY`*`:`*`TAG`* コマンドを実行します。

これらのいずれかのコンピューティングタイプを選択するには:

- AWS CodeBuild コンソールで、[Create Project] ウィザードまたは [Update project] ページの [Show advanced settings] を展開し、[Compute type] からいずれかのオプションを選択します。詳細については、「ビルドプロジェクトの作成 (コンソール) (p. 146)」または「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」を参照してください。
- AWS CodePipeline コンソールで、[Step 3: Build] ページの [Create Pipeline] ウィザード、あるいは [Add action] ペインまたは [Edit action] ペインの [Create a new build project] を選択し、[Advanced] を展開して [Compute type] のいずれかのオプションを選択します。詳細については、「AWS CodeBuild を使用するパイプラインを作成する (AWS CodePipeline コンソール) (p. 128)」または「AWS CodeBuild ビルドアクションをパイプラインに追加する (AWS CodePipeline コンソール) (p. 135)」を参照してください。

- AWS CLI の場合は、`create-project` または `update-project` コマンドを実行し、`environment` オブジェクトの `computeType` の値を指定します。詳細については、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」または「ビルドプロジェクトの設定を変更する (AWS CLI) (p. 165)」を参照してください。
- AWS SDK の場合は、ターゲットのプログラミング言語に `CreateProject` または `UpdateProject` に相当するオペレーションを呼び出し、`environment` オブジェクトの `computeType` に相当する値を指定します。詳細については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

ビルド環境でのシェルとコマンド

ビルドのライフサイクル中にビルド環境で実行するための AWS CodeBuild の一連のコマンドを提供します (たとえば、ビルドの依存関係のインストール、ソースコードのテストおよびコンパイルなど)。これらのコマンドを指定する方法はいくつかあります。

- ビルドスペックファイルを作成し、それをソースコードに組み込みます。このファイルでは、ビルドライフサイクルの各段階で実行するコマンドを指定します。詳細については、「AWS CodeBuild のビルド仕様に関するリファレンス (p. 95)」を参照してください。
- AWS CodeBuild または AWS CodePipeline コンソールを使用してビルドプロジェクトを作成します。[Insert build commands] の [Build command] で、[`build`] フェーズで実行するコマンドを指定します。詳細については、「ビルドプロジェクトの作成 (コンソール) (p. 146)」または「AWS CodeBuild で AWS CodePipeline を使用する (p. 126)」で「ビルドコマンドの挿入」の説明を参照してください。
- AWS CodeBuild コンソールを使用してビルドプロジェクトの設定を変更します。[Insert build commands] の [Build command] で、[`build`] フェーズで実行するコマンドを指定します。詳細については、「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」で「ビルドコマンドの挿入」の説明を参照してください。
- AWS CLI または AWS SDK を使用して、ビルドプロジェクトを作成するか、ビルドプロジェクトの設定を変更します。コマンドを使用してビルドスペックファイルを含むソースコードを参照するか、ビルドスペックファイルと同等の内容を含む単一の文字列を指定します。詳細については、ビルドプロジェクトを作成する (p. 146) または ビルドプロジェクトの設定を変更する (p. 161) にある `buildspec` 値の説明を参照してください。
- AWS CLI または AWS SDK を使用してビルドを開始し、ビルドスペックファイルを指定するか、ビルドスペックファイルと同等の内容を含む単一の文字列を指定します。詳細については、ビルドの実行 (p. 167) にある `buildspecOverride` 値の説明を参照してください。

ビルド環境のデフォルトシェルでサポートされている任意のコマンドを指定できます (`- sh` は選別されたイメージのデフォルトシェルです)。ビルドスペックバージョン 0.1 では、AWS CodeBuild はビルド環境のデフォルトシェルの各インスタンスで各コマンドを実行します。つまり、各コマンドは他のすべてのコマンドとは独立して実行されます。したがって、デフォルトでは、以前のコマンド (ディレクトリの変更や環境変数の設定など) の状態に依存する単一のコマンドを実行することはできません。この制限を回避するには、バージョン 0.2 を使用することをお勧めします。これにより、問題が解決されます。何らかの理由でバージョン 0.1 を使用する場合は、以下のアプローチをお勧めします。

- デフォルトシェルの単一のインスタンスで実行するコマンドを含むシェルスクリプトをソースコードに含めます。たとえば、`my-script.sh` という名前のファイルを、`cd MyDir; mkdir -p mySubDir; cd mySubDir; pwd;` などのコマンドを含むソースコードに含めます。次にビルドスペックファイルで `./my-script.sh` コマンドを指定します。
- ビルドスペックファイル (または `build` フェーズに限ってはコンソールの [Build command] 設定) で、デフォルトシェルの単一のインスタンスで実行するすべてのコマンドが含まれている単一のコマンドを指定します (例: `cd MyDir && mkdir -p mySubDir && cd mySubDir && pwd`)。

AWS CodeBuild でエラーが発生した場合は、デフォルトシェルの独自のインスタンスで単一のコマンドを実行するのに比べて、トラブルシューティングが難しくなる場合があります。

ビルド環境の環境変数

AWS CodeBuild には、ビルドコマンドで使用できるいくつかの環境変数が用意されています。

- `AWS_DEFAULT_REGION`: ビルドが実行されている AWS リージョン (例:`us-east-1`)。この環境変数は、AWS CLI で主に使用されます。
- `AWS_REGION`: ビルドが実行されている AWS リージョン (例:`us-east-1`)。この環境変数は、AWS SDK で主に使用されます。
- `CODEBUILD_BUILD_ARN`: ビルドの Amazon リソースネーム (ARN) (例:`arn:aws:codebuild:region-ID:account-ID:build/codebuild-demo-project:b1e6661e-e4f2-4156-9ab9-82a19EXAMPLE`)。
- `CODEBUILD_BUILD_ID`: ビルドの AWS CodeBuild ID (例:`codebuild-demo-project:b1e6661e-e4f2-4156-9ab9-82a19EXAMPLE`)。
- `CODEBUILD_BUILD_IMAGE`: AWS CodeBuild ビルドイメージ識別子 (例:`aws/codebuild/java:openjdk-8`)。
- `CODEBUILD_BUILD_SUCCEEDING`: 現在のビルドが成功かどうか。ビルドが失敗の場合は `0` に設定され、成功の場合は `1` に設定されます。
- `CODEBUILD_INITIATOR`: ビルドを開始したエンティティ。AWS CodePipeline がビルドを開始した場合、これはパイプラインの名前です (例:`codepipeline/my-demo-pipeline`)。IAM ユーザーがビルドを開始した場合は、これはユーザーの名前です (例:`MyUserName`)。AWS CodeBuild の Jenkins プラグインがビルドを開始した場合、これは文字列 `CodeBuild-Jenkins-Plugin` です。
- `CODEBUILD_KMS_KEY_ID`: AWS CodeBuild がビルド出力アーティファクトを暗号化するために使用している AWS KMS キーの識別子 (例:`arn:aws:kms:region-ID:account-ID:key/key-ID` または `alias/key-alias`)。
- `CODEBUILD_RESOLVED_SOURCE_VERSION`: AWS CodePipeline によって実行されるビルドの場合、ビルドするソースコードのコミット ID または Amazon S3 バージョン ID。この値は、パイプラインの関連するソースアクションが Amazon S3、AWS CodeCommit、または GitHub リポジトリに基づいている場合にのみ使用できます。
- `CODEBUILD_SOURCE_REPO_URL`: 入力アーティファクトまたはソースコードリポジトリの URL。Amazon S3 では、これは `s3://` の後にバケット名と入力アーティファクトへのパスが続きます。AWS CodeCommit および GitHub の場合、これはリポジトリのクローン URL です。
- `CODEBUILD_SOURCE_VERSION`: Amazon S3 の場合、入力アーティファクトに関連付けられたバージョン ID。AWS CodeCommit の場合、ビルドするソースコードのバージョンに関連付けられたコミット ID またはブランチ名。GitHub の場合、ビルドするソースコードのバージョンに関連付けられたコミット ID、ブランチ名、またはタグ名。
- `CODEBUILD_SRC_DIR`: AWS CodeBuild がビルドに使用するディレクトリパス (例:`/tmp/src123456789/src`)。
- `HOME`: この環境変数は常に `/root` に設定されます。

独自の環境変数を持つビルド環境を提供することもできます。詳細については、次のトピックを参照してください。

- AWS CodeBuild で AWS CodePipeline を使用する (p. 126)
- ビルドプロジェクトを作成する (p. 146)
- ビルドプロジェクトの設定を変更する (p. 161)
- ビルドの実行 (p. 167)
- ビルドスペックリファレンス (p. 95)

Important

機密情報、特に AWS アクセスキー ID とシークレットアクセスキーを格納する場合には、環境変数を使用しないことを強くお勧めします。環境変数は、AWS CodeBuild コンソールや AWS CLI などのツールを使用してプレーンテキストで表示できます。
重要な値は Amazon EC2 Systems Manager パラメータストアに保存し、ビルドスペックから取得することをお勧めします。重要な値を保存するには、Amazon EC2 Systems Manager ユーザーガイドの「Systems Manager パラメータストア」および「Systems Manager パラメータストア チュートリアル」を参照してください。重要な値を取得するには、「ビルドスペックの構文 (p. 96)」で `parameter-store` マッピングを参照してください。

ビルド環境で使用可能な環境変数をすべて一覧表示するには、ビルド中に `printenv` コマンドを実行します。前述のものを除いて、`CODEBUILD_` で始まる環境変数は、AWS CodeBuild の内部使用のためのものです。それらはビルドコマンドで使用できません。

ビルド環境のバックグラウンドタスク

ビルド環境でバックグラウンドタスクを実行できます。これを行うには、ビルドプロセスでシェルが終了される場合でも、ビルドスペックで `nohup` コマンドを使用してバックグラウンドのタスクとしてコマンドを実行します。実行中のバックグラウンドタスクを強制終了するには、`disown` コマンドを使用します。

例:

- バックグラウンドプロセスを開始し、その後、完了するまで待機します。

```
nohup sleep 30 & ; echo $! > pidfile
…
wait $(cat pidfile)
```

- バックグラウンドプロセスを開始し、その後、完了するまで待機しません。

```
nohup sleep 30 & ; disown $!
```

- バックグラウンドプロセスを開始し、その後、強制終了します。

```
nohup sleep 30 & ; echo $! > pidfile
…
kill $(cat pidfile)
```

AWS CodeBuild を直接実行する

AWS CodeBuild で直接ビルドを設定、実行、監視するには、AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用できます。

お探しのものではありませんか。AWS CodePipeline を使用して、AWS CodeBuild を実行するには、「AWS CodeBuild で AWS CodePipeline を使用する (p. 126)」を参照してください。

トピック

前提条件

ビルドを計画する (p. 94) の質問に答えてください。

AWS CodeBuild を直接実行する (コンソール)

1. ビルドプロジェクトを作成します。詳細については、ビルドプロジェクトの作成 (コンソール) (p. 146) を参照してください。
2. ビルドを実行します。詳細については、ビルドの実行 (コンソール) (p. 167) を参照してください。
3. ビルドについての情報を取得します。詳細については、ビルドの詳細の表示 (コンソール) (p. 174) を参照してください。

AWS CodeBuild を直接実行する (AWS CLI)

AWS CodeBuild で AWS CLI を使用する方法の詳細については、「コマンドラインリファレンス (p. 191)」を参照してください。

1. ビルドプロジェクトを作成します。詳細については、ビルドプロジェクトを作成する (AWS CLI) (p. 151) を参照してください。
2. ビルドを実行します。詳細については、ビルドの実行 (AWS CLI) (p. 169) を参照してください。
3. ビルドについての情報を取得します。詳細については、ビルドの詳細を表示する (AWS CLI) (p. 174) を参照してください。

Amazon Virtual Private Cloud で AWS CodeBuild を使用する

通常、VPC 内のリソースには AWS CodeBuild でアクセスすることはできません。アクセスを有効にするには、AWS CodeBuild プロジェクト設定の一部として追加の VPC 固有設定情報を指定する必要があります。これには、VPC ID、VPC サブネット ID、および VPC セキュリティグループ ID が含まれます。VPC 対応のビルドは、VPC 内のリソースにアクセスできます。Amazon VPC で VPC を設定する方法の詳細については、「VPC ユーザーガイド」を参照してください。

トピック

ユースケース

AWS CodeBuild からの VPC 接続により、次のことが可能になります。

- プライベートサブネット上に分離された Amazon RDS データベース内のデータに対して、ビルドから統合テストを実行する。
- Amazon ElastiCache クラスターのデータをテストから直接クエリする。
- Amazon EC2、Amazon ECS、または内部 Elastic Load Balancing を使用するサービスでホストされる内部ウェブサービスを操作する。
- Python 用 PyPI、Java 用 Maven、Node.js 用 npm など、セルフホスト型の内部アーティファクトリポジトリから依存関係を取得する。
- Amazon VPC エンドポイント経由でのみアクセスできるように設定された Amazon S3 バケット内のオブジェクトにアクセスする。
- 固定 IP アドレスを必要とする外部ウェブサービスを、サブネットに関連付けられた NAT ゲートウェイまたは NAT インスタンスの Elastic IP アドレスを使用してクエリする。

お客様のビルドは、VPC でホストされている任意のリソースにアクセスできます。

AWS CodeBuild プロジェクトでの Amazon VPC アクセスの有効化

以下の設定を VPC 設定に含めます。

- [VPC ID] で、AWS CodeBuild が使用する VPC ID を選択します。
- [Subnets] で、AWS CodeBuild が使用するリソースを含むサブネットを選択します。

- [Security Groups] で、AWS CodeBuild が VPC 内のリソースへのアクセスを許可するために使用するセキュリティグループを選択します。

ビルドプロジェクトの作成 (コンソール)

ビルドプロジェクトの作成の詳細については、「ビルドプロジェクトの作成 (コンソール) (p. 146)」を参照してください。AWS CodeBuild プロジェクトを作成または変更する場合、[VPC] で、VPC ID、サブネット、セキュリティグループを選択します。

ビルドプロジェクトを作成する (AWS CLI)

ビルドプロジェクトの作成の詳細については、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」を参照してください。AWS CLI で AWS CodeBuild を使用している場合、IAM ユーザーに代わってサービスを操作するために AWS CodeBuild が使用するサービスロールには、次のポリシーがアタッチされている必要があります:「VPC ネットワークインターフェイスの作成をユーザーに許可する (p. 203)」。

vpcConfig オブジェクトには、*vpcId*、*securityGroupIds*、および *subnets* が含まれている必要があります。

- *vpcId*: 必須値。AWS CodeBuild で使用される VPC ID。リージョン内のすべての Amazon VPC ID のリストを取得するには、次のコマンドを実行します。

```
aws ec2 describe-vpcs
```

- *subnets*: 必須値。AWS CodeBuild で使用されるリソースを含むサブネット ID。この ID を取得するには、次のコマンドを実行します。

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=<vpc-id>" --region us-east-1
```

  Note

  us-east-1 を、ご利用のリージョンに置き換えてください。

- *securityGroupIds*: 必須値。VPC 内のリソースへのアクセスを許可するために AWS CodeBuild で使用されるセキュリティグループ ID。この ID を取得するには、次のコマンドを実行します。

```
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=<vpc-id>" --region us-
east-1
```

  Note

  us-east-1 を、ご利用のリージョンに置き換えてください。

VPC のベストプラクティス

AWS CodeBuild で動作するように VPC を設定する場合、このチェックリストを使用してください。

- パブリックおよびプライベートサブネットと NAT ゲートウェイを使用して VPC を設定します。詳細については、「パブリックサブネットとプライベートサブネットを持つ VPC (NAT)」を参照してください。

  Important

  Amazon VPC で AWS CodeBuild を使用するには、NAT ゲートウェイまたは NAT インスタンスが必要です。これにより、AWS CodeBuild がパブリックエンドポイントに到達できるようになります (ビルド実行時に CLI コマンドを実行するなど)。AWS CodeBuild は、作成した

ネットワークインターフェイスに Elastic IP アドレスを割り当てることをサポートしていないため、NAT ゲートウェイまたは NAT インスタンスの代わりにインターネットゲートウェイを使用することはできません。パブリック IP アドレスの自動割り当ては、Amazon EC2 インスタンスによる起動以外で作成されたネットワークインタフェースに対しては Amazon EC2 ではサポートされていません。

- VPC に複数のアベイラビリティーゾーンを含めます。
- セキュリティグループに、ビルドに許可されたインバウンド (進入) トラフィックがないことを確認します。詳細については、「セキュリティグループルール」を参照してください。
- ビルド用に別個のサブネットを設定します。
- VPC にアクセスするように AWS CodeBuild プロジェクトを設定する場合、プライベートサブネットのみを選択します。

Amazon VPC で VPC を設定する方法の詳細については、「Amazon VPC ユーザーガイド」を参照してください。

AWS CloudFormation を使用して AWS CodeBuild VPC 機能を使用するように Amazon VPC を設定する方法の詳細については、「AWS CloudFormation VPC テンプレート (p. 117)」を参照してください。

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、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。

次の一般的な AWS CodeBuild VPC のエラーをトラブルシューティングする際に役立つガイドラインを以下に示します。「ビルドにはインターネット接続がありません。サブネットのネットワーク構成を確認してください。」

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指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. NAT ゲートウェイのトラブルシューティングを行います。
6. プライベートサブネットのルートテーブルが NAT ゲートウェイを指していることを確認します。
7. IAM ユーザーに代わってサービスを操作するために AWS CodeBuild が使用するサービスロールに、次のポリシーがアタッチされていることを確認します:「VPC ネットワークインターフェイスの作成をユーザーに許可する (p. 203)」。

AWS CloudFormation VPC テンプレート

AWS CloudFormation では、テンプレートファイルを使用してリソース群を単体 (スタック) としてまとめて作成および削除することで、AWS インフラストラクチャのデプロイを想定どおりに繰り返し作成およびプロビジョンできます。詳細については、AWS CloudFormation ユーザーガイドを参照してください。

Amazon VPC で AWS CodeBuild VPC 機能を使うよう設定するための AWS CloudFormation YAML テンプレートは次のとおりです。

```
Description:

    This template deploys a VPC, with a pair of public and private subnets spread
```

```
    across two Availability Zones. It deploys an Internet Gateway, with a default
    route on the public subnets. It deploys a pair of NAT Gateways (one in each AZ),
    and default routes for them in the private subnets.

Parameters:

    EnvironmentName:
        Description: An environment name that will be prefixed to resource names
        Type: String

    VpcCIDR:
        Description: Please enter the IP range (CIDR notation) for this VPC
        Type: String
        Default: 10.192.0.0/16

    PublicSubnet1CIDR:
        Description: Please enter the IP range (CIDR notation) for the public subnet in the
 first Availability Zone
        Type: String
        Default: 10.192.10.0/24

    PublicSubnet2CIDR:
        Description: Please enter the IP range (CIDR notation) for the public subnet in the
 second Availability Zone
        Type: String
        Default: 10.192.11.0/24

    PrivateSubnet1CIDR:
        Description: Please enter the IP range (CIDR notation) for the private subnet in
 the first Availability Zone
        Type: String
        Default: 10.192.20.0/24

    PrivateSubnet2CIDR:
```

```
        Description: Please enter the IP range (CIDR notation) for the private subnet in
 the second Availability Zone

        Type: String

        Default: 10.192.21.0/24


Resources:


    VPC:

        Type: AWS::EC2::VPC

        Properties:

            CidrBlock: !Ref VpcCIDR

            Tags:

                - Key: Name

                  Value: !Ref EnvironmentName


    InternetGateway:

        Type: AWS::EC2::InternetGateway

        Properties:

            Tags:

                - Key: Name

                  Value: !Ref EnvironmentName


    InternetGatewayAttachment:

        Type: AWS::EC2::VPCGatewayAttachment

        Properties:

            InternetGatewayId: !Ref InternetGateway

            VpcId: !Ref VPC


    PublicSubnet1:

        Type: AWS::EC2::Subnet

        Properties:

            VpcId: !Ref VPC

            AvailabilityZone: !Select [ 0, !GetAZs '' ]

            CidrBlock: !Ref PublicSubnet1CIDR
```

```
        MapPublicIpOnLaunch: true
        Tags:
            - Key: Name
              Value: !Sub ${EnvironmentName} Public Subnet (AZ1)

PublicSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
        VpcId: !Ref VPC
        AvailabilityZone: !Select [ 1, !GetAZs '' ]
        CidrBlock: !Ref PublicSubnet2CIDR
        MapPublicIpOnLaunch: true
        Tags:
            - Key: Name
              Value: !Sub ${EnvironmentName} Public Subnet (AZ2)

PrivateSubnet1:
    Type: AWS::EC2::Subnet
    Properties:
        VpcId: !Ref VPC
        AvailabilityZone: !Select [ 0, !GetAZs '' ]
        CidrBlock: !Ref PrivateSubnet1CIDR
        MapPublicIpOnLaunch: false
        Tags:
            - Key: Name
              Value: !Sub ${EnvironmentName} Private Subnet (AZ1)

PrivateSubnet2:
    Type: AWS::EC2::Subnet
    Properties:
        VpcId: !Ref VPC
        AvailabilityZone: !Select [ 1, !GetAZs '' ]
        CidrBlock: !Ref PrivateSubnet2CIDR
```

```
        MapPublicIpOnLaunch: false
        Tags:
            - Key: Name
              Value: !Sub ${EnvironmentName} Private Subnet (AZ2)

NatGateway1EIP:
    Type: AWS::EC2::EIP
    DependsOn: InternetGatewayAttachment
    Properties:
        Domain: vpc

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

NatGateway1:
    Type: AWS::EC2::NatGateway
    Properties:
        AllocationId: !GetAtt NatGateway1EIP.AllocationId
        SubnetId: !Ref PublicSubnet1

NatGateway2:
    Type: AWS::EC2::NatGateway
    Properties:
        AllocationId: !GetAtt NatGateway2EIP.AllocationId
        SubnetId: !Ref PublicSubnet2

PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
        VpcId: !Ref VPC
```

```
            Tags:
                - Key: Name
                  Value: !Sub ${EnvironmentName} Public Routes

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

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

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



    PrivateRouteTable1:
        Type: AWS::EC2::RouteTable
        Properties:
            VpcId: !Ref VPC
            Tags:
                - Key: Name
                  Value: !Sub ${EnvironmentName} Private Routes (AZ1)
```

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

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

  PrivateRouteTable2:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC
      Tags:
        - Key: Name
          Value: !Sub ${EnvironmentName} Private Routes (AZ2)

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

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

```
    NoIngressSecurityGroup:
        Type: AWS::EC2::SecurityGroup
        Properties:
            GroupName: "no-ingress-sg"
            GroupDescription: "Security group with no ingress rule"
            VpcId: !Ref VPC

Outputs:

    VPC:
        Description: A reference to the created VPC
        Value: !Ref VPC

    PublicSubnets:
        Description: A list of the public subnets
        Value: !Join [ ",", [ !Ref PublicSubnet1, !Ref PublicSubnet2 ]]

    PrivateSubnets:
        Description: A list of the private subnets
        Value: !Join [ ",", [ !Ref PrivateSubnet1, !Ref PrivateSubnet2 ]]

    PublicSubnet1:
        Description: A reference to the public subnet in the 1st Availability Zone
        Value: !Ref PublicSubnet1

    PublicSubnet2:
        Description: A reference to the public subnet in the 2nd Availability Zone
        Value: !Ref PublicSubnet2

    PrivateSubnet1:
        Description: A reference to the private subnet in the 1st Availability Zone
        Value: !Ref PrivateSubnet1
```

```
PrivateSubnet2:

    Description: A reference to the private subnet in the 2nd Availability Zone

    Value: !Ref PrivateSubnet2


NoIngressSecurityGroup:

    Description: Security group with no ingress rule

    Value: !Ref NoIngressSecurityGroup
```

AWS CodeBuild で AWS CodePipeline を使用してコードをテストし、ビルドを実行する

リリースプロセスを自動化するには、AWS CodePipeline を使用してコードをテストし、AWS CodeBuild でビルドを実行します。

次の表に示しているのは、タスクとその実行に使用できるメソッドです。これらのタスクを AWS SDK で達成する方法については、このトピックの対象外です。

タスク	使用可能なアプローチ	このトピックで説明するアプローチ
AWS CodeBuild でビルドを自動化する AWS CodePipeline を使用して、継続的な配信 (CD) パイプラインを作成する	• AWS CodePipeline コンソール • AWS CLI • AWS SDK	• AWS CodePipeline コンソールを使用して (p. 128) • AWS CLI の使用 (p. 132) • このトピックの情報は、AWS SDK を使用するように調整できます。詳細については、Amazon Web Services 用のツールの SDK セクションから、プログラミング言語のパイプラインアクションの作成ドキュメントを参照するか、または、AWS CodePipeline API リファレンス の `CreatePipeline` を参照してください。
既存の AWS CodePipeline のパイプラインに AWS CodeBuild でのテストおよびビルドの自動化の追加	• AWS CodePipeline コンソール • AWS CLI • AWS SDK	• AWS CodePipeline コンソールを使用してビルドの自動化を追加する (p. 135) • AWS CodePipeline コンソールを使用してテストの自動化を追加する (p. 140) • AWS CLI の場合、このトピックの情報を調整して、AWS CodeBuild ビルドアクションまたはテストアクションを含むパイプラインを作成できます。詳細については、AWS CodePipeline ユーザーガイド の「パイプラインの編集 (AWS CLI)」および「AWS CodePipeline パイプライン構造のリファレンス」を参照してください。 • このトピックの情報は、AWS SDKspipeline を使用するように調整できます。詳細については、Amazon Web Services 用のツールの SDK セクションから、プログラミング言語のパイプラインアクションの更新ドキュメントを参照するか、または、AWS CodePipeline API リファレンス の `UpdatePipeline` を参照してください。

トピック

- AWS CodeBuild ビルドアクションをパイプラインに追加する (AWS CodePipeline コンソール) (p. 135)
- AWS CodeBuild テストアクションをパイプラインに追加する (AWS CodePipeline コンソール) (p. 140)

前提条件

1. ビルドを計画する (p. 94) の質問に答えます。
2. AWS ルートアカウントや管理者 IAM ユーザーの代わりに、IAM ユーザーを AWS CodePipeline へのアクセスに使用している場合、`AWSCodePipelineFullAccess` にという名前の管理ポリシーをユーザー (またはユーザーが属する IAM グループ) にアタッチします。(AWS ルートアカウントを使用することは推奨されません。)これにより、ユーザーは AWS CodePipeline でパイプラインを作成できます。詳細については、IAM ユーザーガイドの「管理ポリシーをアタッチする」を参照してください。

   Note

   ポリシーをユーザー (またはユーザーが属する IAM グループ) にアタッチする IAM エンティティは、ポリシーをアタッチするために IAM でのアクセス許可を持っている必要があります。詳細については、IAM ユーザーガイドの「IAM ユーザー、グループ、および認証情報を管理するための権限の委任」を参照してください。
3. AWS アカウントに AWS CodePipeline のサービスロールがまだない場合は、作成します。このサービスロールにより、AWS CodePipeline は AWS CodeBuild を含む他の AWS のサービスとやり取りできます。たとえば、AWS CLI を使用して AWS CodePipeline のサービスロールを作成するには、IAM の `create-role` コマンドを実行します。

   Linux, macOS, or Unix 用:

   ```
   aws iam create-role --role-name AWS-CodePipeline-CodeBuild-Service-Role --assume-role-
   policy-document '{"Version":"2012-10-17","Statement":{"Effect":"Allow","Principal":
   {"Service":"codepipeline.amazonaws.com"},"Action":"sts:AssumeRole"}}'
   ```

   Windows の場合:

   ```
   aws iam create-role --role-name AWS-CodePipeline-CodeBuild-Service-Role --assume-role-
   policy-document "{\"Version\":\"2012-10-17\",\"Statement\":{\"Effect\":\"Allow\",
   \"Principal\":{\"Service\":\"codepipeline.amazonaws.com\"},\"Action\":\"sts:AssumeRole
   \"}}"
   ```

   Note

   この AWS CodePipeline のサービスロールを作成する IAM エンティティは、サービスロールを作成するために IAM のアクセス許可を持っている必要があります。
4. AWS CodePipeline のサービスロールを作成した後、または、既存のものを識別した後は、ポリシーステートメントを追加する必要があります。AWS CodePipeline ユーザーガイドの「デフォルトの AWS CodePipeline のサービスロールポリシーを確認する」の説明のとおりに、サービスロールにデフォルトの AWS CodePipeline のサービスロールポリシーを追加します。

   Note

   この AWS CodePipeline サービスロールのポリシーを追加する IAM エンティティは、サービスロールポリシーをサービスロールに追加するために IAM のアクセス許可を持っている必要があります。
5. AWS CodeCommit、Amazon S3 または GitHub など、AWS CodeBuild と AWS CodePipeline でサポートされているリポジトリタイプにソースコードを作成してアップロードします。(現在 AWS CodePipeline は Bitbucket をサポートしていません。)ソースコードにビルドスペックファイルが含ま

れていることを確認します (または、このトピックの後半でビルドプロジェクトを定義するときにビルドスペックを宣言できます)。そのファイルがソースコードを構築するための手順を提供します。詳細については、「ビルドスペックリファレンス (p. 95)」を参照してください。

Important

パイプラインを使用してビルドされたソースコードをデプロイする場合、ビルド出力アーティファクトは、使用するデプロイシステムと互換性がなければなりません。

- AWS CodeDeploy については、このガイドの AWS CodeDeploy サンプル (p. 61)、および、AWS CodeDeploy ユーザーガイドの「AWS CodeDeploy のリビジョンを準備する」を参照してください。
- AWS Elastic Beanstalk については、このガイドの Elastic Beanstalk サンプル (p. 65)、および、AWS Elastic Beanstalk 開発者ガイドの「アプリケーションソースバンドルを作成する」を参照してください。
- AWS OpsWorks については、AWS OpsWorks ユーザーガイドの「アプリケーションソース」および「AWS OpsWorks で AWS CodePipeline を使用する」を参照してください。

AWS CodeBuild を使用するパイプラインを作成する (AWS CodePipeline コンソール)

AWS CodeBuild を使用してソースコードをビルドおよびデプロイするパイプラインを作成するには、次の手順を実行します。

ソースコードのみをテストするパイプラインを作成するには、次のようなオプションがあります。

- 次の手順を使用してパイプラインを作成し、パイプラインからビルドステージとベータステージを削除します。次に、このトピックの AWS CodeBuild テストアクションをパイプラインに追加する (AWS CodePipeline コンソール) (p. 140) の手順を使用して、AWS CodeBuild を使用するアクションをパイプラインに追加します。
- このトピックの他の手順のいずれかを使用してパイプラインを作成した後、このトピックの AWS CodeBuild テストアクションをパイプラインに追加する (AWS CodePipeline コンソール) (p. 140) の手順を使用して AWS CodeBuild を使用するテストアクションをパイプラインに追加します。

AWS CodePipeline でパイプライン作成ウィザードを使用して、AWS CodeBuild を使用するパイプラインを作成するには

1. 「前提条件 (p. 127)」の各ステップを実行します。
2. https://console.aws.amazon.com/codepipeline で AWS CodePipeline コンソールを開きます。

   次のいずれかを使用して、AWS マネジメントコンソール に既にサインインしている必要があります。

   - AWS ルートアカウント。これは推奨されません。詳細については、IAM ユーザーガイドの「アカウント root ユーザー」を参照してください。
   - AWS アカウントの管理者 IAM ユーザー。詳細については、IAM ユーザーガイドの「最初の IAM 管理者のユーザーおよびグループの作成」を参照してください。
   - 次の最小限のアクションを使用する権限がある AWS アカウントの IAM ユーザー。

```
codepipeline:*
iam:ListRoles
iam:PassRole
s3:CreateBucket
s3:GetBucketPolicy
```

```
s3:GetObject
s3:ListAllMyBuckets
s3:ListBucket
s3:PutBucketPolicy
codecommit:ListBranches
codecommit:ListRepositories
codedeploy:GetApplication
codedeploy:GetDeploymentGroup
codedeploy:ListApplications
codedeploy:ListDeploymentGroups
elasticbeanstalk:DescribeApplications
elasticbeanstalk:DescribeEnvironments
lambda:GetFunctionConfiguration
lambda:ListFunctions
opsworks:DescribeStacks
opsworks:DescribeApps
opsworks:DescribeLayers
```

3. AWS リージョンセレクタで、パイプラインおよび関連する AWS リソースが配置されているリージョンを選択します。このリージョンも AWS CodeBuild をサポートしている必要があります。詳細については、アマゾン ウェブ サービス全般のリファレンスの「リージョンとエンドポイント」トピックにある「AWS CodeBuild」を参照してください。
4. 次のようにパイプラインを作成します。

   ウェルカムページが表示された場合は、[Get started] を選択します。

   [All Pipelines] ページが表示された場合は、[Create pipeline] を選択します。
5. [Step 1: Name] ページで、[Pipeline name] にパイプライン名を入力します (例: **`CodeBuildDemoPipeline`**)。別の名前を選択する場合は、この手順全体でそれを置き換えてください。[Next step] を選択します。
6. [Step 2: Source] ページの [Source provider] で、次のいずれかの操作を行います。

   - ソースコードが Amazon S3 バケットに保存されている場合は、[Amazon S3] を選択します。[Amazon S3 location] に、ソースコードへのパスを `s3://`*`bucket-name`*`/`*`path`*`/`*`to`*`/`*`file-name`*`.zip` の形式で入力します。[Next step] を選択します。
   - ソースコードが AWS CodeCommit リポジトリに保存されている場合は、[AWS CodeCommit] を選択します。[Repository name] で、ソースコードが含まれているリポジトリの名前を選択します。[Branch name] で、ビルドするソースコードのバージョンを表すブランチの名前を選択します。[Next step] を選択します。
   - ソースコードが GitHub リポジトリに保存されている場合は、[GitHub] を選択します。[Connect to GitHub] を選択し、手順に従って GitHub に対して認証します。[Repository] で、ソースコードが含まれているリポジトリの名前を選択します。[Branch] で、ビルドするソースコードのバージョンを表すブランチの名前を選択します。[Next step] を選択します。
7. [Step 3: Build] ページの [Build provider] で、[AWS CodeBuild] を選択します。
8. 既存のビルドプロジェクトを使用する場合は、[Select an existing build project] を選択します。[Project name] で、ビルドプロジェクトの名前を選択し、この手順のステップ 17 に進みます。

   Note

   既存のビルドプロジェクトを選択した場合、ビルド出力アーティファクトの設定がすでに定義されている必要があります (ただし、AWS CodePipeline によってビルド出力の設定が上書きされます)。詳細については、「ビルドプロジェクトの作成 (コンソール) (p. 146)」または「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」の [Artifacts: Where to put the artifacts from this build project] を参照してください。

   Important

   AWS CodeBuild プロジェクトのウェブフックを有効にして、プロジェクトを AWS CodePipeline のビルドステップとして使用すると、コミットごとに 2 つの等しいビルド

が作成されます。1 つのビルドはウェブフックを通じてトリガーされ、別の 1 つは AWS CodePipeline を通じてトリガーされます。請求はビルド単位で発生するため、両方のビルドに対して課金されます。したがって、AWS CodePipeline を使用する場合は、CodeBuild でウェブフックを無効にすることをお勧めします。AWS CodeBuild コンソールで、[Webhook] ボックスをオフにします。詳細については、「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」のステップ 9 を参照してください。

9. [Create a new build project] を選択します。
10. [Project name] に、このビルドプロジェクトの名前を入力します。ビルドプロジェクトの名前は、各 AWS アカウントで一意である必要があります。
11. (オプション) [Description] ボックスに説明を入力します。
12. [Environment image] で、次のいずれかの操作を行います。

    - AWS CodeBuild で管理される Docker イメージに基づいてビルド環境を使用するには、[Use an image managed by AWS CodeBuild] を選択します。[Operating system]、[Runtime]、および [Version] の各ドロップダウンリストで適切な選択を行います。詳細については、「AWS CodeBuild に用意されている Docker イメージ (p. 103)」を参照してください。
    - AWS アカウントの Amazon ECR リポジトリの Docker イメージに基づいてビルド環境を使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Amazon ECR] を選択します。[Amazon ECR repository] および [Amazon ECR image] の各ドロップダウンリストを使用して、必要な Amazon ECR リポジトリおよびそのリポジトリ内の Docker イメージを指定します。
    - Docker Hub で公開されている Docker イメージに基づいてビルド環境を使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Other] を選択します。[Custom image ID] ボックスに、Docker イメージ ID を *docker-repo-name*/*docker-image-name*:*tag* の形式で入力します。

13. [Build specification] で、次のいずれかの操作を行います。

    - ソースコードにビルドスペックファイルが含まれている場合は、[Use the buildspec.yml in the source code root directory] を選択します。
    - ソースコードにビルドスペックファイルが含まれていない場合は、[Insert build commands] を選択します。[Build command] に、ビルド環境のビルドフェーズで実行するコマンドを入力します。複数を入力する場合は、各コマンドを && で区切ります。[Output files] に、ビルド環境で AWS CodePipeline に送信するビルド出力ファイルへのパスを入力します。複数を入力する場合は、各ファイルパスをカンマで区切ります。詳細については、コンソールのツールヒントを参照してください。

14. [AWS CodeBuild service role] で、次のいずれかの操作を行います。

    - AWS アカウントに AWS CodeBuild サービスロールがない場合は、[Create a service role in your account] を選択します。[Role name] ボックスに、サービスロールの名前を入力するか、提案された名前をそのまま使用します。(サービスロール名は AWS アカウント内で一意でなければなりません)。

      Note

      コンソールを使用して AWS CodeBuild サービスロールを作成する場合、デフォルトでは、このサービスロールはこのビルドプロジェクトでのみ動作します。コンソールを使用して、サービスロールを別のビルドプロジェクトと関連付けると、このロールは他のビルドプロジェクトと連携するように更新されます。1 つの AWS CodeBuild サービスロールは最大 10 個のビルドプロジェクトと連携できます。

    - AWS アカウントに AWS CodeBuild サービスロールがある場合は、[Choose an existing service role from your account] を選択します。[Role name] ボックスで、サービスロールの名前を選択します。

15. [Advanced] を展開します。

    60 分 (デフォルト) 以外のビルドタイムアウトを指定するには、[hours] ボックスと [minutes] ボックスを使用してタイムアウトを 5〜480 分 (8 時間) に設定します。

(オプション) [Privileged] チェックボックスをオンにするのは、このビルドプロジェクトを使用して Docker イメージを作成する予定であり、Docker をサポートする AWS CodeBuild に用意されているビルド環境イメージ以外のイメージを選択している場合に限ります。それ以外の場合、関連付けられているビルドで Docker デーモンと通信しようとすると、すべて失敗します。必要に応じて、ビルドで Docker デーモンを操作できるように、Docker デーモンも起動する必要があります。これを行う 1 つの方法は、以下のビルドコマンドを実行してビルドスペックの `install` フェーズで Docker デーモンを初期化することです。(Docker をサポートする AWS CodeBuild に用意されているビルド環境イメージを選択した場合は、以下のビルドコマンドを実行しないでください。)

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://0.0.0.0:2375 --storage-driver=overlay&
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

[Compute] で、以下のいずれかの利用可能なオプションを選択します。

[Environment variables] で、[Name] および [Value] を使用して、使用するビルド環境のオプションの環境変数を指定します。さらに環境変数を追加するには、[Add row] を選択します。

> **Important**
>
> 機密情報、特に AWS アクセスキー ID とシークレットアクセスキーを格納する場合には、環境変数を使用しないことを強くお勧めします。環境変数は、AWS CodeBuild コンソールや AWS CLI などのツールを使用してプレーンテキストで表示できます。
> 重要な値を保存および取得するには、ビルドコマンドで AWS CLI を使用して Amazon EC2 Systems Manager パラメータストアを操作することをお勧めします。AWS CLI は、AWS CodeBuild に用意されているすべての環境変数でプレインストール済みで構成済みになっています。詳細については、Amazon EC2 Systems Manager ユーザーガイドの「Systems Manager パラメータストア」および「Systems Manager パラメータストア CLI のチュートリアル」を参照してください。

16. ビルドプロジェクトの保存 を選択します。
17. ビルドプロジェクトを保存したら、[Next step] を選択します。
18. [Step 4: Deploy] ページで、次のいずれかの操作を行います。

    - ビルド出力アーティファクトをデプロイしない場合は、[Deployment provider] で [No Deployment] を選択します。
    - ビルド出力アーティファクトをデプロイする場合は、[Deployment provider] でデプロイプロバイダを選択し、次にプロンプトに応じて設定を指定します。

19. [Next step] を選択します。
20. [Step 5: Service Role] ページの [Role name] で、このトピックの前提条件として作成または識別した AWS CodePipeline サービスロールを選択します。

    このページを使用して AWS CodePipeline のサービスロールを作成しないでください。これを行うと、サービスロールに AWS CodeBuild での作業に必要なアクセス許可が与えられません。
21. [Next step] を選択します。
22. [Step 6: Review] ページで、[Create pipeline] を選択します。
23. パイプラインが正常に実行されたら、ビルド出力アーティファクトを取得できます。パイプラインが表示されている AWS CodePipeline コンソールの [Build] アクションで、マウスポインタをツールヒントに合わせます。[Output artifact] の値をメモします (例: MyAppBuild)。

    > **Note**
    >
    > ビルド出力アーティファクトを取得するには、AWS CodeBuild コンソールのビルドの詳細ページで [Build artifacts] リンクを選択することもできます。このページを表示するには、この手順の残りのステップを省略して、「ビルドの詳細の表示 (コンソール) (p. 174)」を参照してください。

24. https://console.aws.amazon.com/s3/ にある Amazon S3 コンソールを開きます。
25. バケットのリストで、パイプラインで使用されるバケットを開きます。バケット名は、`codepipeline-`*`region-ID`*`-`*`random-number`* の形式に従う必要があります。AWS CLI を使用して、AWS CodePipeline `get-pipeline` コマンドを実行すると、バケットの名前を取得できます。*`my-pipeline-name`* は、パイプラインの表示名です。

```
aws codepipeline get-pipeline --name my-pipeline-name
```

出力では、`pipeline` オブジェクトには `artifactStore` オブジェクトが含まれ、それには、バケットの名前と `location` の値が含まれます。

26. パイプラインの名前と一致するフォルダを開きます (パイプライン名の長さによってはフォルダ名が切り捨てられる場合があります)。次に、この手順のステップ 23 でメモした [Output artifact] の値に一致するフォルダを開きます。
27. ファイルの内容を展開します。そのフォルダに複数のファイルがある場合は、[Last Modified] タイムスタンプが最新であるファイルの内容を抽出します。(システムの ZIP ユーティリティで操作できるように、必要に応じて、ファイルに `.zip` 拡張子を付けます。)ビルド出力アーティファクトは、展開されたファイルの内容に含まれます。
28. AWS CodePipeline にビルド出力アーティファクトをデプロイするよう指示した場合は、デプロイプロバイダの説明を活用してデプロイターゲットのビルド出力アーティファクトを取得します。

AWS CodeBuild を使用するパイプラインを作成する (AWS CLI)

AWS CodeBuild を使用してソースコードをビルドするパイプラインを作成するには、次の手順を実行します。

AWS CLI を使用して、ビルドされたソースコードをデプロイする、または、ソースコードのテストのみを行うパイプラインを作成するには、AWS CodePipeline ユーザーガイドの「パイプラインの編集 (AWS CLI)」および「AWS CodePipeline パイプライン構造リファレンス」を参照してください。

1. 「前提条件 (p. 127)」の各ステップを実行します。
2. AWS CodeBuild でビルドプロジェクトを作成または識別します。詳細については、「ビルドプロジェクトを作成する (p. 146)」を参照してください。

   Important

   ビルドプロジェクトは、ビルド出力アーティファクトの設定を定義する必要があります (ただし、AWS CodePipeline が設定を上書きします)。詳細については、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」の `artifacts` の説明を参照してください。
3. このトピックで説明している IAM エンティティの 1 つに対応する AWS アクセスキーと AWS シークレットアクセスキーで AWS CLI を設定していることを確認してください。詳細については、AWS Command Line Interface ユーザーガイドの「AWS Command Line Interface のセットアップ」を参照してください。
4. パイプラインの構造を表す JSON 形式のファイルを作成します。ファイルに `create-pipeline.json` のような名前を付けます。たとえば、この JSON 形式の構造は、Amazon S3 入力バケットを参照するソースアクションと、AWS CodeBuild を使用するビルドアクションを持つパイプラインを作成します。

```
{
 "pipeline": {
  "roleArn": "arn:aws:iam::account-id:role/my-AWS-CodePipeline-service-role-name",
  "stages": [
    {
```

```
      "name": "Source",
      "actions": [
        {
          "inputArtifacts": [],
          "name": "Source",
          "actionTypeId": {
            "category": "Source",
            "owner": "AWS",
            "version": "1",
            "provider": "S3"
          },
          "outputArtifacts": [
            {
              "name": "MyApp"
            }
          ],
          "configuration": {
            "S3Bucket": "my-input-bucket-name",
            "S3ObjectKey": "my-source-code-file-name.zip"
          },
          "runOrder": 1
        }
      ]
    },
    {
      "name": "Build",
      "actions": [
        {
          "inputArtifacts": [
            {
              "name": "MyApp"
            }
          ],
          "name": "Build",
          "actionTypeId": {
            "category": "Build",
            "owner": "AWS",
            "version": "1",
            "provider": "AWS CodeBuild"
          },
          "outputArtifacts": [
      {
              "name": "default"
            }
    ],
          "configuration": {
            "ProjectName": "my-build-project-name"
          },
          "runOrder": 1
        }
      ]
    }
  ],
  "artifactStore": {
    "type": "S3",
    "location": "AWS-CodePipeline-internal-bucket-name"
  },
  "name": "my-pipeline-name",
  "version": 1
 }
}
```

この JSON 形式のデータは以下のようになっています。

- `roleArn` の値は、前提条件の一部として作成または識別した AWS CodePipeline のサービスロールの ARN と一致する必要があります。
- `configuration` の `S3Bucket` と `S3ObjectKey` の値は、ソースコードが Amazon S3 バケットに保存されることを想定しています。その他のソースコードのリポジトリタイプの設定については、AWS CodePipeline ユーザーガイドの「AWS CodePipeline パイプライン構造のリファレンス」を参照してください。
- `ProjectName` の値は、この手順の前半で作成した AWS CodeBuild ビルドプロジェクトの名前です。
- `location` の値は、このパイプラインで使用する Amazon S3 バケットの名前です。詳細については、AWS CodePipeline ユーザーガイドの「AWS CodePipeline のアーティファクトストアとして使用する Amazon S3 バケットのポリシーを作成する」を参照してください。
- `name` の値は、このパイプラインの名前です。すべてのパイプラインの名前はアカウントに対して一意である必要があります。

このデータにはソースアクションとビルドアクションのみが記述されていますが、テスト、ビルド出力アーティファクトのデプロイ、AWS Lambda 関数の呼び出しなどに関連したアクティビティのためにアクションを追加できます。詳細については、AWS CodePipeline ユーザーガイドの「AWS CodePipeline パイプライン構造のリファレンス」を参照してください。

5. JSON ファイルが保存されているフォルダに切り替え、AWS CodePipeline `create-pipeline` コマンドを実行し、ファイル名を指定します。

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

> **Note**
>
> AWS CodeBuild をサポートする AWS リージョンにパイプラインを作成する必要があります。詳細については、アマゾン ウェブ サービス全般のリファレンスの「リージョンとエンドポイント」トピックにある「AWS CodeBuild」を参照してください。

JSON 形式のデータが出力に表示され、AWS CodePipeline がパイプラインを作成します。

6. パイプラインのステータスに関する情報を取得するには、AWS CodePipeline `get-pipeline-state` コマンドを実行して、パイプラインの名前を指定します。

```
aws codepipeline get-pipeline-state --name my-pipeline-name
```

出力で、ビルドが成功したことを確認する情報を探します。省略記号 (`...`) は、簡潔にするために省略されたデータを表すために使用されます。

```
{
  ...
  "stageStates": [
    ...
    {
      "actionStates": [
        {
          "actionName": "AWS CodeBuild",
          "latestExecution": {
            "status": "SUCCEEDED",
            ...
          },
          ...
        }
      ]
    }
  ]
```

```
}
```

このコマンドをあまりに早く実行すると、ビルドアクションに関する情報が表示されないことがあります。パイプラインがビルドアクションの実行を終了するまで、このコマンドを複数回実行する必要があります。

7. ビルドが成功したら、次の手順に従ってビルド出力アーティファクトを取得します。https://console.aws.amazon.com/s3/ にある Amazon S3 コンソールを開きます。

   Note

   ビルド出力アーティファクトを取得するには、AWS CodeBuild コンソールの関連するビルドの詳細ページで [Build artifacts] リンクを選択することもできます。このページを表示するには、この手順の残りのステップを省略して、「ビルドの詳細の表示 (コンソール) (p. 174)」を参照してください。

8. バケットのリストで、パイプラインで使用されるバケットを開きます。バケット名は、`codepipeline-`*`region-ID`*`-`*`random-number`* の形式に従う必要があります。`create-pipeline.json` ファイルからバケット名を取得するか、または、AWS CodePipeline `get-pipeline` コマンドを実行してバケット名を取得します。

```
aws codepipeline get-pipeline --name my-pipeline-name
```

出力では、`pipeline` オブジェクトには `artifactStore` オブジェクトが含まれ、それには、バケットの名前と `location` の値が含まれます。

9. パイプラインの名前と一致するフォルダを開きます (例:*`my-pipeline-name`*)。
10. そのフォルダで、`default` という名前のフォルダを開きます。
11. ファイルの内容を展開します。そのフォルダに複数のファイルがある場合は、[Last Modified] タイムスタンプが最新であるファイルの内容を抽出します。(システムの ZIP ユーティリティで操作できるように、必要に応じて、ファイルに `.zip` 拡張子を付けます。)ビルド出力アーティファクトは、展開されたファイルの内容に含まれます。

AWS CodeBuild ビルドアクションをパイプラインに追加する (AWS CodePipeline コンソール)

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

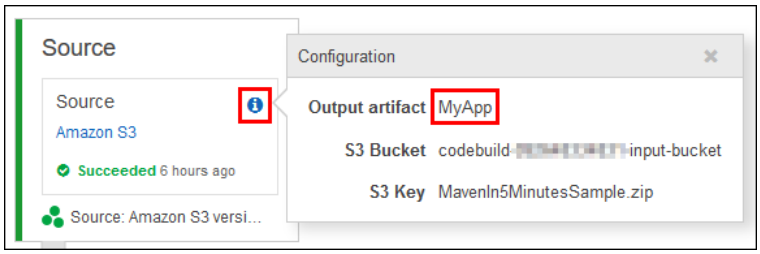
   次のいずれかを使用して、AWS マネジメントコンソール に既にサインインしている必要があります。

   - AWS ルートアカウント。これは推奨されません。詳細については、IAM ユーザーガイドの「アカウント root ユーザー」を参照してください。
   - AWS アカウントの管理者 IAM ユーザー。詳細については、IAM ユーザーガイドの「最初の IAM 管理者ユーザーおよびグループの作成」を参照してください。
   - 次の最小限のアクションを実行する権限がある AWS アカウントの IAM ユーザー。

```
codepipeline:*
iam:ListRoles
iam:PassRole
s3:CreateBucket
s3:GetBucketPolicy
s3:GetObject
s3:ListAllMyBuckets
s3:ListBucket
s3:PutBucketPolicy
```

```
codecommit:ListBranches
codecommit:ListRepositories
codedeploy:GetApplication
codedeploy:GetDeploymentGroup
codedeploy:ListApplications
codedeploy:ListDeploymentGroups
elasticbeanstalk:DescribeApplications
elasticbeanstalk:DescribeEnvironments
lambda:GetFunctionConfiguration
lambda:ListFunctions
opsworks:DescribeStacks
opsworks:DescribeApps
opsworks:DescribeLayers
```

2. AWS リージョンセレクタで、パイプラインが配置されているリージョンを選択します。このリージョンも AWS CodeBuild をサポートしている必要があります。詳細については、アマゾン ウェブ サービス全般のリファレンスの「リージョンとエンドポイント」トピックにある「AWS CodeBuild」を参照してください。
3. [All Pipelines] ページで、パイプラインの名前を選択します。
4. パイプラインの詳細ページの [Source] アクションで、マウスポインタをツールヒントに合わせます。[Output artifact] の値をメモします (例: MyApp)。



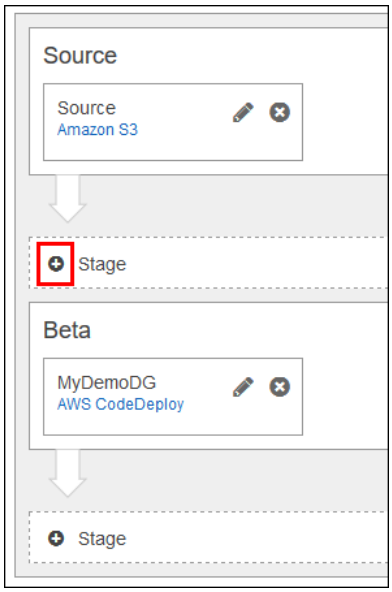
Note

この手順では、[Source] ステージと [Beta] ステージの間のビルドステージ内にビルドアクションを追加することを想定しています。ビルドアクションを別の場所に追加する場合は、ビルドアクションを追加する場所の直前のアクションにマウスポインタを合わせ、[Output artifact] の値をメモします。

5. [Edit] を選択します。
6. [Source] ステージと [Beta] ステージの間で、[Stage] の横にある追加記号 (+) を選択します。

Note

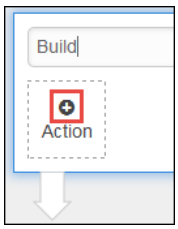
この手順では、パイプラインに新しいビルドステージを追加することを想定しています。既存のステージにビルドアクションを追加するには、既存のステージで編集 (鉛筆) アイコンを選択し、この手順のステップ 8 に進みます。
また、この手順では、[Source] ステージと [Beta] ステージの間にビルドステージを追加することを想定しています。ビルドステージを別の場所に追加するには、目的の場所で追加記号を選択します。



7. [Enter stage name] で、ビルドステージの名前を入力します (例: **`Build`**)。別の名前を選択する場合は、この手順全体でそれを使用します。
8. 選択したステージ内で、[Action] の横にある追加記号 (+) を選択します。

   Note

   この手順では、ビルドステージの中にビルドアクションを追加することを想定しています。ビルドアクションを別の場所に追加するには、目的の場所で追加記号を選択します。まず、ビルドアクションを追加する既存のステージで編集 (鉛筆) アイコンを選択する必要があります。



9. [Add action] ペインの [Action category] で、[Build] を選択します。
10. [Build actions] の [Action name] に、アクション名を入力します (例: **`AWS CodeBuild`**)。別の名前を選択する場合は、この手順全体でそれを使用します。
11. [Build provider] で、[AWS CodeBuild] を選択します。
12. AWS CodeBuild に既存のビルドプロジェクトがある場合は、[Select an existing build project] を選択します。[Project name] で、ビルドプロジェクトの名前を選択し、この手順のステップ 21 に進みます。

    Note

    既存のビルドプロジェクトを選択した場合、ビルド出力アーティファクトの設定がすでに定義されている必要があります (ただし、AWS CodePipeline によってビルド出力の設定が上書きされます)。詳細については、「ビルドプロジェクトの作成 (コンソール) (p. 146)」または「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」の [Artifacts: Where to put the artifacts from this build project] を参照してください。

Important

AWS CodeBuild プロジェクトのウェブフックを有効にして、プロジェクトを AWS CodePipeline のビルドステップとして使用すると、コミットごとに 2 つの等しいビルドが作成されます。1 つのビルドはウェブフックを通じてトリガーされ、別の 1 つは AWS CodePipeline を通じてトリガーされます。請求はビルド単位で発生するため、両方のビルドに対して課金されます。したがって、AWS CodePipeline を使用する場合は、CodeBuild でウェブフックを無効にすることをお勧めします。AWS CodeBuild コンソールで、[Webhook] ボックスをオフにします。詳細については、「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」のステップ 9 を参照してください。

13. [Create a new build project] を選択します。
14. [Project name] に、このビルドプロジェクトの名前を入力します。ビルドプロジェクトの名前は、各 AWS アカウントで一意である必要があります。
15. (オプション) [Description] ボックスに説明を入力します。
16. [Environment image] で、次のいずれかの操作を行います。

    - AWS CodeBuild で管理される Docker イメージに基づいてビルド環境を使用するには、[Use an image managed by AWS CodeBuild] を選択します。[Operating system]、[Runtime]、および [Version] の各ドロップダウンリストで適切な選択を行います。詳細については、「AWS CodeBuild に用意されている Docker イメージ (p. 103)」を参照してください。
    - AWS アカウントの Amazon ECR リポジトリの Docker イメージに基づいてビルド環境を使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Amazon ECR] を選択します。[Amazon ECR repository] および [Amazon ECR image] の各ドロップダウンリストを使用して、必要な Amazon ECR リポジトリおよびそのリポジトリ内の Docker イメージを指定します。
    - Docker Hub の Docker イメージに基づいてビルド環境を使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Other] を選択します。[Custom image ID] ボックスに、Docker イメージ ID を *docker-repo-name*/*docker-image-name*:*tag* の形式で入力します。

17. [Build specification] で、次のいずれかの操作を行います。

    - ソースコードにビルドスペックファイルが含まれている場合は、[Use the buildspec.yml in the source code root directory] を選択します。
    - ソースコードにビルドスペックファイルが含まれていない場合は、[Insert build commands] を選択します。[Build command] に、ビルド環境のビルドフェーズで実行するコマンドを入力します。複数を入力する場合は、各コマンドを && で区切ります。[Output files] に、ビルド環境で AWS CodePipeline に送信するビルド出力ファイルへのパスを入力します。複数を入力する場合は、各ファイルパスをカンマで区切ります。詳細については、コンソールのツールヒントを参照してください。

18. [AWS CodeBuild service role] で、次のいずれかの操作を行います。

    - AWS アカウントに AWS CodeBuild サービスロールがない場合は、[Create a service role in your account] を選択します。[Role name] ボックスに、サービスロールの名前を入力するか、提案された名前をそのまま使用します。(サービスロール名は AWS アカウント内で一意でなければなりません)。

      Note

      コンソールを使用して AWS CodeBuild サービスロールを作成する場合、デフォルトでは、このサービスロールはこのビルドプロジェクトでのみ動作します。コンソールを使用して、サービスロールを別のビルドプロジェクトと関連付けると、このロールは他のビルドプロジェクトと連携するように更新されます。1 つの AWS CodeBuild サービスロールは最大 10 個のビルドプロジェクトと連携できます。

    - AWS アカウントに AWS CodeBuild サービスロールがある場合は、[Choose an existing service role from your account] を選択します。[Role name] ボックスで、サービスロールの名前を選択します。

19. [Advanced] を展開します。

60 分 (デフォルト) 以外のビルドタイムアウトを指定するには、[hours] ボックスと [minutes] ボックスを使用して 5〜480 分 (8 時間) のタイムアウトを指定します。

[Compute] で、以下のいずれかの利用可能なオプションを選択します。

(オプション) [Privileged] チェックボックスをオンにするのは、このビルドプロジェクトを使用して Docker イメージを作成する予定であり、Docker をサポートする AWS CodeBuild に用意されているビルド環境イメージ以外のイメージを選択している場合に限ります。それ以外の場合、関連付けられているビルドで Docker デーモンと通信しようとすると、すべて失敗します。必要に応じて、ビルドで Docker デーモンを操作できるように、Docker デーモンも起動する必要があります。これを行う 1 つの方法は、以下のビルドコマンドを実行してビルドスペックの `install` フェーズで Docker デーモンを初期化することです。(Docker をサポートする AWS CodeBuild に用意されているビルド環境イメージを選択した場合は、以下のビルドコマンドを実行しないでください。)

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://0.0.0.0:2375 --storage-driver=overlay&
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

[Environment variables] で、[Name] および [Value] を使用して、使用するビルド環境のオプションの環境変数を指定します。さらに環境変数を追加するには、[Add row] を選択します。

> **Important**
>
> 機密情報、特に AWS アクセスキー ID とシークレットアクセスキーを格納する場合には、環境変数を使用しないことを強くお勧めします。環境変数は、AWS CodeBuild コンソールや AWS CLI などのツールを使用してプレーンテキストで表示できます。
> 重要な値を保存および取得するには、ビルドコマンドで AWS CLI を使用して Amazon EC2 Systems Manager パラメータストアを操作することをお勧めします。AWS CLI は、AWS CodeBuild に用意されているすべての環境変数でプレインストール済みで構成済みになっています。詳細については、Amazon EC2 Systems Manager ユーザーガイドの「Systems Manager パラメータストア」および「Systems Manager パラメータストア CLI のチュートリアル」を参照してください。

20. ビルドプロジェクトの保存 を選択します。
21. [Input artifact #1] で、この手順のステップ 4 でメモした [Output artifact] の値を入力します。
22. [Output artifact #1] で、出力アーティファクトの名前を入力します (例: **`MyAppBuild`**)。
23. [Add action] を選択します。
24. [Save pipeline changes]、[Save and continue] の順に選択します。
25. [Release change] を選択します。
26. パイプラインが正常に実行されたら、ビルド出力アーティファクトを取得できます。パイプラインが表示されている AWS CodePipeline コンソールの [Build] アクションで、マウスポインタをツールヒントに合わせます。[Output artifact] の値をメモします (例: MyAppBuild)。

    > **Note**
    >
    > ビルド出力アーティファクトを取得するには、AWS CodeBuild コンソールのビルドの詳細ページで [Build artifacts] リンクを選択することもできます。このページの内容を表示するには、「ビルドの詳細の表示 (コンソール) (p. 174)」を参照して、この手順のステップ 31 に進みます。

27. https://console.aws.amazon.com/s3/ にある Amazon S3 コンソールを開きます。
28. バケットのリストで、パイプラインで使用されるバケットを開きます。バケット名は、`codepipeline-`*`region-ID`*`-`*`random-number`* の形式に従う必要があります。AWS CLI を使用して AWS CodePipeline `get-pipeline` コマンドを実行すると、バケットの名前を取得できます。

```
aws codepipeline get-pipeline --name my-pipeline-name
```

出力では、`pipeline` オブジェクトには `artifactStore` オブジェクトが含まれ、それには、バケットの名前と `location` の値が含まれます。

29. パイプラインの名前と一致するフォルダを開きます (パイプライン名の長さによってはフォルダ名が切り捨てられる場合があります)。次に、この手順のステップ 26 でメモした [Output artifact] の値に一致するフォルダを開きます。
30. ファイルの内容を展開します。そのフォルダに複数のファイルがある場合は、[Last Modified] タイムスタンプが最新であるファイルの内容を抽出します。(システムの ZIP ユーティリティで操作できるように、必要に応じて、ファイルに `.zip` 拡張子を付けます。)ビルド出力アーティファクトは、展開されたファイルの内容に含まれます。
31. AWS CodePipeline にビルド出力アーティファクトをデプロイするよう指示した場合は、デプロイプロバイダの説明を活用してデプロイターゲットのビルド出力アーティファクトを取得します。

AWS CodeBuild テストアクションをパイプラインに追加する (AWS CodePipeline コンソール)

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

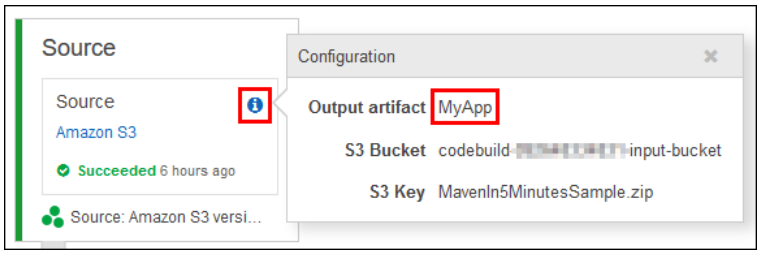
   次のいずれかを使用して、AWS マネジメントコンソール に既にサインインしている必要があります。

   - AWS ルートアカウント。これは推奨されません。詳細については、IAM ユーザーガイドの「アカウント root ユーザー」を参照してください。
   - AWS アカウントの管理者 IAM ユーザー。詳細については、IAM ユーザーガイドの「最初の IAM 管理者ユーザーおよびグループの作成」を参照してください。
   - 次の最小限のアクションを実行する権限がある AWS アカウントの IAM ユーザー。

   ```
   codepipeline:*
   iam:ListRoles
   iam:PassRole
   s3:CreateBucket
   s3:GetBucketPolicy
   s3:GetObject
   s3:ListAllMyBuckets
   s3:ListBucket
   s3:PutBucketPolicy
   codecommit:ListBranches
   codecommit:ListRepositories
   codedeploy:GetApplication
   codedeploy:GetDeploymentGroup
   codedeploy:ListApplications
   codedeploy:ListDeploymentGroups
   elasticbeanstalk:DescribeApplications
   elasticbeanstalk:DescribeEnvironments
   lambda:GetFunctionConfiguration
   lambda:ListFunctions
   opsworks:DescribeStacks
   opsworks:DescribeApps
   opsworks:DescribeLayers
   ```

2. AWS リージョンセレクタで、パイプラインが配置されているリージョンを選択します。このリージョンも AWS CodeBuild をサポートしている必要があります。詳細については、アマゾン ウェブ サービス全般のリファレンスの「リージョンとエンドポイント」トピックにある「AWS CodeBuild」を参照してください。
3. [All Pipelines] ページで、パイプラインの名前を選択します。

4. パイプラインの詳細ページの [Source] アクションで、マウスポインタをツールヒントに合わせます。[Output artifact] の値をメモします (例: MyApp)。



Note

この手順では、[Source] ステージと [Beta] ステージの間のテストステージ内にテストアクションを追加することを想定しています。テストアクションを別の場所に追加する場合は、直前のアクションにマウスポインタを合わせ、[Output artifact] の値をメモします。

5. [Edit] を選択します。
6. [Source] ステージのすぐ後で、[Stage] の横にある追加 (+) を選択します。

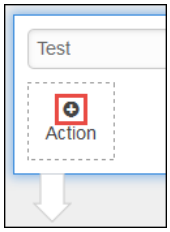
Note

この手順では、パイプラインにテストステージを追加することを想定しています。既存のステージにテストアクションを追加するには、既存のステージで編集 (鉛筆) アイコンを選択し、この手順のステップ 8 に進みます。
また、この手順では、[Source] ステージのすぐ後にテストステージを追加することを想定しています。テストステージを別の場所に追加するには、目的の場所で追加記号を選択します。

7. [Enter stage name] で、テストステージの名前を入力します (例: **`Test`**)。別の名前を選択する場合は、この手順全体でそれを使用します。
8. 選択したステージ内で、[Action] の横にある追加 (+) を選択します。

Note

この手順では、テストステージの中にテストアクションを追加することを想定しています。テストアクションを別の場所に追加するには、目的の場所で追加記号を選択します。まず、テストアクションを追加する既存のステージで編集 (鉛筆) アイコンを選択する必要があります。



9. [Add action] ペインの [Action category] で、[Test] を選択します。

10. [Test actions] の [Action name] に、アクション名を入力します (例: **`Test`**)。別の名前を選択する場合は、この手順全体でそれを使用します。
11. [Test provider] で、[AWS CodeBuild] を選択します。
12. AWS CodeBuild に既存のビルドプロジェクトがある場合は、[Select an existing build project] を選択します。[Project name] で、ビルドプロジェクトの名前を選択し、この手順のステップ 21 に進みます。

    Important

    AWS CodeBuild プロジェクトのウェブフックを有効にして、プロジェクトを AWS CodePipeline のビルドステップとして使用すると、コミットごとに 2 つの等しいビルドが作成されます。1 つのビルドはウェブフックを通じてトリガーされ、別の 1 つは AWS CodePipeline を通じてトリガーされます。請求はビルド単位で発生するため、両方のビルドに対して課金されます。したがって、AWS CodePipeline を使用する場合は、CodeBuild でウェブフックを無効にすることをお勧めします。AWS CodeBuild コンソールで、[Webhook] ボックスをオフにします。詳細については、「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」のステップ 9 を参照してください。
13. [Create a new build project] を選択します。
14. [Project name] に、このビルドプロジェクトの名前を入力します。ビルドプロジェクトの名前は、各 AWS アカウントで一意である必要があります。
15. (オプション) [Description] ボックスに説明を入力します。
16. [Environment image] で、次のいずれかの操作を行います。

    - AWS CodeBuild で管理される Docker イメージに基づいてビルド環境を使用するには、[Use an image managed by AWS CodeBuild] を選択します。[Operating system]、[Runtime]、および [Version] の各ドロップダウンリストで適切な選択を行います。詳細については、「AWS CodeBuild に用意されている Docker イメージ (p. 103)」を参照してください。
    - AWS アカウントの Amazon ECR リポジトリの Docker イメージに基づいてビルド環境を使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Amazon ECR] を選択します。[Amazon ECR repository] および [Amazon ECR image] の各ドロップダウンリストを使用して、必要な Amazon ECR リポジトリおよびそのリポジトリ内の Docker イメージを指定します。
    - Docker Hub の Docker イメージに基づいてビルド環境を使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Other] を選択します。[Custom image ID] ボックスに、Docker イメージ ID を *`docker-repo-name`*`/`*`docker-image-name`*`:`*`tag`* の形式で入力します。
17. [Build specification] で、次のいずれかの操作を行います。

    - ソースコードにビルドスペックファイルが含まれている場合は、[Use the buildspec.yml in the source code root directory] を選択します。
    - ソースコードにビルドスペックファイルが含まれていない場合は、[Insert build commands] を選択します。[Build command] で、ビルド環境のビルドフェーズで実行するコマンドを入力します。複数を入力する場合は、`&&` で各コマンドを区切ります。[Output files] で、ビルド環境で AWS CodePipeline に送信するビルド出力ファイルへのパスを入力します。複数を入力する場合は、各ファイルパスをカンマで区切ります。詳細については、コンソールのツールヒントを参照してください。
18. [AWS CodeBuild service role] で、次のいずれかの操作を行います。

    - AWS アカウントに AWS CodeBuild サービスロールがない場合は、[Create a service role in your account] を選択します。[Role name] ボックスに、サービスロールの名前を入力するか、提案された名前をそのまま使用します。(サービスロール名は AWS アカウント内で一意でなければなりません)。

      Note

      コンソールを使用して AWS CodeBuild サービスロールを作成する場合、デフォルトでは、このサービスロールはこのビルドプロジェクトでのみ動作します。コンソールを使用

して、サービスロールを別のビルドプロジェクトと関連付けると、このロールは他のビルドプロジェクトと連携するように更新されます。1 つの AWS CodeBuild サービスロールは最大 10 個のビルドプロジェクトと連携できます。

- AWS アカウントに AWS CodeBuild サービスロールがある場合は、[Choose an existing service role from your account] を選択します。[Role name] ボックスで、サービスロールの名前を選択します。

19. (オプション) [Advanced] を展開します。

    60 分 (デフォルト) 以外のビルドタイムアウトを指定するには、[hours] ボックスと [minutes] ボックスを使用して 5〜480 分 (8 時間) のタイムアウトを指定します。

    (オプション) [Privileged] チェックボックスをオンにするのは、このビルドプロジェクトを使用して Docker イメージを作成する予定であり、Docker をサポートする AWS CodeBuild に用意されているビルド環境イメージ以外のイメージを選択している場合に限ります。それ以外の場合、関連付けられているビルドで Docker デーモンと通信しようとすると、すべて失敗します。必要に応じて、ビルドで Docker デーモンを操作できるように、Docker デーモンも起動する必要があります。これを行う 1 つの方法は、以下のビルドコマンドを実行してビルドスペックの `install` フェーズで Docker デーモンを初期化することです。(Docker をサポートする AWS CodeBuild に用意されているビルド環境イメージを選択した場合は、以下のビルドコマンドを実行しないでください。)

    ```
    - nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
    host=tcp://0.0.0.0:2375 --storage-driver=overlay&
    - timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
    ```

    [Compute] で、以下のいずれかの利用可能なオプションを選択します。

    [Environment variables] で、[Name] および [Value] を使用して、使用するビルド環境のオプションの環境変数を指定します。さらに環境変数を追加するには、[Add row] を選択します。

    > Important
    >
    > 機密情報、特に AWS アクセスキー ID とシークレットアクセスキーを格納する場合には、環境変数を使用しないことを強くお勧めします。環境変数は、AWS CodeBuild コンソールや AWS CLI などのツールを使用してプレーンテキストで表示できます。
    > 重要な値を保存および取得するには、ビルドコマンドで AWS CLI を使用して Amazon EC2 Systems Manager パラメータストアを操作することをお勧めします。AWS CLI は、AWS CodeBuild に用意されているすべての環境変数でプレインストール済みで構成済みになっています。詳細については、Amazon EC2 Systems Manager ユーザーガイドの「Systems Manager パラメータストア」および「Systems Manager パラメータストア CLI のチュートリアル」を参照してください。

20. ビルドプロジェクトの保存 を選択します。
21. [Input artifacts #1] で、この手順のステップ 4 でメモした [Output artifact] の値を入力します。
22. (オプション) テストアクションで出力アーティファクトを生成し、それに応じてビルドスペックを設定する場合は、[Output artifact #1] に出力アーティファクトに割り当てる値を入力します。
23. [Add action] を選択します。
24. [Save pipeline changes]、[Save and continue] の順に選択します。
25. [Release change] を選択します。
26. パイプラインが正常に実行された後、テスト結果を取得できます。パイプラインの [Test] ステージで、[AWS CodeBuild] ハイパーリンクを選択して、AWS CodeBuild コンソールで関連するビルドプロジェクトのページを開きます。

27. ビルドプロジェクトページの [Build history] エリアで、関連する [Build run] ハイパーリンクを選択します。
28. ビルドの実行ページの [Build logs] エリアで、[View entire log] ハイパーリンクを選択し、Amazon CloudWatch コンソールで関連するビルドログを開きます。
29. ビルドログをスクロールして、テスト結果を表示します。

Jenkins で AWS CodeBuild を使用する

Jenkins は、ソフトウェアプロジェクトを継続的に構築およびテストするために使用できる継続的な統合および継続的な配信アプリケーションです。詳細については、Jenkins のウェブサイトの Jenkins の紹介を参照してください。

機能レベルでは、Jenkins 2 つのコンポーネントがあります。ビルドジョブを作成して実行するスケジューラと、ビルドプラットフォーム、つまり一連の分散ビルドノードです。詳細については、分散ビルドを参照してください。

AWS CodeBuild Jenkins プラグインを使用すると、AWS CodeBuild を Jenkins のビルドジョブと統合することができます。Jenkins ビルドノードにビルドジョブを送信する代わりに、プラグインを使用してビルドジョブを AWS CodeBuild に送信します。これにより、Jenkins ビルドノードのプロビジョニング、設定、および管理が不要になります。

プラグインを入手するには、aws-codebuild.hpi ファイルをダウンロードします。プラグインを Jenkins 内でインストール、設定、実行する手順については、Jenkins のウェブサイトの AWS CodeBuild CodeBuild Pluginページを参照してください。

AWS CodeBuild でのビルドプロジェクトおよびビルドの操作

開始するには、「ビルドプロジェクトを作成する (p. 146)」の手順に従い、次に「ビルドの実行 (p. 167)」の手順に従ってください。ビルドプロジェクトおよびビルドの詳細については、以下のトピックを参照してください。

トピック

ビルドプロジェクトを操作する

ビルドプロジェクトは、AWS CodeBuild がビルドを実行する方法を定義します。これには、ソースコード、使用するビルド環境、実行するビルドの入手先、ビルド出力の保存先などの情報が含まれます。

ビルドプロジェクトを操作して以下のタスクを実行できます。

トピック

AWS CodeBuild でビルドプロジェクトを作成する

AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用して、ビルドプロジェクトを作成できます。

トピック

前提条件

ビルドを計画する (p. 94) の質問に答えます。

ビルドプロジェクトの作成 (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ウェルカムページが表示された場合は、[Get started] を選択します。

   ウェルカムページが表示されない場合は、ナビゲーションペインで [Build projects] を選択し、次に [Create project] を選択します。

3. [Configure your project] ページの [Project name] に、このビルドプロジェクトの名前を入力します。ビルドプロジェクトの名前は、各 AWS アカウントで一意である必要があります。
4. (オプション) [Add description] を選択して、[Description] ボックスに説明を入力します。
5. [Source: What to build] の [Source provider] で、ソースコードプロバイダタイプを選択し、次のいずれかの操作を行います。

   - [Amazon S3] を選択した場合は、[Bucket] でソースコードが含まれている入力バケットの名前を選択します。[S3 object key] に、ソースコードが含まれている ZIP ファイルの名前を入力します。
   - [AWS CodeCommit] を選択した場合は、[Repository] で、リポジトリの名前を選択します。[Build Badge] チェックボックスをオンにして、プロジェクトのビルドステータスを表示可能および埋め込み可能にします。詳細については、「ビルドバッジサンプル (p. 39)」を参照してください。[Git のクローンの深さ] の値を変更します。これによって、指定されるコミット数で切り捨てられる履歴の浅いクローンが作成されます。完全クローンを希望する場合には、[Full] を選択します。
   - [GitHub] を選択した場合は、手順に従って GitHub に接続 (または再接続) します。GitHub の [Authorize application] ページの [Organization access] で、AWS CodeBuild にアクセスを許可する各リポジトリの横にある [Request access] を選択します。[Authorize application] を選択した後で AWS CodeBuild コンソールに戻り、[Repository] でソースコードが含まれているリポジトリの名前を選択します。
     - [Git のクローンの深さ] の値を変更します。これによって、指定されるコミット数で切り捨てられる履歴の浅いクローンが作成されます。完全クローンを希望する場合には、[Full] を選択します。
     - コードの変更がこのリポジトリにプッシュされるたびに AWS CodeBuild がソースコードを再ビルドする場合、[GitHub webhooks] で [Webhook] を選択します。
     - [Build Badge] を選択すると、プロジェクトのビルドステータスが表示可能および埋め込み可能になります。詳細については、「ビルドバッジサンプル (p. 39)」を参照してください。
   - [GitHub Enterprise] を選択した場合、前提条件については「GitHub Enterprise のサンプル (p. 30)」を参照してください。
     - [個人用アクセストークン] には、前提条件の一部としてクリップボードにコピーしたトークンを貼り付けます。[トークンの保存] を選択します。[リポジトリ URL] には、GitHub Enterprise リポジトリの URL を入力します。

       Note

       個人用アクセストークンは、一回のみ入力して保存することが必要となります。次からのすべての AWS CodeBuild プロジェクトには、このトークンが使用されます。
     - [Git のクローンの深さ] の値を変更します。これによって、指定されるコミット数で切り捨てられる履歴の浅いクローンが作成されます。完全クローンを希望する場合には、[Full] を選択します。
     - コードの変更がこのリポジトリにプッシュされるたびに AWS CodeBuild がソースコードを再ビルドするには、[Webhook] を選択します。
     - GitHub Enterprise プロジェクトリポジトリに接続するときに SSL 警告を無視するには、[Insecure SSL] を選択します。

       Note

       [Insecure SSL] はテストのみに使用することが推奨されます。本番環境では使用しないでください。
     - [Build Badge] を選択すると、プロジェクトのビルドステータスが表示可能および埋め込み可能になります。
   - [Bitbucket] を選択した場合は、手順に従って Bitbucket に接続 (または再接続) します。Bitbucket の [Confirm access to your account] ページで、[Organization access] の [Grant access] を選択します。[Grant access] を選択した後で AWS CodeBuild コンソールに戻り、[Repository] でソースコードが含まれているリポジトリの名前を選択します。[Build Badge] を選択すると、プロジェクトのビルドステータスが表示可能および埋め込み可能になります。詳細については、「ビルドバッジサンプル (p. 39)」を参照してください。[Git のクローンの深さ] の値を変更します。これによって、指定されるコミット数で切り捨てられる履歴の浅いクローンが作成されます。完全クローンを希望する場合には、[Full] を選択します

6. [Environment: How to build] で以下の操作を行います。

   [Environment image] で、次のいずれかの操作を行います。

   - AWS CodeBuild で管理されている Docker イメージを使用するには、[Use an image managed by AWS CodeBuild] を選択し、次に [Operating system]、[Runtime]、および [Version] で選択を行います。
   - 別の Docker イメージを使用するには、[Specify a Docker image] を選択します。[Custom image type] で [Other] または [Amazon ECR] を選択します。[Other] を選択した場合は、[Custom image ID] に Docker Hub の Docker イメージの名前とタグを *repository-name*/*image-name*:*image-tag* の形式で入力します。[Amazon ECR] を選択した場合は、[Amazon ECR repository] および [Amazon ECR image] を使用して AWS アカウントの Docker イメージを選択します。

   [Build specification] で、次のいずれかの操作を行います。

   - ソースコードにビルド仕様ファイルが含まれている場合は、[Use the buildspec.yml in the source code root directory] を選択します。
   - ソースコードにビルド仕様ファイルが含まれていない場合、または、ソースコードルートディレクトリの `buildspec.yml` ファイルの `build` フェーズで指定したものと異なるビルドコマンドを実行する場合は、[Insert build commands] を選択します。[Build command] で、`build` フェーズで実行するコマンドを入力します。複数のコマンドについては、`&&` で各コマンドを区切ります (例: `mvn test && mvn package`)。他のフェーズでコマンドを実行する場合、または `build` フェーズのコマンドの長いリストがある場合は、ソースコマンドのルートディレクトリに `buildspec.yml` ファイルを追加し、ファイルにコマンドを追加してから、[Use the buildspec.yml in the source code root directory] を選択します。

   詳細については、「ビルドスペックリファレンス (p. 95)」を参照してください。

7. [Artifacts: Where to put the artifacts from this build project] の、[Artifacts type] で次のいずれかの操作を行います。

   - ビルド出力アーティファクトを作成しない場合は、[No artifacts] を選択します。ビルドテストのみを実行している場合や、Docker イメージを Amazon ECR リポジトリにプッシュする場合には、これを行うことができます。
   - ビルド出力を Amazon S3 バケットに保存する場合は、[Amazon S3] を選択して次のいずれかの操作を行います。
     - ビルド出力 ZIP ファイルまたはフォルダにプロジェクト名を使用する場合は、[Artifacts name] を空白のままにします。それ以外の場合は、[Artifacts name] ボックスに名前を入力します。(ZIP ファイルを出力して、ZIP ファイルにファイル拡張子を付ける場合は、ZIP ファイル名の後に含めるようにしてください。)
     - [Bucket name] で、出力バケットの名前を選択します。
     - この手順の前の方で [Insert build commands] を選択した場合は、[Output files] に、ビルド出力 ZIP ファイルまたはフォルダに格納する、ビルドからのファイルの場所を入力します。複数の場所の場合は、各場所をコンマで区切ります (例: `appspec.yml, target/my-app.jar`)。詳細については、「ビルドスペックの構文 (p. 96)」の `files` の説明を参照してください。

8. [Cache] で、次のいずれかの操作を行います。

   - キャッシュを使用しない場合は、[No cache] を選択します。
   - キャッシュを使用するには、[Amazon S3] を選択し、次の操作を行います。
     - [バケット] では、キャッシュが保存される Amazon S3 バケットの名前を選択します。
     - (オプション) [Path prefix] に、Amazon S3 パスプレフィックスを入力します。[Path prefix] 値は、バケット内の同じディレクトリにキャッシュを保存できるディレクトリ名に似ています。

Important

[Path prefix] の末尾に「/」を追加しないでください。

キャッシュを使用すると、再利用可能なビルド環境がキャッシュに保存され、ビルド全体で使用されるため、かなりのビルド時間が節約されます。

9. [Service role] で、次のいずれかの操作を行います。

   - AWS CodeBuild サービスロールがない場合は、[Create a service role in your account] を選択します。[Role name] で、デフォルト名を受け入れるか、独自の名前を入力します。
   - AWS CodeBuild サービスロールがある場合は、[Choose an service existing role from your account] を選択します。[Role name] で、サービスロールを選択します。

   Note

   コンソールでは、ビルドプロジェクトの作成時や更新時に AWS CodeBuild サービスロールも作成できます。デフォルトでは、ロールはそのビルドプロジェクトでのみ使用できます。コンソールでは、このサービスロールを別のビルドプロジェクトと関連付けると、この別のビルドプロジェクトで使用できるようにロールが更新されます。サービスロールは最大 10 個のビルドプロジェクトで使用できます。

10. [VPC] で、次のいずれかの操作を行います。

    - プロジェクトに VPC を使用していない場合は、[No VPC] を選択します。
    - AWS CodeBuild を使用して VPC で作業する場合は、次のようにします。
      - [VPC] で、AWS CodeBuild が使用する VPC ID を選択します。
      - [Subnets] で、AWS CodeBuild が使用するリソースを含むサブネットを選択します。
      - [Security Groups] で、AWS CodeBuild が VPC 内のリソースへのアクセスを許可するために使用するセキュリティグループを選択します。

    詳細については、「Amazon Virtual Private Cloud で AWS CodeBuild を使用する (p. 115)」を参照してください。

11. [Show advanced settings] を展開します。

    Note

    ウェルカムページから [Get started] 選択してこのページに到着した場合、[Show advanced settings] セクションは表示されません。この手順のステップ 20 に進みます。デフォルト設定の変更については、「ビルドプロジェクトの設定を変更する (コンソール) (p. 162)」を参照してください。

12. (オプション) [Timeout] の場合は、5 分から 480 分 (8 時間) の間の値を指定します。この時間が経過すると、AWS CodeBuild は完了していない場合にビルドを停止します。[hours] と [minutes] を空白のままにすると、デフォルト値の 60 分が使用されます。
13. (オプション) [Encryption key] で次のいずれかの操作を行います。

    - アカウントの Amazon S3 の AWS 管理の顧客マスターキー (CMK) を使用してビルド出力アーティファクトを暗号化するには、[Encryption key] を空白のままにします。これがデフォルト値です。
    - To use a カスタマー管理 CMK を使用してビルド出力アーティファクトを暗号化するには、[Encryption key] に CMK の ARN を入力します。`arn:aws:kms:`*`region-ID`*`:`*`account-ID`*`:key/`*`key-ID`* の形式を使用します。

14. (オプション) このビルドプロジェクトを使用して Docker イメージを作成し、選択したビルド環境イメージが Docker サポート付きの AWS CodeBuild によって提供されない場合にのみ、[特権付与] を選択します。それ以外の場合、関連付けられているビルドで Docker デーモンと通信しようとすると、すべて失敗します。ビルドが Docker デーモンと連係動作できるように、Docker デーモンも

起動する必要があります。これを行う 1 つの方法は、次のビルドコマンドを実行してビルドスペックの `install` フェーズで Docker デーモンを初期化することです。Docker をサポートする AWS CodeBuild によって提供されるビルド環境イメージを選択した場合は、これらのコマンドを実行しないでください。

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://0.0.0.0:2375 --storage-driver=overlay&
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

15. (オプション) この手順で以前に [Artifacts type] で [Amazon S3] を選択した場合は、[Artifacts packaging] で、次のいずれかの操作を行います。
    - AWS CodeBuild にビルド出力を含む ZIP ファイルを作成させるには、[Zip] を選択します。
    - AWS CodeBuild にビルド出力を含むフォルダを作成させるには、[None] を選択します。(これがデフォルトです)
16. [Compute type] で、以下のいずれかの利用可能なオプションを選択します。
17. [Environment variables] には、使用するビルドの各環境変数の名前、値、タイプを入力します。[Add row] を使用して環境変数を追加します。

    Note

    AWS CodeBuild は AWS リージョンの環境変数を自動的に設定します。それらを buildspec.yml に追加しない場合は、次の環境変数を設定する必要があります。

    - AWS_ACCOUNT_ID
    - IMAGE_REPO_NAME
    - IMAGE_TAG

    他のユーザーは、AWS CodeBuild コンソールと AWS CLI を使用して環境変数を確認できます。環境変数の表示に懸念がない場合は、[Name] および [Value] フィールドを設定し、[Type] を [Plaintext] に設定します。

    Amazon EC2 Systems Manager パラメータストアには、AWS アクセスキー ID、AWS シークレットアクセスキー、またはパスワードなどの機密値を持つ環境変数をパラメータとして保存することをお勧めします。[Type] で [Parameter Store] を選択します。[Name] に、参照する AWS CodeBuild の識別子を入力します。[Value] に、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータの名前を入力します。たとえば、`/CodeBuild/dockerLoginPassword` という名前のパラメータを使用して、[Type] で [Parameter Store] を選択します。[Name] に、「`LOGIN_PASSWORD`」と入力します。[Value] に、「`/CodeBuild/dockerLoginPassword`」と入力します。

    Important

    パラメータは `/CodeBuild/` で始まるパラメータ名 (例: `/CodeBuild/dockerLoginPassword`) で、Amazon EC2 Systems Manager パラメータストアに保存することをお勧めします。AWS CodeBuild コンソールを使用して Amazon EC2 Systems Manager にパラメータを作成することができます。[Create a parameter] を選択し、ダイアログボックスの手順に従います。(ダイアログボックスでは、[KMS key] の場合、オプションでアカウントの AWS KMS キーの ARN を指定できます。Amazon EC2 Systems Manager では、このキーを使用して、保存中にパラメータの値を暗号化し、取得中に復号化します。)AWS CodeBuild コンソールを使用してパラメータを作成した場合、コンソールは保存されている `/CodeBuild/` パラメータ名を開始します。詳細については、「Amazon EC2 Systems Manager ユーザーガイド」の「Systems Manager パラメータストア」および「Systems Manager パラメータストアコンソールのチュートリアル」を参照してください。ビルドプロジェクトが Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照する場合、ビルドプロジェクトのサービスロールで `ssm:GetParameters` アクションを許可する必要があります。以前に [Create a service role in your account] を選択した場合、AWS CodeBuild では、ビルドプロジェクトのデフォルトのサービスロールに

このアクションが自動的に含められます。ただし [Choose an existing service role from your account] を選択した場合は、このアクションをサービスロールに個別に含める必要があります。
ビルドプロジェクトが、`/CodeBuild/` で始まらないパラメータ名を持つ、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照し、[Create a service role in your account] を選択した場合、`/CodeBuild/` で始まらないパラメータ名にアクセスできるようにサービスロールを更新する必要があります。これは、サービスロールが `/CodeBuild/` で始まるパラメータ名にのみアクセスできるためです。
既存の環境変数は、設定した環境変数により置き換えられます。たとえば、Docker イメージに `my_value` の値を持つ `MY_VAR` という名前の環境変数が既に含まれていて、`other_value` の値を持つ `MY_VAR` という名前の環境変数を設定した場合、`my_value` が `other_value` に置き換えられます。同様に、Docker イメージに `/usr/local/sbin:/usr/local/bin` の値を持つ `PATH` という名前の環境変数が既に含まれていて、`$PATH:/usr/share/ant/bin` の値を持つ `PATH` という名前の環境変数を設定した場合、`/usr/local/sbin:/usr/local/bin` はリテラル値 `$PATH:/usr/share/ant/bin` に置き換えられます。
`CODEBUILD_` で始まる名前の環境変数は設定しないでください。このプレフィックスは内部使用のために予約されています。
同じ名前の環境変数が複数の場所で定義されている場合は、その値は次のように決定されます。

- ビルド開始オペレーション呼び出しの値が最も優先順位が高くなります。
- ビルドプロジェクト定義の値が次に優先されます。
- ビルドスペック宣言の値の優先順位が最も低くなります。

18. (オプション) [Tags] に、使用する AWS サービスをサポートするタグの名前と値を入力します。[Add row] を使用して、タグを追加します。最大 50 個のタグを追加できます。
19. [Continue] を選択します。
20. [Review] ページで、次のいずれかの操作を行います。

- ビルドを実行するには、[Save and build] を選択します。
- ビルドを実行せずにビルドプロジェクトの作成を終了するには、[Save] を選択します。

ビルドプロジェクトを作成する (AWS CLI)

AWS CLI を AWS CodeBuild と組み合わせて使用する方法については、「コマンドラインリファレンス (p. 191)」を参照してください。

1. `create-project` コマンドを実行します。

```
aws codebuild create-project --generate-cli-skeleton
```

JSON 形式のデータが出力に表示されます。AWS CLI がインストールされているローカルコンピュータまたはインスタンス上の場所にあるファイル (例: *`create-project.json`*) にデータをコピーします。コピーされたデータを次のように変更して、結果を保存します。

```
{
  "name": "project-name",
  "description": "description",
  "source": {
    "type": "source-type",
    "location": "source-location",
    "gitCloneDepth": "gitCloneDepth",
    "buildspec": "buildspec",
    "auth": {
      "type": "auth-type",
```

```
      "resource": "resource"
      "badgeEnabled": "badgeEnabled"
      "InsecureSsl": "InsecureSsl"
    }
  },
  "artifacts": {
    "type": "artifacts-type",
    "location": "artifacts-location",
    "path": "path",
    "namespaceType": "namespaceType",
    "name": "artifacts-name",
    "packaging": "packaging"
  },
  "cache": {
    "type": "cache-type",
    "location": "cache-location",
  },
  "serviceRole": "serviceRole",
  },
  "vpcConfig": {
    "securityGroupIds": [
         "security-group-id"
    ],
    "subnets": [
         "subnet-id"
    ],
    "vpcId": "vpc-id"
  },
  "timeoutInMinutes": timeoutInMinutes,
  "encryptionKey": "encryptionKey",
  "tags": [
    {
      "key": "tag-key",
      "value": "tag-value"
    }
  ],
    "environment": {
    "type": "environment-type",
    "image": "image",
    "computeType": "computeType",
    "certificate": "certificate",
    "environmentVariables": [
      {
        "name": "environmentVariable-name",
        "value": "environmentVariable-value",
        "type": "environmentVariable-type"
      }
    ],
    "privilegedMode": privilegedMode
  },
}
```

以下に置き換えます。

- *project-name*: 必須値。このビルドプロジェクトの名前。この名前は、AWS アカウントのすべてのビルドプロジェクトにわたって一意である必要があります。
- *description*: (オプションの値)。このビルドの説明。
- 必要な source オブジェクトのための、このビルドプロジェクトのソースコードの設定についての情報。これらの設定には以下が含まれます。
  - *source-type*: 必須値。ビルドするソースコードを含むリポジトリのタイプ。有効な値には、CODECOMMIT、CODEPIPELINE、GITHUB、BITBUCKET、および S3 があります。

- *source-location*: 必須値 (CODEPIPELINE に *source-type* を設定しない場合)。指定されたリポジトリタイプのソースコードの場所。
  - AWS CodeCommit の場合、ソースコードとビルド仕様 (例: `https://git-codecommit.`*region-id*`.amazonaws.com/v1/repos/`*repo-name*) を含むリポジトリの、HTTPS クローン URL。
  - Amazon S3 では、ビルド入力バケット名の後に、スラッシュ (/) が続き、ソースコードとビルド仕様 (例: *bucket-name*`/`*object-name*`.zip`) を含む ZIP ファイルの名前が続きます。これは ZIP ファイルがビルド入力バケットのルートにあることを前提としています。(ZIP ファイルがバケット内のフォルダにある場合は、代わりに *bucket-name*`/`*path*`/`*to*`/`*object-name*`.zip` を使用してください)。
  - GitHub の場合、ソースコードとビルド仕様を含むリポジトリへの HTTPS クローン URL。また、AWS アカウントを GitHub アカウントに接続する必要があります。これを行うには、AWS CodeBuild コンソールを使用してプロジェクトを作成します。コンソールを使用して GitHub に接続 (または再接続) するときは、GitHub の [Authorize application] ページの [Organization access] で、AWS CodeBuild にアクセスできるように各リポジトリの横にある [Request access] を選択します。[Authorize application] を選択します。(GitHub アカウントに接続した後、ビルドプロジェクトの作成を完了する必要はありません。AWS CodeBuild コンソールを閉じることができます。)この接続を使用するよう AWS CodeBuild に指示するには、`source` オブジェクトで、`auth` オブジェクトの `type` 値を `OAUTH` に設定します。
  - GitHub Enterprise の場合に、ソースコードとビルド仕様を含むリポジトリへの HTTP または HTTPS クローン URL。また、AWS アカウントを GitHub Enterprise アカウントに接続する必要があります。これを行うには、AWS CodeBuild コンソールを使用してプロジェクトを作成します。

    まず、GitHub Enterprise で個人用アクセストークンを作成します。AWS CodeBuild プロジェクトを作成する際に使用できるように、クリップボードにこのトークンをコピーします。詳細については、GitHub ヘルプウェブサイトの「GitHub Enterprise で個人用アクセストークンを作成する」を参照してください。コンソールを使用して AWS CodeBuild プロジェクトを作成する場合、[ソースプロバイダ] の [ソース: 何をビルドするか] で [GitHub Enterprise] を選択します。[個人用アクセストークン] には、クリップボードにコピーしたトークンを貼り付けます。[トークンの保存] を選択します。これで、AWS CodeBuild アカウントが GitHub Enterprise アカウントに接続されました。
  - Bitbucket の場合、ソースコードとビルド仕様を含むリポジトリへの HTTPS クローン URL。また、AWS アカウントを Bitbucket アカウントに接続する必要があります。これを行うには、AWS CodeBuild コンソールを使用してプロジェクトを作成します。コンソールを使用して Bitbucket に接続 (または再接続) する場合は、Bitbucket の [Confirm access to your account] ページで、[Grant access] を選択します (Bitbucket アカウントに接続した後、ビルドプロジェクトの作成を完了する必要はありません)。AWS CodeBuild コンソールを閉じることができます。)この接続を使用するよう AWS CodeBuild に指示するには、`source` オブジェクトで、`auth` オブジェクトの `type` 値を `OAUTH` に設定します。
  - AWS CodePipeline の場合は、`source` の `location` 値を指定しないでください。AWS CodePipeline ではパイプラインを作成するときに、パイプラインのソースステージでソースコードの場所を指定するため、この値は AWS CodePipeline では無視されます。
- *gitCloneDepth*: オプション値。ダウンロードする履歴の深さ。最小値は 0 です。この値が 0、あるいは 25 より大きいか指定されていない場合、完全な履歴が各ビルドプロジェクトと共にダウンロードされます。ソースタイプが Amazon S3 の場合、この値はサポートされません。
- *buildspec*: (オプションの値)。使用するビルド仕様定義またはファイル。この値が設定されている場合は、インラインのビルド仕様定義か、組み込みの環境変数 `CODEBUILD_SRC_DIR` の値に相対的な代替ビルド仕様ファイルへのパスのいずれかになります。この値が指定されていない場合、または空の文字列に設定されている場合、ソースコードにはルートディレクトリの `buildspec.yml` ファイルが含まれている必要があります。詳細については、「ビルドスペックのファイル名とストレージの場所 (p. 95)」を参照してください。
- *auth*: このオブジェクトは、AWS CodeBuild コンソールでのみ使用されます。*auth-type* (前述のとおり `GITHUB` に *source-type* が設定されていない場合を除く)、または *resource* に値を指定しないでください。

- *badgeEnabled*: オプションの値。ビルドバッジを AWS CodeBuild プロジェクトに含めるには、*badgeEnabled* を true の値で指定する必要があります。詳細については、「AWS CodeBuild のビルドバッジサンプル (p. 39)」を参照してください。
- *InsecureSsl*: オプション値。これは GitHub Enterprise のみで使用されます。GitHub Enterprise プロジェクトリポジトリに接続するときに SSL 警告を無視するには、この値を true に設定します。デフォルト値は false です。*InsecureSsl* は、テスト目的のみで使用してください。本番環境では使用しないでください。

- 必要な artifacts オブジェクトのための、このビルドプロジェクトの出力アーティファクトの設定についての情報。これらの設定には以下が含まれます。
  - *artifacts-type*: 必須値。ビルド出力アーティファクトのタイプ。有効な値は、CODEPIPELINE、NO_ARTIFACTS、および S3 です。
  - *artifacts-location*: 必須値 (CODEPIPELINE または NO_ARTIFACTS に *artifacts-type* を設定しない場合)。ビルド出力アーティファクトの場所。
    - *artifacts-type* に CODEPIPELINE を指定した場合は、artifacts の location を指定しないでください。
    - *artifacts-type* に NO_ARTIFACTS を指定した場合は、artifacts に location を指定しないでください。
    - *artifacts-type* に S3 を指定した場合、これは前提条件で作成または識別された出力バケットの名前です。
  - *path*: (オプションの値)。ビルド出力 ZIP ファイルまたはフォルダのパスと名前。
    - *artifacts-type* に CODEPIPELINE を指定した場合は、artifacts に path を指定しないでください。
    - *artifacts-type* に NO_ARTIFACTS を指定した場合は、artifacts に path を指定しないでください。
    - *artifacts-type* に S3 を指定した場合、これは *artifacts-location* 内のビルド出力 ZIP ファイルまたはフォルダのパスです。*path* の値を指定しない場合、AWS CodeBuild では *namespaceType* (指定されている場合) と *artifacts-name* を使用して、ビルド出力 ZIP ファイルまたはフォルダのパスと名前を決定します。たとえば、*path* に MyPath、*artifacts-name* に MyArtifact.zip を指定すると、パスと名前は MyPath/MyArtifact.zip になります。
  - *namespaceType*: (オプションの値)。ビルド出力 ZIP ファイルまたはフォルダのパスと名前。
    - *artifacts-type* に CODEPIPELINE を指定した場合は、artifacts に namespaceType を指定しないでください。
    - *artifacts-type* に NO_ARTIFACTS を指定した場合は、artifacts に namespaceType を指定しないでください。
    - *artifacts-type* に S3 を指定した場合、有効な値には BUILD_ID と NONE が含まれます。BUILD_ID を使用してビルド出力 ZIP ファイルまたはフォルダのパスにビルド ID を挿入します。それ以外の場合は、NONE を使用します。*namespaceType* の値を指定しない場合、AWS CodeBuild では *path* (指定されている場合) と *artifacts-name* を使用して、ビルド出力 ZIP ファイルまたはフォルダのパスと名前を決定します。たとえば、*path* に MyPath、*namespaceType* に BUILD_ID、*artifacts-name* に MyArtifact.zip を指定する場合、パスと名前は MyPath/*build-ID*/MyArtifact.zip になります。
  - *artifacts-name*: 必須値 (CODEPIPELINE または NO_ARTIFACTS に *artifacts-type* を設定しない場合)。ビルド出力 ZIP ファイルまたはフォルダのパスと名前。
    - *artifacts-type* に CODEPIPELINE を指定した場合は、artifacts に name を指定しないでください。
    - *artifacts-type* に NO_ARTIFACTS を指定した場合は、artifacts に name を指定しないでください。
    - *artifacts-type* に S3 を指定した場合、これは *artifacts-location* 内のビルド出力 ZIP ファイルまたはフォルダの名前です。たとえば、*path* に MyPath、*artifacts-name* に MyArtifact.zip を指定すると、パスと名前は MyPath/MyArtifact.zip になります。
  - *packaging*: (オプションの値)。作成するビルド出力アーティファクトのタイプ。

  - *artifacts-type* に CODEPIPELINE を指定した場合は、artifacts に packaging を指定しないでください。
  - *artifacts-type* に NO_ARTIFACTS を指定した場合は、artifacts に packaging を指定しないでください。
  - *artifacts-type* に S3 を指定した場合、有効な値には ZIP と NONE が含まれます。ビルド出力を含む ZIP ファイルを作成するには、ZIP を使用します。ビルド出力を含むフォルダを作成するには、NONE を使用します。デフォルト値は NONE です。
- 必要な *cache* オブジェクトのための、このビルドプロジェクトのキャッシュ設定についての情報。これらの設定には以下が含まれます。
  - *CacheType*: 必須値。有効な値は S3 または NONE です。
  - *CacheLocation*: 必須値 (*CacheType* を NONE に設定しない場合)。*CacheType* に S3 を指定した場合、これが S3 バケットの ARN およびパスのプレフィックスになります。たとえば、Amazon S3 バケット名が my-bucket で、パスのプレフィックスが build-cache の場合、*CacheLocation* に使用できるフォーマットは my-bucket/build-cache または aws:s3:::my-bucket/build-cache になります。
- *serviceRole*: 必須値。AWS CodeBuild のサービスロールの ARN は、IAM ユーザーに代わってサービスとやり取りするために使用します (例: arn:aws:iam::*account-id*:role/*role-name*)。
- オプションの *vpcConfig* オブジェクトについては、VPC 設定に関する情報を参照してください。設定は次のとおりです。
  - *vpcId*: 必須値。AWS CodeBuild で使用される VPC ID。リージョン内のすべての Amazon VPC ID のリストを取得するには、次のコマンドを実行します。

```
aws ec2 describe-vpcs
```

  - *subnets*: 必須値。AWS CodeBuild で使用されるリソースを含むサブネット ID。この ID を取得するには、次のコマンドを実行します。

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=<vpc-id>" --region us-east-1
```

    Note

    us-east-1 以外のリージョンを使用している場合は、コマンドを実行するときに必ず使用してください。

  - *securityGroupIds*: 必須値。VPC 内のリソースへのアクセスを許可するために AWS CodeBuild で使用されるセキュリティグループ ID。この ID を取得するには、次のコマンドを実行します。

```
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=<vpc-id>" --region
 us-east-1
```

    Note

    us-east-1 以外のリージョンを使用している場合は、コマンドを実行するときに必ず使用してください。

- 必要な environment オブジェクトのための、このプロジェクトのビルド環境設定についての情報。設定は次のとおりです。
  - *environment-type*: 必須値。構築環境のタイプ。許容されている値は LINUX_CONTAINER のみです。
  - *image*: 必須値。このビルド環境で使用される Docker イメージ識別子。通常、この識別子は *image-name*:*tag* として表されます。たとえば、Docker イメージを管理するために AWS CodeBuild が使用する Docker リポジトリでは、これは aws/codebuild/java:openjdk-8 です。Docker Hub では、maven:3.3.9-jdk-8 です。Amazon ECR では、*account-*

id.dkr.ecr.*region-id*.amazonaws.com/*your-Amazon-ECR-repo-name*:*tag* です。詳細については、「AWS CodeBuild に用意されている Docker イメージ (p. 103)」を参照してください。

- *computeType*: 必須値。このビルド環境で使用される CPU コアとメモリの数に対応するカテゴリ。使用できる値は、BUILD_GENERAL1_SMALL、BUILD_GENERAL1_MEDIUM、および BUILD_GENERAL1_LARGE です。
- *###*: オプション値。AWS S3 バケットの ARN、パスプレフィックスと PEM エンコードされた証明書を含むオブジェクトキー。オブジェクトキーは、.pem フェイルのみ、あるいは pem エンコードされた証明書を含む .zip ファイルのどちらでも指定できます。たとえば、Amazon S3 バケットの名前が my-bucket、パスプレフィックスは cert、オブジェクトキー名が certificate.pem の場合、*###*に許容される形式は my-bucket/cert/certificate.pem or arn:aws:s3:::my-bucket/cert/certificate.pem となります。
- オプションの environmentVariables 配列についての、このビルド環境で指定する任意の環境変数に関する情報。各環境変数は、name、value、*environmentVariable-name* の type、*environmentVariable-value*、*environmentVariable-type* を含むオブジェクトとして表されます。

  他のユーザーは、AWS CodeBuild コンソールと AWS CLI を使用して環境変数を確認できます。環境変数の表示に懸念がない場合は、*environmentVariable-name* と *environmentVariable-value* を設定してから *environmentVariable-type* を PLAINTEXT に設定します。

  Amazon EC2 Systems Manager パラメータストアには、AWS アクセスキー ID、AWS シークレットアクセスキー、パスワードなどの機密値を持つ環境変数をパラメータとして保存することをお勧めします。*environmentVariable-name* の場合、保存されているパラメータについては、AWS CodeBuild の識別子を参照するように設定します。*environmentVariable-value* には、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータの名前を設定します。*environmentVariable-type* を PARAMETER_STORE に設定します。たとえば、/CodeBuild/dockerLoginPassword いう名前のパラメータを使用して *environmentVariable-name* を LOGIN_PASSWORD に設定します。*environmentVariable-value* を /CodeBuild/dockerLoginPassword に設定します。*environmentVariable-type* を PARAMETER_STORE に設定します。

  Important

  ビルドプロジェクトが Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照する場合、ビルドプロジェクトのサービスロールで ssm:GetParameters アクションを許可する必要があります。以前に [Create a service role in your account] を選択した場合、AWS CodeBuild では、ビルドプロジェクトのデフォルトのサービスロールにこのアクションが自動的に含められます。ただし [Choose an existing service role from your account] を選択した場合は、このアクションをサービスロールに個別に含める必要があります。
  ビルドプロジェクトが、/CodeBuild/ で始まらないパラメータ名を持つ、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照し、[Create a service role in your account] を選択した場合、/CodeBuild/ で始まらないパラメータ名にアクセスできるようにサービスロールを更新する必要があります。これは、サービスロールが /CodeBuild/ で始まるパラメータ名にのみアクセスできるためです。
  既存の環境変数は、設定した環境変数によって置き換えられます。たとえば、Docker イメージに my_value の値を持つ MY_VAR という名前の環境変数が既に含まれていて、other_value の値を持つ MY_VAR という名前の環境変数を設定した場合、my_value が other_value に置き換えられます。同様に、Docker イメージに /usr/local/sbin:/usr/local/bin の値を持つ PATH という名前の環境変数が既に含まれていて、$PATH:/usr/share/ant/bin の値を持つ PATH という名前の環境変数を設定した場合、/usr/local/sbin:/usr/local/bin はリテラル値 $PATH:/usr/share/ant/bin に置き換えられます。

CODEBUILD_ で始まる名前の環境変数は設定しないでください。このプレフィックスは内部使用のために予約されています。
同じ名前の環境変数が複数の場所で定義されている場合は、その値は次のように決定されます。

- ビルド開始オペレーション呼び出しの値が最も優先順位が高くなります。
- ビルドプロジェクト定義の値が次に優先されます。
- ビルドスペック宣言の値の優先順位が最も低くなります。

- このビルドプロジェクトを使用して Docker イメージを構築する予定があり、指定したビルド環境イメージが Docker をサポートする AWS CodeBuild によって提供されない場合にのみ、*privilegedMode* を true の値で指定する必要があります。それ以外の場合、関連付けられているビルドで Docker デーモンと通信しようとすると、すべて失敗します。ビルドが Docker デーモンと連係動作できるように、Docker デーモンも起動する必要があります。これを行う 1 つの方法は、次のビルドコマンドを実行してビルドスペックの install フェーズで Docker デーモンを初期化することです。Docker をサポートする AWS CodeBuild によって提供されるビルド環境イメージを指定した場合は、これらのコマンドを実行しないでください。

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://0.0.0.0:2375 --storage-driver=overlay&
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

- *timeoutInMinutes*: (オプションの値)。5〜480 分 (8 時間) の分単位の時間。この時間が経過すると、ビルドが完了していない場合に AWS CodeBuild によってビルドが停止されます。指定しない場合は、デフォルトの 60 が使用されます。AWS CodeBuild がタイムアウトによりビルドを停止したかどうかを判断するには、batch-get-builds コマンドを実行します。ビルドが停止しているかどうかを確認するには、buildStatus 出力の FAILED の値を調べます。ビルドがタイムアウトしたかを確認するには、endTime 出力の phaseStatus 値に関連付けられている TIMED_OUT 値を調べます。
- *encryptionKey*: (オプションの値)。AWS CodeBuild がビルド出力の暗号化に使用するエイリアス、または AWS KMS カスタマーマスターキー (CMK) の ARN。エイリアスを指定する場合に、arn:aws:kms:*region-ID*:*account-ID*:key/*key-ID* 形式を使用し、エイリアスが存在する場合には、alias/*key-alias* 形式を使用します。指定しない場合は、Amazon S3 の AWS 管理 CMK が使用されます。
- オプションの *tags* 配列についての、このビルドプロジェクトに関連付けるタグに関する情報。最大 50 個のタグを指定できます。これらのタグは、AWS CodeBuild ビルドプロジェクトタグをサポートする任意の AWS サービスで使用できます。各タグは、*tag-key* の key 値と value 値、および *tag-value* を含むオブジェクトとして表されます。

例については、「ビルドプロジェクト (AWS CLI) を作成するには (p. 10)」を参照してください。

2. 保存したばかりのファイルがあるディレクトリに移動し、create-project コマンドをもう一度実行します。

```
aws codebuild create-project --cli-input-json file://create-project.json
```

3. 成功すると、次のようなデータが出力に表示されます。

```
{
  "project": {
    "name": "project-name",
    "description": "description",
    "serviceRole": "serviceRole",
    "tags": [
      {
        "key": "tags-key",
        "value": "tags-value"
      }
    ],
```

```
    "artifacts": {
      "namespaceType": "namespaceType",
      "packaging": "packaging",
      "path": "path",
      "type": "artifacts-type",
      "location": "artifacts-location",
      "name": "artifacts-name"
    },
    "lastModified": lastModified,
    "timeoutInMinutes": timeoutInMinutes,
    "created": created,
    "environment": {
      "computeType": "computeType",
      "image": "image",
      "type": "environment-type",
      "environmentVariables": [
        {
          "name": "environmentVariable-name",
          "value": "environmentVariable-value",
          "type": "environmentVariable-type"
        }
      ]
    },
    "source": {
      "type": "source-type",
      "location": "source-location",
      "buildspec": "buildspec",
      "auth": {
        "type": "auth-type",
        "resource": "resource"
      }
    },
    "encryptionKey": "encryptionKey",
    "arn": "arn"
  }
}
```

- `project` オブジェクトには、新しいビルドプロジェクトに関する情報が含まれています。
  - `lastModified` 値は、ビルドプロジェクトに関する情報が最後に変更された時刻 (Unix の時刻形式) を表します。
  - `created` 値は、ビルドプロジェクトが作成された時刻 (Unix の時刻形式) を表します。
  - `arn` 値は、ビルドプロジェクトの ARN を表します。

Note

ビルドプロジェクトの名前を除いて、後でビルドプロジェクトの設定を変更することができます。詳細については、「ビルドプロジェクトの設定を変更する (AWS CLI) (p. 165)」を参照してください。

ビルドの実行を開始するには、「ビルドの実行 (AWS CLI) (p. 169)」を参照してください。

ソースコードが GitHub リポジトリに保存されていて、コード変更がリポジトリにプッシュされるたびに AWS CodeBuild でソースコードを再構築する場合は、「自動的にビルドの実行を開始する (AWS CLI) (p. 172)」を参照してください。

ビルドプロジェクトの作成 (AWS SDK)

AWS CodeBuild を AWS SDK と組み合わせて使用する方法については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild のビルドプロジェクト名のリストを表示する

AWS CodeBuild でビルドプロジェクトのリストを表示するには、AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用できます。

トピック

ビルドプロジェクト名のリストを表示する (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ナビゲーションペインで、[Build projects] を選択します。

   Note

   デフォルトでは、最新の 10 件のビルドプロジェクトのみが表示されます。さらに多くのビルドプロジェクトを表示するには、[Projects per page] で別の値を選択するか、[Viewing projects] で前後の矢印を選択します。

ビルドプロジェクト名のリストを表示する (AWS CLI)

`list-projects` コマンドを実行します。

```
aws codebuild list-projects --sort-by sort-by --sort-order sort-order --next-token next-token
```

上記のコマンドで、次のプレースホルダを置き換えます。

- *sort-by*: オプションの文字列です。ビルドプロジェクト名を一覧表示するために使用される条件です。有効な値を次に示します。
  - `CREATED_TIME`: 各ビルドプロジェクトがいつ作成されたかに基づいて、ビルドプロジェクト名を一覧表示します。
  - `LAST_MODIFIED_TIME`: 各ビルドプロジェクトに関する情報が最後に変更されたときに基づいてビルドプロジェクト名を一覧表示します。
  - `NAME` 各ビルドプロジェクト名に基づいて、ビルドプロジェクト名を一覧表示します。
- *sort-order*: オプションの文字列。*sort-by* に基づいてビルドプロジェクトを一覧表示する順序です。有効な値は、`ASCENDING` および `DESCENDING` です。
- *next-token*: オプションの文字列。以前の実行中に、リストに 100 を超える項目がある場合、最初の 100 項目だけが、next token と呼ばれる一意の文字列と共に返されます。リスト内の項目の次のバッチを取得するには、次のコマンドを再度実行し、次のトークンを呼び出しに追加します。リスト内のすべての項目を取得するには、次のトークンが返されなくなるまで、このコマンドを、以後のすべての次のトークンで実行し続けます。

たとえば、次のコマンドを実行するとします。

```
aws codebuild list-projects --sort-by NAME --sort-order ASCENDING
```

次のような結果が出力に表示されることがあります。

```
{
  "nextToken": "Ci33ACF6...The full token has been omitted for brevity...U+AkMx8=",
  "projects": [
    "codebuild-demo-project",
    "codebuild-demo-project2",
    ... The full list of build project names has been omitted for brevity ...
    "codebuild-demo-project99"
  ]
}
```

このコマンドをもう一度実行します。

```
aws codebuild list-projects  --sort-by NAME --sort-order ASCENDING --next-token
 Ci33ACF6...The full token has been omitted for brevity...U+AkMx8=
```

次のような結果が出力に表示されることがあります。

```
{
  "projects": [
    "codebuild-demo-project100",
    "codebuild-demo-project101",
    ... The full list of build project names has been omitted for brevity ...
    "codebuild-demo-project122"
  ]
}
```

ビルドプロジェクト名のリストを表示する (AWS SDKs)

AWS SDK で AWS CodeBuild を使用する方法の詳細については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild でビルドプロジェクトの詳細を表示する

AWS CodeBuild でビルドプロジェクトの詳細を表示するには、AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用できます。

トピック

ビルドプロジェクトの詳細を表示する (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ナビゲーションペインで、[Build projects] を選択します。

   Note

   デフォルトでは、最新の 10 件のビルドプロジェクトのみが表示されます。さらに多くのビルドプロジェクトを表示するには、[Projects per page] で別の値を選択するか、[Viewing projects] で前後の矢印を選択します。
3. ビルドプロジェクトのリストの [Project] 列で、ビルドプロジェクトに対応するリンクを選択します。
4. [Build project: ***project-name***] ページで、[Project details] を展開します。

ビルドプロジェクトの詳細を表示する (AWS CLI)

AWS CLI を AWS CodeBuild と組み合わせて使用する方法については、「コマンドラインリファレンス (p. 191)」を参照してください。

`batch-get-projects` コマンドを実行します。

```
aws codebuild batch-get-projects --names names
```

上記のコマンドで、次のプレースホルダを置き換えます。

- *##*: 必須の文字列。詳細を表示する 1 つ以上のビルドプロジェクト名。複数のビルドプロジェクトを指定するには、各ビルドプロジェクトの名前をスペースで区切ります。最大 100 のビルドプロジェクト名を指定できます。ビルドプロジェクトのリストを表示するには、「ビルドプロジェクト名のリストを表示する (AWS CLI) (p. 159)」を参照してください。

たとえば、次のコマンドを実行するとします。

```
aws codebuild batch-get-projects --names codebuild-demo-project codebuild-demo-project2 my-other-demo-project
```

次のような結果が出力に表示されます。省略記号 (`...`) は簡潔にするために省略されたデータを表します。

```
{
  "projectsNotFound": [
    "my-other-demo-project"
  ],
  "projects": [
    {
      ...
      "name": codebuild-demo-project,
      ...
    },
    {
      ...
      "name": codebuild-demo-project2",
      ...
    }
  ]
}
```

上記の出力では、指定されたビルドプロジェクト名はすべて `projectsNotFound` 配列にリストされていますが、情報は見つかりませんでした。`projects` 配列は、情報が見つかった各ビルドプロジェクトの詳細を示しています。ビルドプロジェクトの詳細は、簡潔にするために前の出力から省略されています。詳細については、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」の出力を参照してください。

ビルドプロジェクトの詳細を表示する (AWS SDK)

AWS CodeBuild を AWS SDK と組み合わせて使用する方法については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild でビルドプロジェクトの設定を変更する

AWS CodeBuild でビルドプロジェクトの設定を変更するには、AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用できます。

トピック

ビルドプロジェクトの設定を変更する (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ナビゲーションペインで、[Build projects] を選択します。
3. 次のいずれかを行ってください。

   - 変更するビルドプロジェクトの横にあるラジオボタンを選択して、[Actions] を選択し、[Update] を選択します。
   - 変更するビルドプロジェクトのリンクを選択し、[Edit project] を選択します。

   > **Note**
   >
   > デフォルトでは、最新の 10 個のビルドプロジェクトのみが表示されます。さらに多くのビルドプロジェクトを表示するには、[Projects per page] で別の値を選択するか、[Viewing projects] で前後の矢印を選択します。

4. プロジェクトの詳細ページで、[Description] に説明を入力します。

   この手順ので参照される設定の詳細については、「ビルドプロジェクトの作成 (コンソール) (p. 146)」を参照してください。

5. ソースコードの場所に関する情報を変更するには、[Source: What to build] エリアで、[Update source] を選択します。表示されるフィールドは、ソースプロバイダタイプ (例: [Source provider]、[Bucket]、[S3 object]、または [Repository]) に応じて変更します。

   - ソースプロバイダが AWS CodeCommit、BitBucket、GitHub あるいは GitHub Enterprise の場合、[Git のクローンの深さ] の値を変更することができます。これによって、指定されるコミット数で切り捨てられる履歴の浅いクローンが作成されます。完全クローンを希望する場合には、[Full] を選択します。
   - ソースコードが GitHub または GitHub Enterprise リポジトリに保存されていて、コード変更がリポジトリにプッシュされるたびに AWS CodeBuild でソースコードを再構築する場合は、[Webhook] を選択します。
   - [ソースプロバイダ] が AWS CodeCommit、BitBucket、GitHub または GitHub Enterprise の場合、[Build Badge] チェックボックスが使用可能になります。[Build Badge] を選択すると、プロジェクトのビルドステータスが表示可能および埋め込み可能になります。詳細については、「ビルドバッジサンプル (p. 39)」を参照してください。

     > **Important**
     >
     > プロジェクトソースを更新すると、プロジェクトのビルドバッジの正確性に影響する場合があります。

6. ビルド環境に関する情報を変更するには、[Environment: How to build] で、[Update image] を選択します。ビルド環境タイプ (例: [Environment image]、[Operating system]、[Runtime]、[Version]、[Custom image type]、[Custom image ID]、[Amazon ECR repository]、または [Amazon ECR image]) に適切な変更を行います。
7. 次のいずれかを行ってください。

   - 以前は、ソースコードに buildspec.yml ファイルが含まれていなかったが現在は含まれている場合、[Update build specification] を選択し、[Use buildspec.yml from source code] を選択します。

- 以前は、ソースコードに buildspec.yml ファイルが含まれていたが現在は含まれていない場合、[ビルド仕様の更新] を選択した後、[ビルドコマンドの挿入] を選択し、[ビルドコマンド] にコマンドを入力します。

8. ビルド出力アーティファクトの場所と名前に関する情報を変更するには、[Artifacts: Where to put the artifacts from this build project] で、[Artifacts type]、[Artifact name]、[Bucket name]、または [Output files] の値を変更します。
9. キャッシュに関する情報を変更するには、[Cache] で、次のいずれかの操作を行います。

   - 以前にキャッシュを選択したが今はキャッシュを使用しない場合、[No cache] を選択します。
   - 以前に [No cache] を選択したが今はキャッシュを使用する場合、[Amazon S3] を選択し、次のいずれかを実行します。
     - [バケット] では、キャッシュが保存される Amazon S3 バケットの名前を選択します。
     - (オプション) [Path prefix] に、Amazon S3 パスプレフィックスを入力します。[Path prefix] 値は、バケット内の同じディレクトリにキャッシュを保存できるディレクトリ名に似ています。

       Important

       [Path prefix] の末尾に「/」を追加しないでください。

   キャッシュを使用すると、再利用可能なビルド環境がキャッシュに保存され、ビルド全体で使用されるため、かなりのビルド時間が節約されます。
10. AWS CodeBuild サービスロールに関する情報を変更するには、[Service role] で、[Create a role]、[Choose an existing service role from your account]、または [Role name] の値を変更します。

    Note

    コンソールでは、ビルドプロジェクトの作成時や更新時に AWS CodeBuild サービスロールも作成できます。デフォルトでは、ロールはそのビルドプロジェクトでのみ使用できます。コンソールでは、このサービスロールを別のビルドプロジェクトと関連付けると、この別のビルドプロジェクトで使用できるようにロールが更新されます。サービスロールは最大 10 個のビルドプロジェクトで使用できます。
11. [VPC] で、次のいずれかの操作を行います。

    - プロジェクトに VPC を使用していない場合は、[No VPC] を選択します。
    - AWS CodeBuild を VPC で動作させる場合、
      - [VPC] で、AWS CodeBuild が使用する VPC ID を選択します。
      - [Subnets] で、AWS CodeBuild が使用するリソースを含むサブネットを選択します。
      - [Security Groups] で、AWS CodeBuild が VPC 内のリソースへのアクセスを許可するために使用するセキュリティグループを選択します。

    詳細については、「Amazon Virtual Private Cloud で AWS CodeBuild を使用する (p. 115)」を参照してください。
12. ビルドタイムアウトに関する情報を変更するには、[Show advanced settings] で、[Timeout] の、[hours] と [minutes] の値を変更します。[hours] と [minutes] が空白のままの場合、デフォルト値は 60 分になります。
13. AWS KMS カスタマーマスターキー (CMK) に関する情報を変更するには、[Show advanced settings] の [Encryption key] の値を変更します。

    Important

    [暗号化キー] を空白のままにした場合、AWS CodeBuild は 代わりに AWS アカウントの Amazon S3 用 AWS マネージド CMK を使用します。
14. このビルドプロジェクトを使用して Docker イメージを作成する場合、指定されたビルド環境が Docker をサポートする AWS CodeBuild に用意されていない場合は、[Show advanced settings] で

[Privileged] を選択します。それ以外の場合、関連付けられているビルドで Docker デーモンと通信しようとすると、すべて失敗します。必要に応じて、ビルドで Docker デーモンを操作できるように、Docker デーモンも起動する必要があります。これを行う 1 つの方法は、次のビルドコマンドを実行してビルドスペックの `install` フェーズで Docker デーモンを初期化することです。(Docker をサポートする AWS CodeBuild に用意されているビルド環境イメージを指定した場合は、以下のビルドコマンドを実行しないでください。)

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://0.0.0.0:2375 --storage-driver=overlay&
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

15. ビルド出力アーティファクトの保存方法に関する情報を変更するには、[Show advanced settings] で、[Artifacts packaging] の値を変更します。
16. ビルドの実行に使用するメモリと vCPU の量を変更するには、[Show advanced settings] で [Compute type] の値を変更します。
17. ビルドに使用する環境変数に関する情報を変更するには、[Show advanced settings] で、[Environment variables] の、[Name]、[Value]、および [Type] の値を変更します。[Add row] を使用して環境変数を追加します。もう使用しない環境変数の横にある削除 (X) ボタンを選択します。

    他のユーザーは、AWS CodeBuild コンソールと AWS CLI を使用して環境変数を確認できます。環境変数の表示に懸念がない場合は、[Name] および [Value] フィールドを設定し、[Type] を [Plaintext] に設定します。

    Amazon EC2 Systems Manager パラメータストアには、AWS アクセスキー ID、AWS シークレットアクセスキー、またはパスワードなどの機密値を持つ環境変数をパラメータとして保存することをお勧めします。[Type] で [Parameter Store] を選択します。[Name] に、参照する AWS CodeBuild の識別子を入力します。[Value] に、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータの名前を入力します。たとえば、`/CodeBuild/dockerLoginPassword` という名前のパラメータを使用して、[Type] で [Parameter Store] を選択します。[Name] に、「`LOGIN_PASSWORD`」と入力します。[Value] に、「`/CodeBuild/dockerLoginPassword`」と入力します。

    > **Important**
    >
    > パラメータは `/CodeBuild/` で始まるパラメータ名 (例: `/CodeBuild/dockerLoginPassword`) で、Amazon EC2 Systems Manager パラメータストアに保存することをお勧めします。AWS CodeBuild コンソールを使用して Amazon EC2 Systems Manager にパラメータを作成することができます。[Create a parameter] を選択し、ダイアログボックスの手順に従います。(ダイアログボックスでは、[KMS key] の場合、オプションでアカウントの AWS KMS キーの ARN を指定できます。Amazon EC2 Systems Manager では、このキーを使用して、保存中にパラメータの値を暗号化し、取得中に復号化します。)AWS CodeBuild コンソールを使用してパラメータを作成した場合、コンソールは保存されている `/CodeBuild/` パラメータ名を開始します。詳細については、「Amazon EC2 Systems Manager ユーザーガイド」の「Systems Manager パラメータストア」および「Systems Manager パラメータストアコンソールのチュートリアル」を参照してください。
    > ビルドプロジェクトが Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照する場合、ビルドプロジェクトのサービスロールで `ssm:GetParameters` アクションを許可する必要があります。以前に [Create a service role in your account] を選択した場合、AWS CodeBuild では、ビルドプロジェクトのデフォルトのサービスロールにこのアクションが自動的に含められます。ただし [Choose an existing service role from your account] を選択した場合は、このアクションをサービスロールに個別に含める必要があります。
    > ビルドプロジェクトが、`/CodeBuild/` で始まらないパラメータ名を持つ、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照し、[Create a service role in your account] を選択した場合、`/CodeBuild/` で始まらないパラメータ名にアクセスできるようにサービスロールを更新する必要があります。これは、サービスロールが `/CodeBuild/` で始まるパラメータ名にのみアクセスできるためです。
    > 既存の環境変数は、設定した環境変数により置き換えられます。たとえば、Docker イメージに `my_value` の値を持つ `MY_VAR` という名前の環境変数が既に含まれてい

て、other_value の値を持つ MY_VAR という名前の環境変数を設定した場合、my_value が other_value に置き換えられます。同様に、Docker イメージに /usr/local/sbin:/usr/local/bin の値を持つ PATH という名前の環境変数が既に含まれていて、$PATH:/usr/share/ant/bin の値を持つ PATH という名前の環境変数を設定した場合、/usr/local/sbin:/usr/local/bin はリテラル値 $PATH:/usr/share/ant/bin に置き換えられます。
CODEBUILD_ で始まる名前の環境変数は設定しないでください。このプレフィックスは内部使用のために予約されています。
同じ名前の環境変数が複数の場所で定義されている場合は、その値は次のように決定されます。

- ビルド開始オペレーション呼び出しの値が最も優先順位が高くなります。
- ビルドプロジェクト定義の値が次に優先されます。
- ビルドスペック宣言の値の優先順位が最も低くなります。

18. このビルドプロジェクトのタグに関する情報を変更するには、[Show advanced settings] で、[Tags] の、[Name] と [Value] の値を変更します。[Add row] を使用して、タグを追加します。最大 50 個のタグを追加できます。もう使用しないタグの横にある削除 (X) アイコンを選択します。
19. [Update] を選択します。

ビルドプロジェクトの設定を変更する (AWS CLI)

AWS CLI を AWS CodeBuild と組み合わせて使用する方法については、「コマンドラインリファレンス (p. 191)」を参照してください。

1. 次のように update-project コマンドを実行します。

```
aws codebuild update-project --generate-cli-skeleton
```

JSON 形式のデータが出力に表示されます。AWS CLI がインストールされているローカルコンピュータまたはインスタンス上の場所にあるファイル (例: *update-project.json*) にデータをコピーします。次に、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」の説明に従って、コピーしたデータを変更して、結果を保存します。

Note

JSON 形式のデータでは、設定を変更するビルドプロジェクトの名前を指定する必要があります。その他のすべての設定はオプションです。ビルドプロジェクトの名前を変更することはできませんが、他の設定を変更することはできます。

2. たった今保存したファイルを含むディレクトリに切り替えて、update-project コマンドをもう一度実行します。

```
aws codebuild update-project --cli-input-json file://update-project.json
```

3. 成功した場合は、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」で説明されているのと同様のデータが出力に表示されます。

ビルドプロジェクトの設定を変更する (AWS SDK)

AWS CodeBuild を AWS SDK と組み合わせて使用する方法については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild でビルドプロジェクトを削除する

AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用して、AWS CodeBuild のビルドプロジェクトを削除できます。

> Warning
>
> ビルドプロジェクトを削除すると、復元できません。ビルドに関するすべての情報も削除され、復元することはできません。

トピック

ビルドプロジェクトの削除 (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ナビゲーションペインで、[Build projects] を選択します。
3. 次のいずれかを行ってください。
   - 削除するビルドプロジェクトの横にあるラジオボタンを選択して、[Actions] を選択し、次に [Delete] を選択します。
   - 削除するビルドプロジェクトのリンクを選択し、[Delete] を選択します。

   > Note
   >
   > デフォルトでは、最新の 10 個のビルドプロジェクトのみが表示されます。さらに多くのビルドプロジェクトを表示するには、[Projects per page] で別の値を選択するか、[Viewing projects] で前後の矢印を選択します。

ビルドプロジェクトを削除する (AWS CLI)

AWS CLI を AWS CodeBuild と組み合わせて使用する方法については、「コマンドラインリファレンス (p. 191)」を参照してください。

1. `delete-project` コマンドを実行します。

   ```
   aws codebuild delete-project --name name
   ```

   次のプレースホルダを置き換えます。

   - *name*: 必須の文字列。削除するビルドプロジェクトの名前。使用可能なビルドプロジェクトのリストを取得するには、`list-projects` コマンドを実行します。詳細については、「ビルドプロジェクト名のリストを表示する (AWS CLI) (p. 159)」を参照してください。
2. 成功した場合、データは出力されず、エラーも出力に表示されません。

ビルドプロジェクトを削除する (AWS SDK)

AWS CodeBuild を AWS SDK と組み合わせて使用する方法については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild でビルドを操作する

ビルド は、一連の入力アーティファクト (Java クラスファイルのコレクションなど) に基づいて出力アーティファクト (JAR ファイルなど) を作成するために AWS CodeBuild によって実行される一連のアクションを表します。

ビルドを操作するときに、次のタスクを実行できます。

トピック

AWS CodeBuild でビルドを実行する

AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用して、AWS CodeBuild でビルドを実行できます。

トピック

ビルドの実行 (コンソール)

AWS CodeBuild で AWS CodePipeline を使用してビルドを実行するには、この手順をスキップして「AWS CodeBuild で AWS CodePipeline を使用する (p. 126)」の手順に従います。

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. 次のいずれかを行ってください。
   - ビルドプロジェクトの作成を完了したばかりの場合、[Build project: ***project-name***] ページが表示されます。[Start build] を選択します。
   - 以前にビルドプロジェクトを作成した場合は、ナビゲーションペインで、[Build projects] を選択します。ビルドプロジェクトを選択した後、[Start build] を選択します。
3. [Start new build] ページで、次のいずれかの操作を行います。
   - Amazon S3 の場合、オプションの [Source version] 値に、構築する入力アーティファクトのバージョンに対応するバージョン ID を入力します。[Source version] が空白のままの場合、最新バージョンが使用されます。
   - AWS CodeCommit の場合、オプションの [Source version] 値では、[Branch] で、構築するソースコードのバージョンを含むブランチの名前を選択します。[Source version] では、表示されている HEAD コミット ID をそのまま使用するか、別の値を入力します。[Source version] が空白の場合は、デフォルトのブランチの HEAD コミット ID が使用されます。[Source version] のタグ名は入力できません。タグを指定するには、タグのコミット ID を入力します。[Git のクローンの深さ] の値

を変更します。これによって、指定されるコミット数で切り捨てられる履歴の浅いクローンが作成されます。完全クローンを希望する場合には、[Full] を選択します。

- GitHub または GitHub Enterprise の場合は、オプションの [ソースバージョン] 値に、構築するソースコードのバージョンに対応するコミット ID、プルリクエスト ID、ブランチ名、またはタグ名を入力します。プルリクエスト ID を指定する場合、`pr/`*`pull-request-ID`* (例: `pr/25`) 形式を使用する必要があります。ブランチ名を指定すると、ブランチの HEAD コミット ID が使用されます。[Source version] が空白の場合は、デフォルトのブランチの HEAD コミット ID が使用されます。[Git のクローンの深さ] の値を変更します。これによって、指定されるコミット数で切り捨てられる履歴の浅いクローンが作成されます。完全クローンを希望する場合には、[Full] を選択します。
- Bitbucket の場合は、オプションの [Source version] 値に、構築するソースコードのバージョンに対応するコミット ID、ブランチ名、またはタグ名を入力します。ブランチ名を指定すると、ブランチの HEAD コミット ID が使用されます。[Source version] が空白の場合は、デフォルトのブランチの HEAD コミット ID が使用されます。[Git のクローンの深さ] の値を変更します。これによって、指定されるコミット数で切り捨てられる履歴の浅いクローンが作成されます。完全クローンを希望する場合には、[Full] を選択します。

4. [Show advanced options] を展開します。

   - このビルドのみの出力アーティファクトのタイプを変更する場合は、[Artifacts type] で代わりのアーティファクトタイプを選択します。
   - このビルドのみの出力アーティファクトの名前を変更する場合は、[Artifacts name] に代わりのアーティファクト名を入力します。
   - このビルドのみの出力バケットの名前を変更する場合は、[Bucket name] で代わりの名前を選択します。
   - このビルドのみの出力アーティファクトがパッケージされる方法を変更する場合は、[Artifacts packaging] で代わりのパッケージングタイプを選択します。
   - このビルドのみのビルドタイムアウトを変更する場合は、[Timeout] で新しい値を指定します。

5. [Environment variables] を展開します。

   このビルドのみの環境変数を変更する場合は、[Name]、[Value] および [Type] の値を変更します。[Add row] を使用して、このビルドのみの新しい環境変数を追加します。このビルドで使用しない場合は、環境変数の横にある削除 (X) ボタンを選択します。

   他のユーザーは、AWS CodeBuild コンソールと AWS CLI を使用して環境変数を確認できます。環境変数の表示に懸念がない場合は、[Name] および [Value] フィールドを設定し、[Type] を [Plaintext] に設定します。

   Amazon EC2 Systems Manager パラメータストアには、AWS アクセスキー ID、AWS シークレットアクセスキー、またはパスワードなどの機密値を持つ環境変数をパラメータとして保存することをお勧めします。[Type] で [Parameter Store] を選択します。[Name] に、参照する AWS CodeBuild の識別子を入力します。[Value] に、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータの名前を入力します。たとえば、`/CodeBuild/dockerLoginPassword` という名前のパラメータを使用して、[Type] で [Parameter Store] を選択します。[Name] に、「`LOGIN_PASSWORD`」と入力します。[Value] に、「`/CodeBuild/dockerLoginPassword`」と入力します。

   > **Important**
   >
   > パラメータは `/CodeBuild/` で始まるパラメータ名 (例: `/CodeBuild/dockerLoginPassword`) で、Amazon EC2 Systems Manager パラメータストアに保存することをお勧めします。AWS CodeBuild コンソールを使用して Amazon EC2 Systems Manager にパラメータを作成することができます。[Create a parameter] を選択し、ダイアログボックスの手順に従います。(ダイアログボックスでは、[KMS key] の場合、オプションでアカウントの AWS KMS キーの ARN を指定できます。Amazon EC2 Systems Manager では、このキーを使用して、保存中にパラメータの値を暗号化し、取得中に復号化します。)AWS CodeBuild コンソールを使用してパラメータを作成した場合、コンソールは保存されている `/CodeBuild/` パラメータを開始します。詳細については、「Amazon

EC2 Systems Manager ユーザーガイド」の「Systems Manager パラメータストア」および「Systems Manager パラメータストアコンソールのチュートリアル」を参照してください。
ビルドプロジェクトが Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照する場合、ビルドプロジェクトのサービスロールで `ssm:GetParameters` アクションを許可する必要があります。以前に [Create a service role in your account] を選択した場合、AWS CodeBuild では、ビルドプロジェクトのデフォルトのサービスロールにこのアクションが自動的に含められます。ただし [Choose an existing service role from your account] を選択した場合は、このアクションをサービスロールに個別に含める必要があります。
ビルドプロジェクトが、`/CodeBuild/` で始まらないパラメータ名を持つ、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照し、[Create a service role in your account] を選択した場合、`/CodeBuild/` で始まらないパラメータ名にアクセスできるようにサービスロールを更新する必要があります。これは、サービスロールが `/CodeBuild/` で始まるパラメータ名にのみアクセスできるためです。
既存の環境変数は、設定した環境変数によって置き換えられます。たとえば、Docker イメージに `my_value` の値を持つ `MY_VAR` という名前の環境変数が既に含まれていて、`other_value` の値を持つ `MY_VAR` という名前の環境変数を設定した場合、`my_value` が `other_value` に置き換えられます。同様に、Docker イメージに `/usr/local/sbin:/usr/local/bin` の値を持つ `PATH` という名前の環境変数が既に含まれていて、`$PATH:/usr/share/ant/bin` の値を持つ `PATH` という名前の環境変数を設定した場合、`/usr/local/sbin:/usr/local/bin` はリテラル値 `$PATH:/usr/share/ant/bin` に置き換えられます。
`CODEBUILD_` で始まる名前の環境変数は設定しないでください。このプレフィックスは内部使用のために予約されています。
同じ名前の環境変数が複数の場所で定義されている場合は、その値は次のように決定されます。

- ビルド開始オペレーション呼び出しの値が最も優先順位が高くなります。
- ビルドプロジェクト定義の値が次に優先されます。
- ビルドスペック宣言の値の優先順位が最も低くなります。

6. [Start build] を選択します。

   このビルドの詳細については、「ビルドの詳細の表示 (コンソール) (p. 174)」を参照してください。

ビルドの実行 (AWS CLI)

Note

AWS CodeBuild で AWS CodePipeline を使用してビルドを実行するには、この手順をスキップして「AWS CodeBuild を使用するパイプラインを作成する (AWS CLI) (p. 132)」の手順に従います。
AWS CLI を AWS CodeBuild と組み合わせて使用する方法については、「コマンドラインリファレンス (p. 191)」を参照してください。

1. 次のいずれかの方法で `start-build` コマンドを実行します。

```
aws codebuild start-build --project-name project-name
```

ビルド入力アーティファクトの最新バージョンとビルドプロジェクトの既存の設定を使用するビルドを実行する場合は、これを使用します。

```
aws codebuild start-build --generate-cli-skeleton
```

以前のバージョンのビルド入力アーティファクトを使用してビルドを実行する場合、またはビルド出力アーティファクト、環境変数、ビルドスペック、またはデフォルトのビルドタイムアウト期間の設定をオーバーライドする場合は、これを使用します。

2. `--project-name` オプションを指定して `start-build` コマンドを実行する場合は、*project-name* をビルドプロジェクトの名前に置き換えて、この手順のステップ 6 に進みます。ビルドプロジェクトのリストを表示するには、「ビルドプロジェクト名のリストを表示する (p. 159)」を参照してください。

3. `start-build` オプションを指定して `--generate-cli-skeleton` コマンドを実行すると、出力に JSON 形式のデータが表示されます。AWS CLI がインストールされているローカルコンピュータまたはインスタンス上の場所にあるファイル (例: *start-build.json*) にデータをコピーします。コピーしたデータを次の形式に変更して、結果を保存します。

```
{
  "projectName": "projectName",
  "sourceVersion": "sourceVersion",
  "artifactsOverride": {
    "type": "type",
    "location": "location",
    "path": "path",
    "namespaceType": "namespaceType",
    "name": "artifactsOverride-name",
    "packaging": "packaging"
  },
  "environmentVariablesOverride": [
    {
      "name": "environmentVariablesOverride-name",
      "value": "value",
      "type": "environmentVariablesOverride-type"
    }
  ],
  "buildspecOverride": "buildspecOverride",
  "gitCloneDepthOverride": "gitCloneDepthOverride",
  "timeoutInMinutesOverride": timeoutInMinutesOverride
}
```

次のプレースホルダを置き換えます。

- *projectName*: 必須の文字列。このビルドに使用するビルドプロジェクトの名前。
- *sourceVersion*: オプションの文字列。作成するソースコードのバージョンで、次のようになります。
  - Amazon S3 の場合、ビルドする入力 ZIP ファイルのバージョンに対応するバージョン ID。*sourceVersion* が指定されなければ、最新のバージョンが使用されます。
  - AWS CodeCommit の場合、ビルドするソースコードのバージョンに対応するコミット ID。*sourceVersion* が指定されなければ、デフォルトブランチの HEAD コミット ID が使用されます。(*sourceVersion* にタグ名は指定できません。しかし、タグのコミット ID は指定できます。)
  - GitHub の場合、ビルドするソースコードのバージョンに対応するコミット ID、プルリクエスト ID、ブランチ名、またはタグ名。プルリクエスト ID を指定する場合、`pr/`*pull-request-ID* (例: `pr/25`) 形式を使用する必要があります。ブランチ名を指定すると、ブランチの HEAD コミット ID が使用されます。*sourceVersion* が指定されなければ、デフォルトブランチの HEAD コミット ID が使用されます。
  - Bitbucket の場合、ビルドするソースコードのバージョンに対応するコミット ID、ブランチ名、またはタグ名。ブランチ名を指定すると、ブランチの HEAD コミット ID が使用されます。*sourceVersion* が指定されなければ、デフォルトブランチの HEAD コミット ID が使用されます。

- *type*: オプションの文字列。このビルドで上書きするビルド出力アーティファクトタイプは、ビルドプロジェクトで定義されたものです。
- *location*: オプションの文字列。このビルドで上書きするビルド出力アーティファクトの場所は、ビルドプロジェクトで定義されたものです。
- *path*: オプションの文字列。このビルドで上書きするビルド出力アーティファクトパスは、ビルドプロジェクトで定義されたものです。
- *namespaceType*: オプションの文字列。このビルドで上書きするビルド出力アーティファクトパスのタイプは、ビルドプロジェクトで定義されたものです。
- *name*: オプションの文字列。このビルドで上書きするビルド出力アーティファクト名は、ビルドプロジェクトで定義されたものです。
- *packaging*: オプションの文字列。このビルドで上書きするビルド出力アーティファクトパッケージタイプは、ビルドプロジェクトで定義されたものです。
- *environmentVariablesOverride-name*: オプションの文字列。このビルドで値を上書きするビルドプロジェクトの環境変数の名前。
- *value*: オプションの文字列。このビルドで値を上書きするビルドプロジェクトで定義された環境変数の値。
- *environmentVariablesOverride-type*: オプションの文字列。このビルドで値を上書きするビルドプロジェクトの環境変数のタイプ。

  Important

  Amazon EC2 Systems Manager パラメータストアには、AWS アクセスキー ID、AWS シークレットアクセスキー、パスワードなどの機密値を持つ環境変数をパラメータとして保存することをお勧めします。AWS CodeBuild では、Amazon EC2 Systems Manager パラメータストアに保存されているパラメータは、そのパラメータの名前が `/CodeBuild/` (例: `/CodeBuild/dockerLoginPassword`) で始まる場合にのみ使用できます。AWS CodeBuild コンソールを使用して Amazon EC2 Systems Manager にパラメータを作成することができます。[Create a parameter] を選択し、ダイアログボックスの手順に従います。(ダイアログボックスでは、[KMS key] の場合、オプションでアカウントの AWS KMS キーの ARN を指定できます。Amazon EC2 Systems Manager では、このキーを使用して、保存中にパラメータの値を暗号化し、取得中に復号化します。)AWS CodeBuild コンソールを使用してパラメータを作成した場合、コンソールは保存されている `/CodeBuild/` パラメータを開始します。ただし、Amazon EC2 Systems Manager パラメータストアコンソールを使用してパラメータを作成する場合、パラメータの名前を `/CodeBuild/` で開始する必要があり、[Type] を [Secure String] に設定する必要があります。詳細については、「Amazon EC2 Systems Manager ユーザーガイド」の「Systems Manager パラメータストア」および「Systems Manager パラメータストアコンソールのチュートリアル」を参照してください。
  ビルドプロジェクトが Amazon EC2 Systems Manager パラメータストアに保存されているパラメータを参照する場合、ビルドプロジェクトのサービスロールで `ssm:GetParameters` アクションを許可する必要があります。以前に [Create a new service role in your account] を選択した場合、AWS CodeBuild では、ビルドプロジェクトのデフォルトのサービスロールにこのアクションが自動的に含められます。ただし [Choose an existing service role from your account] を選択した場合は、このアクションをサービスロールに個別に含める必要があります。
  既存の環境変数は、設定した環境変数により置き換えられます。たとえば、Docker イメージに `my_value` の値を持つ `MY_VAR` という名前の環境変数が既に含まれていて、`other_value` の値を持つ `MY_VAR` という名前の環境変数を設定した場合、`my_value` が `other_value` に置き換えられます。同様に、Docker イメージに `/usr/local/sbin:/usr/local/bin` の値を持つ `PATH` という名前の環境変数が既に含まれていて、`$PATH:/usr/share/ant/bin` の値を持つ `PATH` という名前の環境変数を設定した場合、`/usr/local/sbin:/usr/local/bin` はリテラル値 `$PATH:/usr/share/ant/bin` に置き換えられます。
  `CODEBUILD_` で始まる名前の環境変数は設定しないでください。このプレフィックスは内部使用のために予約されています。

同じ名前の環境変数が複数の場所で定義されている場合、環境変数の値は次のように決定されます。

- ビルド開始オペレーション呼び出しの値が最も優先順位が高くなります。
- ビルドプロジェクト定義の値が次に優先されます。
- ビルドスペック宣言の値の優先順位が最も低くなります。

- *buildspecOverride*: オプションの文字列。このビルドで上書きするビルドスペック宣言は、ビルドプロジェクトで定義されたものです。この値が設定されている場合は、インラインのビルド仕様定義か、組み込みの環境変数 CODEBUILD_SRC_DIR の値に相対的な代替ビルド仕様ファイルへのパスのいずれかになります。
- *gitCloneDepthOverride*: オプションの文字列。このビルドで上書きする、ビルドプロジェクトの [Git のクローンの深さ] の値。ソースタイプが Amazon S3 の場合、この値はサポートされません。
- *timeoutInMinutesOverride*: オプション番号。このビルドで上書きするビルドタイムアウトの分数は、ビルドプロジェクトで定義されたものです。

これらのプレースホルダの有効な値の詳細については、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」を参照してください。ビルドプロジェクトの最新の設定の一覧については、「ビルドプロジェクトの詳細を表示する (p. 160)」を参照してください。

4. 保存したばかりのファイルがあるディレクトリに移動し、start-build コマンドをもう一度実行します。

```
aws codebuild start-build --cli-input-json file://start-build.json
```

5. 成功した場合は、「ビルドを実行するには (AWS CLI) (p. 13)」の手順で説明されているのと同様のデータが出力に表示されます。

このビルドの詳細情報を使用するには、出力の id の値を書き留めてから、「ビルドの詳細を表示する (AWS CLI) (p. 174)」を参照してください。

自動的にビルドの実行を開始する (AWS CLI)

ソースコードを GitHub または GitHub Enterprise リポジトリに保存している場合は、コード変更がリポジトリにプッシュされるたびに AWS CodeBuild でソースコードを再構築するのに GitHub ウェブフックを使用できます。

次のように create-webhook コマンドを実行します。

```
aws codebuild create-webhook --project-name
```

- *project-name* は、再構築するソースコードを含むビルドプロジェクトの名前です。

GitHub では、次のような情報が出力に表示されます。

```
{
  "webhook": {
    "url": "url"
  }
}
```

- *url* は GitHub ウェブフックへの URL です。

GitHub Enterprise では、次のような情報が出力に表示されます。

出力からシークレットキーとペイロード URL をコピーします。GitHub Enterprise にウェブフックを追加するために必要となります。GitHub Enterprise で AWS CodeBuild プロジェクトが保存されているリポジトリを選択し、[設定]、[Hooks & services]、[Add webhook] の順に選択します。ペイロード URL とシークレットキーを入力し、その他のフィールドにはデフォルト値を選択して、[Add webhook] を選択します。

自動的にビルドの実行を停止する (AWS CLI)

ソースコードを GitHub または GitHub Enterprise リポジトリに保存している場合は、コード変更がリポジトリにプッシュされるたびに AWS CodeBuild でソースコードを再構築するのに GitHub ウェブフックを設定できます。詳細については、「自動的にビルドの実行を開始する (AWS CLI) (p. 172)」を参照してください。

この動作を有効にしている場合、次の `delete-webhook` コマンドを実行して無効化できます。

```
aws codebuild delete-webhook --project-name
```

- *`project-name`* は、再構築するソースコードを含むビルドプロジェクトの名前です。

このコマンドが成功すると、情報やエラーはなにも出力に表示されません。

> Note
>
> これは、AWS CodeBuild プロジェクトからのみウェブフックを削除します。GitHub あるいは GitHub Enterprise でもウェブフックを削除する必要があります。

ビルドの実行 (AWS SDK)

AWS CodeBuild で AWS CodePipeline を使用してビルドを実行するには、この手順をスキップして、代わりに「AWS CodeBuild で AWS CodePipeline を使用してコードをテストし、ビルドを実行する (p. 126)」の手順に従います。

AWS CodeBuild を AWS SDK と組み合わせて使用する方法については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild のビルドの詳細を表示する

AWS CodeBuild で管理するビルドの詳細を表示するには、AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用できます。

トピック

ビルドの詳細の表示 (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. 次のいずれかを行ってください。

   - ナビゲーションペインで、[Build history] を選択します。ビルドのリストの [Build run] 列で、ビルドに対応するリンクを選択します。
   - ナビゲーションペインで、[Build projects] を選択します。ビルドプロジェクトのリストの [Project] 列で、ビルドプロジェクト名に対応するリンクを選択します。次に、ビルドのリストの [Build run] 列で、ビルドに対応するリンクを選択します。

     Note

     デフォルトでは、最新の 10 個のビルドまたはビルドプロジェクトのみが表示されます。さらに多くのビルドやビルドプロジェクトを表示するには、[Builds per page] や [Projects per page] で別の値を選択するか、[Viewing builds] や [Viewing projects] で前後の矢印を選択します。

ビルドの詳細を表示する (AWS CLI)

AWS CLI を AWS CodeBuild と組み合わせて使用する方法については、「コマンドラインリファレンス (p. 191)」を参照してください。

`batch-get-builds` コマンドを実行します。

```
aws codebuild batch-get-builds --ids ids
```

次のプレースホルダを置き換えます。

- *ids*: 必須の文字列。詳細を表示する 1 つ以上のビルド ID。複数のビルド ID を指定するには、各ビルド ID をスペースで区切ります。最大 100 のビルド ID を指定できます。ビルド ID のリストを取得するには、次のトピックを 1 つ以上参照してください。
  - ビルド ID のリストを表示する (AWS CLI) (p. 176)
  - ビルドプロジェクトのビルド ID のリストを表示する (AWS CLI) (p. 177)

たとえば、次のコマンドを実行するとします。

```
aws codebuild batch-get-builds --ids codebuild-demo-project:e9c4f4df-3f43-41d2-
ab3a-60fe2EXAMPLE codebuild-demo-project:815e755f-bade-4a7e-80f0-efe51EXAMPLE my-other-
project:813bb6c6-891b-426a-9dd7-6d8a3EXAMPLE
```

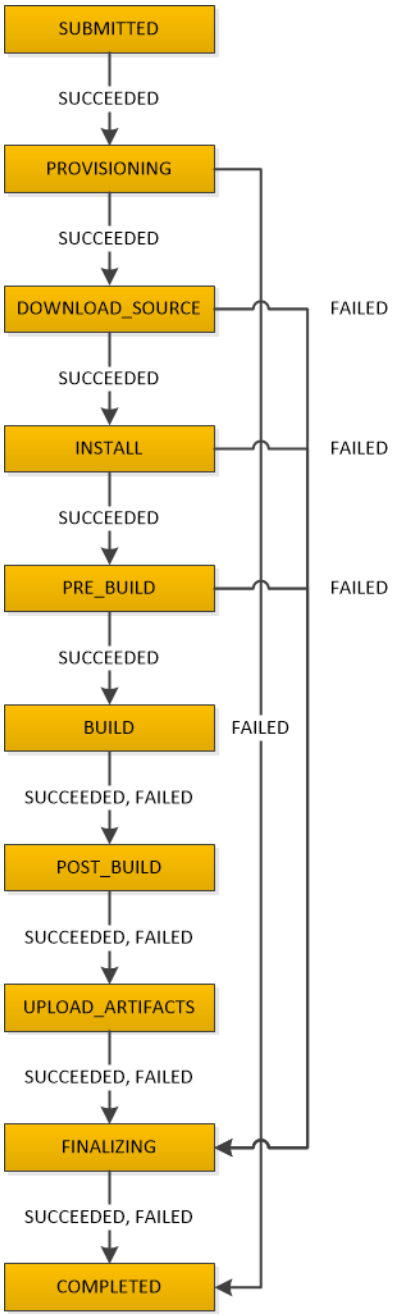
コマンドが成功した場合は、「ご利用開始にあたって」の「要約されたビルド情報を表示するには (AWS CLI) (p. 15)」の手順に説明されているのと同様のデータが出力に表示されます。

ビルドの詳細を表示する (AWS SDK)

AWS CodeBuild を AWS SDK と組み合わせて使用する方法については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

ビルドフェーズの移行

AWS CodeBuild のビルドはフェーズごとに進められます。



ここで注意すべき重要な点は、`BUILD` フェーズが失敗しても、`UPLOAD_ARTIFACTS` フェーズは常に試行されることです。

AWS CodeBuild のビルド ID のリストを表示する

AWS CodeBuild によって管理されるビルドのビルド ID のリストを表示するには、AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用できます。

トピック

- ビルド ID のリストを表示する (AWS SDK) (p. 177)

ビルド ID のリストを表示する (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ナビゲーションペインで、[Build history] を選択します。

   Note

   デフォルトでは、最新の 10 個のビルドのみが表示されます。さらに多くのビルドを表示するには、[Builds per page] で別の値を選択するか、[Viewing Builds] で前後の矢印を選択します。

ビルド ID のリストを表示する (AWS CLI)

AWS CLI を AWS CodeBuild と組み合わせて使用する方法については、「コマンドラインリファレンス (p. 191)」を参照してください。

- `list-builds` コマンドを実行します。

```
aws codebuild list-builds --sort-order sort-order --next-token next-token
```

上記のコマンドで、次のプレースホルダーを置き換えます。

- *`sort-order`*: オプションの文字列。ビルド ID を一覧表示する方法。有効な値は、`ASCENDING` および `DESCENDING` です。
- *`next-token`*: オプションの文字列。前の実行中に、リストに 100 を超える項目がある場合、最初の 100 項目だけが、次のトークンと呼ばれる一意の文字列と共に返されます。リスト内の項目の次のバッチを取得するには、次のコマンドを再度実行し、次のトークンを呼び出しに追加します。リスト内のすべての項目を取得するには、次のトークンが返されなくなるまで、このコマンドを、以後のすべての次のトークンで実行し続けます。

たとえば、次のコマンドを実行するとします。

```
aws codebuild list-builds --sort-order ASCENDING
```

次のような結果が出力に表示されることがあります。

```
{
  "nextToken": "4AEA6u7J...The full token has been omitted for brevity...MzY2OA==",
  "ids": [
    "codebuild-demo-project:815e755f-bade-4a7e-80f0-efe51EXAMPLE"
    "codebuild-demo-project:84a7f3d1-d40e-4956-b4cf-7a9d4EXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:931d0b72-bf6f-4040-a472-5c707EXAMPLE"
  ]
}
```

このコマンドをもう一度実行します。

```
aws codebuild list-builds --sort-order ASCENDING --next-token 4AEA6u7J...The full token
 has been omitted for brevity...MzY2OA==
```

次のような結果が出力に表示されることがあります。

```
{
  "ids": [
    "codebuild-demo-project:49015049-21cf-4b50-9708-df115EXAMPLE",
    "codebuild-demo-project:543e7206-68a3-46d6-a4da-759abEXAMPLE",
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:c282f198-4582-4b38-bdc0-26f96EXAMPLE"
  ]
}
```

ビルド ID のリストを表示する (AWS SDK)

AWS CodeBuild を AWS SDK と組み合わせて使用する方法については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild でビルドプロジェクトのビルド ID のリストを表示する

AWS CodeBuild コンソール、AWS CLI または AWS SDK を使用して、AWS CodeBuild でビルドプロジェクトのビルド ID のリストを表示できます。

トピック

ビルドプロジェクト (コンソール) のビルド ID の一覧を表示する

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. ナビゲーションペインで、[Build projects] を選択します。ビルドプロジェクトのリストの [Project] 列で、ビルドプロジェクトを選択します。

   Note

   デフォルトでは、最新の 10 個のビルドまたはビルドプロジェクトのみが表示されます。さらに多くのビルドやビルドプロジェクトを表示するには、[Builds per page] や [Projects per page] で異なる値を選択するか、[Viewing builds] や [Viewing projects] で前後の矢印を選択します。

ビルドプロジェクトのビルド ID のリストを表示する (AWS CLI)

AWS CLI を AWS CodeBuild と組み合わせて使用する方法については、「コマンドラインリファレンス (p. 191)」を参照してください。

次のように `list-builds-for-project` コマンドを実行します。

```
aws codebuild list-builds-for-project --project-name project-name --sort-order sort-order
 --next-token next-token
```

上記のコマンドで、次のプレースホルダーを置き換えます。

- *project-name*: 必須の文字列。ビルド ID をリスト表示するビルドプロジェクトの名前です。ビルドプロジェクトのリストを表示するには、「ビルドプロジェクト名のリストを表示する (AWS CLI) (p. 159)」を参照してください。
- *sort-order*: オプションの文字列。ビルド ID を一覧表示する方法。有効な値は、ASCENDING および DESCENDING です。
- *next-token*: オプションの文字列。以前の実行中に、リストに 100 を超える項目がある場合、最初の 100 項目だけが、next token と呼ばれる一意の文字列と共に返されます。リスト内の項目の次のバッチを取得するには、次のコマンドを再度実行し、次のトークンを呼び出しに追加します。リスト内のすべての項目を取得するには、次のトークンが返されなくなるまで、次に続くトークンが返されるごとにこのコマンドを実行し続けます。

たとえば、このコマンドを次のように実行するとします。

```
aws codebuild list-builds-for-project --project-name codebuild-demo-project --sort-order
 ASCENDING
```

次のような結果が出力に表示されます。

```
{
  "nextToken": "4AEA6u7J...The full token has been omitted for brevity...MzY2OA==",
  "ids": [
    "codebuild-demo-project:9b175d16-66fd-4e71-93a0-50a08EXAMPLE"
    "codebuild-demo-project:a9d1bd09-18a2-456b-8a36-7d65aEXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:fe70d102-c04f-421a-9cfa-2dc15EXAMPLE"
  ]
}
```

このコマンドをもう一度実行します。

```
aws codebuild list-builds-for-project --project-name codebuild-demo-project --sort-order
 ASCENDING --next-token 4AEA6u7J...The full token has been omitted for brevity...MzY2OA==
```

次のような結果が出力されます。

```
{
  "ids": [
    "codebuild-demo-project:98253670-7a8a-4546-b908-dc890EXAMPLE"
    "codebuild-demo-project:ad5405b2-1ab3-44df-ae2d-fba84EXAMPLE"
    ... The full list of build IDs has been omitted for brevity ...
    "codebuild-demo-project:f721a282-380f-4b08-850a-e0ac1EXAMPLE"
  ]
}
```

ビルドプロジェクトのビルド ID のリストを表示する (AWS SDKs)

AWS SDK で AWS CodeBuild を使用する方法の詳細については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild でビルドを停止する

AWS CodeBuild でビルドを停止するには、AWS CodeBuild コンソール、AWS CLI、または AWS SDK を使用できます。

トピック

ビルドを停止する (コンソール)

1. Open the AWS CodeBuild console at https://console.aws.amazon.com/codebuild/.
2. 次のいずれかを行ってください。

   - [*build-project-name*:*build-ID*] ページが表示された場合は、[Stop] を選択します。
   - ナビゲーションペインで、[Build history] を選択します。ビルドのリストで、ビルドに対応するボックスを選択し、次に [Stop] を選択します。
   - ナビゲーションペインで、[Build projects] を選択します。ビルドプロジェクトのリストの [Project] 列で、ビルドプロジェクト名に対応するリンクを選択します。ビルドのリストで、ビルドに対応するボックスを選択し、次に [Stop] を選択します。

   Note

   デフォルトでは、最新の 10 個のビルドまたはビルドプロジェクトのみが表示されます。さらに多くのビルドやビルドプロジェクトを表示するには、[Builds per page] や [Projects per page] で異なる値を選択するか、[Viewing builds] や [Viewing projects] で前後の矢印を選択します。
   AWS CodeBuild でビルドを正常に停止できない場合 (ビルドプロセスが完了済みである場合など)、[Stop] ボタンは使用できないか、まったく表示されないことがあります。

ビルドを停止する (AWS CLI)

- `stop-build` コマンドを実行します。

```
aws codebuild stop-build --id id
```

上記のコマンドで、次のプレースホルダを置き換えます。

- *id*: 必須の文字列。停止するビルドの ID。ビルド ID のリストを取得するには、次のトピックを参照してください。
  - ビルド ID のリストを表示する (AWS CLI) (p. 176)
  - ビルドプロジェクトのビルド ID のリストを表示する (AWS CLI) (p. 177)

AWS CodeBuild がビルドを正常に停止した場合、出力で `build` オブジェクトの `buildStatus` 値が `STOPPED` になります。

AWS CodeBuild がビルドを正常に停止できない場合 (たとえば、ビルドがすでに完了している場合)、オブジェクトの出力の `build` オブジェクトの `buildStatus` 値が最終的なビルドステータス (例: `SUCCEEDED`) になります。

ビルドを停止する (AWS SDK)

AWS SDK で AWS CodeBuild を使用する方法の詳細については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

AWS CodeBuild でのビルドの削除

AWS CodeBuild でビルドを削除するには、AWS CLI または AWS SDK を使用できます。

ビルドの削除 (AWS CLI)

`batch-delete-builds` コマンドを実行します。

```
aws codebuild batch-delete-builds --ids ids
```

上記のコマンドで、次のプレースホルダを置き換えます。

- *ids*: 必須の文字列。削除するビルドの ID。複数のビルドを指定するには、各ビルド ID をスペースで区切ります。ビルド ID のリストを取得するには、次のトピックを参照してください。
  - ビルド ID のリストを表示する (AWS CLI) (p. 176)
  - ビルドプロジェクトのビルド ID のリストを表示する (AWS CLI) (p. 177)

成功すると、`buildsDeleted` 配列が出力に表示されます。この配列には、正常に削除された各ビルドの Amazon リソースネーム (ARN) が含まれています。正常に削除されなかったビルドに関する情報は、出力の `buildsNotDeleted` 配列内に表示されます。

たとえば、次のコマンドを実行するとします。

```
aws codebuild batch-delete-builds --ids my-demo-build-project:f8b888d2-5e1e-4032-8645-
b115195648EX my-other-demo-build-project:a18bc6ee-e499-4887-b36a-8c90349c7eEX
```

次のような情報が出力に表示されます。

```
{
  "buildsNotDeleted": [
    {
      "id": "arn:aws:codebuild:us-west-2:123456789012:build/my-demo-build-
project:f8b888d2-5e1e-4032-8645-b115195648EX",
      "statusCode": "BUILD_IN_PROGRESS"
    }
  ],
  "buildsDeleted": [
    "arn:aws:codebuild:us-west-2:123456789012:build/my-other-demo-build-project:a18bc6ee-
e499-4887-b36a-8c90349c7eEX"
  ]
}
```

ビルドの削除 (AWS SDK)

AWS SDK で AWS CodeBuild を使用する方法の詳細については、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

高度なトピック

このセクションでは、経験豊富な AWS CodeBuild ユーザーに役立ついくつかの高度なトピックを示します。

トピック

高度な設定

AWS CodeBuild に初めてアクセスするための「ご利用開始にあたって (p. 4)」の手順を実行する場合、このトピックの情報を参照する必要はほとんどありません。ただし、引き続き AWS CodeBuild を使用する場合は、組織の IAM グループやユーザーに AWS CodeBuild へのアクセスを付与したり、AWS CodeBuild にアクセスするため IAM の既存のサービスロールや AWS KMS のカスタマーマスターキーを変更したり、AWS CodeBuild にアクセスするため組織のワークステーション全体で AWS CLI をセットアップしたりする場合があります。このトピックでは、関連するセットアップ手順の実行方法について説明します。

AWS アカウントは既にあるものとします。ただし、まだアカウントがない場合は、http://aws.amazon.com に移動し、[Sign In to the Console] を選択してオンラインの指示に従ってください。

トピック

IAM グループまたは IAM ユーザーに AWS CodeBuild アクセス許可を追加する

AWS CodeBuild にアクセスするには、IAM グループまたは IAM ユーザーにアクセス許可を追加する必要があります。このセクションでは、IAM コンソールまたは AWS CLI でこれを行う方法について説明します。

AWS ルートアカウント (非推奨) または AWS アカウントの管理者 IAM ユーザーで AWS CodeBuild にアクセスする場合は、これらの手順に従う必要はありません。

AWS ルートアカウントと管理者 IAM ユーザーについては、IAM ユーザーガイドの、「アカウント root ユーザー」および「最初の IAM 管理者ユーザーおよびグループの作成」を参照してください。

IAM グループまたは IAM ユーザー (コンソール) に AWS CodeBuild アクセス許可を追加するには

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。

   次のいずれかを使用して、AWS マネジメントコンソール に既にサインインしている必要があります。

- AWS ルートアカウント。これは推奨されません。詳細については、IAM ユーザーガイドの「アカウント root ユーザー」を参照してください。
- AWS アカウントの管理者 IAM ユーザー。詳細については、IAM ユーザーガイドの「最初の IAM 管理者ユーザーおよびグループの作成」を参照してください。
- 次の最小限のアクションを実行する権限がある AWS アカウントの IAM ユーザー。

```
iam:AttachGroupPolicy
iam:AttachUserPolicy
iam:CreatePolicy
iam:ListAttachedGroupPolicies
iam:ListAttachedUserPolicies
iam:ListGroups
iam:ListPolicies
iam:ListUsers
```

詳細については、IAM ユーザーガイドの「IAM ポリシーの概要」を参照してください。

2. ナビゲーションペインで、[Policies] を選択します。
3. カスタムセットの AWS CodeBuild アクセス許可を IAM グループまたは IAM ユーザーに追加するには、この手順のステップ 4 に進んでください。

   IAM グループや IAM ユーザーにデフォルトの AWS CodeBuild アクセス許可セットを追加するには、[Policy Type]、[AWS Managed] の順に選択し、以下の操作を行います。

   - AWS CodeBuild にフルアクセス許可を追加するには、[AWSCodeBuildAdminAccess] という名前のボックスを選択します。次に [Policy Actions]、[Attach] の順に選択します。対象の IAM グループや IAM ユーザーの横にあるボックスを選択し、[Attach Policy] を選択します。AmazonS3ReadOnlyAccess ポリシーおよび IAMFullAccess ポリシーに対して、この操作を繰り返します。
   - AWS CodeBuild にビルドプロジェクト管理以外のすべてのアクセス許可を追加するには、[AWSCodeBuildDeveloperAccess] という名前のボックスを選択します。次に [Policy Actions]、[Attach] の順に選択します。対象の IAM グループや IAM ユーザーの横にあるボックスを選択し、[Attach Policy] を選択します。AmazonS3ReadOnlyAccess ポリシーに対して、この操作を繰り返します。
   - AWS CodeBuild に読み取り専用アクセス許可を追加するには、[AWSCodeBuildReadOnlyAccess] という名前のボックスを選択します。対象の IAM グループや IAM ユーザーの横のボックスを選択し、[Attach Policy] を選択します。AmazonS3ReadOnlyAccess ポリシーに対して、この操作を繰り返します。

   これで、IAM グループまたは IAM ユーザーへの AWS CodeBuild のアクセス許可のデフォルトセットが追加されました。この手順の残りの手順をスキップします。
4. [Create Policy] を選択します。
5. [Create Policy] ページで、[Create Your Own Policy] の横にある [Select] を選択します。
6. [Review Policy] ページで、[Policy Name] にポリシー名を入力します (例: **`CodeBuildAccessPolicy`**)。別の名前を使用する場合は、この手順全体でそれを置き換えてください。
7. [Policy Document] に、以下のように入力し、[Create Policy] を選択します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CodeBuildDefaultPolicy",
      "Effect": "Allow",
      "Action": [
```

```
        "codebuild:*",
        "iam:PassRole"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsAccessPolicy",
      "Effect": "Allow",
      "Action": [
        "logs:FilterLogEvents",
        "logs:GetLogEvents"
      ],
      "Resource": "*"
    },
    {
      "Sid": "S3AccessPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:GetObject",
        "s3:List*",
        "s3:PutObject"
      ],
      "Resource": "*"
    }
  ]
}
```

Note

このポリシーにより、すべての AWS CodeBuild アクションへのアクセスが許可され、多数の AWS リソースへのアクセスが許可される可能性があります。アクセス許可を特定の AWS CodeBuild アクションに限定するには、AWS CodeBuild ポリシーステートメントの `codebuild:*` の値を変更します。詳細については、「認証とアクセスコントロール (p. 193)」を参照してください。特定の AWS リソースへのアクセスを制限するには、`Resource` オブジェクトの値を変更します。詳細については、「認証とアクセスコントロール (p. 193)」を参照してください。

8. ナビゲーションペインで、[Groups] または [Users] を選択します。
9. グループまたはユーザーのリストで、AWS CodeBuild アクセス許可を追加する IAM グループまたは IAM ユーザーの名前を選択します。
10. グループの場合は、グループ設定ページの [Permissions] タブで [Managed Policies] を展開し、[Attach Policy] を選択します。

    ユーザーの場合は、ユーザー設定ページの [Permissions] タブで、[Add permissions] を選択します。
11. グループの場合は、[Attach Policy] ページで [CodeBuildAccessPolicy] を選択し、[Attach Policy] を選択します。

    ユーザーの場合は、[Add permissions] ページで [Attach existing policies directly] を選択します。[CodeBuildAccessPolicy] を選択し、[Next: Reivew]、[Add permissions] の順に選択します。

IAM グループまたは IAM ユーザー (AWS CLI) に AWS CodeBuild アクセス許可を追加するには

1. 前の手順で説明しているように、IAM エンティティのいずれかに対応する AWS アクセスキーと AWS シークレットアクセスキーを使用して AWS CLI が設定されていることを確認します。詳細については、AWS Command Line Interface User Guide の「AWS Command Line Interface のセットアップ」を参照してください。
2. カスタムセットの AWS CodeBuild アクセス許可を IAM グループまたは IAM ユーザーに追加するには、この手順のステップ 3 に進んでください。

IAM グループまたは IAM ユーザーに、AWS CodeBuild アクセス許可のデフォルトセットを追加するには以下を実行します。

IAM グループまたは IAM ユーザーのどちらにアクセス許可を追加するかに応じて、次のいずれかのコマンドを実行します。

```
aws iam attach-group-policy --group-name group-name --policy-arn policy-arn

aws iam attach-user-policy --user-name user-name --policy-arn policy-arn
```

group-name または *user-name* と IAM グループ名または IAM ユーザー名を置き換えて、以下のポリシー Amazon リソースネーム (ARN) ごとに *policy-arn* を置き換えるために、コマンドを 3 回実行する必要があります。

- AWS CodeBuild にフルアクセス許可を追加するには、以下のポリシー ARN を使用します。
  - `arn:aws:iam::aws:policy/AWSCodeBuildAdminAccess`
  - `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`
  - `arn:aws:iam::aws:policy/IAMFullAccess`
- ビルドプロジェクトの管理以外のすべてに対して AWS CodeBuild にアクセス許可を追加するには、次のポリシー ARN を使用します。
  - `arn:aws:iam::aws:policy/AWSCodeBuildDeveloperAccess`
  - `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`
- AWS CodeBuild に読み取り専用アクセス許可を追加するには、以下のポリシー ARN を使用します。
  - `arn:aws:iam::aws:policy/AWSCodeBuildReadOnlyAccess`
  - `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`

これで、IAM グループまたは IAM ユーザーへの AWS CodeBuild のアクセス許可のデフォルトセットが追加されました。この手順の残りの手順をスキップします。

3. AWS CLI がインストールされているローカルワークステーションまたはインスタンス上の空のディレクトリに、`put-group-policy.json` または `put-user-policy.json` という名前のファイルを作成します。別のファイル名を使用する場合は、この手順全体でそれを置き換えてください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CodeBuildAccessPolicy",
      "Effect": "Allow",
      "Action": [
        "codebuild:*",
        "iam:PassRole"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CloudWatchLogsAccessPolicy",
      "Effect": "Allow",
      "Action": [
        "logs:FilterLogEvents",
        "logs:GetLogEvents"
      ],
      "Resource": "*"
    },
    {
      "Sid": "S3AccessPolicy",
```

```
      "Effect": "Allow",
      "Action": [
        "s3:CreateBucket",
        "s3:GetObject",
        "s3:List*",
        "s3:PutObject"
      ],
      "Resource": "*"
    }
  ]
}
```

Note

このポリシーにより、すべての AWS CodeBuild アクションへのアクセスが許可され、多数の AWS リソースへのアクセスが許可される可能性があります。アクセス許可を特定の AWS CodeBuild アクションに限定するには、AWS CodeBuild ポリシーステートメントの `codebuild:*` の値を変更します。詳細については、「認証とアクセスコントロール (p. 193)」を参照してください。特定の AWS リソースへのアクセスを制限するには、関連する `Resource` オブジェクトの値を変更します。詳細については、「認証とアクセスコントロール (p. 193)」または特定の AWS サービスのセキュリティドキュメントを参照してください。

4. ファイルを保存したディレクトリに移動し、以下のいずれかのコマンドを実行します。`CodeBuildGroupAccessPolicy` および `CodeBuildUserAccessPolicy` に異なる値を使用できます。異なる値を使用する場合は、ここで置き換えてください。

   IAM グループの場合:

   ```
   aws iam put-group-policy --group-name group-name --policy-name
    CodeBuildGroupAccessPolicy --policy-document file://put-group-policy.json
   ```

   IAM ユーザーの場合:

   ```
   aws iam put-user-policy --user-name user-name --policy-name CodeBuildUserAccessPolicy
    --policy-document file://put-user-policy.json
   ```

   前述のコマンドでは、*group-name* または *user-name* をターゲット IAM グループまたは IAM ユーザーの名前と置き換えます。

AWS CodeBuild サービスロールの作成

AWS CodeBuild サービスロールが必要です。これにより、AWS CodeBuild が、ユーザーに代わって依存 AWS サービスとやり取りできるようになります。AWS CodeBuild または AWS CodePipeline コンソールを使用して、AWS CodeBuild サービスロールを作成できます。詳細については、以下を参照してください。

- ビルドプロジェクトの作成 (コンソール) (p. 146)
- AWS CodeBuild を使用するパイプラインを作成する (AWS CodePipeline コンソール) (p. 128)
- AWS CodeBuild ビルドアクションをパイプラインに追加する (AWS CodePipeline コンソール) (p. 135)
- ビルドプロジェクトの設定を変更する (コンソール) (p. 162)

これらのコンソールを使用する予定がない場合のために、このセクションでは、IAM コンソールまたは AWS CLI を使用して AWS CodeBuild サービスロールを作成する方法について説明します。

AWS CodeBuild サービスロールを作成するには (コンソール)

1. https://console.aws.amazon.com/iam/ にある IAM コンソールを開きます。

   次のいずれかを使用して、コンソールに既にサインインしている必要があります。

   - AWS ルートアカウント。これは推奨されません。詳細については、IAM ユーザーガイドの「アカウント root ユーザー」を参照してください。
   - AWS アカウントの管理者 IAM ユーザー。詳細については、IAM ユーザーガイドの「最初の IAM 管理者ユーザーおよびグループの作成」を参照してください。
   - 次の最小限のアクションを実行する権限がある AWS アカウントの IAM ユーザー。

   ```
   iam:AddRoleToInstanceProfile
   iam:AttachRolePolicy
   iam:CreateInstanceProfile
   iam:CreatePolicy
   iam:CreateRole
   iam:GetRole
   iam:ListAttachedRolePolicies
   iam:ListPolicies
   iam:ListRoles
   iam:PassRole
   iam:PutRolePolicy
   iam:UpdateAssumeRolePolicy
   ```

   詳細については、IAM ユーザーガイドの「IAM ポリシーの概要」を参照してください。
2. ナビゲーションペインで、[Policies] を選択します。
3. [Create Policy] を選択します。
4. [Create Policy] ページで、[JSON] を選択します。
5. [JSON Policy] に以下の内容を入力し、[Review Policy] を選択します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogsPolicy",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CodeCommitPolicy",
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3GetObjectPolicy",
      "Effect": "Allow",
      "Action": [
```

```
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3PutObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Note

このポリシーに含まれているステートメントでは、多数の AWS リソースへのアクセスが許可される可能性があります。AWS CodeBuild に特定の AWS リソースへのアクセスを制限するには、`Resource` 配列の値を変更します。詳細については、AWS サービスのセキュリティドキュメントを参照してください。

6. [Review Policy] ページで、[Policy Name] にポリシー名を入力します (例: **`CodeBuildServiceRolePolicy`**)。次に [Create policy] を選択します。

   Note

   別の名前を使用する場合は、この手順全体でそれを置き換えてください。

7. ナビゲーションペインで [Roles (ロール)] を選択します。
8. [Create role] を選択します。
9. [Create role] ページで [AWS Service] が選択済みの状態で、このロールを使用するサービスとして [CodeBuild] を選択し、次に [Next:Permissions] を選択します。
10. [Attach permissions policies] ページで、[CodeBuildServiceRolePolicy]、[Next: Review] の順に選択します。
11. [Create role and review] ページで、[Role name] にロール名を入力します (例: **`CodeBuildServiceRole`**)。次に [Create role] を選択します。

AWS CodeBuild サービスロールを作成するには (AWS CLI)

1. 前の手順で説明しているように、IAM エンティティのいずれかに対応する AWS アクセスキーと AWS シークレットアクセスキーを使用して AWS CLI が設定されていることを確認します。詳細については、AWS Command Line Interface User Guide の「AWS Command Line Interface のセットアップ」を参照してください。
2. AWS CLI がインストールされているローカルワークステーションまたはインスタンスの空のディレクトリに、`create-role.json` および `put-role-policy.json` という名前の 2 つのファイルを作成します。別のファイル名を選択した場合は、この手順で置き換えてください。

   `create-role.json`:

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

```
      "Principal": {
        "Service": "codebuild.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

put-role-policy.json:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogsPolicy",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CodeCommitPolicy",
      "Effect": "Allow",
      "Action": [
        "codecommit:GitPull"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3GetObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "S3PutObjectPolicy",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

Note

このポリシーに含まれているステートメントでは、多数の AWS リソースへのアクセスが許可される可能性があります。AWS CodeBuild に特定の AWS リソースへのアクセスを制限す

るには、Resource 配列の値を変更します。詳細については、AWS サービスのセキュリティドキュメントを参照してください。

3. 上記のファイルを保存したディレクトリに移動し、以下の 2 つのコマンドをこの順番で 1 つずつ実行します。CodeBuildServiceRole と CodeBuildServiceRolePolicy には異なる値を使用することができますが、ここでそれらを代用してください。

```
aws iam create-role --role-name CodeBuildServiceRole --assume-role-policy-document
 file://create-role.json
```

```
aws iam put-role-policy --role-name CodeBuildServiceRole --policy-name
 CodeBuildServiceRolePolicy --policy-document file://put-role-policy.json
```

AWS CodeBuild の AWS KMSCMK の作成と設定

AWS CodeBuild がビルド出力アーティファクトを暗号化するには、AWS KMS カスタマーマスタキー (CMK) にアクセスする必要があります。デフォルトでは、AWS CodeBuild は、AWS アカウントの Amazon S3 の AWS 管理 CMK を使用します。

この CMK を使用しない場合は、お客様が管理する CMK を自分で作成して設定する必要があります。このセクションでは、IAM コンソールを使用してこれを行う方法を説明します。

CMK の詳細については、AWS KMS 開発者ガイドの「AWS Key Management Service の概念」と「キーの作成」を参照してください。

AWS CodeBuild で使用する CMK を設定するには、AWS KMS 開発者ガイドの「キーポリシーの変更」の手順に従ってください。次に、キーポリシーに以下のステートメント (*###BEGIN ADDING STATEMENTS HERE###* と *###END ADDING STATEMENTS HERE###* の間) を追加します。省略記号 (...) は、簡潔にするために使用され、ステートメントを追加する場所の特定に役立ちます。ステートメントを削除しないでください、また、これらの省略記号をキーポリシーに入力しないでください。

```
{
  "Version": "2012-10-17",
  "Id": "...",
  "Statement": [
    ### BEGIN ADDING STATEMENTS HERE ###
    {
      "Sid": "Allow access through Amazon S3 for all principals in the account that are
 authorized to use Amazon S3",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "s3.region-ID.amazonaws.com",
          "kms:CallerAccount": "account-ID"
        }
      }
    },
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::account-ID:role/CodeBuild-service-role"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    ### END ADDING STATEMENTS HERE ###
    {
      "Sid": "Enable IAM User Permissions",
      ...
    },
    {
      "Sid": "Allow access for Key Administrators",
      ...
    },
    {
      "Sid": "Allow use of the key",
      ...
    },
    {
      "Sid": "Allow attachment of persistent resources",
      ...
    }
  ]
}
```

- *region-ID* は、AWS CodeBuild に関連付けられた Amazon S3 バケットが配置されている AWS リージョンの ID を表します (たとえば、`us-east-1`)。
- *account-ID* は、CMK を所有する AWS アカウントの ID を表します。
- *CodeBuild-service-role* は、このトピックの前半で作成または識別した AWS CodeBuild サービスロールの名前を表します。

> **Note**
>
> IAM コンソールで CMK を作成または設定するには、まず次のいずれかを使用して AWS マネジメントコンソール にサインインする必要があります。
>
> - AWS ルートアカウント。これは推奨されません。詳細については、IAM ユーザーガイドの「アカウント root ユーザー」を参照してください。
> - AWS アカウントの管理者 IAM ユーザー。詳細については、IAM ユーザーガイドの「最初の IAM 管理者ユーザーおよびグループの作成」を参照してください。
> - CMK を作成または変更する権限が与えられている AWS アカウントの IAM ユーザー。詳細については、AWS KMS 開発者ガイドの、「AWS KMS コンソールを使用するために必要なアクセス許可」を参照してください。

AWS CLI のインストールと設定

AWS CodeBuild にアクセスするには、AWS CLI を使用できます (AWS CodeBuild コンソール、AWS CodePipeline コンソール、または AWS SDK と一緒に使用するか、これらの代わりに使用できます)。AWS CLI をインストールして設定するには、AWS Command Line Interface ユーザーガイドの「AWS Command Line Interface のセットアップ」を参照してください。

1. 次のコマンドを実行して、AWS CLI のインストールが AWS CodeBuild をサポートしているかどうかを確認します。

```
aws codebuild list-builds
```

成功すると、次のような情報が出力に表示されます。

```
{
  "ids": []
}
```

空の角括弧は、まだビルドを実行していないことを示しています。

2. エラーが出力された場合は、現在のバージョンの AWS CLI をアンインストールしてから、最新バージョンをインストールする必要があります。詳細については、AWS Command Line Interface ユーザーガイドの「AWS CLI のアンインストール」および「AWS Command Line Interface のインストール」を参照してください。

AWS CodeBuild のコマンドラインリファレンス

AWS CLI は、AWS CodeBuild を自動化するためのコマンドを提供します。このトピックの情報は、AWS Command Line Interface ユーザーガイド および AWS CLI Reference for AWS CodeBuild の補足資料として活用してください。

お探しのものではありませんか。AWS SDK を使用して AWS CodeBuild を呼び出す場合は、「AWS SDK とツールのリファレンス (p. 192)」を参照してください。

このトピックの情報を使用するには、「AWS CLI のインストールと設定 (p. 190)」の説明に従って AWS CLI をインストールし、AWS CodeBuild 用に設定しておく必要があります。

次のコマンドでは、AWS CodeBuild のコマンドのリストを取得できます。

```
aws codebuild help
```

次のコマンドでは、AWS CodeBuild コマンドに関する情報を取得できます。*command-name* はコマンド名です。

```
aws codebuild command-name help
```

AWS CodeBuild のコマンドは以下のとおりです。

- `batch-delete-builds`: AWS CodeBuild の 1 つ以上のビルドを削除します。詳細については、「ビルドの削除 (AWS CLI) (p. 180)」を参照してください。
- `batch-get-builds`: AWS CodeBuild の複数のビルドに関する情報を取得します。詳細については、「ビルドの詳細を表示する (AWS CLI) (p. 174)」を参照してください。
- `batch-get-projects`: 指定された 1 つ以上のビルドプロジェクトに関する情報を取得します。詳細については、「ビルドプロジェクトの詳細を表示する (AWS CLI) (p. 161)」を参照してください。
- `create-project`: ビルドプロジェクトを作成します。詳細については、「ビルドプロジェクトを作成する (AWS CLI) (p. 151)」を参照してください。
- `delete-project`: ビルドプロジェクトを削除します。詳細については、「ビルドプロジェクトを削除する (AWS CLI) (p. 166)」を参照してください。
- `list-builds`: AWS CodeBuild でのビルドの Amazon リソースネーム (ARN) をリスト表示します。詳細については、「ビルド ID のリストを表示する (AWS CLI) (p. 176)」を参照してください。

- `list-builds-for-project`: 指定されたビルドプロジェクトに関連付けられているビルド ID のリストを取得します。詳細については、「ビルドプロジェクトのビルド ID のリストを表示する (AWS CLI) (p. 177)」を参照してください。
- `list-curated-environment-images`: ビルドに使用できる AWS CodeBuild によって管理される Docker イメージのリストを取得します。詳細については、「AWS CodeBuild に用意されている Docker イメージ (p. 103)」を参照してください。
- `list-projects`: ビルドプロジェクト名のリストを取得します。詳細については、「ビルドプロジェクト名のリストを表示する (AWS CLI) (p. 159)」を参照してください。
- `start-build`: ビルドの実行を開始します。詳細については、「ビルドの実行 (AWS CLI) (p. 169)」を参照してください。
- `stop-build`: 停止されたビルドの実行を停止しようとします。詳細については、「ビルドを停止する (AWS CLI) (p. 179)」を参照してください。
- `update-project`: 指定されたビルドプロジェクトに関する情報を変更します。詳細については、「ビルドプロジェクトの設定を変更する (AWS CLI) (p. 165)」を参照してください。

AWS CodeBuild の AWS SDK とツールのリファレンス

いずれかの AWS SDK またはツールを使用して AWS CodeBuild を自動化するには、以下のリソースを参照してください。

AWS CLI を使用して AWS CodeBuild を実行する場合は、「コマンドラインリファレンス (p. 191)」を参照してください。

AWS CodeBuild でサポートされる AWS SDK とツール

AWS CodeBuild でサポートしている AWS SDK とツールは以下のとおりです。

- AWS SDK for C++。詳細については、AWS SDK for C++ API リファレンスの Aws::CodeBuild 名前空間のセクションを参照してください。
- AWS SDK for Go。詳細については、AWS SDK for Go API リファレンスの codebuild セクションを参照してください。
- AWS SDK for Java。詳細については、`com.amazonaws.services.codebuild` および `com.amazonaws.services.codebuild.model` および AWS SDK for Java API リファレンスを参照してください。
- ブラウザでの AWS SDK for JavaScript および Node.js での AWS SDK for JavaScript。詳細については、クラス: AWS AWS CodeBuild セクション (AWS SDK for JavaScript API リファレンス) を参照してください。
- AWS SDK for .NET。詳細については、Amazon.CodeBuild および Amazon.CodeBuild.Model 名前空間のセクション (AWS SDK for .NET API リファレンス) を参照してください。
- AWS SDK for PHP。詳細については、AWS SDK for PHP API リファレンスの 名前空間 Aws\CodeBuild セクションを参照してください。
- AWS SDK for Python (Boto3)。詳細については、Boto 3 ドキュメントの CodeBuild セクションを参照してください。
- AWS SDK for Ruby。詳細については、AWS SDK for Ruby API リファレンスの モジュール: Aws\CodeBuild セクションを参照してください。
- AWS Tools for PowerShell。詳細については、AWS Tools for PowerShell Cmdlet リファレンスの「AWS CodeBuild」セクションを参照してください。

AWS CodeBuild に対する認証とアクセスコントロール

AWS CodeBuild へのアクセスには、認証情報が必要です。 認証情報は、Amazon S3 バケットのビルドアーティファクトの保存や取得、および、ビルドのための Amazon CloudWatch Logs の表示など、AWS リソースにアクセスするための権限を持っている必要があります。以下のセクションでは、AWS Identity and Access Management (IAM) と AWS CodeBuild を使用してリソースに安全にアクセスする方法について説明します。

認証

AWS には、次のタイプのアイデンティティでアクセスできます。

- AWS アカウントのルートユーザー – AWS にサインアップするときは、AWS アカウントに関連付けられた E メールアドレスとパスワードを指定します。これらはルート認証情報であり、これらの情報を使用すると、すべての AWS リソースへの完全なアクセスが可能になります。

  Important

  セキュリティ上の理由から、AWS アカウントへの完全なアクセス権限を持つ管理者ユーザー (IAM ユーザー) を作成するためにのみ、ルート認証情報を使用することをお勧めします。その後、この管理者ユーザーを使用して、制限されたアクセス権限を持つ他の IAM ユーザーとロールを作成できます。詳細については、IAM ユーザーガイド ガイドの「IAM のベストプラクティス」および「管理者のユーザーおよびグループの作成」を参照してください。

- IAM ユーザー – IAM ユーザーは、カスタム権限 (たとえば、AWS CodeBuild でビルドプロジェクトを作成する権限) を持つ AWS アカウントの ID です。 IAM のユーザー名およびパスワードを使用して、AWS マネジメントコンソール、AWS ディスカッションフォーラム、AWS Support Center などの安全な AWS ウェブページにサインインできます。

  ユーザー名とパスワードに加えて、各ユーザーのアクセスキーを生成することもできます。AWS SDK の 1 つまたは AWS Command Line Interface (AWS CLI) を使ってプログラムで AWS サービスにアクセスするときに、これらのキーを使用します。AWS SDK と AWS CLI ツールでは、アクセスキーを使用してリクエストが暗号で署名されます。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。AWS CodeBuild では、署名バージョン 4 がサポートされています。これは、インバウンド API リクエストを認証するためのプロトコルです。リクエストの認証の詳細については、AWS General Referenceの「署名バージョン 4 の署名プロセス」を参照してください。

- IAM ロール – IAM ロールは IAM ユーザーに似ていますが、特定のユーザーに関連付けられていません。IAM ロールでは、AWS サービスおよびリソースにアクセスするために使用できる一時的なアクセスキーを取得することができます。IAM ロールと一時的な認証情報は、次の状況で役立ちます。

  - フェデレーティッドユーザーアクセス – IAM ユーザーを作成するのではなく、AWS Directory Service、エンタープライズユーザーディレクトリ、またはウェブアイデンティティプロバイダーの既存のユーザーアイデンティティを使用することもできます。このようなユーザーはフェデレーティッドユーザーと呼ばれます。AWS では、ID プロバイダーを通じてアクセスがリクエストされたとき、フェデレーティッドユーザーにロールを割り当てます。フェデレーティッドユーザーの詳細については、IAM ユーザーガイド ガイドの「フェデレーティッドユーザーとロール」を参照してください。
  - クロスアカウントアクセス – アカウントの IAM ロールを使用して、アカウントのリソースにアクセスするための権限を別の AWS アカウントに付与することができます。この例については、IAM ユーザーガイド ガイドの「チュートリアル: AWS アカウント間の IAM ロールを使用したアクセスの委任」を参照してください。

- AWS サービスのアクセス – アカウントで IAM ロールを使用して、アカウントのリソースにアクセスするために AWS サービスにアクセス許可を付与できます。たとえば、Amazon Redshift がお客様に代わって Amazon S3 バケットにアクセスし、バケットに保存されたデータを Amazon Redshift クラスターにロードすることを許可するロールを作成できます。 詳細については、IAM ユーザーガイド ガイドの「AWS ユーザー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。
- Amazon EC2 で実行されるアプリケーション – インスタンスで実行し、AWS API リクエストを作成するアプリケーションで使用されるアクセスキーを Amazon EC2 インスタンスに保存する代わりに、IAM ロールを使用して、これらのアプリケーション用の一時認証情報を管理できます。AWS ロールを Amazon 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、Amazon EC2 インスタンスで実行されるプログラムは一時認証情報を取得することができます。詳細については、IAM ユーザーガイド ガイドの「Amazon EC2 上のアプリケーションに対するロールの使用」を参照してください。

アクセスコントロール

有効な認証情報があればリクエストを認証できますが、許可を持っていないかぎり AWS CodeBuild リソースの作成やアクセスはできません。たとえば、ビルドプロジェクトの作成、表示、削除、および、ビルドの開始、停止、表示には、アクセス許可が必要です。

以下のセクションでは、AWS CodeBuild のアクセス許可を管理する方法について説明します。最初に概要のセクションを読むことをお勧めします。

- AWS CodeBuild リソースへのアクセス許可の管理の概要 (p. 194)
- AWS CodeBuild でアイデンティティベースのポリシー (IAM ポリシー) を使用する (p. 197)
- AWS CodeBuild の権限リファレンス (p. 204)

AWS CodeBuild リソースへのアクセス許可の管理の概要

すべての AWS リソースは AWS アカウントによって所有され、リソースを作成またはアクセスするためのアクセス権限は、アクセス権限ポリシーによって管理されます。アカウント管理者は、IAM ID (ユーザー、グループ、ロール) にアクセス権限ポリシーをアタッチできます。

> Note
>
> アカウント管理者 (または管理者ユーザー) は、管理者権限を持つユーザーです。詳細については、IAM ユーザーガイド ガイドの「IAM ベストプラクティス」を参照してください。

アクセス許可を付与するときは、アクセス許可を取得するユーザー、アクセスできるリソース、およびそれらのリソースに対して実行できるアクションを決定します。

トピック

- AWS CodeBuild リソースおよびオペレーション (p. 194)
- リソース所有権について (p. 195)
- リソースへのアクセスの管理 (p. 196)
- ポリシー要素の指定 : アクション、効果、プリンシパル (p. 197)

AWS CodeBuild リソースおよびオペレーション

AWS CodeBuild で、プライマリリソースはビルドプロジェクトです。ポリシーで Amazon リソースネーム (ARN) を使用して、ポリシーを適用するリソースを識別します。ビルドもリソースで、ARN が関連付

けられています。詳細については、アマゾン ウェブ サービス全般のリファレンス の「Amazon リソースネーム (ARN) と AWS のサービスの名前空間」を参照してください。

リソースタイプ	ARN 形式
ビルドプロジェクト	`arn:aws:codebuild:region-ID:account-ID:project/project-name`
構築	`arn:aws:codebuild:region-ID:account-ID:build/build-ID`
すべての AWS CodeBuild リソース	`arn:aws:codebuild:*`
特定リージョンの特定アカウントが所有するすべての AWS CodeBuild リソース	`arn:aws:codebuild:region-ID:account-ID:*`

> **Note**
>
> AWS のほとんどのサービスでは、ARN 内のコロン (:) またはスラッシュ (/) は同じ文字として扱われます。ただし、AWS CodeBuild では、リソースパターンとルールで完全一致が使用されます。イベントパターンの作成時に正しい文字を使用して、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。

たとえば、以下のように ARN を使用して、ステートメント内で特定のビルドプロジェクト (*myBuildProject*) を指定できます。

```
"Resource": "arn:aws:codebuild:us-east-2:123456789012:project/myBuildProject"
```

すべてのリソースを指定する場合、または API アクションが ARN をサポートしていない場合は、以下の要領で、`Resource` エレメント内でワイルドカード文字 (*) を使用します。

```
"Resource": "*"
```

一部の AWS CodeBuild API アクションは複数のリソースを受け入れます (例:`BatchGetProjects`)。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": [
  "arn:aws:codebuild:us-east-2:123456789012:project/myBuildProject",
  "arn:aws:codebuild:us-east-2:123456789012:project/myOtherBuildProject"
]
```

AWS CodeBuild には、AWS CodeBuild リソースを操作するための一連のオペレーションが用意されています。リストについては、「AWS CodeBuild の権限リファレンス (p. 204)」を参照してください。

リソース所有権について

AWS アカウントは、誰がリソースを作成したかにかかわらず、アカウントで作成されたリソースを所有します。具体的には、リソース所有者は、リソースの作成リクエストを認証するプリンシパルエンティティ (ルートアカウント、IAM ユーザー、または IAM ロール) の AWS アカウントです。以下の例では、このしくみを示しています。

- AWS アカウントのルートアカウントの認証情報を使用してルールを作成する場合、AWS アカウントは AWS CodeBuild リソースの所有者です。

- AWS アカウントに IAM ユーザーを作成し、そのユーザーに AWS CodeBuild リソースを作成するためのアクセス許可を付与する場合、そのユーザーは AWS CodeBuild リソースを作成できます。ただし、ユーザーが属する AWS アカウントが AWS CodeBuild リソースを所有します。
- AWS CodeBuild リソースを作成するためのアクセス許可を持つ AWS アカウントに IAM ロールを作成する場合は、ロールを引き受けることのできるいずれのユーザーも AWS CodeBuild リソースを作成できます。ロールが属する AWS アカウントは AWS CodeBuild リソースを所有しているとします。

リソースへのアクセスの管理

アクセスポリシーでは、誰がどのリソースにアクセスできるかを記述します。

> Note
>
> このセクションでは、AWS CodeBuild での IAM の使用について説明します。これは、IAM サービスに関する詳細情報を取得できません。完全な IAM ドキュメントについては、「IAM とは?」(IAM ユーザーガイド ガイド) を参照してください。IAM ポリシー構文の詳細および説明については、IAM ユーザーガイド ガイドの「AWS IAM ポリシーリファレンス」を参照してください。

IAM アイデンティティにアタッチされたポリシーはアイデンティティベースのポリシー (IAM ポリシー) と呼ばれます。リソースにアタッチされたポリシーはリソースベースのポリシーと呼ばれます。AWS CodeBuild では、アイデンティティベースのポリシー (IAM ポリシー) のみがサポートされています。

アイデンティティベースのポリシー (IAM ポリシー)

ポリシーを IAM アイデンティティにアタッチできます。

- アカウントのユーザーまたはグループにアクセス許可ポリシーをアタッチする – ユーザーに、ビルドプロジェクトおよび AWS CodeBuild コンソールの AWS CodeBuild リソースを表示するアクセス許可を付与するためには、ユーザーが所属するユーザーまたはグループにアクセス許可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アクセス権限を付与) – アイデンティティ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アクセス権限を付与することができます。たとえば、アカウント A の管理者は、次のように別の AWS アカウント (たとえば、アカウント B) または AWS サービスにクロスアカウントアクセス権限を付与するロールを作成できます。
  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 のリソースに対するアクセス権限を付与するロールに、アクセス権限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 (ロールを継承できるプリンシパル) を識別するロールに信頼ポリシーをアタッチします。
  3. アカウント B の管理者は、アカウント B のユーザーにロールを引き受ける権限を委任できるようになります。これにより、アカウント B のユーザーにアカウント A のリソースの作成とアクセスが許可されます。AWS サービスのアクセス許可を付与してロールを引き受けさせたい場合は、信頼ポリシー内のプリンシパルも、AWS サービスのプリンシパルである必要があります。

  IAM を使用したアクセス許可の委任の詳細については、IAM ユーザーガイド ガイドの「アクセス管理」を参照してください。

AWS CodeBuild で、アイデンティティベースのポリシーは、デプロイプロセスに関連するリソースに対するアクセス許可を管理するために使用されます。たとえば、ビルドプロジェクトへのアクセスを制御できます。

お客様のアカウントのユーザーがアクセスを許可される呼び出しとリソースを制限する IAM ポリシーを作成し、IAM ユーザーにそれらのポリシーをアタッチできます。IAM ロールを作成する方法、および AWS CodeBuild の IAM ポリシーステートメントの例を調べる方法の詳細については、「AWS CodeBuild リソースへのアクセス許可の管理の概要 (p. 194)」を参照してください。

ポリシー要素の指定 : アクション、効果、プリンシパル

AWS CodeBuild リソースごとに、このサービスは、一連の API オペレーションを定義します。 これらの API オペレーションを実行するためのアクセス許可を付与するために、AWS CodeBuild ではポリシーに一連のアクションを定義できます。一部の API オペレーションは、API オペレーションを実行するために複数のアクションに対するアクセス許可を要求できます。 詳細については、「AWS CodeBuild リソースおよびオペレーション (p. 194)」および「AWS CodeBuild の権限リファレンス (p. 204)」を参照してください。

以下は、基本的なポリシーの要素です。

- リソース – Amazon リソースネーム (ARN) を使用して、ポリシーの適用先のリソースを識別します。
- アクション – アクションのキーワードを使用して、許可または拒否するリソースオペレーションを識別します。たとえば、`codebuild:CreateProject` 権限は、`CreateProject` オペレーションを実行する権限をユーザーに与えます。
- 効果 - ユーザーがアクションをリクエストする際の効果を指定します。許可または拒否のいずれかになります。リソースへのアクセスを明示的に許可していない場合、アクセスは暗黙的に拒否されます。リソースへのアクセスを明示的に拒否できます。これは、別のポリシーがアクセスを許可している場合でも、ユーザーがリソースにアクセスできないようにするために行うことができます。
- プリンシパル – アイデンティティベースのポリシー (IAM ポリシー) で、ポリシーがアタッチされているユーザーが暗黙のプリンシパルとなります。リソースベースのポリシーでは、権限を受け取りたいユーザー、アカウント、サービス、またはその他のエンティティを指定します。

IAM ポリシーの構文と記述の詳細については、IAM ユーザーガイド ガイドの「AWS IAM ポリシーリファレンス」を参照してください。

すべての AWS CodeBuild API アクションとそれらが適用されるリソースの表については、「AWS CodeBuild の権限リファレンス (p. 204)」を参照してください。

AWS CodeBuild でアイデンティティベースのポリシー (IAM ポリシー) を使用する

このトピックでは、アカウント管理者が IAM ID (ユーザー、グループ、ロール) にアクセス許可ポリシーをアタッチし、それによって AWS CodeBuild リソースでオペレーションを実行するアクセス許可を付与する方法を示すアイデンティティベースのポリシーの例を示します。

> Important
>
> 初めに、AWS CodeBuild リソースへのアクセスを管理するための基本概念と使用可能なオプションについて説明する概要トピックをお読みになることをお勧めします。詳細については、「AWS CodeBuild リソースへのアクセス許可の管理の概要 (p. 194)」を参照してください。

トピック

- AWS CodeBuild コンソールを使用するために必要なアクセス許可 (p. 198)
- OAuth を採用しているソースプロバイダに接続するために AWS CodeBuild コンソールに必要なアクセス許可 (p. 198)
- AWS CodeBuild の AWS 管理 (定義済み) ポリシー (p. 198)
- お客様が管理するポリシーの例 (p. 199)

`us-east-2` リージョンにあり、`123456789012` のアカウントで、名前が `my` で始まるビルドプロジェクトについての情報のみをユーザーが取得するのを許可するアクセス許可ポリシーの例を次に示します。

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetProjects",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

AWS CodeBuild コンソールを使用するために必要なアクセス許可

AWS CodeBuild コンソールを使用するユーザーは、他の AWS リソースを AWS アカウントで記述できる、最小限のアクセス許可を持っている必要があります。次のサービスからのアクセス許可を持っている必要があります。

- AWS CodeBuild
- Amazon CloudWatch
- AWS CodeCommit (ソースコードを AWS CodeCommit リポジトリに保存している場合)
- Amazon Elastic Container Registry (Amazon ECR) (Amazon ECR リポジトリの Docker イメージに依存するビルド環境を使用している場合)
- Amazon Elastic Container Service (Amazon ECS) (Amazon ECR リポジトリの Docker イメージに依存するビルド環境を使用している場合)
- AWS Identity and Access Management (IAM)
- AWS Key Management Service (AWS KMS)
- Amazon Simple Storage Service (Amazon S3)

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、コンソールは意図したとおりには機能しません。

OAuth を採用しているソースプロバイダに接続するために AWS CodeBuild コンソールに必要なアクセス許可

コンソールでは、以下の API アクションにより、OAuth を採用しているソースプロバイダ (GitHub リポジトリなど) に接続します。

- `codebuild:ListConnectedOAuthAccounts`
- `codebuild:ListRepositories`
- `codebuild:PersistOAuthToken`

OAuth を採用しているソースプロバイダ (GitHub リポジトリなど) をビルドプロジェクトに関連付けるには、AWS CodeBuild コンソールを使用する必要があります。これを行うには、まず上記の API アクションを、AWS CodeBuild コンソールにアクセスするために使用する IAM ユーザーに関連付けられた IAM アクセスポリシーに追加します。

これらの API アクションは、コードで呼び出すためのものではないため、AWS CLI と AWS SDK には含まれていません。

AWS CodeBuild の AWS 管理 (定義済み) ポリシー

AWS は、多くの一般的ユースケースに対処するために、AWS で作成および管理するスタンドアロンの IAM ポリシーを用意しています。これらの AWS 管理ポリシーでは、一般的なユースケースに必要なアク

セス権限が付与されているため、どのアクセス権限が必要なのかを調べる必要がありません。詳細については、IAM ユーザーガイドの「AWS 管理ポリシー」を参照してください。

アカウントのユーザーにアタッチ可能な以下の AWS 管理ポリシーは、AWS CodeBuild に固有のものです。

- AWSCodeBuildAdminAccess – AWS CodeBuild へのフルアクセスを提供します。これには、AWS CodeBuild ビルドプロジェクトを管理するためのアクセス許可が含まれます。
- AWSCodeBuildDeveloperAccess – AWS CodeBuild へのアクセスを提供します。ただし、ビルドプロジェクトの管理は許可されません。
- AWSCodeBuildReadOnlyAccess – AWS CodeBuild への読み取り専用アクセスを提供します。

AWS CodeBuild が作成するビルド出力アーティファクトにアクセスするには、AWS 管理ポリシーとして AmazonS3ReadOnlyAccess もアタッチする必要があります。

AWS CodeBuild サービスロールを作成および管理するには、AWS 管理ポリシーとして IAMFullAccess もアタッチする必要があります。

独自のカスタム IAM ポリシーを作成して、AWS CodeBuild アクションとリソースのためのアクセス許可を許可することもできます。これらのカスタムポリシーは、それらのアクセス権限が必要な IAM ユーザーまたはグループにアタッチできます。

お客様が管理するポリシーの例

このセクションでは、AWS CodeBuild 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例を示しています。AWS CodeBuild API、AWS SDK、または AWS CLI を使用している場合、これらのポリシーは機能します。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、AWS CodeBuild コンソールを使用するために必要なアクセス許可 (p. 198) を参照してください。

以下のサンプル IAM ポリシーを使用して、IAM ユーザーとロールに対して AWS CodeBuild アクセスを制限できます。

トピック

ビルドプロジェクトに関する情報の取得をユーザーに許可する

次のポリシーステートメントの例では、`us-east-2` リージョンにあり、`123456789012` のアカウントで、名前が `my` で始まるビルドプロジェクトについての情報のみをユーザーが取得するのを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetProjects",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

ビルドプロジェクトの作成をユーザーに許可する

以下のポリシーステートメントの例では、名前は問いませんが、us-east-2 リージョンだけにある 123456789012 というアカウントで、特定の AWS CodeBuild のサービスロールのみを使用したビルドプロジェクトの作成をユーザーに許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:CreateProject",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam:123456789012:role/CodeBuildServiceRole"
    }
  ]
}
```

ビルドプロジェクトの削除をユーザーに許可する

次のポリシーステートメントの例では、us-east-2 リージョンにあり、123456789012 のアカウントで、名前が my で始まるビルドプロジェクトをユーザーが削除するのを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:DeleteProject",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

ビルドプロジェクト名のリストをユーザーが取得するのをに許可する

以下のポリシーステートメントの例では、同じアカウントのビルドプロジェクト名のリストをユーザーが取得するのを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:ListProjects",
```

```
      "Resource": "*"
    }
  ]
}
```

ビルドプロジェクトに関する情報の変更をユーザーに許可する

以下のポリシーステートメントの例では、名前は問いませんが、us-east-2 リージョンだけにある 123456789012 というアカウントで、特定の AWS CodeBuild のサービスロールのみを使用したビルドプロジェクトに関する情報の変更をユーザーに許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:UpdateProject",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam:123456789012:role/CodeBuildServiceRole"
    }
  ]
}
```

ビルドに関する情報の取得をユーザーに許可する

次のポリシーステートメントの例では、us-east-2 リージョンにあり、123456789012 のアカウントで、my-build-project および my-other-build-project という名前のビルドプロジェクトについての情報のみをユーザーが取得するのを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetBuilds",
      "Resource": [
        "arn:aws:codebuild:us-east-2:123456789012:project/my-build-project",
        "arn:aws:codebuild:us-east-2:123456789012:project/my-other-build-project"
      ]
    }
  ]
}
```

ビルドプロジェクトのビルド ID のリストの取得をユーザーに許可する

次のポリシーステートメントの例では、us-east-2 リージョンにあり、123456789012 のアカウントで、my-build-project および my-other-build-project という名前のビルドプロジェクトのみのビルド ID のリストをユーザーが取得するのを許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:ListBuildsForProject",
      "Resource": [
```

```
        "arn:aws:codebuild:us-east-2:123456789012:project/my-build-project",
        "arn:aws:codebuild:us-east-2:123456789012:project/my-other-build-project"
      ]
    }
  ]
}
```

ビルド ID のリストの取得をユーザーに許可する

以下のポリシーステートメントの例では、同じアカウントのすべてのビルド ID のリストをユーザーが取得するのを許可します。

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

ビルドの実行の開始をユーザーに許可する

以下のポリシーステートメントの例では、us-east-2 リージョンの 123456789012 というアカウントの、my という名前で始まるビルドプロジェクトのみのビルドをユーザーが実行する許可を与えます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:StartBuild",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

ビルドの停止を試みる許可をユーザーに与える

次のポリシーステートメントの例では、us-east-2 リージョンにあり、123456789012 のアカウントで、名前が my で始まるビルドの停止を試みるのをユーザーに許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:StopBuild",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

ビルドの削除試行をユーザーに許可する

次のポリシーステートメントの例では、us-east-2 リージョンに限り、123456789012 アカウントで名前が my で始まるビルドプロジェクトの削除試行をユーザーに許可します。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchDeleteBuilds",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

AWS CodeBuild によって管理される Docker イメージに関する情報の取得をユーザーに許可する

次のポリシーステートメントの例では、AWS CodeBuild で管理するすべての Docker イメージに関する情報を取得することをユーザーに許可します。

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

VPC ネットワークインターフェイスの作成をユーザーに許可する

次のポリシーステートメントの例では、Amazon VPC でネットワークインターフェイスを作成することをユーザーに許可します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterface",
                "ec2:DescribeDhcpOptions",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DeleteNetworkInterface",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeVpcs"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateNetworkInterfacePermission"
            ],
            "Resource": "arn:aws:ec2:{{region}}:{{account-id}}:network-interface/*",
            "Condition": {
                "StringEquals": {
                    "ec2:Subnet": [
                        "arn:aws:ec2:{{region}}:{{account-id}}:subnet/[[subnets]]"
                    ],
                    "ec2:AuthorizedService": "codebuild.amazonaws.com"
```

```
                }
            }
        }
    ]
}
```

AWS CodeBuild の権限リファレンス

アクセスコントロール (p. 194) をセットアップし、IAM アイデンティティ (アイデンティティベースのポリシー) にアタッチできるアクセス許可ポリシーを作成するときは、以下の表をリファレンスとして使用できます。

AWS CodeBuild ポリシーで AWS 全体の条件キーを使用して、条件を表現することができます。リストについては、IAM ユーザーガイドの「利用可能なキー」を参照してください。

アクションは、ポリシーの `Action` フィールドで指定します。アクションを指定するには、API オペレーション名 (たとえば、`codebuild:CreateProject` や `codebuild:StartBuild`) の前に `codebuild:` プレフィックスを使用します。単一のステートメントに複数のアクションを指定するには、コンマで区切ります (たとえば、`"Action": [ "codebuild:CreateProject", "codebuild:StartBuild" ]`)。

ワイルドカード文字の使用

ポリシーの `Resource` フィールドでリソース値として、ワイルドカード文字 (*) を使用して、または使用せずに ARN を指定します。ワイルドカードを使用して複数のアクションまたはリソースを指定することができます。たとえば、`codebuild:*` は、すべての AWS CodeBuild アクションを指定し、`codebuild:Batch*` は、`Batch` という単語で始まるすべての AWS CodeBuild アクションを指定します。次の例では、`my` で始まる名前のすべてのビルドプロジェクトへのアクセスを許可します。

```
arn:aws:codebuild:us-east-2:123456789012:project/my*
```

AWS CodeBuild API オペレーションおよびアクションで必要なアクセス許可

`BatchDeleteBuilds`

アクション: `codebuild:BatchDeleteBuilds`

ビルドを削除するのに必要です。

リソース: `arn:aws:codebuild:`*`region-ID`*`:`*`account-ID`*`:project/`*`project-name`*

`BatchGetBuilds`

アクション: `codebuild:BatchGetBuilds`

ビルドに関する情報を取得するのに必要です。

リソース: `arn:aws:codebuild:`*`region-ID`*`:`*`account-ID`*`:project/`*`project-name`*

`BatchGetProjects`

アクション: `codebuild:BatchGetProjects`

ビルドプロジェクトに関する情報を取得するのに必要です。

リソース: `arn:aws:codebuild:`*`region-ID`*`:`*`account-ID`*`:project/`*`project-name`*

`CreateProject`

アクション: `codebuild:CreateProject`, `iam:PassRole`

ビルドプロジェクトを作成するのに必要です。

リソース: arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*, arn:aws:iam:*account-ID*:role/*role-name*

DeleteProject

アクション: codebuild:DeleteProject

ビルドプロジェクトを削除するのに必要です。

リソース: arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

ListBuilds

アクション: codebuild:ListBuilds

ビルド ID のリストを取得するのに必要です。

リソース: *

ListBuildsForProject

アクション: codebuild:ListBuildsForProject

ビルドプロジェクトのビルド ID のリストを取得するために必要です。

リソース: arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

ListCuratedEnvironmentImages

アクション: codebuild:ListCuratedEnvironmentImages

AWS CodeBuild によって管理されるすべての Docker イメージに関する情報を取得するのに必要です。

リソース: * (必須ですが、アドレスで呼び出せる AWS リソースを参照していません)

ListProjects

アクション: codebuild:ListProjects

ビルドプロジェクト名のリストを取得するのに必要です。

リソース: *

StartBuild

アクション: codebuild:StartBuild

ビルドの実行を開始するために必要です。

リソース: arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

StopBuild

アクション: codebuild:StopBuild

実行中のビルドを停止しようとするのに必要です。

リソース: arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

UpdateProject

アクション: codebuild:UpdateProject, iam:PassRole

ビルドに関する情報を変更するのに必要です。

リソース: arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*, arn:aws:iam:*account-ID*:role/*role-name*

AWS CloudTrail を使用した AWS CodeBuild API 呼び出しのログ作成

AWS CodeBuild は CloudTrail と統合されています。これは、AWS アカウントで AWS CodeBuild に代わって行われた API 呼び出しをキャプチャし、指定した Amazon S3 バケットにログファイルを送信するサービスです。CloudTrail は、AWS CodeBuild コンソール、AWS CLI、および AWS SDK からの API 呼び出しをキャプチャします。CloudTrail によって収集された情報を使用して、AWS CodeBuild に対してどのようなリクエストが行われたかを判断することができます。リクエストの作成元のソース IP アドレス、リクエストの実行者、リクエストの実行日時などです。CloudTrail の詳細 (設定する方法や有効にする方法など) については、AWS CloudTrail User Guide を参照してください。

CloudTrail 内の AWS CodeBuild 情報

AWS アカウントで CloudTrail のログ記録を有効にすると、AWS CodeBuild アクションに対する呼び出しがログファイルに記録されます。AWS CodeBuild レコードは、他の AWS サービスレコードと一緒にログファイルに記録されます。CloudTrail は、期間とファイルサイズに基づいて、新しいファイルをいつ作成して書き込むかを決定します。

ログには、すべての AWS CodeBuild アクションが記録されます。これらのアクションについては、コマンドラインリファレンス (p. 191) を参照してください。たとえば、ビルドプロジェクトを作成してビルドを実行する呼び出しは、CloudTrail ログファイルにエントリを生成します。

各ログエントリには、誰がリクエストを生成したかに関する情報が含まれます。ログのユーザー ID 情報により、リクエストが、ルートまたは IAM ユーザーの認証情報を使用して送信されたか、ロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して送信されたか、あるいは別の AWS のサービスによって送信されたかを確認できます。詳細については、`userIdentity`CloudTrail Event Reference の フィールドを参照してください。

必要な場合はログファイルを自身の バケットに保管できますが、ログファイルを自動的にアーカイブまたは削除するように Amazon S3 ライフサイクルルールを定義することもできます。デフォルトでは Amazon S3 のサーバー側の暗号化 (SSE) を使用して、ログファイルが暗号化されます。

新しいログファイルが配信されるときに、CloudTrail が Amazon SNS 通知を発行するようにできます。詳細については、CloudTrail 用の Amazon SNS 通知の構成を参照してください。

また、複数の AWS リージョンと複数の AWS アカウントからの AWS CodeBuild ログファイルを 1 つの Amazon S3 バケットに集約することもできます。詳細については、「Receiving CloudTrail Log Files From Multiple Regions」を参照してください。

AWS CodeBuild ログファイルエントリの概要

CloudTrail ログファイルには、複数の JSON 形式イベントで構成される 1 つ以上のログエントリが記録されます。ログエントリは任意の送信元からの単一のリクエストを表し、リクエストされたアクション、パラメータ、アクションの日時などに関する情報を含みます。ログエントリは、特定の順序になるように生成されるわけではありません。つまり、パブリック 呼び出しの順序付けられたスタックトレースではありません。

次の例は、AWS CodeBuild でビルドプロジェクトを作成する方法を示す CloudTrail ログエントリを示しています。

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "FederatedUser",
    "principalId": "account-ID:user-name",
```

```
    "arn": "arn:aws:sts::account-ID:federated-user/user-name",
    "accountId": "account-ID",
    "accessKeyId": "access-key-ID",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2016-09-06T17:59:10Z"
      },
      "sessionIssuer": {
        "type": "IAMUser",
        "principalId": "access-key-ID",
        "arn": "arn:aws:iam::account-ID:user/user-name",
        "accountId": "account-ID",
        "userName": "user-name"
      }
    }
  },
  "eventTime": "2016-09-06T17:59:11Z",
  "eventSource": "codebuild.amazonaws.com",
  "eventName": "CreateProject",
  "awsRegion": "region-ID",
  "sourceIPAddress": "127.0.0.1",
  "userAgent": "user-agent",
  "requestParameters": {
    "awsActId": "account-ID"
  },
  "responseElements": {
    "project": {
      "environment": {
        "image": "image-ID",
        "computeType": "BUILD_GENERAL1_SMALL",
        "type": "LINUX_CONTAINER",
        "environmentVariables": []
      },
      "name": "codebuild-demo-project",
      "description": "This is my demo project",
      "arn": "arn:aws:codebuild:region-ID:account-ID:project/codebuild-demo-
project:project-ID",
      "encryptionKey": "arn:aws:kms:region-ID:key-ID",
      "timeoutInMinutes": 10,
      "artifacts": {
        "location": "arn:aws:s3:::codebuild-region-ID-account-ID-output-bucket",
        "type": "S3",
        "packaging": "ZIP",
        "outputName": "MyOutputArtifact.zip"
      },
      "serviceRole": "arn:aws:iam::account-ID:role/CodeBuildServiceRole",
      "lastModified": "Sep 6, 2016 10:59:11 AM",
      "source": {
        "type": "GITHUB",
        "location": "https://github.com/my-repo.git"
      },
      "created": "Sep 6, 2016 10:59:11 AM"
    }
  },
  "requestID": "9d32b228-745b-11e6-98bb-23b67EXAMPLE",
  "eventID": "581f7dd1-8d2e-40b0-aeee-0dbf7EXAMPLE",
  "eventType": "AwsApiCall",
  "recipientAccountId": "account-ID"
}
```

AWS CodeBuild のトラブルシューティング

このトピックの情報を使用して、問題を特定、診断、対処します。

トピック

エラー : ビルドプロジェクトを作成または更新するときに「CodeBuild は、次の実行を許可されていません。sts:AssumeRole」

問題: プロジェクトのビルドを作成または更新しようとすると、以下のようなエラーが表示されます。「Code:InvalidInputException、Message:CodeBuild は、次の実行を許可されていません。sts:AssumeRole on arn:aws:iam::*account-ID*:role/*service-role-name*」

考えられる原因:

- ビルドプロジェクトを作成または更新しようとしている AWS リージョン向けの AWS Security Token Service (AWS STS) が非アクティブ化されました。
- ビルドプロジェクトに関連付けられた AWS CodeBuild サービスロールは存在しないか、AWS CodeBuild を信頼するための十分なアクセス権限がありません。

推奨される解決策:

- ビルドプロジェクトを作成または更新しようとしている AWS リージョン向けの AWS STS がアクティブ化されていることを確認します。詳細については、IAM ユーザーガイド で「AWS リージョンでの AWS STS のアクティブ化と非アクティブ化」を参照してください。
- 対象 AWS CodeBuild サービスロールが AWS アカウント内に存在することを確認します。コンソールを使用していない場合は、ビルドプロジェクトを作成または更新したときにサービスロールの Amazon リソースネーム (ARN) のスペルを間違えていないことを確認してください。
- 対象の AWS CodeBuild サービスロールに、AWS CodeBuild を信頼するための十分な権限があることを確認します。詳細については、『AWS CodeBuild サービスロールの作成 (p. 185)』の「信頼関係のポリシーステートメント」を参照してください。

エラー: ビルドの実行時に「アクセスしようとしているバケットは、指定されたエンドポイントを使用してアドレス指定する必要があります...」

問題: ビルドを実行すると、`DOWNLOAD_SOURCE` ビルドフェーズは失敗し、「アクセスしようとしているバケットは、指定されたエンドポイントを使用してアドレス指定する必要があります。今後のリクエストはこのエンドポイントに送信してください」というエラーが発生します。

考えられる原因: 事前に構築されたソースコードは Amazon S3 バケットに格納されており、そのバケットは AWS CodeBuild ビルドプロジェクトとは異なる AWS リージョンにあります。

推奨される解決策: ビルドプロジェクトの設定を、事前に構築されたソースコードを含むバケットを指し、そのバケットがビルドプロジェクトと同じリージョンにあるように更新します。

エラー: ビルドの実行時に「アーティファクトのアップロードに失敗しました: 無効な ARN」

問題: ビルドを実行すると、`UPLOAD_ARTIFACTS` ビルドフェーズが失敗し、「アーティファクトのアップロードに失敗しました : 無効な ARN」というエラーが発生します。

考えられる原因: Amazon S3 出力バケット (AWS CodeBuild がビルドからの出力を格納するバケット) が、AWS CodeBuild ビルドプロジェクトとは異なる AWS リージョンにあります。

推奨される解決策: ビルドプロジェクトの設定を、ビルドプロジェクトと同じ地域にある出力バケットを指すように更新します。

エラー:「認証情報を見つけることができません」

問題: AWS CLI を実行しようとするとき、AWS SDK を使用したり、別の同様のコンポーネントをビルドの一部として呼び出すと、AWS CLI、AWS SDK、またはコンポーネントに直接関連するビルドエラーが発生します。たとえば、「認証情報を見つけることができません」などのビルドエラーが発生することがあります。

考えられる原因:

- ビルド環境の AWS CLI、AWS SDK またはコンポーネントのバージョンが AWS CodeBuild と互換性がありません。
- Docker を使用するビルド環境内で Docker コンテナを実行しており、Docker コンテナは必要な AWS 認証情報にデフォルトでアクセスできません。

推奨される解決策:

- ビルド環境に以下のバージョン以降の AWS CLI、AWS SDK、またはコンポーネントがインストールされていることを確認してください。
  - AWS CLI: 1.10.47
  - AWS SDK for C++: 0.2.19
  - AWS SDK for Go: 1.2.5
  - AWS SDK for Java: 1.11.16
  - AWS SDK for JavaScript: 2.4.7
  - AWS SDK for PHP: 3.18.28
  - AWS SDK for Python (Boto3): 1.4.0
  - AWS SDK for Ruby: 2.3.22
  - Botocore: 1.4.37
  - CoreCLR: 3.2.6-beta
  - Node.js: 2.4.7
- ビルド環境で Docker コンテナを実行する必要があり、コンテナが AWS 認証情報を必要とする場合は、ビルド環境からコンテナに資格情報を渡す必要があります。ビルド仕様に、次のような Docker `run` コマンドを含めます。この例では、`aws s3 ls` コマンドを使用して使用可能な Amazon S3 バケットを一覧表示しています。`-e` オプションは、コンテナが AWS 認証情報にアクセスするために必要な環境変数を渡します。

```
docker run -e AWS_DEFAULT_REGION -e AWS_CONTAINER_CREDENTIALS_RELATIVE_URI your-image-tag
 aws s3 ls
```

- Docker イメージを作成していて、ビルドに AWS 認証情報が必要な場合 (たとえば、Amazon S3 からファイルをダウンロードする場合)、ビルド環境から Docker ビルドプロセスに認証情報を次のようにパススルーする必要があります。
  1. Docker イメージ用ソースコードの Dockerfile に、次の `ARG` インストラクションを指定します。

```
ARG AWS_DEFAULT_REGION
ARG AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

  2. ビルドスペックでは、以下のような Docker `build` コマンドを組み込みます。`--build-arg` オプションは、Docker ビルドプロセスが AWS 認証情報にアクセスするために必要な環境変数を設定します。

```
docker build --build-arg AWS_DEFAULT_REGION=$AWS_DEFAULT_REGION --build-arg
 AWS_CONTAINER_CREDENTIALS_RELATIVE_URI=$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI -
t your-image-tag .
```

ビルド仕様の以前のコマンドが、それ以降のコマンドで認識されない

問題: ビルド仕様の 1 つ以上のコマンドの結果が、同じビルド仕様の後のコマンドで認識されません。たとえば、コマンドによってローカル環境変数が設定される場合がありますが、後で実行されるコマンドがその環境変数の値を取得できない可能性があります。

考えられる原因: ビルドスペックバージョン 0.1 では、AWS CodeBuild はビルド環境のデフォルトシェルの各インスタンスで各コマンドを実行します。つまり、各コマンドは他のすべてのコマンドとは独立して実行されます。デフォルトでは、以前のコマンドの状態に依存する単一のコマンドを実行することはできません。

推奨される解決策: ビルドスペックバージョン 0.2 を使用することをお勧めします。これにより、問題が解決されます。何らかの理由でビルドスペックバージョン 0.1 を使用する必要がある場合は、複数のコマンドを 1 つのコマンドにまとめるためにシェルコマンド連鎖演算子 (Linux の `&&` など) を使用することをお勧めします。または、複数のコマンドを含むソースコードにシェルスクリプトを組み込み、そのシェルスクリプトをビルド仕様の 1 つのコマンドから呼び出します。詳細については、「ビルド環境でのシェルとコマンド (p. 111)」および「ビルド環境の環境変数 (p. 112)」を参照してください。

Apache Maven が間違ったリポジトリからリファレンスアーティファクトを構築する

問題: AWS CodeBuild 提供の Java ビルド環境で Maven を使用すると、Maven はビルドおよびプラグインの依存関係を https://repo1.maven.org/maven2 で安全な中央 Maven リポジトリから取得します。 これは、ビルドプロジェクトの `pom.xml` ファイルが別の場所を使用すると明示的に宣言した場合でも発生します。

考えられる原因: AWS CodeBuild 提供の Java ビルド環境には、ビルド環境の `/root/.m2` ディレクトリに事前にインストールされている `settings.xml` という名前のファイルが含まれています。この `settings.xml` には次の宣言が含まれています。これにより、Maven が常に https://repo1.maven.org/maven2 で、セキュアな中央 Maven リポジトリからビルドおよびプラグインの依存関係を引き出すように指示されます。

```
<settings>
  <activeProfiles>
    <activeProfile>securecentral</activeProfile>
  </activeProfiles>
  <profiles>
    <profile>
      <id>securecentral</id>
      <repositories>
        <repository>
          <id>central</id>
          <url>https://repo1.maven.org/maven2</url>
          <releases>
            <enabled>true</enabled>
          </releases>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>https://repo1.maven.org/maven2</url>
          <releases>
            <enabled>true</enabled>
          </releases>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
</settings>
```

推奨される解決策: 以下を実行してください。

1. `settings.xml` ファイルをソースコードに追加します。
2. この `settings.xml` ファイルでは、前述の `settings.xml` 形式をガイドとして使用して、Maven が代わりにビルドとプラグインの依存関係を取得するリポジトリを宣言します。

3. ビルドプロジェクトの `install` フェーズで、`settings.xml` ファイルをビルド環境の `/root/.m2` ディレクトリにコピーするよう AWS CodeBuild に指示します。たとえば、この動作を示す `buildspec.yml` ファイルの次のスニペットを考えてみましょう。

```
version 0.2

phases:
  install:
    commands:
      - cp ./settings.xml /root/.m2/settings.xml
```

ビルドコマンドがデフォルトで root として実行される

問題: AWS CodeBuild は、ビルドコマンドを root ユーザーとして実行します。これは、関連するビルドイメージの Dockerfile によって `USER` インストラクションが別のユーザーに設定された場合でも発生します。

原因: AWS CodeBuild は、デフォルトですべてのビルドコマンドを root ユーザーとして実行します。

推奨される解決策: なし。

Bourne シェル (sh) がビルドイメージに存在する必要がある

問題: AWS CodeBuild で提供されていないビルドイメージを使用し、「ビルドを完了する前にビルドコンテナが終了しました」というメッセージが表示されてビルドに失敗します。

考えられる原因: Bourne シェル (`sh`) がビルドイメージに含まれていません。ビルドコマンドとスクリプトを実行するために、AWS CodeBuild は `sh` を必要とします。

推奨される解決策: ビルドイメージに `sh` が含まれていない場合、イメージを使用するビルドを開始する前に必ずそれを含めてください。(AWS CodeBuild は、ビルドイメージにすでに `sh` を含んでいます。)

エラー: ビルドを実行中に「AWS CodeBuild に問題が発生しています」

問題: ビルドプロジェクトを実行しようとすると、ビルドの `PROVISIONING` フェーズで中に次のエラーが表示されます:「AWS CodeBuild に問題が発生しています。」

考えられる原因: AWS CodeBuild には大きすぎる環境変数がビルドで使用されています。AWS CodeBuild では、すべての環境変数の長さ (すべての名前と値を一緒にしたもの) が最大約 5,500 文字に達すると エラーが発生します。

推奨される解決策: Amazon EC2 Systems Manager パラメータストアを使用して大きな環境変数を保存し、ビルドスペックでそれらを取得します。Amazon EC2 Systems Manager のパラメータストアには、4,096 文字以下の組み合わせの個々の環境変数 (名前と値を一緒にしたもの) を保存できます。大きな環境変数を保存するには、Amazon EC2 Systems Manager ユーザーガイドの「Systems Manager パラメータストア」および「Systems Manager パラメータストアコンソールチュートリアル」を参照してください。それらを取得するには、「ビルドスペックの構文 (p. 96)」の `parameter-store` マッピングを参照してください。

エラー: 「BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE」 カスタムビルドイメージを使用した場合

問題: カスタムビルドイメージを使用するビルドを実行しようとすると、ビルドは `BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE` というエラーで失敗します。

考えられる原因:

- ビルドイメージの全体的な非圧縮サイズが、ビルド環境のコンピューティングタイプの使用可能ディスクスペースよりも大きい。ビルドイメージのサイズを確認するには、Docker を使用して `docker images` *`REPOSITORY`*`:`*`TAG`* コマンドを実行します。コンピューティングタイプで使用可能なディスク容量のリストについては、「ビルド環境コンピューティングタイプ (p. 110)」を参照してください。
- AWS CodeBuild には、Amazon Elastic Container Registry (Amazon ECR) からビルドイメージを取得する権限がありません。

推奨される解決策:

- 使用可能なディスク容量の大きなコンピューティングタイプを使用するか、カスタムビルドイメージのサイズを縮小します。
- AWS CodeBuild がカスタムビルドイメージをビルド環境にプルできるように、Amazon ECR のリポジトリの権限を更新します。詳細については、「Amazon ECR サンプル (p. 22)」を参照してください。

ファイル名に英語以外の文字が含まれているとビルドが失敗する

問題: 英語以外の文字 (中国語の文字など) を含むファイル名のファイルを使用するビルドを実行すると、ビルドが失敗します。

考えられる原因: AWS CodeBuild によって提供されるビルド環境は、デフォルトのロケールが `POSIX` に設定されています。`POSIX` のローカリゼーション設定は、米国英語以外の文字を含む AWS CodeBuild やファイル名との互換性が低く、関連するビルドが失敗する可能性があります。

推奨される解決策: ビルド仕様の `pre_build` セクションに次のコマンドを追加します。これらのコマンドは、ビルド環境のローカライゼーション設定として米国英語 UTF-8 を使用するよう設定します。これは、米国英語以外の文字を含む AWS CodeBuild やファイル名と高い互換性があります。

Ubuntu ベースのビルド環境の場合:

```
pre_build:
  commands:
    - export LC_ALL="en_US.UTF-8"
    - locale-gen en_US en_US.UTF-8
    - dpkg-reconfigure locales
```

Amazon Linux ベースのビルド環境の場合:

```
pre_build:
  commands:
    - export LC_ALL="en_US.utf8"
```

Amazon EC2 パラメータストアからパラメータを取得する場合にビルドが失敗する

問題: ビルドが Amazon EC2 パラメータストアに保存されている 1 つ以上のパラメータの値を取得しようとすると、DOWNLOAD_SOURCE フェーズで「パラメータが存在しません」というエラーが発生し、ビルドが失敗します。

考えられる原因: ビルドプロジェクトが依存するサービスロールに ssm:GetParameters アクションを呼び出す権限がありません。または、ビルドプロジェクトは AWS CodeBuild によって生成された ssm:GetParameters アクションを呼び出すことができるサービスロールを使用していますが、パラメータには /CodeBuild/ で始まらない名前が付けられています。

推奨される解決策:

- AWS CodeBuild によって生成されていないサービスロールの場合は、その定義を更新して AWS CodeBuild が ssm:GetParameters アクションを呼びだせるようにします。たとえば、次のポリシーステートメントでは、ssm:GetParameters アクションを呼び出して /CodeBuild/ で始まる名前を持つパラメータを取得できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "ssm:GetParameters",
      "Effect": "Allow",
      "Resource": "arn:aws:ssm:REGION_ID:ACCOUNT_ID:parameter/CodeBuild/*"
    }
  ]
}
```

- AWS CodeBuild によって生成されたサービスロールの場合は、その定義を更新して、Amazon EC2 パラメータストア内の /CodeBuild/ で始まる名前以外の名前のパラメータに AWS CodeBuild がアクセスできるようにしますたとえば、次のポリシーステートメントでは、ssm:GetParameters アクションを呼び出して指定された名前のパラメータを取得できます。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "ssm:GetParameters",
      "Effect": "Allow",
      "Resource": "arn:aws:ssm:REGION_ID:ACCOUNT_ID:parameter/PARAMETER_NAME"
    }
  ]
}
```

キャッシュをダウンロードしようとすると、「アクセスが拒否されました」というエラーメッセージが表示される

問題: キャッシュが有効なビルドプロジェクトでキャッシュをダウンロードしようとすると、次の一般的なエラーが発生します。「アクセスが拒否されました」。

考えられる原因:

- ビルドプロジェクトの一部としてキャッシングが設定されています。
- キャッシュは、`InvalidateProjectCache` API を介して最近無効化されています。
- CodeBuild によって使用されているサービスロールには、キャッシュを保持している Amazon S3 バケットへの `s3:GetObject` および `s3:PutObject` アクセス権限がありません。

推奨される解決策: キャッシュ設定を更新した直後に初めて使用する場合、このエラーが表示されるのは普通です。このエラーが解消されない場合は、キャッシュを保持している Amazon S3 バケットへの`s3:GetObject` および `s3:PutObject` アクセス権限をサービスロールが持っているかどうかを確認する必要があります。詳細については、「S3 アクセス権限の指定」を参照してください。

エラー：「S3 から証明書をダウンロードできません。AccessDenied"

問題: ビルドプロジェクトを実行しようとすると、ビルドが失敗して次のエラーが表示されます「S3 から証明書をダウンロードできません。AccessDenied".

考えられる原因:

- 証明書に正しくない S3 バケットが選択されています。
- 証明書に誤ったオブジェクトキーが入力されています。

推奨される解決策:

- プロジェクトを編集します。[証明書のバケット] では、SSL 証明書が保存されている S3 バケットを選択します。
- プロジェクトを編集します。[証明書のオブジェクトキー] に S3 オブジェクトキーの名前を入力します。

エラー:「Git のクローンに失敗しました: 'your-repository-URL' にアクセスできません: SSL 証明書の問題: 自己署名証明書」

問題: ビルドプロジェクトを実行しようとすると、ビルドが失敗し、「Git のクローン失敗: `'your-repository-URL'` にアクセスできません: SSL 証明書の問題: 自己署名証明書」エラーが表示されます。

考えられる原因:

- ソースリポジトリには自己署名証明書がありますが、S3 バケットから証明書をビルドプロジェクトの一部としてインストールする選択をしていません。

推奨される解決策:

- プロジェクトを編集します。[証明書] で [S3 から証明書をインストールする] を選択します。[証明書のバケット] では、SSL 証明書が保存されている S3 バケットを選択します。[証明書のオブジェクトキー] に S3 オブジェクトキーの名前を入力します。
- プロジェクトを編集します。GitHub Enterprise プロジェクトリポジトリに接続するときに SSL 警告を無視するには、[Insecure SSL] を選択します。

Note

[Insecure SSL] はテストのみに使用することが推奨されます。本番環境では使用しないでください。

AWS CodeBuild の制限

次の表に、AWS CodeBuild の現在の制限を示します。これらの制限は、特に指定のない限り、AWS アカウントごとにサポートされている AWS リージョンごとです。

ビルドプロジェクト

リソース	デフォルトの制限
ビルドするプロジェクトの最大数	1,000
ビルドプロジェクト名の長さ	2~255 文字以内
ビルドプロジェクト名に使用できる文字	文字 `A-Z` および `a-z`、数字 `0-9`、特殊文字 `-` および `_`
ビルドプロジェクトの説明の最大長	255 文字
ビルドプロジェクトの説明に使用できる文字	すべて
AWS CLI または AWS SDK を使用して、同時に関連情報をリクエストできるビルドプロジェクトの最大数	100
ビルドプロジェクトに関連付けることができるタグの最大数	50
すべての関連ビルドのビルドタイムアウトのためにビルドプロジェクトで指定できる時間 (分)	5~480 (8 時間)
VPC 設定で追加できるサブネットの数	1〜16
VPC 設定で追加できるセキュリティグループの数	1〜5

ビルド数

リソース	デフォルトの制限
同時実行のビルドの最大数 *	20
AWS CLI または AWS SDK を使用して、同時に関連情報をリクエストできるビルドの最大数	100
1 つのビルドのビルドタイムアウトのために指定できる時間 (分)	5~480 (8 時間)

* 同時実行ビルドの最大数の制限は、コンピューティングタイプによって異なります。一部のコンピューティングタイプでは、デフォルトは 20 です。同時ビルドの制限の引き上げをリクエストするには、または「アカウントのアクティブなビルドは X 以上持つことはできません」というエラーが発生した場合は、AWS サポートに連絡してください。

AWS CodeBuild ユーザーガイドドキュメントの履歴

AWS CodeBuild ユーザーガイドの重要な変更点の一覧を示します。

- 最新の API バージョン: 2016-10-06
- 最新のドキュメンテーションの更新: 2018 年 1 月 25 日

変更	説明	変更日
GitHub Enterprise サポート	AWS CodeBuild は、GitHub Enterprise リポジトリに保存されたソースコードから構築できるようになりました。詳細については、「GitHub Enterprise のサンプル (p. 30)」を参照してください。	2018 年 1 月 25 日
Git クローンの深さサポート	AWS CodeBuild は、指定されるコミット数で切り捨てられる履歴の浅いクローンの作成をサポートするようになりました。詳細については、「ビルドプロジェクトを作成する (p. 146)」を参照してください。	2018 年 1 月 25 日
VPC サポート	VPC 対応のビルドが VPC 内のリソースにアクセスできるようになりました。詳細については、「VPC サポート (p. 115)」を参照してください。	2017 年 11 月 27 日
依存関係のキャッシュのサポート	AWS CodeBuild で依存関係のキャッシュがサポートされるようになりました。これにより、AWS CodeBuild は、ビルド環境の特定の再利用可能部分をキャッシュに保存し、これを複数のビルドにわたって使用できます。	2017 年 11 月 27 日
ビルドバッジのサポート	AWS CodeBuild でビルドバッジが使用可能になりました。ビルドバッジは、埋め込み可能なイメージ (バッジ) として動的に生成され、プロジェクトの最新ビルドのステータスを示します。詳細については、「ビルドバッジサンプル (p. 39)」を参照してください。	2017 年 11 月 27 日

変更	説明	変更日
AWS Config の統合	AWS Config で AWS CodeBuild が AWS リソースとしてサポートされるようになりました。これにより、サービスは AWS CodeBuild のプロジェクトを追跡できます。AWS Config の詳細については、「AWS CodeBuild サンプルを使用した AWS Config (p. 38)」を参照してください。	2017 年 10 月 20 日
GitHub リポジトリで更新されたソースコードの自動的な再構築	ソースコードを GitHub リポジトリに保存している場合は、コード変更がリポジトリにプッシュされるたびに AWS CodeBuild でソースコードを再構築できます。詳細については、「GitHub プルリクエストサンプル (p. 36)」を参照してください。	2017 年 9 月 21 日
Amazon EC2 Systems Manager パラメータストアでの新しい方法による重要または大規模な環境変数の保存と取得	AWS CodeBuild コンソールまたは AWS CLI を使用して Amazon EC2 Systems Manager パラメータストアに保存された重要または大規模な環境変数を取得できるようになりました。また、AWS CodeBuild コンソールを使用し、これらの種類の環境変数を Amazon EC2 Systems Manager パラメータストアに保存することもできます。これまでは、これらの種類の環境変数を取得するには、これらの変数をビルドスペックに含めるか、ビルドコマンドを実行して AWS CLI を自動化する以外にありませんでした。また、これらの種類の環境変数を保存するには、Amazon EC2 Systems Manager パラメータストアコンソールを使用する以外にありませんでした。詳細については、「ビルドプロジェクトを作成する (p. 146)」、「ビルドプロジェクトの設定を変更する (p. 161)」、および「ビルドの実行 (p. 167)」を参照してください。	2017 年 9 月 14 日
ビルドの削除のサポート	AWS CodeBuild でビルドを削除できるようになりました。詳細については、「ビルドの削除 (p. 180)」を参照してください。	2017 年 8 月 31 日

変更	説明	変更日
Amazon EC2 Systems Manager パラメータストアに保存された重要または大規模な環境変数をビルドスペックを使用して取得する新しい方法	AWS CodeBuild ではビルドスペックを使用し、Amazon EC2 Systems Manager パラメータストアに保存された重要または大規模な環境変数を簡単に取得できるようになりました。これまでは、これらの種類の環境変数を取得するには、ビルドコマンドを実行して AWS CLI を自動化する以外にありませんでした。詳細については、「ビルドスペックの構文 (p. 96)」の `parameter-store` マッピングを参照してください。	2017 年 8 月 10 日
AWS CodeBuild での Bitbucket のサポート	AWS CodeBuild は、Bitbucket リポジトリに保存されたソースコードから構築できるようになりました。詳細については、「ビルドプロジェクトを作成する (p. 146)」および「ビルドの実行 (p. 167)」を参照してください。	2017 年 8 月 10 日
AWS CodeBuild が 米国西部 (北カリフォルニア)、region-eu-west-2;、および カナダ (中部) で使用可能に	AWS CodeBuild は、米国西部 (北カリフォルニア)、region-eu-west-2;、および カナダ (中部) の各リージョンで使用可能になりました。詳細については、アマゾン ウェブ サービス全般のリファレンスの「AWS リージョンとエンドポイント」トピックの「AWS CodeBuild」セクションを参照してください。	2017 年 6 月 29 日
Linux における .NET Core のサンプル	AWS CodeBuild と .NET Core を使用して、C# で記述されたコードから実行可能ファイルをビルドする方法を示すサンプルが追加されました。詳細については、「Linux における .NET Core のサンプル (p. 91)」を参照してください。	2017 年 6 月 29 日
ビルドスペックファイルの代替の名前および場所のサポート	ビルドプロジェクトで使用するビルドスペックファイル名として、ソースコードのルートにあるデフォルトの名前 (`buildspec.yml`) の代わりに、別の名前や場所を指定できるようになりました。詳細については、「ビルドスペックのファイル名とストレージの場所 (p. 95)」を参照してください。	2017 年 6 月 27 日

変更	説明	変更日
更新されたビルド通知のサンプル	AWS CodeBuild は、Amazon CloudWatch Events および Amazon Simple Notification Service (Amazon SNS) を通じてビルド通知に対する組み込みサポートを提供するようになりました。この新しい動作を反映するために、従来の ビルド通知サンプル (p. 42) が更新されています。	2017 年 6 月 22 日
カスタムイメージの Docker のサンプルを追加	AWS CodeBuild およびカスタム Docker ビルドイメージを使用して Docker イメージをビルドして実行する方法を示すサンプルを追加しました。詳細については、「カスタム Docker イメージのサンプル (p. 59)」を参照してください。	2017 年 6 月 7 日
GitHub のプル要求に応じたソースコードの取得	GitHub リポジトリに保存されたソースコードに依存するビルドを AWS CodeBuild で実行するときに、ビルドに対する GitHub プル要求 ID を指定できるようになりました。代わりに、コミット ID、ブランチ名、またはタグ名を指定することもできます。詳細については、「ビルドの実行 (コンソール) (p. 167)」の Source version 値または「ビルドの実行 (AWS CLI) (p. 169)」の `sourceVersion` 値を参照してください。	2017 年 6 月 6 日
ビルド仕様バージョンの更新	新しいバージョンのビルド仕様形式がリリースされました。バージョン 0.2 では、AWS CodeBuild でデフォルトシェルのインスタンス別に各ビルドコマンドを実行する場合の課題に対処しています。また、バージョン 0.2 では `environment_variables` の名前が `env` に変更され、`plaintext` の名前が `variables` 変更されています。詳細については、「AWS CodeBuild のビルド仕様に関するリファレンス (p. 95)」を参照してください。	2017 年 5 月 9 日

変更	説明	変更日
GitHub で使用可能なビルドイメージの Dockerfiles	AWS CodeBuild に用意されているビルドイメージの多くの定義は、GitHub で Dockerfiles として使用できます。詳細については、「AWS CodeBuild に用意されている Docker イメージ (p. 103)」にある表の「定義」列を参照してください。	2017 年 5 月 2 日
欧州 (フランクフルト)、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)、and アジアパシフィック (東京) で利用可能な AWS CodeBuild	AWS CodeBuild は 欧州 (フランクフルト)、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)、and アジアパシフィック (東京) regions で利用可能になりました。詳細については、アマゾン ウェブ サービス全般のリファレンスの「AWS リージョンとエンドポイント」トピックの 「AWS CodeBuild」セクションを参照してください。	2017 年 3 月 21 日
AWS CodeBuild の AWS CodePipeline テストアクションのサポート	AWS CodePipeline のパイプラインに AWS CodeBuild を使用するテストアクションを追加できるようになりました。詳細については、「AWS CodeBuild テストアクションをパイプラインに追加する (AWS CodePipeline コンソール) (p. 140)」を参照してください。	2017 年 3 月 8 日
ビルドスペックは、選択した最上位ディレクトリからのビルド出力の取得をサポートします。	ビルドスペックでは、コンテンツをビルド出力アーティファクトに含めるように AWS CodeBuild に指示できる個々の最上位ディレクトリを指定できるようになりました。これは、`base-directory` マッピングを使用して行います。詳細については、「ビルドスペックの構文 (p. 96)」を参照してください。	2017 年 2 月 8 日
組み込み環境変数	AWS CodeBuild には、ビルドで使用するための追加の組み込みの環境変数が用意されています。 これらには、ビルドを開始したエンティティを記述する環境変数、ソースコードリポジトリへの URL、ソースコードのバージョン ID などが含まれます。詳細については、「ビルド環境の環境変数 (p. 112)」を参照してください。	2017 年 1 月 30 日

変更	説明	変更日
AWS CodeBuild は 米国東部 (オハイオ) で利用可能	AWS CodeBuild 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。詳細については、アマゾン ウェブ サービス全般のリファレンスの「AWS リージョンとエンドポイント」トピックの 「AWS CodeBuild」セクションを参照してください。	2017 年 1 月 19 日
AWS Lambda サンプル	Lambda、AWS CloudFormation、および AWS CodePipeline とともに AWS CodeBuild を使用して、AWS Serverless Application Model (AWS SAM) 標準に準拠したサーバーレスアプリケーションをビルドおよびデプロイする方法を示すサンプルにリファレンスが追加されました。詳細については、「AWS Lambda サンプル (p. 64)」を参照してください。	2016 年 20 月 12 日
C++ と Go のサンプル	AWS CodeBuild を使用して C++ および Go 出力アーティファクトをビルドする方法を示すサンプルが追加されました。詳細については、「C++ サンプル (p. 73)」および「Go サンプル (p. 75)」を参照してください。	2016 年 12 月 9 日
シェルおよびコマンドの動作情報	AWS CodeBuild は、ビルド環境のデフォルトのシェルの個別のインスタンスで指定した各コマンドを実行します。 このデフォルトの動作によって、コマンドに予期しない悪影響が生じることがあります。必要に応じて、このデフォルトの動作を回避するいくつかの方法をお勧めします。詳細については、「ビルド環境でのシェルとコマンド (p. 111)」を参照してください。	2016 年 12 月 9 日
環境変数の情報	AWS CodeBuild には、ビルドコマンドで使用できるいくつかの環境変数が用意されています。独自の環境変数を定義することもできます。詳細については、「ビルド環境の環境変数 (p. 112)」を参照してください。	2016 年 7 月 12 日

変更	説明	変更日
トラブルシューティング情報	トラブルシューティング情報が利用できるようになりました。詳細については、「VPC 設定のトラブルシューティング (p. 117)」を参照してください。	2016 年 5 月 12 日
Jenkins プラグインの初回リリース	これは、AWS CodeBuild Jenkins プラグインの初回のリリースです。詳細については、「Jenkins で AWS CodeBuild を使用する (p. 145)」を参照してください。	2016 年 5 月 12 日
ユーザーガイド初回リリース	これは、AWS CodeBuild ユーザーガイドの初回リリースです。	2016 年 12 月 1 日

AWS の用語集

最新の AWS の用語については、『AWS General Reference』の「AWS の用語集」を参照してください。