
AWS CodeBuild

用户指南

API 版本 2016-10-06



AWS CodeBuild: 用户指南

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

什么是 AWS CodeBuild ?	1
如何运行 AWS CodeBuild	1
AWS CodeBuild 定价	2
如何开始使用 AWS CodeBuild ?	2
概念	2
AWS CodeBuild 如何工作	2
后续步骤	3
入门	4
步骤 1 : 创建或使用 Amazon S3 存储桶来存储构建输入和输出	4
步骤 2 : 创建用于构建的源代码	5
步骤 3 : 创建构建规范	6
步骤 4 : 将源代码和构建规范添加到输入存储桶中	8
步骤 5 : 创建构建项目	8
步骤 6 : 运行构建	12
步骤 7 : 查看汇总的构建信息	13
查看汇总的构建信息 (控制台)	13
查看汇总的构建信息 (AWS CLI)	14
步骤 8 : 查看详细的构建信息	15
步骤 9 : 获取构建输出项目	17
第 10 步 : 清除	17
后续步骤	18
示例	19
Windows 示例	20
运行示例	20
目录结构	21
文件	22
基于使用案例的示例	38
Amazon ECR 示例	38
Docker 示例	41
GitHub Enterprise 示例	46
GitHub 拉取请求示例	51
将 AWS Config 与 AWS CodeBuild 结合使用的示例	53
构建徽章示例	54
构建通知示例	56
自定义映像示例中的 Docker	73
AWS CodeDeploy 示例	75
AWS Lambda 示例	78
Elastic Beanstalk 示例	79
基于代码的示例	85
C++ 示例	85
Go 示例	87
Maven 示例	89
Node.js 示例	92
Python 示例	94
Ruby 示例	96
Scala 示例	98
Java 示例	101
Linux 中的 .NET 核心示例	103
规划构建	106
构建规范参考	107
构建规范文件名称和存储位置	107
构建规范语法	107
构建规范示例	112
构建规范版本	113

构建环境参考	113
AWS CodeBuild 提供的 Docker 映像	114
构建环境计算类型	119
构建环境中的 Shell 和命令	120
构建环境中的环境变量	120
构建环境中的后台任务	122
在本地测试和调试	122
使用 AWS CodeBuild 代理在本地计算机上测试和调试	122
接收有关新的 AWS CodeBuild 代理版本的通知	122
直接运行 AWS CodeBuild	124
先决条件	124
直接运行 AWS CodeBuild (控制台)	124
直接运行 AWS CodeBuild (AWS CLI)	124
VPC 支持	125
使用案例	125
在您的 AWS CodeBuild 项目中启用 Amazon VPC 访问	125
VPC 的最佳实践	126
VPC 设置问题排查	127
使用 VPC 终端节点	127
在您创建 VPC 终端节点前	127
创建 AWS CodeBuild 的 VPC 终端节点	128
CloudFormation VPC 模板	128
将 AWS CodePipeline 与 AWS CodeBuild 结合使用	136
先决条件	136
创建使用了 AWS CodeBuild 的管道 (AWS CodePipeline 控制台)	137
创建使用 AWS CodeBuild 的管道 (AWS CLI)	141
将 AWS CodeBuild 构建操作添加到管道 (AWS CodePipeline 控制台)	144
将 AWS CodeBuild 测试操作添加到管道 (AWS CodePipeline 控制台)	148
将 AWS CodeBuild 与 Jenkins 结合使用	152
设置 Jenkins	152
安装插件	152
使用插件	152
使用构建项目和构建	154
使用构建项目	154
创建构建项目	154
查看构建项目名称的列表	168
查看构建项目的详细信息	169
创建构建触发器	171
编辑构建触发器	172
更改构建项目的设置	174
删除构建项目	182
使用构建	183
运行构建项目	183
查看构建详细信息	190
查看构建 ID 的列表	192
查看构建项目的构建 ID 列表	194
停止构建	195
删除构建	196
高级主题	198
高级设置	198
为 IAM 组或 IAM 用户添加 AWS CodeBuild 访问权限	198
创建 AWS CodeBuild 服务角色	202
为 AWS CodeBuild 创建和配置 AWS KMS CMK	205
安装和配置 AWS CLI	207
命令行参考	207
AWS 开发工具包和工具参考	208
适用于 AWS CodeBuild 的受支持的 AWS 开发工具包和工具	208

指定终端节点	209
指定 AWS CodeBuild 终端节点 (AWS CLI)	209
指定 AWS CodeBuild 终端节点 (AWS 开发工具包)	209
身份验证和访问控制	211
身份验证	211
访问控制	212
访问管理概述	212
使用基于身份的策略 (IAM 策略)	214
AWS CodeBuild 权限参考	220
使用 CloudTrail 记录 API 调用	222
CloudTrail 中的 AWS CodeBuild 信息	223
了解 AWS CodeBuild 日志文件条目	223
故障排除	225
创建或更新构建项目时收到错误：“CodeBuild 无权执行：sts:AssumeRole”	225
运行构建时收到错误：“必须使用指定的终端节点来寻址当前尝试访问的存储桶...”	226
运行构建时收到错误：“无法上传项目：arn 无效”	226
错误：“Unable to Locate Credentials”	226
构建规范中的前期命令无法被后续命令识别	227
来自错误存储库的 Apache Maven 构建参考项目	227
默认情况下，构建命令以根用户身份运行	228
Bourne 外壳 (sh) 必须存在于构建映像中	228
运行构建时出现错误“AWS CodeBuild is experiencing an issue”	229
使用自定义构建映像时出现错误“BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE”	229
当文件名包含非美国英语字符时，构建可能失败	229
当从 Amazon EC2 Parameter Store 获取参数时，构建可能失败	230
尝试下载缓存时出现“Access denied”错误消息	230
错误：“Unable to download cache: RequestError: send request failed caused by: x509: failed to load system roots and no roots provided”(无法下载缓存：请求错误：发送请求失败原因：x509：无法加载系统根，没有提供任何根)	231
错误：“Unable to download certificate from S3. AccessDenied”	231
错误：“Git Clone Failed: unable to access 'your-repository-URL': SSL certificate problem: self signed certificate”	232
错误：“The policy's default version was not created by enhanced zero click role creation or was not the most recent version created by enhanced zero click role creation”(策略的默认版本不是通过增强的零单击角色创建来创建的或不是通过增强的零单击角色创建来创建的最新版本)。	232
限制	234
构建项目	234
构建	234
适用于 Windows 的 AWS CodeBuild – 第三方说明	235
1) 基本 Docker 映像 – windowsservercore	235
2) 基于 Windows 的 Docker 映像 – Choco	236
3) 基于 Windows 的 Docker 映像 – git -- 版本 2.16.2	236
4) 基于 Windows 的 Docker 映像 – microsoft-build-tools -- 版本 15.0.26320.2	236
5) 基于 Windows 的 Docker 映像 – nuget.commandline -- 版本 4.5.1	236
7) 基于 Windows 的 Docker 映像 – netfx-4.6.2-devpack	236
8) 基于 Windows 的 Docker 映像 – visualfsharpertools , v 4.0	236
9) 基于 Windows 的 Docker 映像 – netfx-pcl-reference-assemblies-4.6	240
10) 基于 Windows 的 Docker 映像 – visualcppbuildtools , v 14.0.25420.1	241
11) 基于 Windows 的 Docker 映像 – microsoft-windows-netfx3-ondemand-package.cab	243
12) 基于 Windows 的 Docker 映像 – dotnet-sdk	244
文档历史记录	245
早期更新	245
AWS 词汇表	251

什么是 AWS CodeBuild ?

AWS CodeBuild 是一项在云中完全托管的生成服务。AWS CodeBuild 可编译源代码，运行单元测试，并生成可供部署的项目。使用 AWS CodeBuild，您无需预置、管理和扩展自己的构建服务器。它提供了适用于最热门编程语言的预先打包的构建环境以及 Apache Maven 和 Gradle 等构建工具。您还可以在 AWS CodeBuild 中自定义构建环境以使用自己的构建工具。AWS CodeBuild 会自动扩展以满足峰值构建请求。

AWS CodeBuild 具有以下优势：

- 完全托管 – 利用 AWS CodeBuild，您无需设置、修补、更新和管理自己的构建服务器。
- 按需 – AWS CodeBuild 可以按需扩展，以满足您的构建需求。您只需为使用的构建分钟数付费。
- 开箱即用 – AWS CodeBuild 提供了适用于最热门编程语言的预配置构建环境。您只需指向您的构建脚本以开始首次构建即可。

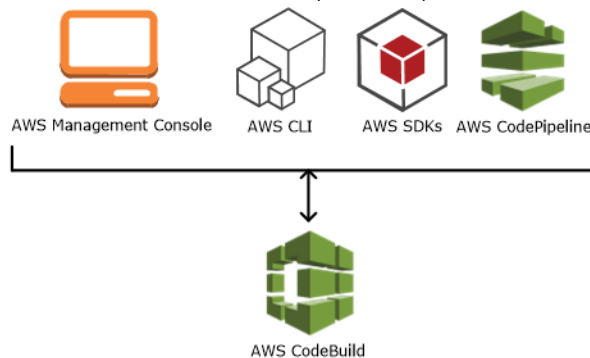
有关更多信息，请参阅 [AWS CodeBuild](#)。

主题

- [如何运行 AWS CodeBuild \(p. 1\)](#)
- [AWS CodeBuild 定价 \(p. 2\)](#)
- [如何开始使用 AWS CodeBuild ? \(p. 2\)](#)
- [AWS CodeBuild 概念 \(p. 2\)](#)

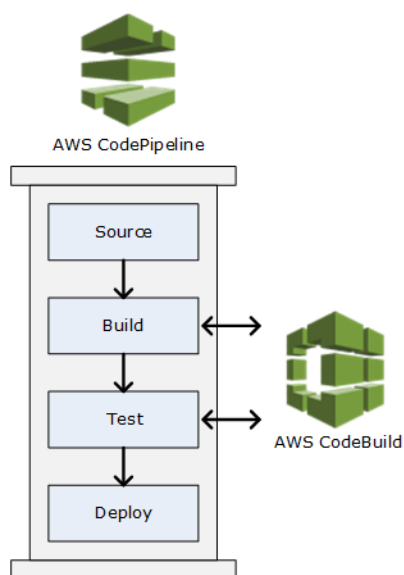
如何运行 AWS CodeBuild

您可以使用 AWS CodeBuild 控制台或 AWS CodePipeline 控制台来运行 AWS CodeBuild。您也可以使用 AWS Command Line Interface (AWS CLI) 或 AWS 软件开发工具包来自动运行 AWS CodeBuild。



要使用 AWS CodeBuild 控制台、AWS CLI、AWS 开发工具包运行 AWS CodeBuild，请参阅[直接运行 AWS CodeBuild \(p. 124\)](#)。

如下图所示，您可以添加 AWS CodeBuild 作为用于 AWS CodePipeline 中管道的构建或测试阶段的构建或测试操作。AWS CodePipeline 是一种持续交付服务，它可以建模、可视化和自动执行发布代码所需的步骤。其中包括构建您的代码。管道是一个描述发布流程中代码更改情况的工作流程结构。



要使用 AWS CodePipeline 创建管道并添加 AWS CodeBuild 构建或测试操作，请参阅 [将 AWS CodePipeline 与 AWS CodeBuild 结合使用 \(p. 136\)](#)。有关 AWS CodePipeline 的更多信息，请参阅 [AWS CodePipeline 用户指南](#)。

AWS CodeBuild 定价

有关信息，请参阅 [AWS CodeBuild 定价](#)。

如何开始使用 AWS CodeBuild ？

我们建议您完成以下步骤：

1. 阅读 [概念 \(p. 2\)](#) 中的信息，深入了解 AWS CodeBuild。
2. 按照 [入门 \(p. 4\)](#) 中的说明在示例方案中试验 AWS CodeBuild。
3. 按照 [规划构建 \(p. 106\)](#) 中的说明在自己的方案中使用 AWS CodeBuild。

AWS CodeBuild 概念

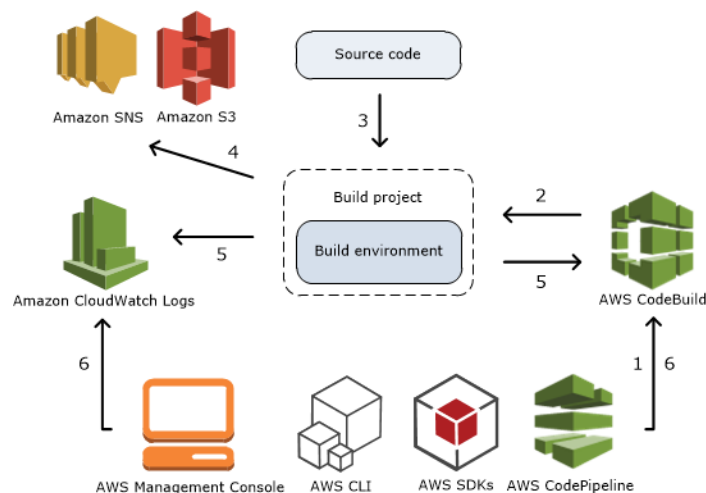
以下概念对了解 AWS CodeBuild 的工作原理来说很重要。

主题

- [AWS CodeBuild 如何工作 \(p. 2\)](#)
- [后续步骤 \(p. 3\)](#)

AWS CodeBuild 如何工作

下图显示了当您借助 AWS CodeBuild 运行构建时会发生的情况：



1. 作为输入，您必须为 AWS CodeBuild 提供构建项目。生成项目 定义 AWS CodeBuild 如何运行生成任务。它包括的信息有源代码获取位置、要使用的生成环境、要运行的生成命令和存储生成输出的位置等。生成环境 是由操作系统、编程语言运行时和 AWS CodeBuild 用于运行生成任务的工具组成的。有关更多信息，请参阅：
 - [创建构建项目 \(p. 154\)](#)
 - [构建环境参考 \(p. 113\)](#)
2. AWS CodeBuild 使用构建项目创建构建环境。
3. AWS CodeBuild 将源代码下载到构建环境中，然后使用构建项目中定义的或源代码中直接包含的构建规范 (build spec)。生成规范 是生成命令和相关设置的集合，采用 YAML 格式，由 AWS CodeBuild 用来运行生成任务。有关更多信息，请参阅 [构建规范参考 \(p. 107\)](#)。
4. 如果存在任何构建输出，则该构建环境会将其输出上传到 Amazon S3 存储桶。构建环境也可以执行您在构建规范中指定的任务 (例如，将构建通知发送到 Amazon SNS 主题)。有关示例，请参阅 [构建通知示例 \(p. 56\)](#)。
5. 在构建运行时，构建环境会将信息发送给 AWS CodeBuild 和 Amazon CloudWatch Logs。
6. 在构建运行时，您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包从 AWS CodeBuild 中获取汇总的构建信息，并从 Amazon CloudWatch Logs 中获取详细的构建信息。如果您使用 AWS CodePipeline 运行构建，则可以从 AWS CodePipeline 获取有限的构建信息。

后续步骤

现在，您了解了有关 AWS CodeBuild 的更多信息，我们建议您完成以下步骤：

1. 按照 [入门 \(p. 4\)](#) 中的说明在示例方案中试验 AWS CodeBuild。
2. 按照 [规划构建 \(p. 106\)](#) 中的说明在自己的方案中使用 AWS CodeBuild。

AWS CodeBuild 入门

在本演练中，您将使用 AWS CodeBuild 将一系列示例源代码输入文件（我们称之为构建输入项目或构建输入）构建为一个可部署的源代码版本（我们称之为构建输出项目或构建输出）。具体来说，您将指示 AWS CodeBuild 使用 Apache Maven（一种常用的构建工具）将一组 Java 类文件构建为 Java 存档（JAR）文件。您无需熟悉 Apache Maven 或 Java 即可完成本演练。

Important

完成本演练会对您的 AWS 账户产生费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

主题

- [步骤 1：创建或使用 Amazon S3 存储桶来存储构建输入和输出 \(p. 4\)](#)
- [步骤 2：创建用于构建的源代码 \(p. 5\)](#)
- [步骤 3：创建构建规范 \(p. 6\)](#)
- [步骤 4：将源代码和构建规范添加到输入存储桶中 \(p. 8\)](#)
- [步骤 5：创建构建项目 \(p. 8\)](#)
- [步骤 6：运行构建 \(p. 12\)](#)
- [步骤 7：查看汇总的构建信息 \(p. 13\)](#)
- [步骤 8：查看详细的构建信息 \(p. 15\)](#)
- [步骤 9：获取构建输出项目 \(p. 17\)](#)
- [第 10 步：清除 \(p. 17\)](#)
- [后续步骤 \(p. 18\)](#)

步骤 1：创建或使用 Amazon S3 存储桶来存储构建输入和输出

要完成本演练，您需要两个 Amazon S3 存储桶：

- 其中一个存储桶用于存储构建输入（我们称之为输入存储桶）。在本演练中，我们将此输入存储桶命名为 `codebuild-region-ID-account-ID-input-bucket`，其中 *region-ID* 表示存储桶的 AWS 区域，*account-ID* 表示您的 AWS 账户 ID。
- 另一个存储桶用于存储构建输出（我们称之为输出存储桶）。在本演练中，我们将此输出存储桶命名为 `codebuild-region-ID-account-ID-output-bucket`。

如果您为任何一个存储桶选择其他名称，请在整个演练中使用此名称进行替换。

这两个存储桶必须与您的构建项目处在同一个 AWS 区域中。例如，如果您指示 AWS CodeBuild 在美国东部（俄亥俄州）区域中运行构建，则这些存储桶也必须位于美国东部（俄亥俄州）区域中。

要创建存储桶，请参阅 Amazon Simple Storage Service 用户指南中的 [创建存储桶](#)。

Note

在本演练中，您可以使用一个存储桶。但是，使用两个存储桶更容易查看构建输入的来源以及构建输出的去向。

尽管 AWS CodeBuild 也支持存储在 AWS CodeCommit、GitHub 和 Bitbucket 存储库中的构建输入，但本演练不说明如何使用它们。有关更多信息，请参阅 [规划构建 \(p. 106\)](#)。

步骤 2：创建用于构建的源代码

在此步骤中，您将创建需要 AWS CodeBuild 构建到输出存储桶的源代码。此源代码包含两个 Java 类文件和一个 Apache Maven 项目对象模型 (POM) 文件。

1. 在您的本地计算机或实例上的空目录中，创建此目录结构。

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

2. 使用您选择的文本编辑器创建此文件，将其命名为 `MessageUtil.java`，然后保存在 `src/main/java` 目录中。

```
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;
    }
}
```

创建此类文件是用来输出传入的字符串。`MessageUtil` 构造函数用于设置字符串。`printMessage` 方法用于创建输出。`salutationMessage` 方法用于输出 `Hi!` 后跟字符串。

3. 创建此文件，将其命名为 `TestMessageUtil.java`，然后将它保存在 `/src/test/java` 目录中。

```
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());
}
}
```

此类文件用于将 MessageUtil 类中的 message 变量设置为 Robert。然后，通过检查输出中是否出现字符串 Robert 和 Hi!Robert 来测试是否成功设置 message 变量。

4. 创建此文件，将其命名为 pom.xml，然后保存在根 (顶级) 目录中。

```
<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 将使用此文件中的指令将 MessageUtil.java 和 TestMessageUtil.java 文件转换为名为 messageUtil-1.0.jar 的文件，然后运行指定测试。

此时，您的目录结构应如下所示。

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

步骤 3：创建构建规范

在此步骤中，您将创建一个构建规范文件。生成规范是生成命令和相关设置的集合，采用 YAML 格式，由 AWS CodeBuild 用来运行生成任务。如果没有构建规范，AWS CodeBuild 就无法将您的构建输入成功转换为构建输出，也无法在构建环境中找到构建输出项目以便上传到输出存储桶中。

创建此文件，将其命名为 buildspec.yml，然后保存在根 (顶级) 目录中。

```
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

因为构建规范声明必须为有效的 YAML，所以构建规范声明中的间隔至关重要。如果构建规范声明中的空格数与此不匹配，则构建可能会立即失败。您可以使用 YAML 验证程序测试构建规范声明是否为有效的 YAML。

Note

您可以在创建构建项目时单独声明构建命令，而不是将构建规范文件包含在源代码中。如果您需要使用其他构建命令来构建源代码，而不是每次更新源代码存储库，这个方法很有用。有关更多信息，请参阅 [构建规范语法 \(p. 107\)](#)。

在此构建规范声明中：

- `version` 表示正在使用的构建规范标准的版本。此构建规范声明使用最新版本 0.2。
- `phases` 表示您可以指示 AWS CodeBuild 运行命令的构建阶段。这些构建阶段包括 `install`、`pre_build`、`build` 和 `post_build`。您无法更改这些构建阶段名称的拼写，也无法创建其他构建阶段名称。

本示例中，在 `build` 阶段，AWS CodeBuild 运行 `mvn install` 命令。此命令指示 Apache Maven 编译和测试 Java 类文件，然后将编译完的文件打包为构建输出项目。出于完整性考虑，本示例的每个构建阶段中都放了几条 `echo` 命令。您稍后查看本演练中详细的构建信息时，这些 `echo` 命令的输出可以帮助您更好地理解 AWS CodeBuild 运行命令的方式以及顺序。（尽管本示例中包含了所有构建阶段，但如果您不打算在某个构建阶段运行任何命令，则无需包含该构建阶段。）对于包含的每个构建阶段，AWS CodeBuild 将按照列出的顺序，从头到尾运行每个指定命令（一次运行一个命令）。

- `artifacts` 表示 AWS CodeBuild 会上传到输出存储桶的一组构建输出项目。`files` 表示要包含在构建输出中的文件。AWS CodeBuild 会上传在构建环境的 `messageUtil-1.0.jar` 相对目录中找到的单个 `target` 文件。文件 `messageUtil-1.0.jar` 和目录 `target` 只是根据本示例中 Apache Maven 创建和存储构建输出项目的方式来命名的。在您自己的构建项目中，这些文件和目录名称会有所不同。

有关更多信息，请参见 [构建规范参考 \(p. 107\)](#)。

此时，您的目录结构应如下所示。

```
(root directory name)
|-- pom.xml
|-- buildspec.yml
`-- src
```

```
|-- main
|   |-- java
|   |   |-- MessageUtil.java
|-- test
|   |-- java
|   |   |-- TestMessageUtil.java
```

步骤 4：将源代码和构建规范添加到输入存储桶中

在此步骤中，您将把源代码和构建规范文件添加到输入存储桶中。

使用操作系统的 ZIP 实用工具，创建一个名为 MessageUtil.zip 的文件，其中包含 MessageUtil.java、TestMessageUtil.java、pom.xml 和 buildspec.yml。

MessageUtil.zip 文件的目录结构必须如下所示。

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

Important

请不要包含 (*root directory name*) 目录，而只包含 (*root directory name*) 目录中包含的目录和文件。

将 MessageUtil.zip 文件上传至名为 codebuild-*region-ID-account-ID*-input-bucket 的输入存储桶中。

Important

对于 AWS CodeCommit、GitHub 和 Bitbucket 存储库，按照惯例，您必须在每个存储库的根 (顶级) 位置存储一个名为 buildspec.yml 的构建规范文件，或者将构建规范声明作为构建项目定义的一部分包含。请勿创建包含存储库源代码和构建规范文件的 ZIP 文件。

仅对于存储在 Amazon S3 存储桶中的构建输入，您需要创建一个包含源代码的 ZIP 文件和一个 (按照惯例) 位于根 (顶级) 位置的名为 buildspec.yml 的构建规范文件，或者将构建规范声明作为构建项目定义的一部分包含。

如果您要为构建规范文件使用其他名称，或者要在根位置之外的位置引用构建规范，则可指定构建规范覆盖作为构建项目定义的一部分。有关更多信息，请参阅 [构建规范文件名称和存储位置 \(p. 107\)](#)。

步骤 5：创建构建项目

在此步骤中，您将创建一个构建项目，AWS CodeBuild 将使用它来运行构建项目。生成项目定义 AWS CodeBuild 如何运行生成任务。它包括的信息有源代码获取位置、要使用的生成环境、要运行的生成命令和存储生成输出的位置等。生成环境是由操作系统、编程语言运行时和 AWS CodeBuild 用于运行生成任务的工具组成的。构建环境以 Docker 镜像的形式表示。(有关更多信息，请参阅 [Docker 文档网站上的 Docker](#)

概述主题。) 对于此构建环境，您需要指示 AWS CodeBuild 使用包含 Java 开发工具包 (JDK) 和 Apache Maven 的 Docker 镜像。

您可以通过 [AWS CodeBuild 控制台 \(p. 9\)](#) 或 [AWS CLI \(p. 10\)](#) 完成此步骤。

Note

您可以通过多种方式使用 AWS CodeBuild：通过 AWS CodeBuild 控制台、AWS CodePipeline、AWS CLI 或 AWS 开发工具包。本演练将展示如何使用 AWS CodeBuild 控制台和 AWS CLI。要了解如何使用 AWS CodePipeline，请参阅[将 AWS CodePipeline 与 AWS CodeBuild 结合使用 \(p. 136\)](#)。要了解如何使用 AWS 开发工具包，请参阅[直接运行 AWS CodeBuild \(p. 124\)](#)。

创建构建项目 (控制台)

1. Sign in to the AWS 管理控制台 and open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 在 AWS 区域选择器中，选择一个支持 AWS CodeBuild 的区域。有关更多信息，请参阅 Amazon Web Services 一般参考 中的“区域和终端节点”主题中的 [AWS CodeBuild](#)。
3. 如果显示欢迎页面，请选择 Get started (开始使用)。

如果未显示欢迎页面，请在导航窗格上，选择 Build projects，然后选择 Create project。

4. 在 Configure your project 页面上，对于 Project name，键入此构建项目的名称 (此示例中为 codebuild-demo-project)。构建项目名称在您的各个 AWS 账户内必须是唯一的。如果您使用其他名称，请在整个演练中使用此名称进行替换。

Note

在 Configure project 页面上，您可能会看到类似于以下内容的错误消息：User: **user-ARN** is not authorized to perform: codebuild:ListProjects。这很可能是因为您以 IAM 用户身份登录 AWS 管理控制台，而此用户身份没有足够的权限，无法在控制台中使用 AWS CodeBuild。要修复此问题，请从 AWS 管理控制台 注销，然后使用属于以下任一 IAM 实体的凭证重新登录：

- 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅 IAM 用户指南中的[账户根用户](#)。
- AWS 账户中的 IAM 管理员用户。有关更多信息，请参阅 IAM 用户指南中的[创建您的第一个 IAM 管理员用户和组](#)。
- 您的 AWS 账户中的 IAM 用户，已向该用户或其所属的 IAM 组附加名为 AWSCodeBuildAdminAccess、AmazonS3ReadOnlyAccess 和 IAMFullAccess 的 AWS 托管策略。如果您的 AWS 账户中没有具备这些权限的 IAM 用户或组，并且您无法为 IAM 用户或组添加这些权限，请联系您的 AWS 账户管理员寻求帮助。有关更多信息，请参阅[适用于 AWS CodeBuild 的 AWS 托管 \(预定义\) 策略 \(p. 216\)](#)。

5. 在 Source: What to build 中，对于 Source provider，选择 Amazon S3。
6. 对于 Bucket，选择 codebuild-**region-ID-account-ID**-input-bucket。
7. 对于 S3 object key，键入 MessageUtil.zip。
8. 在 Environment: How to build 中，对于 Environment image，将 Use an image managed by AWS CodeBuild 保持选中状态。
9. 对于 Operating system，选择 Ubuntu。
10. 对于 Runtime，选择 Java。
11. 对于 Version，选择 aws/codebuild/java:openjdk-8。
12. 对于 Build specification，将 Use the buildspec.yml in the source code root directory 保持选中状态。
13. 在 Artifacts: Where to put the artifacts from this build project 中，对于 Artifacts type，选择 Amazon S3。
14. 将 Artifacts name 留空。

15. 对于 Bucket name，选择 codebuild-**region-ID-account-ID**-output-bucket。
16. 在 Service role 中，将 Create a service role in your account 保持选中状态，并将 Role name 保持不变。
17. 选择 Continue (继续)。
18. 在 Review 页面上，选择 Save。

向前跳至 [步骤 6：运行构建 \(p. 12\)](#)。

创建构建项目 (AWS CLI)

1. 使用 AWS CLI 运行 create-project 命令，如下所示：

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

输出中将显示 JSON 格式的数据。将数据复制到已安装 AWS CLI 的本地计算机或实例上某个位置的名称为 create-project.json 的文件中。如果您选择使用其他文件名，请务必在本演练过程中将其替换为 create-project.json。

按照以下格式修改所复制的数据，然后保存结果：

```
{
  "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"
}
```

将 **serviceIAMRole** 替换为 AWS CodeBuild 服务角色的 Amazon 资源名称 (ARN) (如 `arn:aws:iam::account-ID:role/role-name`)。如需创建一个角色，请参阅 [创建 AWS CodeBuild 服务角色 \(p. 202\)](#)。

在此数据中：

- name 表示此构建项目的必需标识符 (在本示例中为 codebuild-demo-project)。如果您使用了其他名称，请在整个过程中使用此名称进行替换。构建项目名称在您账户的所有构建项目中必须是唯一的。
- 对于 source，type 是一个必需值，表示源代码的存储库类型 (在本示例中，S3 表示 Amazon S3 存储桶)。
- 对于 source，location 表示源代码的路径 (在本示例中，为输入存储桶名称后跟 ZIP 文件名称)。
- 对于 artifacts，type 是一个必需值，表示构建输出项目的存储库类型 (在本示例中，S3 表示 Amazon S3 存储桶)。
- 对于 artifacts，location 表示您先前创建或识别的输出存储桶的名称 (在本示例中为 codebuild-**region-ID-account-ID**-output-bucket)。
- 对于 environment，type 是一个必需值，表示构建环境的类型 (LINUX_CONTAINER 是目前唯一允许的值)。

- 对于 `environment`，`image` 是一个必需值，表示此构建项目将会使用的 Docker 镜像名称和标签结合，由 Docker 镜像存储库类型指定（在本示例中，`aws/codebuild/java:openjdk-8` 表示 AWS CodeBuild Docker 镜像存储库中的 Docker 镜像）。`aws/codebuild/java` 是 Docker 镜像的名称。`openjdk-8` 是 Docker 镜像的标签。

要查找您可以在自己方案中使用的更多 Docker 镜像，请参阅 [构建环境参考](#) (p. 113)。

- 对于 `environment`，`computeType` 是一个必需值，表示 AWS CodeBuild 将会使用的计算资源（在本示例中为 `BUILD_GENERAL1_SMALL`）。

Note

原始 JSON 格式数据中的其他可用值，如 `description`、`buildspec`、`auth`（包括 `type` 和 `resource`）、`path`、`namespaceType`、`name`（对于 `artifacts`）、`packaging`、`environmentVariables`（包括 `name` 和 `value`）、`timeoutInMinutes`、`encryptionKey` 和 `tags`（包括 `key` 和 `value`）为可选的值。本演练中未使用这些值，因此没有在这里显示。有关更多信息，请参阅 [创建构建项目 \(AWS CLI\)](#) (p. 162)。

2. 切换到您刚才保存的文件所在的目录，然后再次运行 `create-project` 命令。

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

如果成功，输出中将显示与此类似的数据。

```
{
  "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` 表示有关此构建项目的信息。
- `tags` 表示已经声明的所有标签。
- `packaging` 表示构建输出项目将如何存储在输出存储桶中。`NONE` 表示将在输出存储桶内部创建一个文件夹，然后构建输出项目将存储在该文件夹内。
- `lastModified` 表示构建项目最后一次更改的时间，采用 Unix 时间格式。

- `timeoutInMinutes` 表示如果构建未完成，AWS CodeBuild 会在多少分钟后停止构建。(默认为 60 分钟。)
- `created` 表示构建项目的创建时间，采用 Unix 时间格式。
- `environmentVariables` 表示已经声明且可供 AWS CodeBuild 在构建过程中使用的所有环境变量。
- `encryptionKey` 表示 AWS CodeBuild 用于加密构建输出项目的 AWS KMS 客户主密钥 (CMK) 的 Amazon 资源名称 (ARN)。
- `arn` 表示构建项目的 ARN。

Note

在运行 `create-project` 命令后，可能会形式类似于以下内容的错误消息：User: **`user-ARN`** is not authorized to perform: `codebuild:CreateProject`。这很可能是因为在您使用 IAM 用户的凭证配置 AWS CLI 时，该用户没有足够的权限，无法使用 AWS CodeBuild 创建构建项目。要修复此问题，请使用属于以下任一 IAM 实体的凭证配置 AWS CLI：

- 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅 IAM 用户指南中的[账户根用户](#)。
- AWS 账户中的 IAM 管理员用户。有关更多信息，请参阅 IAM 用户指南中的[创建您的第一个 IAM 管理员用户和组](#)。
- 您的 AWS 账户中的 IAM 用户，已向该用户或其所属的 IAM 组附加名为 `AWSCodeBuildAdminAccess`、`AmazonS3ReadOnlyAccess` 和 `IAMFullAccess` 的 AWS 托管策略。如果您的 AWS 账户中没有具备这些权限的 IAM 用户或组，并且您无法为 IAM 用户或组添加这些权限，请联系您的 AWS 账户管理员寻求帮助。有关更多信息，请参阅[适用于 AWS CodeBuild 的 AWS 托管 \(预定义\) 策略 \(p. 216\)](#)。

步骤 6：运行构建

在此步骤中，您将指示 AWS CodeBuild 使用构建项目中的设置来运行构建。

您可以通过 [AWS CodeBuild 控制台 \(p. 12\)](#) 或 [AWS CLI \(p. 12\)](#) 完成此步骤。

运行构建 (控制台)

1. 如果未显示 Build projects 页面，请在导航窗格中，选择 Build projects。
2. 在构建项目列表中，选择 `codebuild-demo-project`，然后选择 Start build。
3. 在 Start new build 页面上，选择 Start build。
4. 向前跳至 [步骤 7：查看汇总的构建信息 \(p. 13\)](#)。

运行构建 (AWS CLI)

1. 使用 AWS CLI 运行 `start-build` 命令：

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

将 **`project-name`** 替换为上一步中的构建项目名称 (如 `codebuild-demo-project`)。

2. 如果成功，输出中将显示与以下内容类似的数据：

```
{
  "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 表示有关此构建的信息。
 - buildComplete 表示构建是否完成，true 表示完成，否则为 false。
 - initiator 表示启动构建的实体。
 - artifacts 表示有关构建输出的信息，包括其位置。
 - projectName 表示构建项目的名称。
 - buildStatus 表示运行 start-build 命令时当前构建的状态。
 - currentPhase 表示运行 start-build 命令时的当前构建阶段。
 - startTime 表示构建过程开始的时间，采用 Unix 时间格式。
 - id 表示构建的 ID。
 - arn 表示构建的 Amazon 资源名称 (ARN)。

记下此 id 值。您在下一个步骤中需要使用此值。

步骤 7：查看汇总的构建信息

在此步骤中，您将查看有关构建状态的汇总信息。

您可以通过 [AWS CodeBuild 控制台](#) (p. 13) 或 [AWS CLI](#) (p. 14) 完成此步骤。

查看汇总的构建信息 (控制台)

1. 如果未显示 codebuild-demo-project:*build-ID* 页面，请在导航栏中，选择 Build history。接下来，在构建项目列表中，选择与 Project 的 codebuild-demo-project 对应的 Build run 链接。应该只有一个匹配的链接。(如果您以前完成过本演练，请选择与 Completed 列中最新的值对应的链接。)
2. 在构建详细信息页面上的 Phase details 中，应显示构建阶段的列表，并且 Status 列中的值为 Succeeded：
 - SUBMITTED
 - PROVISIONING

- DOWNLOAD_SOURCE
- INSTALL
- PRE_BUILD
- BUILD
- POST_BUILD
- UPLOAD_ARTIFACTS
- FINALIZING
- COMPLETED

在页面标题区域中，应显示一个带有 Succeeded 的绿色框。

如果您看到的是带 In Progress 的蓝色框，请选择刷新按钮以查看最新进度。

3. 在每个构建阶段的旁边，Duration 值表示构建阶段持续的时间。Completed 值表示构建阶段的结束时间。

如果您展开一个构建阶段，会看到此阶段的开始和结束时间。

向前跳至 [步骤 8：查看详细的构建信息 \(p. 15\)](#)。

查看汇总的构建信息 (AWS CLI)

使用 AWS CLI 运行 batch-get-builds 命令。

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

将 *id* 替换为上一步的输出中显示的 id 值。

如果成功，输出中将显示与此类似的数据。

```
{
  "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=38calc4a-e9ca-4dbc-bef1-d52bfEXAMPLE",
        "streamName": "38calc4a-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` 表示所有不具备信息的构建的构建 ID。在本示例中，其应该为空。
- `builds` 表示有关每个具备信息的构建项目的信息。在本示例中，输出中只显示了有关一个构建项目的信息。
- `phases` 表示 AWS CodeBuild 在构建过程中运行的一组构建阶段。有关每个构建阶段的信息将分别列出，其中包含：`startTime`、`endTime` 和 `durationInSeconds` (采用 Unix 时间格式的构建阶段开始时间和结束时间，以及构建阶段的持续时间，以秒为单位)，以及 `phaseType` (如 `SUBMITTED`、`PROVISIONING`、`DOWNLOAD_SOURCE`、`INSTALL`、`PRE_BUILD`、`BUILD`、`POST_BUILD`、`UPLOAD`、`FINALIZING` 或 `COMPLETED`)，还有 `phaseStatus` (如 `SUCCEEDED`、`FAILED`、`FAULT`、`TIMED_OUT`、`IN_PROGRESS` 或 `STOPPED`)。首次运行 `batch-get-builds` 命令时，可能不会有很多 (或没有) 阶段。使用相同构建 ID 再次运行 `batch-get-builds` 命令后，输出中应当会出现更多构建阶段。
- `logs` 表示 Amazon CloudWatch Logs 中有关构建日志的信息。
- `md5sum` 和 `sha256sum` 表示构建输出项目的 MD5 和 SHA-256 哈希值。只有将相关构建项目的 `packaging` 值设为 `ZIP` (您未在本演练中设置)，输出中才会显示这些值。您可以将这些哈希值和校验和工具一起使用，确认文件的完整性和真实性。

Note

您还可以使用 Amazon S3 控制台查看这些哈希值。选中构建输出项目旁边的框，然后依次选择 **Actions** 和 **Properties**。在 **Properties** 窗格中，展开 **Metadata**，然后查看 `x-amz-meta-codebuild-content-md5` 和 `x-amz-meta-codebuild-content-sha256` 的值。(在 Amazon S3 控制台中，构建输出项目的 ETag 值不应解释为 MD5 或 SHA-256 哈希值。)

如果您使用 AWS 软件开发工具包来获取这些哈希值，这些值会被命名为 `codebuild-content-md5` 和 `codebuild-content-sha256`。

- `endTime` 表示构建过程结束的时间，采用 Unix 时间格式。

步骤 8：查看详细的构建信息

在此步骤中，您将查看有关 CloudWatch Logs 中构建项目的详细信息。

您可以通过 [AWS CodeBuild 控制台 \(p. 16\)](#) 或 [AWS CLI \(p. 16\)](#) 完成此步骤。

查看详细的构建信息 (控制台)

1. 上一步完成后，构建详细页面继续显示，Build logs 中显示了构建日志的最后 10,000 行内容。要在 CloudWatch Logs 中查看完整构建日志，请选择 View entire log 链接。
2. 在 CloudWatch Logs 日志流中，您可以浏览日志事件。默认情况下，只显示最近的一组日志事件。要查看以前的日志事件，请滚动到列表开头。
3. 在本演练中，大多数日志事件包含的都是关于 AWS CodeBuild 下载构建相关文件并将其安装到构建环境中的详细信息，您可能并不关心这些信息。您可以使用 Filter events 框来减少显示的信息。例如，如果您在 Filter events 框中键入 "[INFO]" 并按下 Enter 键，则仅显示那些包含 [INFO] 字符的事件。有关更多信息，请参阅 Amazon CloudWatch 用户指南 中的 [筛选条件和模式语法](#)。

向前跳至 [步骤 9：获取构建输出项目 \(p. 17\)](#)。

查看详细的构建信息 (AWS CLI)

1. 使用您的 Web 浏览器，转到上一步的输出中显示的 deepLink 位置 (如 `https://console.aws.amazon.com/cloudwatch/home?region=region-ID#logEvent:group=/aws/codebuild/codebuild-demo-project;stream=38ca1c4a-e9ca-4dbc-bef1-d52bfEXAMPLE`)。
2. 在 CloudWatch Logs 日志流中，您可以浏览日志事件。默认情况下，只显示最近的一组日志事件。要查看以前的日志事件，请滚动到列表开头。
3. 在本演练中，大多数日志事件包含的都是关于 AWS CodeBuild 下载构建相关文件并将其安装到构建环境中的详细信息，您可能并不关心这些信息。您可以使用 Filter events 框来减少显示的信息。例如，如果您在 Filter events 框中键入 "[INFO]" 并按下 Enter 键，则仅显示那些包含 [INFO] 字符的事件。有关更多信息，请参阅 Amazon CloudWatch 用户指南 中的 [筛选条件和模式语法](#)。

CloudWatch Logs 日志流的这些部分与本演练有关。

```
...
[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
```

在本示例中，AWS CodeBuild 成功完成了预构建、构建和构建后这些构建阶段。它运行单元测试并成功生成 messageUtil-1.0.jar 文件。

步骤 9：获取构建输出项目

在此步骤中，您将获取 AWS CodeBuild 生成的 messageUtil-1.0.jar 文件并将其上传到输出存储桶中。

您可以通过 [AWS CodeBuild 控制台 \(p. 17\)](#) 或 [Amazon S3 控制台 \(p. 17\)](#) 完成此步骤。

获取构建输出项目 (AWS CodeBuild 控制台)

1. 上一步完成后，AWS CodeBuild 控制台仍处于打开状态，构建详细信息页面也继续显示，您可以展开 Build details，然后选择 Build artifacts 链接。这将打开 Amazon S3 中用于构建输出项目的文件夹。(如果未显示构建详细信息页面，请在导航栏中选择 Build history，然后选择 Build run 链接。)
2. 打开名为 target 的文件夹，您可以在此处找到名为 messageUtil-1.0.jar 的构建输出项目文件。

向前跳至 [第 10 步：清除 \(p. 17\)](#)。

获取构建输出项目 (Amazon S3 控制台)

1. 通过以下网址打开 Amazon S3 控制台：<https://console.aws.amazon.com/s3/>。
2. 打开名为 codebuild-**region-ID-account-ID**-output-bucket 的存储桶。
3. 打开名为 codebuild-demo-project 的文件夹。
4. 打开名为 target 的文件夹，您可以在此处找到名为 messageUtil-1.0.jar 的构建输出项目文件。

第 10 步：清除

为防止您的 AWS 账户持续产生费用，您可以删除本演练中使用的输入存储桶。有关说明，请参阅 Amazon Simple Storage Service 开发人员指南 中的 [删除或清空存储桶](#)。

如果您使用 IAM 用户而不是 AWS 根账户或管理员 IAM 用户来删除此存储桶，则该用户必须拥有额外的访问权限。(不建议使用 AWS 根账户。)将标记 (**### BEGIN ADDING STATEMENT HERE ###** 和 **### END ADDING STATEMENTS HERE ###**) 之间的语句添加到用户的现有访问策略中。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (...)。请勿删除现有访问策略中的任何语句，也不要将这些省略号写入现有策略中。

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

后续步骤

在本演练中，您使用 AWS CodeBuild 将一组 Java 类文件构建为一个 JAR 文件。然后查看了构建的结果。

您现在可以按照 [规划构建 \(p. 106\)](#) 中的说明，尝试在自己的方案中使用 AWS CodeBuild。如果您觉得自己还没准备好，可以尝试构建我们提供的一些示例。有关更多信息，请参阅 [示例 \(p. 19\)](#)。

AWS CodeBuild 示例

这些基于使用案例的示例可用于试验 AWS CodeBuild：

名称	描述
Amazon ECR 示例 (p. 38)	使用 Amazon ECR 存储库中的 Docker 镜像，以使用 Apache Maven 生成单个 JAR 文件。
Docker 示例 (p. 41)	使用由支持 Docker 的 AWS CodeBuild 提供的构建映像来通过 Apache Maven 生成一个 Docker 映像。将 Docker 镜像推送到 Amazon ECR 中的存储库。您可以调整该示例以将 Docker 映像推送到 Docker Hub。
GitHub Enterprise 示例 (p. 46)	将 AWS CodeBuild 和 GitHub Enterprise 一起用作源存储库，并安装了证书、启用了 Webhook，可在每次将代码更改被推送到存储库时重建源代码。
GitHub 拉取请求示例 (p. 51)	将 AWS CodeBuild 和 GitHub 一起用作源存储库并启用 Webhook，可在每次将代码更改被推送到存储库时重建源代码。
将 AWS Config 与 AWS CodeBuild 结合使用的示例 (p. 53)	说明如何设置 AWS Config。列出跟踪的 AWS CodeBuild 资源并描述如何在 AWS Config 中查找 AWS CodeBuild 项目。
构建徽章示例 (p. 54)	说明如何使用构建徽章设置 AWS CodeBuild。
构建通知示例 (p. 56)	使用 Apache Maven 生成单个 JAR 文件。给 Amazon SNS 主题的订阅者发送构建通知。
自定义映像示例中的 Docker (p. 73)	使用自定义 Docker 映像生成 Docker 映像。
AWS CodeDeploy 示例 (p. 75)	使用 Apache Maven 生成单个 JAR 文件。使用 AWS CodeDeploy 将 JAR 文件部署到 Amazon Linux 实例。您也可以使用 AWS CodePipeline 来构建和部署示例。
AWS Lambda 示例 (p. 78)	使用 AWS CodeBuild 以及 Lambda、AWS CloudFormation 和 AWS CodePipeline 来构建和部署遵循了 AWS 无服务器应用程序模型 (AWS SAM) 标准的无服务器应用程序。
Elastic Beanstalk 示例 (p. 79)	使用 Apache Maven 生成单个 WAR 文件。使用 Elastic Beanstalk 将 WAR 文件部署到 Elastic Beanstalk 实例。

可以使用这些基于代码的示例试验 AWS CodeBuild：

名称	描述
C++ 示例 (p. 85)	使用 C++ 输出单个 .out 文件。

名称	描述
Go 示例 (p. 87)	使用 Go 输出单个二进制文件。
Maven 示例 (p. 89)	使用 Apache Maven 生成单个 JAR 文件。
Node.js 示例 (p. 92)	使用 Mocha 测试代码中的内部变量是否包含特定字符串值。生成单个 .js 文件。
Python 示例 (p. 94)	使用 Python 测试代码中的内部变量是否已设置为特定字符串值。生成单个 .py 文件。
Ruby 示例 (p. 96)	使用 RSpec 测试代码中的内部变量是否已设置为特定字符串值。生成单个 .rb 文件。
Scala 示例 (p. 98)	使用 sbt 生成单个 JAR 文件。
Java 示例 (p. 101)	使用 Apache Maven 生成单个 WAR 文件。
Windows 示例 (p. 20)	从 C#、F# 或 Visual Basic 代码使用 Microsoft .NET Framework 或 .NET 核心来构建可执行文件。
Linux 中的 .NET 核心示例 (p. 103)	使用 .NET 核心通过用 C# 编写的代码构建可执行文件。

适用于 AWS CodeBuild 的 Microsoft Windows 示例

这些示例使用运行 Microsoft Windows Server 2016、Microsoft .NET Framework 和 .NET 核心开发工具包的 AWS CodeBuild 构建环境，通过使用 C#、F# 和 Visual Basic 编写的代码生成可执行文件。

Important

运行这些示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

运行示例

要运行这些示例：

1. 按照本主题的“目录结构和文件”部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit 或 GitHub 存储库中。

Important

请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 创建生成项目、运行生成，并遵循[直接运行 AWS CodeBuild \(p. 124\)](#)中的步骤。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "name": "sample-windows-build-project",
  "source": {
```

```
{
  "type": "S3",
  "location": "codebuild-region-ID-account-ID-input-bucket/windows-build-input-artifact.zip"
},
"artifacts": {
  "type": "S3",
  "location": "codebuild-region-ID-account-ID-output-bucket",
  "packaging": "ZIP",
  "name": "windows-build-output-artifact.zip"
},
"environment": {
  "type": "WINDOWS_CONTAINER",
  "image": "aws/codebuild/windows-base:1.0",
  "computeType": "BUILD_GENERAL1_MEDIUM"
},
"serviceRole": "arn:aws:iam::account-ID:role/role-name",
"encryptionKey": "arn:aws:kms:region-ID:account-ID:key/key-ID"
}
```

3. 要获取构建输出项目，请在您的 Amazon S3 输出存储桶中，将 *windows-build-output-artifact.zip* 文件下载到您的本地计算机或实例。提取内容以获取可执行文件和其他文件。
- 在 CSharpHelloWorld\bin\Debug 目录中可以找到使用 Microsoft .NET Framework 的 C# 示例的可执行文件 CSharpHelloWorld.exe。
 - 在 FSharpHelloWorld\bin\Debug 目录中可以找到使用 Microsoft .NET Framework 的 F# 示例的可执行文件 FSharpHelloWorld.exe。
 - 在 VBHelloWorld\bin\Debug 目录中可以找到使用 Microsoft .NET Framework 的 Visual Basic 示例的可执行文件 VBHelloWorld.exe。
 - 在 bin\Debug\netcoreapp1.0 目录中可以找到使用 .NET 核心的 C# 示例的可执行文件 HelloWorldSample.dll。

目录结构

这些示例采用以下目录结构。

C# 和 Microsoft .NET Framework

```
(root directory name)
|-- buildspec.yml
|-- CSharpHelloWorld.sln
`-- CSharpHelloWorld
    |-- App.config
    |-- CSharpHelloWorld.csproj
    |-- Program.cs
    `-- Properties
        `-- AssemblyInfo.cs
```

F# 和 Microsoft .NET Framework

```
(root directory name)
|-- buildspec.yml
|-- FSharpHelloWorld.sln
`-- FSharpHelloWorld
    |-- App.config
    |-- AssemblyInfo.fs
    |-- FSharpHelloWorld.fsproj
    `-- Program.fs
```

Visual Basic 和 Microsoft .NET Framework

```
(root directory name)
|-- buildspec.yml
|-- VBHelloWorld.sln
`-- VBHelloWorld
    |-- App.config
    |-- HelloWorld.vb
    |-- VBHelloWorld.vbproj
    `-- My Project
        |-- Application.Designer.vb
        |-- Application.myapp
        |-- AssemblyInfo.vb
        |-- Resources.Designer.vb
        |-- Resources.resx
        |-- Settings.Designer.vb
        `-- Settings.settings
```

C# 和 .NET 核心

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

文件

这些示例使用以下文件：

C# 和 Microsoft .NET Framework

buildspec.yml (在 *(root directory name)*):

```
version: 0.2

env:
  variables:
    SOLUTION: .\CSharpHelloWorld.sln
    PACKAGE_DIRECTORY: .\packages
    DOTNET_FRAMEWORK: 4.6.2

phases:
  build:
    commands:
      - '& "C:\ProgramData\chocolatey\bin\NuGet.exe" restore $env:SOLUTION -
        PackagesDirectory $env:PACKAGE_DIRECTORY'
      - '& "C:\Program Files (x86)\MSBuild\14.0\Bin\MSBuild.exe" -
        p:FrameworkPathOverride="C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework
        \.NETFramework\v$env:DOTNET_FRAMEWORK" $env:SOLUTION'
    artifacts:
      files:
        - .\CSharpHelloWorld\bin\Debug\*
```

CSharpHelloWorld.sln (在 *(root directory name)*):

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 14
```

```
VisualStudioVersion = 14.0.25420.1
MinimumVisualStudioVersion = 10.0.40219.1
Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "CSharpHelloWorld", "CSharpHelloWorld\
CSharpHelloWorld.csproj", "{2F8752D5-E628-4A38-AA7E-BC4B4E697CBB}"
EndProject
Global
    GlobalSection(SolutionConfigurationPlatforms) = preSolution
        Debug|Any CPU = Debug|Any CPU
        Release|Any CPU = Release|Any CPU
    EndGlobalSection
    GlobalSection(ProjectConfigurationPlatforms) = postSolution
        {2F8752D5-E628-4A38-AA7E-BC4B4E697CBB}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
        {2F8752D5-E628-4A38-AA7E-BC4B4E697CBB}.Debug|Any CPU.Build.0 = Debug|Any CPU
        {2F8752D5-E628-4A38-AA7E-BC4B4E697CBB}.Release|Any CPU.ActiveCfg = Release|Any CPU
        {2F8752D5-E628-4A38-AA7E-BC4B4E697CBB}.Release|Any CPU.Build.0 = Release|Any CPU
    EndGlobalSection
    GlobalSection(SolutionProperties) = preSolution
        HideSolutionNode = FALSE
    EndGlobalSection
EndGlobal
```

App.config (在 *(root directory name)*\CSharpHelloWorld):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.2" />
  </startup>
</configuration>
```

CSharpHelloWorld.csproj (在 *(root directory name)*\CSharpHelloWorld):

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="14.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/
developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props"
    Condition="Exists('$(MSBuildExtensionsPath)\
$(MSBuildToolsVersion)\Microsoft.Common.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{2F8752D5-E628-4A38-AA7E-BC4B4E697CBB}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>CSharpHelloWorld</RootNamespace>
    <AssemblyName>CSharpHelloWorld</AssemblyName>
    <TargetFrameworkVersion>v4.6.2</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
```

```
<Optimize>true</Optimize>
<OutputPath>bin\Release\</OutputPath>
<DefineConstants>TRACE</DefineConstants>
<ErrorReport>prompt</ErrorReport>
<WarningLevel>4</WarningLevel>
</PropertyGroup>
<ItemGroup>
  <Reference Include="System" />
  <Reference Include="System.Core" />
  <Reference Include="System.Xml.Linq" />
  <Reference Include="System.Data.DataSetExtensions" />
  <Reference Include="Microsoft.CSharp" />
  <Reference Include="System.Data" />
  <Reference Include="System.Net.Http" />
  <Reference Include="System.Xml" />
</ItemGroup>
<ItemGroup>
  <Compile Include="Program.cs" />
  <Compile Include="Properties\AssemblyInfo.cs" />
</ItemGroup>
<ItemGroup>
  <None Include="App.config" />
</ItemGroup>
<Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
<!-- To modify your build process, add your task inside one of the targets below and
uncomment it.
      Other similar extension points exist, see Microsoft.Common.targets.
<Target Name="BeforeBuild">
</Target>
<Target Name="AfterBuild">
</Target>
-->
</Project>
```

Program.cs (在 *(root directory name)*\CSharpHelloWorld):

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CSharpHelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello World");
            System.Threading.Thread.Sleep(10);
        }
    }
}
```

AssemblyInfo.cs (在 *(root directory name)*\CSharpHelloWorld\Properties):

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
```

```
[assembly: AssemblyTitle("CSharpHelloWorld")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("CSharpHelloWorld")]
[assembly: AssemblyCopyright("Copyright © 2017")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components. If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("2f8752d5-e628-4a38-aa7e-bc4b4e697cbb")]

// Version information for an assembly consists of the following four values:
//
// Major Version
// Minor Version
// Build Number
// Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

F# 和 Microsoft .NET Framework

buildspec.yml (在 *(root directory name)*):

```
version: 0.2

env:
  variables:
    SOLUTION: .\FSharpHelloWorld.sln
    PACKAGE_DIRECTORY: .\packages
    DOTNET_FRAMEWORK: 4.6.2

phases:
  build:
    commands:
      - '& "C:\ProgramData\chocolatey\bin\NuGet.exe" restore $env:SOLUTION -
        PackagesDirectory $env:PACKAGE_DIRECTORY'
      - '& "C:\Program Files (x86)\MSBuild\14.0\Bin\MSBuild.exe" -
        p:FrameworkPathOverride="C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework
        \.NETFramework\v$env:DOTNET_FRAMEWORK" $env:SOLUTION'
    artifacts:
      files:
        - .\FSharpHelloWorld\bin\Debug\*
```

FSharpHelloWorld.sln (在 *(root directory name)*):

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 14
VisualStudioVersion = 14.0.25420.1
MinimumVisualStudioVersion = 10.0.40219.1
Project("{F2A71F9B-5D33-465A-A702-920D77279786}") = "FSharpHelloWorld", "FSharpHelloWorld
\FSharpHelloWorld.fsproj", "{D60939B6-526D-43F4-9A89-577B2980DF62}"
```

```
EndProject
Global
  GlobalSection(SolutionConfigurationPlatforms) = preSolution
    Debug|Any CPU = Debug|Any CPU
    Release|Any CPU = Release|Any CPU
  EndGlobalSection
  GlobalSection(ProjectConfigurationPlatforms) = postSolution
    {D60939B6-526D-43F4-9A89-577B2980DF62}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
    {D60939B6-526D-43F4-9A89-577B2980DF62}.Debug|Any CPU.Build.0 = Debug|Any CPU
    {D60939B6-526D-43F4-9A89-577B2980DF62}.Release|Any CPU.ActiveCfg = Release|Any CPU
    {D60939B6-526D-43F4-9A89-577B2980DF62}.Release|Any CPU.Build.0 = Release|Any CPU
  EndGlobalSection
  GlobalSection(SolutionProperties) = preSolution
    HideSolutionNode = FALSE
  EndGlobalSection
EndGlobal
```

App.config (在 *(root directory name)*\FSharpHelloWorld):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.2" />
  </startup>
</configuration>
```

AssemblyInfo.fs (在 *(root directory name)*\FSharpHelloWorld):

```
namespace FSharpHelloWorld.AssemblyInfo

open System.Reflection
open System.Runtime.CompilerServices
open System.Runtime.InteropServices

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[<assembly: AssemblyTitle("FSharpHelloWorld")>]
[<assembly: AssemblyDescription("")>]
[<assembly: AssemblyConfiguration("")>]
[<assembly: AssemblyCompany("")>]
[<assembly: AssemblyProduct("FSharpHelloWorld")>]
[<assembly: AssemblyCopyright("Copyright © 2017")>]
[<assembly: AssemblyTrademark("")>]
[<assembly: AssemblyCulture("")>]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components. If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[<assembly: ComVisible(false)>]

// The following GUID is for the ID of the typelib if this project is exposed to COM
[<assembly: Guid("d60939b6-526d-43f4-9a89-577b2980df62")>]

// Version information for an assembly consists of the following four values:
//
// Major Version
// Minor Version
// Build Number
// Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
```

```
// [<assembly: AssemblyVersion("1.0.*")>]
[<assembly: AssemblyVersion("1.0.0.0")>]
[<assembly: AssemblyFileVersion("1.0.0.0")>]

do
    (

```

FSharpHelloWorld.fsproj (在 *(root directory name)* \FSharpHelloWorld):

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="14.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/
developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props"
    Condition="Exists('$(MSBuildExtensionsPath)\
$(MSBuildToolsVersion)\Microsoft.Common.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>d60939b6-526d-43f4-9a89-577b2980df62</ProjectGuid>
    <OutputType>Exe</OutputType>
    <RootNamespace>FSharpHelloWorld</RootNamespace>
    <AssemblyName>FSharpHelloWorld</AssemblyName>
    <TargetFrameworkVersion>v4.6.2</TargetFrameworkVersion>
    <AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
    <TargetFSharpCoreVersion>4.4.0.0</TargetFSharpCoreVersion>
    <Name>FSharpHelloWorld</Name>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>>false</Optimize>
    <Tailcalls>>false</Tailcalls>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <WarningLevel>3</WarningLevel>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DocumentationFile>bin\Debug\FSharpHelloWorld.XML</DocumentationFile>
    <Prefer32Bit>true</Prefer32Bit>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <Tailcalls>true</Tailcalls>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <WarningLevel>3</WarningLevel>
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DocumentationFile>bin\Release\FSharpHelloWorld.XML</DocumentationFile>
    <Prefer32Bit>true</Prefer32Bit>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="mscorlib" />
    <Reference Include="FSharp.Core, Version=$(TargetFSharpCoreVersion), Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a">
      <Private>True</Private>
    </Reference>
    <Reference Include="System" />
    <Reference Include="System.Core" />
    <Reference Include="System.Numerics" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="AssemblyInfo.fs" />
    <Compile Include="Program.fs" />
    <None Include="App.config" />
  </ItemGroup>

```



```

</ItemGroup>
<PropertyGroup>
  <MinimumVisualStudioVersion Condition="'$(MinimumVisualStudioVersion)' == ''">11</MinimumVisualStudioVersion>
</PropertyGroup>
<Choose>
  <When Condition="'$(VisualStudioVersion)' == '11.0'">
    <PropertyGroup Condition="Exists('$(MSBuildExtensionsPath32)\..\Microsoft SDKs\F#\3.0\Framework\v4.0\Microsoft.FSharp.Targets')">
      <FSharpTargetsPath>$(MSBuildExtensionsPath32)\..\Microsoft SDKs\F#\3.0\Framework\v4.0\Microsoft.FSharp.Targets</FSharpTargetsPath>
    </PropertyGroup>
  </When>
  <Otherwise>
    <PropertyGroup Condition="Exists('$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)\FSharp\Microsoft.FSharp.Targets')">
      <FSharpTargetsPath>$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v$(VisualStudioVersion)\FSharp\Microsoft.FSharp.Targets</FSharpTargetsPath>
    </PropertyGroup>
  </Otherwise>
</Choose>
<Import Project="$(FSharpTargetsPath)" />
<!-- To modify your build process, add your task inside one of the targets below and uncomment it.
      Other similar extension points exist, see Microsoft.Common.targets.
-->
<Target Name="BeforeBuild">
</Target>
<Target Name="AfterBuild">
</Target>
-->
</Project>

```

Program.fs (在 *(root directory name)* \FSharpHelloWorld):

```

// Learn more about F# at http://fsharp.org
// See the 'F# Tutorial' project for more help.

[<EntryPoint>]
let main argv =
    printfn "Hello World"
    0 // return an integer exit code

```

Visual Basic 和 Microsoft .NET Framework

buildspec.yml (在 *(root directory name)*):

```

version: 0.2

env:
  variables:
    SOLUTION: .\VBHelloWorld.sln
    PACKAGE_DIRECTORY: .\packages
    DOTNET_FRAMEWORK: 4.6.2

phases:
  build:
    commands:
      - '& "C:\ProgramData\chocolatey\bin\NuGet.exe" restore $env:SOLUTION - PackagesDirectory $env:PACKAGE_DIRECTORY'
      - '& "C:\Program Files (x86)\MSBuild\14.0\Bin\MSBuild.exe" -p:FrameworkPathOverride="C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v$env:DOTNET_FRAMEWORK" $env:SOLUTION'.2

```

```
artifacts:
  files:
    - .\VBHelloWorld\bin\Debug\*
```

VBHelloWorld.sln (在 *(root directory name)*):

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 14
VisualStudioVersion = 14.0.25420.1
MinimumVisualStudioVersion = 10.0.40219.1
Project("{F184B08F-C81C-45F6-A57F-5ABD9991F28F}") = "VBHelloWorld", "VBHelloWorld\VBHelloWorld.vbproj", "{4DCEC446-7156-4FE6-8CCC-219E34DD409D}"
EndProject
Global
  GlobalSection(SolutionConfigurationPlatforms) = preSolution
    Debug|Any CPU = Debug|Any CPU
    Release|Any CPU = Release|Any CPU
  EndGlobalSection
  GlobalSection(ProjectConfigurationPlatforms) = postSolution
    {4DCEC446-7156-4FE6-8CCC-219E34DD409D}.Debug|Any CPU.ActiveCfg = Debug|Any CPU
    {4DCEC446-7156-4FE6-8CCC-219E34DD409D}.Debug|Any CPU.Build.0 = Debug|Any CPU
    {4DCEC446-7156-4FE6-8CCC-219E34DD409D}.Release|Any CPU.ActiveCfg = Release|Any CPU
    {4DCEC446-7156-4FE6-8CCC-219E34DD409D}.Release|Any CPU.Build.0 = Release|Any CPU
  EndGlobalSection
  GlobalSection(SolutionProperties) = preSolution
    HideSolutionNode = FALSE
  EndGlobalSection
EndGlobal
```

App.config (在 *(root directory name)\VBHelloWorld*):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6.2" />
  </startup>
</configuration>
```

HelloWorld.vb (在 *(root directory name)\VBHelloWorld*):

```
Module HelloWorld

  Sub Main()
    MsgBox("Hello World")
  End Sub

End Module
```

VBHelloWorld.vbproj (在 *(root directory name)\VBHelloWorld*):

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="14.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props"
    Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{4DCEC446-7156-4FE6-8CCC-219E34DD409D}</ProjectGuid>
```

```

<OutputType>Exe</OutputType>
<StartupObject>VBHelloWorld.HelloWorld</StartupObject>
<RootNamespace>VBHelloWorld</RootNamespace>
<AssemblyName>VBHelloWorld</AssemblyName>
<FileAlignment>512</FileAlignment>
<MyType>Console</MyType>
<TargetFrameworkVersion>v4.6.2</TargetFrameworkVersion>
<AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
</PropertyGroup>
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
  <PlatformTarget>AnyCPU</PlatformTarget>
  <DebugSymbols>true</DebugSymbols>
  <DebugType>full</DebugType>
  <DefineDebug>true</DefineDebug>
  <DefineTrace>true</DefineTrace>
  <OutputPath>bin\Debug\</OutputPath>
  <DocumentationFile>VBHelloWorld.xml</DocumentationFile>
  <NoWarn>42016,41999,42017,42018,42019,42032,42036,42020,42021,42022</NoWarn>
</PropertyGroup>
<PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
  <PlatformTarget>AnyCPU</PlatformTarget>
  <DebugType>pdbonly</DebugType>
  <DefineDebug>>false</DefineDebug>
  <DefineTrace>true</DefineTrace>
  <Optimize>true</Optimize>
  <OutputPath>bin\Release\</OutputPath>
  <DocumentationFile>VBHelloWorld.xml</DocumentationFile>
  <NoWarn>42016,41999,42017,42018,42019,42032,42036,42020,42021,42022</NoWarn>
</PropertyGroup>
<PropertyGroup>
  <OptionExplicit>On</OptionExplicit>
</PropertyGroup>
<PropertyGroup>
  <OptionCompare>Binary</OptionCompare>
</PropertyGroup>
<PropertyGroup>
  <OptionStrict>Off</OptionStrict>
</PropertyGroup>
<PropertyGroup>
  <OptionInfer>On</OptionInfer>
</PropertyGroup>
<ItemGroup>
  <Reference Include="System" />
  <Reference Include="System.Data" />
  <Reference Include="System.Deployment" />
  <Reference Include="System.Xml" />
  <Reference Include="System.Core" />
  <Reference Include="System.Xml.Linq" />
  <Reference Include="System.Data.DataSetExtensions" />
  <Reference Include="System.Net.Http" />
</ItemGroup>
<ItemGroup>
  <Import Include="Microsoft.VisualBasic" />
  <Import Include="System" />
  <Import Include="System.Collections" />
  <Import Include="System.Collections.Generic" />
  <Import Include="System.Data" />
  <Import Include="System.Diagnostics" />
  <Import Include="System.Linq" />
  <Import Include="System.Xml.Linq" />
  <Import Include="System.Threading.Tasks" />
</ItemGroup>
<ItemGroup>
  <Compile Include="HelloWorld.vb" />
  <Compile Include="My Project\AssemblyInfo.vb" />
  <Compile Include="My Project\Application.Designer.vb">

```

```

    <AutoGen>True</AutoGen>
    <DependentUpon>Application.myapp</DependentUpon>
  </Compile>
  <Compile Include="My Project\Resources.Designer.vb">
    <AutoGen>True</AutoGen>
    <DesignTime>True</DesignTime>
    <DependentUpon>Resources.resx</DependentUpon>
  </Compile>
  <Compile Include="My Project\Settings.Designer.vb">
    <AutoGen>True</AutoGen>
    <DependentUpon>Settings.settings</DependentUpon>
    <DesignTimeSharedInput>True</DesignTimeSharedInput>
  </Compile>
</ItemGroup>
<ItemGroup>
  <EmbeddedResource Include="My Project\Resources.resx">
    <Generator>VbMyResourcesResXFileCodeGenerator</Generator>
    <LastGenOutput>Resources.Designer.vb</LastGenOutput>
    <CustomToolNamespace>My.Resources</CustomToolNamespace>
    <SubType>Designer</SubType>
  </EmbeddedResource>
</ItemGroup>
<ItemGroup>
  <None Include="My Project\Application.myapp">
    <Generator>MyApplicationCodeGenerator</Generator>
    <LastGenOutput>Application.Designer.vb</LastGenOutput>
  </None>
  <None Include="My Project\Settings.settings">
    <Generator>SettingsSingleFileGenerator</Generator>
    <CustomToolNamespace>My</CustomToolNamespace>
    <LastGenOutput>Settings.Designer.vb</LastGenOutput>
  </None>
  <None Include="App.config" />
</ItemGroup>
<Import Project="$(MSBuildToolsPath)\Microsoft.VisualBasic.targets" />
<!-- To modify your build process, add your task inside one of the targets below and
uncomment it.
    Other similar extension points exist, see Microsoft.Common.targets.
<Target Name="BeforeBuild">
</Target>
<Target Name="AfterBuild">
</Target>
-->
</Project>

```

Application.Designer.vb (在 *(root directory name)*\VBHelloWorld\My Project):

```

'-----
' <auto-generated>
'   This code was generated by a tool.
'   Runtime Version:4.0.30319.42000
'
'   Changes to this file may cause incorrect behavior and will be lost if
'   the code is regenerated.
' </auto-generated>
'-----

Option Strict On
Option Explicit On

```

Application.myapp (在 *(root directory name)*\VBHelloWorld\My Project):

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<MyApplicationData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <MySubMain>false</MySubMain>
  <SingleInstance>false</SingleInstance>
  <ShutdownMode>0</ShutdownMode>
  <EnableVisualStyles>true</EnableVisualStyles>
  <AuthenticationMode>0</AuthenticationMode>
  <ApplicationType>2</ApplicationType>
  <SaveMySettingsOnExit>true</SaveMySettingsOnExit>
</MyApplicationData>
```

AssemblyInfo.vb (在 (root directory name)\VBHelloWorld\My Project):

```
Imports System
Imports System.Reflection
Imports System.Runtime.InteropServices

' General Information about an assembly is controlled through the following
' set of attributes. Change these attribute values to modify the information
' associated with an assembly.

' Review the values of the assembly attributes

<Assembly: AssemblyTitle("VBHelloWorld")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("VBHelloWorld")>
<Assembly: AssemblyCopyright("Copyright © 2017")>
<Assembly: AssemblyTrademark("")>

<Assembly: ComVisible(False)>

'The following GUID is for the ID of the typelib if this project is exposed to COM
<Assembly: Guid("137c362b-36ef-4c3e-84ab-f95082487a5a")>

' Version information for an assembly consists of the following four values:
'
' Major Version
' Minor Version
' Build Number
' Revision
'
' You can specify all the values or you can default the Build and Revision Numbers
' by using the '*' as shown below:
' <Assembly: AssemblyVersion("1.0.*")>

<Assembly: AssemblyVersion("1.0.0.0")>
<Assembly: AssemblyFileVersion("1.0.0.0")>
```

Resources.Designer.vb (在 (root directory name)\VBHelloWorld\My Project):

```
'-----
' <auto-generated>
'   This code was generated by a tool.
'   Runtime Version:4.0.30319.42000
'
'   Changes to this file may cause incorrect behavior and will be lost if
'   the code is regenerated.
' </auto-generated>
'-----

Option Strict On
Option Explicit On
```

```

Namespace My.Resources

    'This class was auto-generated by the StronglyTypedResourceBuilder
    'class via a tool like ResGen or Visual Studio.
    'To add or remove a member, edit your .ResX file then rerun ResGen
    'with the /str option, or rebuild your VS project.
    '''<summary>
    ''' A strongly-typed resource class, for looking up localized strings, etc.
    '''</summary>

    <Global.System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools.StronglyTypedResourceBu
    "4.0.0.0"), _
    Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
    Global.System.Runtime.CompilerServices.CompilerGeneratedAttribute(), _
    Global.Microsoft.VisualBasic.HideModuleNameAttribute()> _
    Friend Module Resources

        Private resourceMan As Global.System.Resources.ResourceManager

        Private resourceCulture As Global.System.Globalization.CultureInfo

        '''<summary>
        ''' Returns the cached ResourceManager instance used by this class.
        '''</summary>

        <Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrowsableSta
        -
        Friend ReadOnly Property ResourceManager() As Global.System.Resources.ResourceManager
            Get
                If Object.ReferenceEquals(resourceMan, Nothing) Then
                    Dim temp As Global.System.Resources.ResourceManager = New
                    Global.System.Resources.ResourceManager("VBHelloWorld.Resources",
                    GetType(Resources).Assembly)
                    resourceMan = temp
                End If
                Return resourceMan
            End Get
        End Property

        '''<summary>
        ''' Overrides the current thread's CurrentUICulture property for all
        ''' resource lookups using this strongly typed resource class.
        '''</summary>

        <Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrowsableSta
        -
        Friend Property Culture() As Global.System.Globalization.CultureInfo
            Get
                Return resourceCulture
            End Get
            Set(ByVal value As Global.System.Globalization.CultureInfo)
                resourceCulture = value
            End Set
        End Property
    End Module
End Namespace

```

Resources.resx (在 *(root directory name)* \VBHelloWorld\My Project):

```

<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema

```

Version 2.0

The primary goals of this format is to allow a simple XML format that is mostly human readable. The generation and parsing of the various data types are done through the TypeConverter classes associated with the data types.

Example:

```
... ado.net/XML headers & schema ...
<resheader name="resmime" type="text/microsoft-resx"/>
<resheader name="version">2.0</resheader>
<resheader name="reader">System.Resources.ResXResourceReader,
System.Windows.Forms, ...</resheader>
<resheader name="writer">System.Resources.ResXResourceWriter,
System.Windows.Forms, ...</resheader>
<data name="Name1"><value>this is my long string</value><comment>this is a comment</comment></data>
<data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
<data name="Bitmap1" mime="application/x-microsoft.net.object.binary.base64">
  <value>[base64 mime encoded serialized .NET Framework object]</value>
</data>
<data name="Icon1" type="System.Drawing.Icon, System.Drawing" mime="application/x-microsoft.net.object.bytearray.base64">
  <value>[base64 mime encoded string representing a byte array form of the .NET Framework object]</value>
  <comment>This is a comment</comment>
</data>
```

There are any number of "resheader" rows that contain simple name/value pairs.

Each data row contains a name, and value. The row also contains a type or mime. Type corresponds to a .NET class that support text/value conversion through the TypeConverter architecture. Classes that don't support this are serialized and stored with the mime set.

The mime is used for serialized objects, and tells the ResXResourceReader how to depersist the object. This is currently not extensible. For a given mime the value must be set accordingly:

Note - application/x-microsoft.net.object.binary.base64 is the format that the ResXResourceWriter will generate, however the reader can read any of the formats listed below.

```
mimetypes: application/x-microsoft.net.object.binary.base64
value      : The object must be serialized with
              : System.Serialization.Formatters.Binary.BinaryFormatter
              : and then encoded with base64 encoding.

mimetypes: application/x-microsoft.net.object.soap.base64
value      : The object must be serialized with
              : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
              : and then encoded with base64 encoding.

mimetypes: application/x-microsoft.net.object.bytearray.base64
value      : The object must be serialized into a byte array
              : using a System.ComponentModel.TypeConverter
              : and then encoded with base64 encoding.

-->
<xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xsd:element name="root" msdata:IsDataSet="true">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
```

```

<xsd:element name="metadata">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" />
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="mimetype" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
<xsd:element name="assembly">
  <xsd:complexType>
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="name" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
<xsd:element name="data">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
      <xsd:element name="comment" type="xsd:string" minOccurs="0"
msdata:Ordinal="2" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" msdata:Ordinal="1" />
    <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
    <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
  </xsd:complexType>
</xsd:element>
<xsd:element name="resheader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0"
msdata:Ordinal="1" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:complexType>
</xsd:element>
</xsd:schema>
<resheader name="resmimetype">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="version">
  <value>2.0</value>
</resheader>
<resheader name="reader">
  <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<resheader name="writer">
  <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
</root>

```

Settings.Designer.vb (在 (root directory name)\VBHelloWorld\My Project):

```

'-----
' <auto-generated>
'   This code was generated by a tool.
'   Runtime Version:4.0.30319.42000
'

```



```
'
    Changes to this file may cause incorrect behavior and will be lost if
    the code is regenerated.
' </auto-generated>
'-----

Option Strict On
Option Explicit On

Namespace My

    <Global.System.Runtime.CompilerServices.CompilerGeneratedAttribute(), _
    Global.System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.SettingsDesigner
    "11.0.0.0"), _
    Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrowsableStat
    -
    Partial Friend NotInheritable Class MySettings
        Inherits Global.System.Configuration.ApplicationSettingsBase

        Private Shared defaultInstance As MySettings =
        CType(Global.System.Configuration.ApplicationSettingsBase.Synchronized(New MySettings),
        MySettings)

        #Region "My.Settings Auto-Save Functionality"
        #If _MyType = "WindowsForms" Then
            Private Shared addedHandler As Boolean

            Private Shared addedHandlerLockObject As New Object

            <Global.System.Diagnostics.DebuggerNonUserCodeAttribute(),
            Global.System.ComponentModel.EditorBrowsableAttribute(Global.System.ComponentModel.EditorBrowsableStat
            -
            Private Shared Sub AutoSaveSettings(ByVal sender As Global.System.Object, ByVal e
            As Global.System.EventArgs)
                If My.Application.SaveMySettingsOnExit Then
                    My.Settings.Save()
                End If
            End Sub
        #End If
    #End Region

    Public Shared ReadOnly Property [Default]() As MySettings
        Get

            #If _MyType = "WindowsForms" Then
                If Not addedHandler Then
                    SyncLock addedHandlerLockObject
                        If Not addedHandler Then
                            AddHandler My.Application.Shutdown, AddressOf AutoSaveSettings
                            addedHandler = True
                        End If
                    End SyncLock
                End If
            #End If
            Return defaultInstance
        End Get
    End Property
End Class
End Namespace

Namespace My

    <Global.Microsoft.VisualBasic.HideModuleNameAttribute(), _
    Global.System.Diagnostics.DebuggerNonUserCodeAttribute(), _
    Global.System.Runtime.CompilerServices.CompilerGeneratedAttribute()> _
```

```
Friend Module MySettingsProperty

    <Global.System.ComponentModel.Design.HelpKeywordAttribute("My.Settings")> _
    Friend ReadOnly Property Settings() As Global.VBHelloWorld.My.MySettings
        Get
            Return Global.VBHelloWorld.My.MySettings.Default
        End Get
    End Property
End Module
End Namespace
```

Settings.settings (在 *(root directory name)*\VBHelloWorld\My Project):

```
<?xml version='1.0' encoding='utf-8'?>
<SettingsFile xmlns="http://schemas.microsoft.com/VisualStudio/2004/01/settings"
    CurrentProfile="(Default)" UseMySettingsClassName="true">
    <Profiles>
        <Profile Name="(Default)" />
    </Profiles>
    <Settings />
</SettingsFile>
```

C# 和 .NET 核心

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

```
version: 0.2

phases:
  build:
    commands:
      - dotnet restore
      - dotnet build
artifacts:
  files:
    - .\bin\Debug\netcoreapp1.0\*
```

HelloWorldSample.csproj (在 *(root directory name)*)

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp1.0</TargetFramework>
  </PropertyGroup>
</Project>
```

Program.cs (在 *(root directory name)*)

```
using System;

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

基于 AWS CodeBuild 使用案例的示例

您可以使用这些基于使用案例的示例来试验 AWS CodeBuild：

名称	描述
Amazon ECR 示例 (p. 38)	使用 Amazon ECR 存储库中的 Docker 镜像，以使用 Apache Maven 生成单个 JAR 文件。
Docker 示例 (p. 41)	使用由支持 Docker 的 AWS CodeBuild 提供的构建映像来通过 Apache Maven 生成一个 Docker 映像。将 Docker 镜像推送到 Amazon ECR 中的存储库。您还可以调整此示例，以将 Docker 镜像推送到 Docker Hub。
GitHub Enterprise 示例 (p. 46)	将 AWS CodeBuild 和 GitHub Enterprise 一起用作源存储库，并安装了证书、启用了 Webhook，可在每次将代码更改被推送到存储库时重建源代码。
GitHub 拉取请求示例 (p. 51)	将 AWS CodeBuild 和 GitHub 一起用作源存储库并启用 Webhook，可在每次将代码更改被推送到存储库时重建源代码。
将 AWS Config 与 AWS CodeBuild 结合使用的示例 (p. 53)	说明如何设置 AWS Config。列出跟踪的 AWS CodeBuild 资源并描述如何在 AWS Config 中查找 AWS CodeBuild 项目。
构建徽章示例 (p. 54)	说明如何使用构建徽章设置 AWS CodeBuild。
构建通知示例 (p. 56)	使用 Apache Maven 生成单个 JAR 文件。给 Amazon SNS 主题的订阅者发送构建通知。
自定义映像示例中的 Docker (p. 73)	使用自定义 Docker 映像生成 Docker 映像。
AWS CodeDeploy 示例 (p. 75)	使用 Apache Maven 生成单个 JAR 文件。使用 AWS CodeDeploy 将 JAR 文件部署到 Amazon Linux 实例。您也可以使用 AWS CodePipeline 来构建和部署示例。
AWS Lambda 示例 (p. 78)	使用 AWS CodeBuild 以及 Lambda、AWS CloudFormation 和 AWS CodePipeline 来构建和部署遵循了 AWS 无服务器应用程序模型 (AWS SAM) 标准的无服务器应用程序。
Elastic Beanstalk 示例 (p. 79)	使用 Apache Maven 生成单个 WAR 文件。使用 Elastic Beanstalk 将 WAR 文件部署到 Elastic Beanstalk 实例。

适用于 AWS CodeBuild 的 Amazon ECR 示例

此示例使用 Amazon Elastic Container Registry (Amazon ECR) 映像存储库中的 Docker 映像构建适用于 AWS CodeBuild 的 [Go 示例 \(p. 87\)](#)。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS、CloudWatch Logs 和 Amazon ECR 相关的操作收取的

费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#)、[Amazon CloudWatch 定价](#) 和 [Amazon Elastic Container Registry 定价](#)。

运行示例

要运行此示例，请：

1. 要创建 Docker 映像并将其推送到 Amazon ECR 中的映像存储库，请完成 [Docker 示例 \(p. 41\)](#) 的“运行示例”部分的步骤。
 2. 要创建并上传要生成的源代码，请完成 [Go 示例 \(p. 87\)](#) 的“运行示例”部分中的步骤 1 到 4。
 3. 为 Amazon ECR 中的映像存储库分配权限，以便 AWS CodeBuild 可以将存储库中的 Docker 映像拉取到以下构建环境中：
1. 如果您通过 IAM 用户而不是 AWS 根账户或 IAM 管理员用户来使用 Amazon ECR，请向用户 (或与用户关联的 IAM 组) 添加语句 (在 `### BEGIN ADDING STATEMENT HERE ###` 和 `### END ADDING STATEMENT HERE ###` 之间)。(不建议使用 AWS 根账户。)通过此语句，可访问 Amazon ECR 存储库的管理权限。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (...)。请勿删除任何语句，也不要将这些省略号键入策略中。有关更多信息，请参阅 IAM 用户指南中的[通过 AWS 管理控制台 使用内联策略](#)。

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

Note

修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。

2. 在 <https://console.aws.amazon.com/ecs/> 上打开 Amazon ECS 控制台。
3. 选择 Repositories。
4. 在存储库名称列表中，选择您创建或选择的存储库的名称。
5. 选择 Permissions 选项卡，选择 Add，然后创建语句。
6. 对于 Sid，请键入标识符 (例如，**CodeBuildAccess**)。
7. 对于 Effect，请将 Allow 保持选中状态，因为您希望允许对 AWS CodeBuild 的访问。
8. 对于 Principal，请键入 **codebuild.amazonaws.com**。请将 Everybody 保持清除状态，因为您希望仅允许对 AWS CodeBuild 的访问。
9. 跳过 All IAM entities 列表。
10. 对于 Action，选择 Pull only actions。

将选择所有 pull-only 操作 (ecr:GetDownloadUrlForLayer、ecr:BatchGetImage 和 ecr:BatchCheckLayerAvailability)。

11. 选择 Save all。

此策略将显示在 Policy document 中。

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

4. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "name": "amazon-ecr-sample-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": "account-ID.dkr.ecr.region-ID.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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
6. 将 *GoOutputArtifact.zip* 文件下载到您的本地计算机或实例，然后提取 *GoOutputArtifact.zip* 文件的内容。在提取出来的内容中，获取 hello 文件。

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

适用于 AWS CodeBuild 的 Docker 示例

该示例会生成一个 Docker 映像作为构建输出，然后将该 Docker 映像推送到 Amazon Elastic Container Registry (Amazon ECR) 映像存储库。您可以调整该示例以将 Docker 映像推送到 Docker Hub。有关更多信息，请参阅 [调整示例以将映像推送到 Docker Hub \(p. 44\)](#)。

要了解如何改用自定义 Docker 构建映像来构建 Docker 映像 (Docker Hub 中的 `docker:dind`)，请参阅我们的 [自定义映像示例中的 Docker \(p. 73\)](#)。

此示例参考 `golang:1.9` 进行了测试

此示例使用新的多阶段 Docker 构建功能，该功能将生成一个 Docker 映像作为构建输出。然后，它将 Docker 映像推送到 Amazon ECR 映像存储库。多阶段 Docker 映像构建有助于减小最终 Docker 映像的大小。有关更多信息，请参阅 [将多阶段构建和 Docker 结合使用](#)。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS、CloudWatch Logs 和 Amazon ECR 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#)、[Amazon CloudWatch 定价](#) 和 [Amazon Elastic Container Registry 定价](#)。

主题

- [运行示例 \(p. 41\)](#)
- [目录结构 \(p. 43\)](#)
- [文件 \(p. 43\)](#)
- [调整示例以将映像推送到 Docker Hub \(p. 44\)](#)
- [相关资源 \(p. 46\)](#)

运行示例

要运行此示例，请：

1. 如果 Amazon ECR 中已存在要使用的映像存储库，请跳至第 3 步。如果您通过 IAM 用户而不是 AWS 根账户或 IAM 管理员用户来使用 Amazon ECR，请向用户 (或与用户关联的 IAM 组) 添加此语句 (在 `### BEGIN ADDING STATEMENT HERE ###` 和 `### END ADDING STATEMENT HERE ###` 之间)。(不建议使用 AWS 根账户。)此语句可创建用于存储 Docker 映像的 Amazon ECR 存储库。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (...)。请勿删除任何语句，也不要将这些省略号键入策略中。有关更多信息，请参阅 IAM 用户指南中的 [通过 AWS 管理控制台 使用内联策略](#)。

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

Note

修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。

2. 在 Amazon ECR 中创建映像存储库。请务必在您要创建构建环境并运行构建的同一 AWS 区域中创建存储库。有关更多信息，请参阅 Amazon ECR 用户指南中的[创建存储库](#)。该存储库的名称必须与您将在此过程的稍后部分指定的存储库名称相匹配，并以 `IMAGE_REPO_NAME` 环境变量的形式表示。
3. 将此语句 (在 `### BEGIN ADDING STATEMENT HERE ###` 和 `### END ADDING STATEMENT HERE ###` 之间) 添加到已附加到 AWS CodeBuild 服务角色的策略中。AWS CodeBuild 可利用此语句将 Docker 映像上传到 Amazon ECR 存储库。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (...)。请勿删除任何语句，也不要将这些省略号键入策略中。

```
{
  "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

修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。

4. 按照本主题的“目录结构”和“文件”中部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 (*root directory name*)，而只上传 (*root directory name*) 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 (*root directory name*) 添加到 ZIP 文件中，而只添加 (*root directory name*) 中的文件。

5. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 `create-project` 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 确认 AWS CodeBuild 已成功将 Docker 映像推送到存储库：

1. 在 <https://console.aws.amazon.com/ecs/> 上打开 Amazon ECS 控制台。
2. 选择 Repositories。
3. 选择存储库名称。映像应在 Images 选项卡上列出。

目录结构

此示例假定有这样一个目录结构。

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

文件

此示例将使用这些文件。

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

Note

如果您使用的是 17.06 版之前的 Docker 版本，请删除 --no-include-email 选项。

```
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 (在 *(root directory name)*)

```
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"]
```

调整示例以将映像推送到 Docker Hub

要将 Docker 映像推送到 Docker Hub 而非 Amazon ECR，请修改此示例的代码。

1. 将 `buildspec.yml` 文件中的这些特定于 Amazon ECR 的代码行替换为：

Note

如果您使用的是 17.06 版之前的 Docker 版本，请删除 `--no-include-email` 选项。

```
...
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
...
```

特定于 Docker Hub 的代码行。

```
...
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. 将修改后的代码上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 (*root directory name*)，而只上传 (*root directory name*) 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 (*root directory name*) 添加到 ZIP 文件中，而只添加 (*root directory name*) 中的文件。

3. 将 create-project 命令的 JSON 格式输入中的这些代码行替换为：

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

利用这些代码行。

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

4. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建环境、运行构建并查看相关构建信息。

5. 确认 AWS CodeBuild 已成功将 Docker 映像推送到存储库。登录 Docker Hub，再转至存储库，然后选择 Tags 选项卡。latest 标签应包含最新的 Last Updated 值。

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

适用于 AWS CodeBuild 的 GitHub Enterprise 示例

AWS CodeBuild 现已支持将 GitHub Enterprise 作为源存储库。此示例介绍了您的 GitHub Enterprise 存储库安装证书后，如何设置您的 AWS CodeBuild 项目。还解释了如何启用 Webhook，这样在每次代码更改推送到您的私有 GitHub Enterprise 存储库后，AWS CodeBuild 都可以重建源代码。

先决条件

1. 生成个人访问令牌 (该令牌会输入您的 AWS CodeBuild 项目)。我们建议您创建一个新的 GitHub Enterprise 用户，并为该用户生成个人访问令牌。将它复制到您的剪贴板，以便在创建 AWS CodeBuild 项目时使用。有关更多信息，请参阅 GitHub 帮助网站上的 [在 GitHub Enterprise 中创建个人访问令牌](#)。

在创建个人访问令牌时，请在定义中包含存储库范围。

Select scopes
Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories

2. 从 GitHub Enterprise 下载您的证书。AWS CodeBuild 使用此证书与存储库建立可信 SSL 连接。

Linux/macOS 客户端：

从您的 终端窗口中运行以下命令：

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

将命令中的占位符替换为以下值：

HOST。您的 GitHub Enterprise 存储库的 IP 地址。

PORTNUMBER。用于连接的端口号 (例如，443)。

folder。下载证书的文件夹。

filename。证书文件的文件名。

Important

将证书另存为 .pem 文件。

Windows 客户端:

使用浏览器从 GitHub Enterprise 下载您的证书。要查看站点的证书详细信息，请选择挂锁图标。有关如何导出证书的信息，请参阅浏览器文档。

Important

将证书另存为 .pem 文件。

3. 将您的证书文件上传到 Amazon S3 存储桶。有关如何创建 Amazon S3 存储桶的信息，请参阅[如何创建 Amazon S3 存储桶？](#) 有关如何将对象上传到 Amazon S3 存储桶的信息，请参阅[如何将文件和文件夹上传至存储桶？](#)

Note

此存储桶必须与您的构建项目处在同一个 AWS 区域中。例如，如果您指示 AWS CodeBuild 在美国东部 (俄亥俄) 区域运行构建任务，则存储桶也必须位于美国东部 (俄亥俄) 区域中。

创建将 GitHub Enterprise 作为源存储库的构建项目，并启用 Webhook (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 如果显示欢迎页面，请选择 Get started (开始使用)。如果未显示欢迎页面，则在导航窗格中选择 Build projects，然后选择 Create project。
3. 在 Configure your project 页面上，对于 Project name，键入此构建项目的名称。构建项目名称在您的各个 AWS 账户内必须是唯一的。
4. 在 Source: What to build 中，对于 Source provider，选择 GitHub Enterprise。
 - 对于 Personal Access Token，粘贴您复制到剪贴板的令牌，然后选择 Save Token。在 Repository URL 中输入您的 GitHub Enterprise 存储库的 URL。

Note

您只需输入并保存一次个人访问令牌。未来所有 AWS CodeBuild 项目均会使用此令牌。

- 选择 Webhook，每次将代码更改推送到此存储库时都会重新构建。
- 选择 Insecure SSL，在连接到您的 GitHub Enterprise 项目存储库时忽略 SSL 警告。

Note

建议您仅将 Insecure SSL 用于测试。它不应在生产环境中使用。

Source: What to build

Source provider*

Repository URL
[Delete Personal Access Token](#)
https://<host-name>/<user-name>/<repository-name>

Git clone depth

Webhook ☒ Rebuild every time a code change is pushed to this repository

Insecure SSL ☐ Enable this flag to ignore SSL warnings while connecting to project source

Build Badge ☐ Enable this flag to make your project's build status visible and embeddable.

5. 在 Environment: How to build 中：

对于 Environment image，执行下列操作之一：

- 要使用由 AWS CodeBuild 托管的 Docker 映像，请选择 Use an image managed by AWS CodeBuild，然后从 Operating system、Runtime 和 Version 中进行相应选择。
- 要使用其他 Docker 映像，请选择 Specify a Docker image。对于 Custom image type，选择 Other 或 Amazon ECR。如果您选择了 Other，那么对于 Custom image ID，请在 Docker Hub 中以格式 `repository-name/image-name: image-tag` 键入 Docker 映像的名称和标签。如果您选择 Amazon ECR，请使用 Amazon ECR repository 和 Amazon ECR image 在您的 AWS 账户中选择 Docker 映像。

对于 Build specification，执行下列操作之一：

- 使用源代码根目录中的 buildspec.yml 文件。
- 通过插入构建命令来覆盖构建规范。

有关更多信息，请参见 [构建规范参考 \(p. 107\)](#)。

对于 Certificate，选择 Install certificate from your S3。对于 Bucket of certificate，选择存储您的 SSL 证书的 S3 存储桶。对于 Object key of certificate，键入您的 S3 对象键的名称。

6. 在 Artifacts: Where to put the artifacts from this build project 中，对于 Artifacts type，执行下列操作之一：

- 如果您不想创建任何构建输出项目，请选择 No artifacts。
- 要将构建输出存储在 Amazon S3 存储桶中，请选择 Amazon S3，然后执行以下操作：

- 如果您要将项目名称用于构建输出 ZIP 文件或文件夹，请将 Artifacts name 留空。否则，请在 Artifacts name 框中键入名称。默认情况下，构件名称是项目名称。如果您要指定其他名称，请在构件名称框中键入该名称。如果您要输出 ZIP 文件，请包含 zip 扩展名。
- 对于 Bucket name，请选择输出存储桶的名称。
- 如果您在此过程的前面部分选择了 Insert build commands，那么对于 Output files，请键入您要放到构建输出 ZIP 文件或文件夹的构建中的文件位置。对于多个位置，使用逗号分开各个位置 (appspec.yml, target/my-app.jar)。有关更多信息，请参阅 [构建规范语法 \(p. 107\)](#) 中 files 的描述。

7. 在 Cache 中，执行下列操作之一：

- 如果您不想使用缓存，请选择 No cache。
- 要使用缓存，请选择 Amazon S3，然后执行以下操作：
 - 对于 Bucket，选择存储缓存的 Amazon S3 存储桶的名称。
 - (可选) 对于 Path prefix，键入一个 Amazon S3 路径前缀。Path prefix 值类似于支持您将缓存存储在存储桶内同一目录下的目录名称。

Important

请不要在 Path prefix 的末尾附加“/”。

使用缓存可节省大量构建时间，因为构建环境的可重用部分被存储在缓存中，并且可跨构建使用。

8. 在 Service role 中，执行下列操作之一：

- 如果您没有 AWS CodeBuild 服务角色，请选择 Create a service role in your account。在 Role name 中，接受默认名称或键入您自己的名称。
- 如果您有 AWS CodeBuild 服务角色，请选择 Choose an service existing role from your account。在 Role name 中，选择服务角色。

Note

当您使用控制台来创建或更新构建项目时，您可以同时创建 AWS CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

9. 在 VPC 中，执行下列操作之一：

- 如果您没有将 VPC 用于项目，请选择 No VPC。
- 如果您要将 AWS CodeBuild 与您的 VPC 结合使用，请执行以下操作：
 - 对于 VPC，选择 AWS CodeBuild 使用的 VPC ID。
 - 对于 Subnets，选择包含 AWS CodeBuild 使用的资源的子网。
 - 对于 Security Groups，选择 AWS CodeBuild 用来支持对 VPC 中资源的访问的安全组。

有关更多信息，请参阅 [将 AWS CodeBuild 与 Amazon Virtual Private Cloud 结合使用 \(p. 125\)](#)。

10. 展开 Show advanced settings，然后根据情况设置它们。

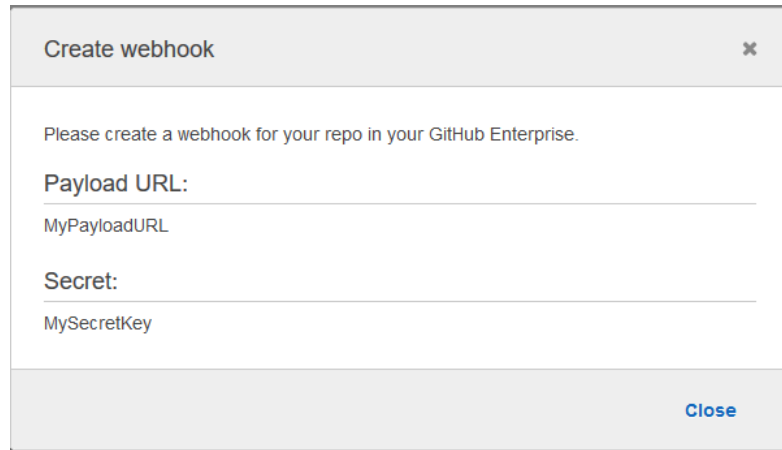
11. 选择 Continue (继续)。在 Review 页面上，选择 Save and build；或者要稍后运行构建，请选择 Save。

12. 如果您在 Source: What to Build 中启用了 Webhook，将出现 Create webhook 对话框，其中显示 Payload URL 和 Secret 的值。

Important

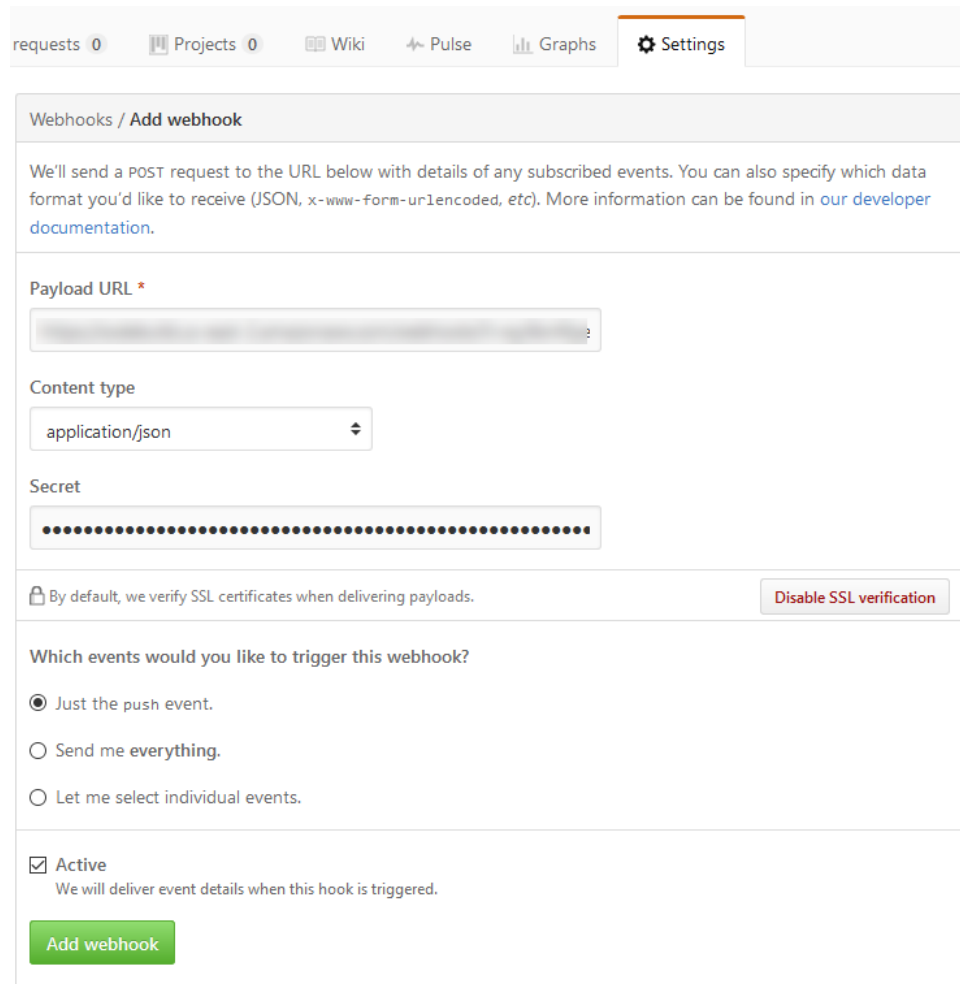
Create webhook 对话框只出现一次。请复制负载 URL 和私有密钥。在 GitHub Enterprise 中添加 Webhook 时会用到它们。API 版本 2016-10-06

如果您需要再次生成负载 URL 和私有密钥，必须首先删除 GitHub Enterprise 存储库中的 Webhook。在您的 AWS CodeBuild 项目中，清除 Webhook 复选框，然后选择 Save。然后您可以在创建或更新 AWS CodeBuild 项目时选中 Webhook 复选框。Create webhook 对话框将再次出现。



A dialog box titled "Create webhook" with a close button (X) in the top right corner. The text inside says "Please create a webhook for your repo in your GitHub Enterprise." Below this, there are two input fields. The first is labeled "Payload URL:" and contains the text "MyPayloadURL". The second is labeled "Secret:" and contains the text "MySecretKey". At the bottom right, there is a blue button labeled "Close".

13. 在 GitHub Enterprise 中，选择存储您的 AWS CodeBuild 项目的存储库，然后依次选择 Settings、Hooks & services 和 Add webhook。输入负载 URL 和私有密钥，接受其他字段的默认值，然后选择 Add webhook。



A screenshot of the GitHub Enterprise "Add webhook" page. At the top, there is a navigation bar with tabs for "requests 0", "Projects 0", "Wiki", "Pulse", "Graphs", and "Settings" (which is selected). Below the navigation bar, the page title is "Webhooks / Add webhook". The main content area contains the following elements: a paragraph explaining that a POST request will be sent to the URL below; a "Payload URL *" input field; a "Content type" dropdown menu set to "application/json"; a "Secret" input field with a masked password; a checkbox for "By default, we verify SSL certificates when delivering payloads." with a "Disable SSL verification" button; a section titled "Which events would you like to trigger this webhook?" with three radio button options: "Just the push event." (selected), "Send me everything.", and "Let me select individual events."; a checkbox for "Active" with the text "We will deliver event details when this hook is triggered."; and a green "Add webhook" button at the bottom.

14. 返回您的 AWS CodeBuild 项目。关闭 Create webhook 对话框，然后选择 Start build。

AWS CodeBuild 的 GitHub 拉取请求

当源存储库为 GitHub 时，AWS CodeBuild 现在支持 webhook。这意味着，对于将源代码存储在私有 GitHub 存储库中的 AWS CodeBuild 构建项目，webhooks 使 AWS CodeBuild 能够在每次代码更改发布到该私有存储库时开始自动重新构建源代码。

创建将 GitHub 作为源存储库的构建项目并启用 Webhooks (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 如果显示欢迎页面，请选择 Get started (开始使用)。如果未显示欢迎页面，则在导航窗格中选择 Build projects，然后选择 Create project。
3. 在 Configure your project 页面上，对于 Project name，键入此构建项目的名称。构建项目名称在您的各个 AWS 账户内必须是唯一的。
4. 在 Source: What to build 中，对于 Source provider，选择 GitHub。按照说明与 GitHub 连接 (或重新连接) 并选择 Authorize。

对于 Webhook，请选中 Rebuild every time a code change is pushed to this repository 复选框。仅当您已选中 Repository 下的 Use a repository in my account 时才选中此复选框。

Source provider* GitHub

Repository* ☐ Use a public repository ☒ Use a repository in my account

Choose a repository

[Disconnect from GitHub](#)

Webhook ☒ Rebuild every time a code change is pushed to this repository

5. 在 Environment: How to build 中：

对于 Environment image，执行下列操作之一：

- 要使用由 AWS CodeBuild 托管的 Docker 映像，请选择 Use an image managed by AWS CodeBuild，然后从 Operating system、Runtime 和 Version 中进行相应选择。
- 要使用其他 Docker 映像，请选择 Specify a Docker image。对于 Custom image type，选择 Other 或 Amazon ECR。如果您选择了 Other，那么对于 Custom image ID，请在 Docker Hub 中以格式 `repository-name/image-name: image-tag` 键入 Docker 映像的名称和标签。如果您选择 Amazon ECR，请使用 Amazon ECR repository 和 Amazon ECR image 在您的 AWS 账户中选择 Docker 映像。

对于 Build specification，执行下列操作之一：

- 使用源代码根目录中的 buildspec.yml 文件。
- 通过插入构建命令来覆盖构建规范。

有关更多信息，请参见 [构建规范参考 \(p. 107\)](#)。

6. 在 Artifacts: Where to put the artifacts from this build project 中，对于 Artifacts type，执行下列操作之一：
 - 如果您不想创建任何构建输出项目，请选择 No artifacts。
 - 要将构建输出存储在 Amazon S3 存储桶中，请选择 Amazon S3，然后执行以下操作：
 - 如果您要将项目名称用于构建输出 ZIP 文件或文件夹，请将 Artifacts name 留空。否则，请在 Artifacts name 框中键入名称。(默认情况下，构件名称是项目名称。)如果您要指定其他名称，请在构件名称框中键入该名称。如果您要输出 ZIP 文件，请包含 zip 扩展名。
 - 对于 Bucket name，请选择输出存储桶的名称。
 - 如果您在此过程的前面部分选择了 Insert build commands，那么对于 Output files，请键入您要放到构建输出 ZIP 文件或文件夹的构建中的文件位置。对于多个位置，使用逗号分开各个位置 (appspec.yml, target/my-app.jar)。有关更多信息，请参阅 [构建规范语法 \(p. 107\)](#) 中 files 的描述。
7. 在 Service role 中，执行下列操作之一：
 - 如果您没有 AWS CodeBuild 服务角色，请选择 Create a service role in your account。在 Role name 中，接受默认名称或键入您自己的名称。
 - 如果您有 AWS CodeBuild 服务角色，请选择 Choose an service existing role from your account。在 Role name 中，选择服务角色。

Note

当您使用控制台来创建或更新构建项目时，您可以同时创建 AWS CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

8. 展开 Show advanced settings，然后根据情况设置它们。
9. 选择 Continue (继续)。在 Review 页面上，选择 Save and build；或者，要稍后运行构建，请选择 Save。

验证检查

1. 在 AWS CodeBuild project 页面上，执行以下操作：
 - 选择 Project Details，然后选择 Webhook URL 链接。在您的 GitHub 存储库中，在 Settings 页面上的 Webhooks 下，确认已选中 Pull Request 和 Push。
2. 在 GitHub 中的 Accounts、Settings 和 Authorized OAuth Apps 下，您应该会看到已获得授权的 AWS CodeBuild 区域。

You have granted 1 application access to your account.

Sort ▼

Revoke all



AWS CodeBuild us-east-1

Last used within the last week · Owned by codebuild-oauth-prod

Revoke

将 AWS Config 与 AWS CodeBuild 结合使用的示例

AWS Config 提供了您的 AWS 资源清单以及这些资源的配置更改历史记录。AWS Config 现在支持 AWS CodeBuild 作为 AWS 资源，这表示该服务可以跟踪您的 AWS CodeBuild 项目。有关 AWS Config 的更多信息，请参阅[什么是 AWS Config ?](#) (位于 AWS Config 开发人员指南 中)。

您可以在 AWS Config 控制台中的 Resource Inventory 页面上查看有关 AWS CodeBuild 资源的以下信息：

- 您的 AWS CodeBuild 配置更改的时间线。
- 每个 AWS CodeBuild 项目的配置详细信息。
- 与其他 AWS 资源的关系。
- 您的 AWS CodeBuild 项目的更改列表。

本主题中的过程展示了如何设置 AWS Config 以及如何查找和查看 AWS CodeBuild 项目。

主题

- [先决条件](#) (p. 53)
- [设置 AWS Config](#) (p. 53)
- [查找 AWS CodeBuild 项目](#) (p. 53)
- [在 AWS Config 控制台中查看 AWS CodeBuild 配置详细信息](#) (p. 54)

先决条件

创建您的 AWS CodeBuild 项目。有关更多信息，请参阅[创建构建项目](#) (p. 154)。

设置 AWS Config

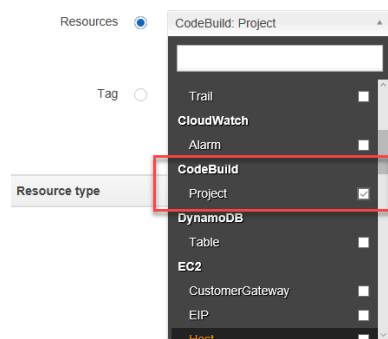
- [设置 AWS Config \(控制台\)](#)
- [设置 AWS Config \(AWS CLI\)](#)

Note

用户可能需要长达 10 分钟才能在 AWS Config 控制台中看到 AWS CodeBuild 项目。

查找 AWS CodeBuild 项目

1. 登录到 AWS 管理控制台，然后通过以下网址打开 AWS Config 控制台：<https://console.aws.amazon.com/config>。
2. 在 Resource inventory 页面上，选择 Resources。向下滚动并选中 CodeBuild project 复选框。



3. 选择 Look up。
4. 在添加 AWS CodeBuild 项目的列表之后，选择 Config timeline 列中的 AWS CodeBuild 项目名称链接。

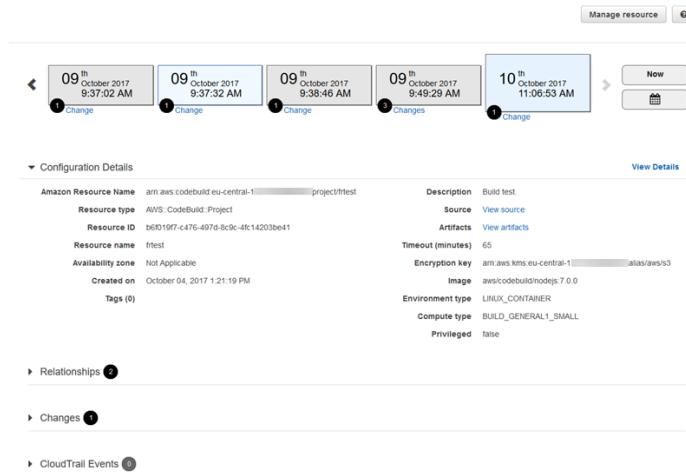
在 AWS Config 控制台中查看 AWS CodeBuild 配置详细信息

当您在 Resource inventory 页面上查找资源时，可选择 AWS Config 时间线以查看有关您的 AWS CodeBuild 项目的详细信息。资源的详细信息页面提供了有关该资源的配置、关系和更改次数的信息。

页面顶部的块统称为时间线。时间线显示了记录的创建日期和时间。

有关更多信息，请参阅 AWS Config 开发人员指南 中的 [在 AWS Config 控制台中查看配置详细信息](#)。

AWS Config 中 AWS CodeBuild 项目的示例：



使用 AWS CodeBuild 构建徽章示例

AWS CodeBuild 现在支持使用构建徽章，该徽章提供一个动态生成的可嵌入映像 (徽章)，用以显示项目的最新构建状态。可通过为您的 AWS CodeBuild 项目生成的公开可用的 URL 访问此映像。这将允许任何人查看 AWS CodeBuild 项目的状态。构建徽章不包含任何安全信息，因此它们无需身份验证。

创建已启用构建徽章的构建项目 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 如果显示欢迎页面，请选择 Get started (开始使用)。如果未显示欢迎页面，则在导航窗格中选择 Build projects，然后选择 Create project。
3. 在 Configure your project 页面上，对于 Project name，键入此构建项目的名称。构建项目名称在您的各个 AWS 账户内必须是唯一的。
4. 在 Source: What to build 中，对于 Source provider，选择源提供商类型，然后执行以下操作之一：
 - 如果您选择了 Amazon S3，那么对于 Bucket，请选择包含源代码的输入存储桶的名称。对于 S3 object key，请键入包含源代码的 ZIP 文件的名称。
 - 如果您选择了 AWS CodeCommit，那么对于 Repository，请选择存储库的名称。选中 Build Badge 复选框以使您的项目的构建状态可见且可嵌入。
 - 如果您选择了 GitHub，请按照说明与 GitHub 连接 (或重新连接)。在 GitHub Authorize application 页面上，对于 Organization access，选择您希望 AWS CodeBuild 能够访问的每个存储库旁边的 Request access。选择 Authorize application 后，返回 AWS CodeBuild 控制台，对于 Repository，选择包含源代码的存储库的名称。选中 Build Badge 复选框以使您的项目的构建状态可见且可嵌入。

- 如果您选择了 Bitbucket，请按照说明与 Bitbucket 连接 (或重新连接)。在 Bitbucket Confirm access to your account 页面上，对于 Organization access，选择 Grant access。选择 Grant access 后，返回 AWS CodeBuild 控制台，对于 Repository，选择包含源代码的存储库的名称。选中 Build Badge 复选框以使您的项目的构建状态可见且可嵌入。

Important

如果您更新项目源，这可能会影响项目的构建徽章的准确性。

5. 在 Environment: How to build 中：

对于 Environment image，执行下列操作之一：

- 要使用由 AWS CodeBuild 托管的 Docker 映像，请选择 Use an image managed by AWS CodeBuild，然后从 Operating system、Runtime 和 Version 中进行相应选择。
- 要使用其他 Docker 映像，请选择 Specify a Docker image。对于 Custom image type，选择 Other 或 Amazon ECR。如果您选择了 Other，那么对于 Custom image ID，请在 Docker Hub 中以格式 `repository-name/image-name:image-tag` 键入 Docker 映像的名称和标签。如果您选择 Amazon ECR，请使用 Amazon ECR repository 和 Amazon ECR image 在您的 AWS 账户中选择 Docker 映像。

对于 Build specification，执行下列操作之一：

- 使用源代码根目录中的 buildspec.yml 文件。
- 通过插入构建命令来覆盖构建规范。

有关更多信息，请参见 [构建规范参考 \(p. 107\)](#)。

6. 在 Artifacts: Where to put the artifacts from this build project 中，对于 Artifacts type，执行下列操作之一：

- 如果您不想创建任何构建输出项目，请选择 No artifacts。
- 要将构建输出存储在 Amazon S3 存储桶中，请选择 Amazon S3，然后执行以下操作：
 - 如果您要将项目名称用于构建输出 ZIP 文件或文件夹，请将 Artifacts name 留空。否则，请在 Artifacts name 框中键入名称。(默认情况下，构件名称是项目名称。)如果您要指定其他名称，请在构件名称框中键入该名称。如果您要输出 ZIP 文件，请包含 zip 扩展名。
 - 对于 Bucket name，请选择输出存储桶的名称。
 - 如果您在此过程的前面部分选择了 Insert build commands，那么对于 Output files，请键入您要放到构建输出 ZIP 文件或文件夹的构建中的文件位置。对于多个位置，使用逗号分开各个位置 (appspect.yml, target/my-app.jar)。有关更多信息，请参阅 [构建规范语法 \(p. 107\)](#) 中 files 的描述。

7. 在 Service role 中，执行下列操作之一：

- 如果您没有 AWS CodeBuild 服务角色，请选择 Create a service role in your account。在 Role name 中，接受默认名称或键入您自己的名称。
- 如果您有 AWS CodeBuild 服务角色，请选择 Choose an service existing role from your account。在 Role name 中，选择服务角色。

Note

当您使用控制台来创建或更新构建项目时，您可以同时创建 AWS CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

8. 展开 Show advanced settings 并根据需要设置其他高级设置。

9. 选择 Continue (继续)。在 Review 页面上，选择 Save and build；或者要稍后运行构建，请选择 Save。

创建已启用构建徽章的构建项目 (CLI)

有关创建构建项目的信息，请参阅[创建构建项目 \(AWS CLI\)](#) (p. 162)。要在您的 AWS CodeBuild 项目中包含构建徽章，您必须指定值为 true 的 `badgeEnabled`。

访问您的 AWS CodeBuild 构建徽章

您可以使用 AWS CodeBuild 控制台或 AWS CLI 访问构建徽章。

- 在 AWS CodeBuild 控制台中，在构建项目列表中的 Project 列中，选择与构建项目对应的链接。在 Build project: **project-name** 页面上，展开 Project details。构建徽章 URL 将显示在 Advanced 下方。有关更多信息，请参阅[查看构建项目的详细信息 \(控制台\)](#) (p. 170)。
- 在 AWS CLI 中，运行 `batch-get-projects` 命令。构建徽章 URL 包含在输出的项目环境详细信息部分中。有关更多信息，请参阅[查看构建项目的详细信息 \(AWS CLI\)](#) (p. 170)。

Important

给定构建徽章请求 URL 适用于主分支，但您可以指定源存储库中已运行构建的任何分支。

发布您的 AWS CodeBuild 构建徽章

您可以将构建徽章请求 URL 包含在您的首选存储库 (例如，GitHub 或 AWS CodeCommit) 中的 markdown 文件中以显示最新构建的状态。

示例 markdown 代码：

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

AWS CodeBuild 徽章状态

- PASSING 给定分支上的最新构建已传递。
- FAILING 给定分支上的最新构建已超时、失败、出现故障或停止。
- IN_PROGRESS 给定分支上的最新构建正在进行中。
- UNKNOWN 项目尚未为给定分支运行构建或根本未运行。此外，构建徽章功能可能已禁用。

适用于 AWS CodeBuild 的构建通知示例

Amazon CloudWatch Events 具有对 AWS CodeBuild 的内置支持。CloudWatch Events 是描述您的 AWS 资源中的变化的系统事件流。利用 CloudWatch Events，您可以写入声明性规则以将相关事件与要执行的自动操作关联。每当构建成功、失败、从一个构建阶段转到另一个构建阶段或出现这些事件的任意组合时，本示例都会使用 Amazon CloudWatch Events 和 Amazon Simple Notification Service (Amazon SNS) 向订阅者发送构建通知。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon CloudWatch 和 Amazon SNS 相关的操作收取的费用。有关更多信息，请参阅[AWS CodeBuild 定价](#)、[Amazon CloudWatch 定价](#)和[Amazon SNS 定价](#)。

运行示例

要运行此示例，请：

1. 如果您已在 Amazon SNS 中设置并订阅用于此示例的主题，请跳至第 4 步。如果您通过 IAM 用户而不是 AWS 根账户或 IAM 管理员用户来使用 Amazon SNS，请向用户 (或与用户关联的 IAM 组) 添加以下语句 (在 **### BEGIN ADDING STATEMENT HERE ###** 和 **### END ADDING STATEMENT HERE ###** 之间)。(不建议使用 AWS 根账户。)此语句可用于查看、创建、订阅和测试向 Amazon SNS 中的主题发送通知的情况。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (...)。请勿删除任何语句，也不要将这些省略号键入现有策略中。

```
{
  "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

修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。

有关更多信息，请参阅[编辑客户托管策略](#)或 IAM 用户指南的[使用内联策略 \(控制台\)](#)中的“编辑或删除组、用户或角色的内联策略”部分。

2. 在 Amazon SNS 中创建或标识主题。AWS CodeBuild 将使用 CloudWatch Events 通过 Amazon SNS 向该主题发送构建通知。要创建主题，请执行以下操作：
 1. 打开 Amazon SNS 控制台：<https://console.aws.amazon.com/sns>。
 2. 选择 Create topic。
 3. 在 Create new topic 对话框中，对于 Topic name，键入该主题的名称，例如 **CodeBuildDemoTopic**。(如果您选择了其他名称，请用该名称替换掉本示例中对应的名称。)
 4. 选择 Create topic。
 5. 在 Topic details: CodeBuildDemoTopic 页面上，复制 Topic ARN 值，如以下屏幕截图所示。在下一个步骤中，您需要用到此值。

Topic details

Publish to topic

Topic ARN

Topic owner

Region

Display name

有关更多信息，请参阅 Amazon SNS 开发人员指南中的[创建主题](#)。

3. 为一个或多个收件人订阅主题以接收电子邮件通知。为收件人订阅主题：
 1. 使用上一步中打开的 Amazon SNS 控制台，在导航窗格中，选择 Subscriptions，然后选择 Create subscription。
 2. 在 Create subscription 对话框中，对于 Topic ARN，粘贴您在上一步中复制的主题 ARN。
 3. 对于 Protocol，选择 Email。
 4. 对于终端节点，请键入收件人的完整电子邮件地址。将您的结果与以下屏幕截图进行比较。

Create subscrip

Topic A

Proto

Endpo

5. 选择 Create Subscription。

6. Amazon SNS 向收件人发送订阅确认电子邮件。要开始接收电子邮件通知，收件人必须在订阅确认电子邮件中选择 Confirm subscription 链接。在收件人单击该链接后，如果成功订阅，Amazon SNS 将在收件人的 Web 浏览器中显示一条确认消息。

有关更多信息，请参阅 Amazon SNS 开发人员指南中的 [订阅主题](#)。

4. 如果您通过 IAM 用户而不是 AWS 根账户或 IAM 管理员用户来使用 CloudWatch Events，请向用户 (或与用户关联的 IAM 组) 添加以下语句 (在 `### BEGIN ADDING STATEMENT HERE ###` 和 `### END ADDING STATEMENT HERE ###` 之间)。(不建议使用 AWS 根账户。)此语句使您可以使用 CloudWatch Events。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (...)。请勿删除任何语句，也不要将这些省略号键入现有策略中。

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

Note

修改该策略的 IAM 实体必须拥有在 IAM 中修改策略的权限。

有关更多信息，请参阅[编辑客户托管策略](#)或 IAM 用户指南的[使用内联策略 \(控制台\)](#)中的“编辑或删除组、用户或角色的内联策略”部分。

5. 在 CloudWatch Events 中创建规则。为此，请通过以下网址打开 CloudWatch 控制台：<https://console.aws.amazon.com/cloudwatch>。
6. 在导航窗格中的 Events 下，选择 Rules，然后选择 Create rule。
7. 在 Step 1: Create rule page 上，以下项应该已被选定：Event Pattern 和 Build event pattern to match events by service。
8. 对于 Service Name，选择 CodeBuild。对于 Event Type，All Events 应该已被选定。
9. Event Pattern Preview 应显示以下代码。

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

将您迄今为止的结果与以下屏幕截图进行比较：

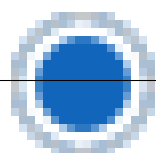
Step 1: Create

Create rules to invoke T

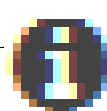
Event Source

Build or customize an E

Schedule to invoke Targ



Event Pattern



10. 通过选择 Edit，将 Event Pattern Preview 中的代码替换为以下两个规则模式之一。

每当一个构建开始或完成时，第一个规则模式就会为 AWS CodeBuild 中的指定构建项目触发一个事件。

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

在前面的规则中，根据需要更改以下代码。

- 要在每次构建开始或完成时触发事件，请保留 build-status 阵列中显示的所有值，或删除整个 build-status 阵列。
- 要仅在构建完成时触发事件，请从 build-status 阵列中删除 IN_PROGRESS。
- 要仅在构建开始时触发事件，请从 build-status 阵列中删除除 IN_PROGRESS 以外的所有值。
- 要为所有构建项目触发事件，请删除整个 project-name 阵列。
- 要仅为单个构建项目触发事件，请在 project-name 阵列中指定每个构建项目的名称。

每当构建从一个构建阶段转到另一个构建阶段时，第二个规则模式将为 AWS CodeBuild 中的指定构建项目触发一个事件。

```
{
  "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"
  ]
}
```

在前面的规则中，根据需要更改以下代码。

- 要针对所有构建阶段更改触发事件 (这可以为每个构建发送最多 9 条通知)，请保留 `completed-phase` 数组中显示的所有值，或删除整个 `completed-phase` 阵列。
- 要仅针对单个构建阶段更改触发事件，请删除 `completed-phase` 阵列中您不希望为其触发事件的每个构建阶段的名称。
- 要针对所有构建阶段状态更改触发事件，请保留 `completed-phase-status` 阵列中显示的所有值，或删除整个 `completed-phase-status` 阵列。
- 要仅针对单个构建阶段状态更改触发事件，请删除 `completed-phase-status` 阵列中您不希望对其触发事件的每个构建阶段状态的名称。
- 要为所有构建项目触发事件，请删除 `project-name` 阵列。
- 要为单个构建项目触发事件，请在 `project-name` 阵列中指定每个构建项目的名称。

Note

如果要同时为构建状态更改和构建阶段更改触发事件，则必须创建两个单独的规则，一个针对构建状态更改，另一个针对构建阶段更改。如果您尝试将两个规则合并为一个规则，则合并后的规则可能产生意外结果或停止协作。

替换完代码后，选择 **Save**。

11. 对于 **Targets**，选择 **Add target**。
12. 在目标列表中，选择 **SNS 主题**。
13. 对于 **Topic**，选择您之前标识或创建的主题。
14. 展开配置输入，然后选择输入转换器。
15. 在 **Input Path** 框中，键入以下输入路径之一。

对于 `detail-type` 值为 **CodeBuild Build State Change** 的规则，键入以下内容。

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

对于 `detail-type` 值为 **CodeBuild Build Phase Change** 的规则，键入以下内容。

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

Note

要获取其他类型的信息，请参阅[构建通知输入格式参考](#) (p. 69)。

16. 在 **Input Template** 框中，键入以下输入模板。

API 版本 2016-10-06

对于 detail-type 值为 CodeBuild Build State Change 的规则，键入以下内容。

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

对于 detail-type 值为 CodeBuild Build Phase Change 的规则，键入以下内容。

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

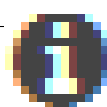
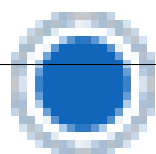
将您迄今为止的结果与以下屏幕截图进行比较，该截图显示了一条 detail-type 值为 CodeBuild Build State Change 的规则：

Step 1: Create

Create rules to invoke T

Event Source

Build or customize an E
Schedule to invoke Targ



17. 选择 Configure details。
18. 在 Step 2: Configure rule details 页面上，键入一个 Name 和可选的 Description。对于 State，将 Enabled 框保持选中状态。

将您迄今为止的结果与以下屏幕截图进行比较：

Step 2: Con

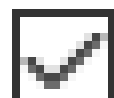
Rule definition

Name*

CodeB

Description

State



Ena

19. 选择 Create rule。
20. 按照[直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤 (举例) 创建构建项目、运行构建并查看构建信息。
21. 确认 AWS CodeBuild 立即成功发送构建通知。例如，检查在您的收件箱中现在是否具有构建通知电子邮件。

要更改规则的行为，请在 CloudWatch 控制台中选择要更改的规则，选择 Actions，然后选择 Edit。对规则进行更改，然后选择 Configure details，接下来选择 Update rule。

要停止使用规则发送构建通知，请在 CloudWatch 控制台中选择要停止使用的规则，选择 Actions，然后选择 Disable。

要删除整个规则，请在 CloudWatch 控制台中选择要删除的规则，选择 Actions，然后选择 Delete。

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

构建通知输入格式参考

CloudWatch 以 JSON 格式发送通知。

构建状态更改通知使用以下格式：

```
{
  "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"
}
}

```

构建阶段更改通知使用以下格式：

```

{
  "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"
}
}
```

适用于 AWS CodeBuild 的自定义映像示例中的 Docker

此示例通过使用 AWS CodeBuild 和自定义 Docker 构建映像 (Docker Hub 中的 `docker:dind`) 来构建和运行 Docker 映像。

要了解如何改用由支持 Docker 的 AWS CodeBuild 提供的构建映像来构建 Docker 映像，请参阅我们的 [Docker 示例 \(p. 41\)](#)。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

主题

- [运行示例 \(p. 73\)](#)
- [目录结构 \(p. 74\)](#)
- [文件 \(p. 74\)](#)
- [相关资源 \(p. 75\)](#)

运行示例

要运行此示例，请：

1. 按照本主题的“目录结构”和“文件”中部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要查看构建结果，请在构建的日志中查找字符串 Hello, World!。有关更多信息，请参阅 [查看构建详细信息 \(p. 190\)](#)。

目录结构

此示例假定有这样一个目录结构。

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

文件

在此示例中使用的操作系统的基本映像是 Ubuntu。此示例将使用这些文件。

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

```
version: 0.2

phases:
  install:
    commands:
      - nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --
host=tcp://127.0.0.1:2375 --storage-driver=overlay&
      - timeout 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!"
```

Note

如果基本操作系统是 Alpine Linux，请在 `buildspec.yml` 中向 `timeout` 添加 `-t` 参数：

```
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

Dockerfile (在 *(root directory name)*)

```
FROM maven:3.3.9-jdk-8

RUN echo "Hello World"
```

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

适用于 AWS CodeBuild 的 AWS CodeDeploy 示例

此示例指示 AWS CodeBuild 使用 Maven 来生成一个名为 `my-app-1.0-SNAPSHOT.jar` 的 JAR 文件作为构建输出。然后，此示例使用 AWS CodeDeploy 将 JAR 文件部署到 Amazon Linux 实例。(或者，您可以使用 AWS CodePipeline 来自动化使用 AWS CodeDeploy 以将 JAR 文件部署到 Amazon Linux 实例。)该示例以 Apache Maven 网站上的 [5 分钟学会 Maven](#) 主题为基础。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS、CloudWatch Logs 和 Amazon EC2 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#)、[Amazon CloudWatch 定价](#) 和 [Amazon EC2 定价](#)。

运行示例

要运行此示例，请：

- 下载并安装 Maven。有关更多信息，请参阅 Apache Maven 网站上的 [下载 Apache Maven](#) 和 [安装 Apache Maven](#)。
- 切换到您的本地计算机或实例上的空目录，然后运行此 Maven 命令。

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

如果成功，将创建此目录结构和文件。

```
(root directory name)
|-- my-app
```



```

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

```

3. 使用此内容创建文件。将文件命名为 `buildspec.yml`，然后将其添加到 `(root directory name)/my-app` 目录。

```

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. 使用此内容创建文件。将文件命名为 `appspec.yml`，然后将其添加到 `(root directory name)/my-app` 目录。

```

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

```

完成后，您的目录结构和文件应该如下所示。

```

(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. 创建一个在 *(root directory name)/my-app* 中包含目录结构和文件的 ZIP 文件，然后将此 ZIP 文件上传到 AWS CodeBuild 和 AWS CodeDeploy 支持的源代码存储库类型，如 Amazon S3 输入存储桶或 GitHub 或 Bitbucket 存储库。

Important

如果您要使用 AWS CodePipeline 部署构建的构建输出项目，则无法将源代码上传到 Bitbucket 存储库。

请勿将 *(root directory name)* 或 *(root directory name)/my-app* 添加到 ZIP 文件，仅添加 *(root directory name)/my-app* 内的目录和文件。ZIP 文件应包含以下目录和文件：

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

6. 请按照 [创建构建项目 \(p. 154\)](#) 中的步骤创建构建项目。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 如果您计划使用 AWS CodeDeploy 部署构建输出项目，则请按照 [运行构建项目 \(p. 183\)](#) 中的步骤操作。否则，请跳过此步骤。(这是因为如果您计划使用 AWS CodePipeline 部署构建输出项目，则 AWS CodePipeline 将使用 AWS CodeBuild 自动运行构建。)
8. 完成使用 AWS CodeDeploy 的设置步骤，包括：

- 授予 IAM 用户访问 AWS CodeDeploy 以及 AWS CodeDeploy 依赖的 AWS 服务和操作的权限。有关更多信息，请参阅 AWS CodeDeploy 用户指南中的[预置 IAM 用户](#)。
 - 创建或标识服务角色，使 AWS CodeDeploy 能够标识将在其中部署构建输出项目的实例。有关更多信息，请参阅 AWS CodeDeploy 用户指南中的[为 AWS CodeDeploy 创建服务角色](#)。
 - 创建或标识 IAM 实例配置文件，使您的实例能够访问包含构建输出项目的 Amazon S3 输入存储桶或 GitHub 存储库。有关更多信息，请参阅 AWS CodeDeploy 用户指南中的[为 Amazon EC2 实例创建 IAM 实例配置文件](#)。
9. 创建或标识在其中部署构建输出项目且与 AWS CodeDeploy 兼容的 Amazon Linux 实例。有关更多信息，请参阅 AWS CodeDeploy 用户指南中的[使用适用于 AWS CodeDeploy 的实例](#)。
 10. 创建或标识 AWS CodeDeploy 应用程序和部署组。有关更多信息，请参阅 AWS CodeDeploy 用户指南中的[使用 AWS CodeDeploy 创建应用程序](#)。
 11. 将构建输出项目部署到实例。

要使用 AWS CodeDeploy 进行部署，请参阅 AWS CodeDeploy 用户指南中的[使用 AWS CodeDeploy 部署修订](#)。

要使用 AWS CodePipeline 进行部署，请参阅[将 AWS CodePipeline 与 AWS CodeBuild 结合使用 \(p. 136\)](#)。

12. 要在完成部署后查找构建输出项目，请登录到实例，然后在 /tmp 目录中查找名为 my-app-1.0-SNAPSHOT.jar 的文件。

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅[AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅[AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅[AWS CodeBuild 的限制 \(p. 234\)](#)。

适用于 AWS CodeBuild 的 AWS Lambda 示例

要使用 Lambda 等资源的无服务器应用程序定义标准模型，AWS 创建了 AWS 无服务器应用程序模型 (AWS SAM)。有关更多信息，请参阅 GitHub 上的[AWS 无服务器应用程序模型](#)存储库。

您可以使用 AWS CodeBuild 打包和部署遵循 AWS SAM 标准的无服务器应用程序。对于部署步骤，AWS CodeBuild 可以使用 AWS CloudFormation。要通过 AWS CodeBuild 和 AWS CloudFormation 自动构建和部署无服务器应用程序，您可以使用 AWS CodePipeline。

有关更多信息，请参阅 AWS Lambda Developer Guide 中的[部署基于 Lambda 的应用程序](#)。要尝试使用 AWS CodeBuild 以及 Lambda、AWS CloudFormation 和 AWS CodePipeline 的无服务器应用程序示例，请参阅 AWS Lambda Developer Guide 中的[基于 Lambda 应用程序的自动化部署](#)。

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅[AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅[AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅[AWS CodeBuild 的限制 \(p. 234\)](#)。

适用于 AWS CodeBuild 的 AWS Elastic Beanstalk 示例

该示例指示 AWS CodeBuild 使用 Maven 来生成一个名为 `my-web-app.war` 的 WAR 文件作为构建输出。然后该示例会将 WAR 文件部署到 Elastic Beanstalk 环境中的实例中。该示例以 [Java 示例 \(p. 101\)](#) 为基础。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS、CloudWatch Logs 和 Amazon EC2 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#)、[Amazon CloudWatch 定价](#) 和 [Amazon EC2 定价](#)。

创建源代码

在本节中，您将使用 Maven 生成要生成的源代码。稍后，您将使用 AWS CodeBuild 基于该源代码构建 WAR 文件。

1. 下载并安装 Maven。有关信息，请参阅 Apache Maven 网站上的 [下载 Apache Maven](#) 和 [安装 Apache Maven](#)。
2. 切换到您的本地计算机或实例上的空目录，然后运行此 Maven 命令。

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

如果成功，将创建此目录结构和文件。

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

在您运行 Maven 后，请继续执行以下方案之一：

- [方案 A：手动运行 AWS CodeBuild 并手动部署到 Elastic Beanstalk \(p. 79\)](#)
- [方案 B：使用 AWS CodePipeline 运行 AWS CodeBuild 并部署到 Elastic Beanstalk 中 \(p. 81\)](#)
- [方案 C：使用 Elastic Beanstalk 命令行界面 \(EB CLI\) 运行 AWS CodeBuild 并将其部署到 Elastic Beanstalk 环境中 \(p. 83\)](#)

方案 A：手动运行 AWS CodeBuild 并手动部署到 Elastic Beanstalk

在此方案中，您将手动创建并上传要生成的源代码。然后，您将使用 AWS CodeBuild 和 Elastic Beanstalk 控制台构建源代码，创建 Elastic Beanstalk 应用程序和环境，并将构建输出部署到环境中。

步骤 A1：将文件添加至源代码

在此步骤中，您将把 Elastic Beanstalk 配置文件和构建规范文件添加到 [创建源代码 \(p. 79\)](#) 中的代码中。然后，您要将源代码上传至 Amazon S3 输入存储桶或者 AWS CodeCommit 或 GitHub 存储库。

1. 在 `(root directory name)/my-web-app` 目录内创建一个名为 `.ebextensions` 的子目录。在 `.ebextensions` 子目录中，使用此内容创建一个名为 `fix-path.config` 的文件。

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

2. 使用以下内容创建名为 `buildspec.yml` 的文件。将此文件存储到 `(root directory name)/my-web-app` 目录。

```
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. 您的文件结构现在应如下所示。

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

4. 将 `my-web-app` 目录的此内容上传到 Amazon S3 输入存储桶或者上传到 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请勿上传 `(root directory name)` 或 `(root directory name)/my-web-app`，仅上传 `(root directory name)/my-web-app` 内的目录和文件。

如果您使用的是 Amazon S3 输入存储桶，请确保创建一个包含目录结构和文件的 ZIP 文件，然后将其上传至输入存储桶。请勿将 `(root directory name)` 或 `(root directory name)/my-web-app` 添加到 ZIP 文件，仅添加 `(root directory name)/my-web-app` 内的目录和文件。

步骤 A2：创建构建项目并运行构建

在此步骤中，您将使用 AWS CodeBuild 控制台创建构建项目，然后运行构建。

1. 创建或标识 Amazon S3 输出存储桶以存储构建输出。如果您将源代码存储在 Amazon S3 输入存储桶中，则输出存储桶必须与输入存储桶位于同一个 AWS 区域中。
2. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
使用 AWS 区域选择器选择一个支持 AWS CodeBuild 并与存储 Amazon S3 输出存储桶的区域相匹配的区域。
3. 创建构建项目，然后运行构建。有关更多信息，请参阅 [创建构建项目 \(控制台\)](#) (p. 154) 和 [运行构建项目 \(控制台\)](#) (p. 183)。除这些设置以外，将所有设置保留为默认值。
 - 在 Environment: How to build 中：
 - 对于 Environment image，选择 Use an image managed by AWS CodeBuild。
 - 对于 Operating system，选择 Ubuntu。
 - 对于 Runtime，选择 Java。
 - 对于 Version，选择 aws/codebuild/java:openjdk-8。
 - 在 Artifacts: Where to put the artifacts from this build project 中：
 - 对于 Artifacts name，请键入好记的构建输出文件名。包括 .zip 扩展名。
 - 在 Show advanced settings 中：
 - 对于 Artifacts packaging，选择 Zip。

步骤 A3：创建应用程序和环境并部署

在此步骤中，您将使用 Elastic Beanstalk 控制台创建应用程序和环境。作为创建环境的一部分，您要将之前步骤的构建输出部署到环境中。

1. 通过 <https://console.aws.amazon.com/elasticbeanstalk> 打开 Elastic Beanstalk 控制台。
使用 AWS 区域选择器选择一个与存储 Amazon S3 输出存储桶的区域相匹配的区域。
2. 创建 Elastic Beanstalk 应用程序。有关更多信息，请参阅[管理和配置 AWS Elastic Beanstalk 应用程序](#)。
3. 为此应用程序创建 Elastic Beanstalk 环境。有关更多信息，请参阅[新建环境向导](#)。除这些设置以外，将所有设置保留为默认值。
 - 对于 Platform，选择 Tomcat。
 - 对于 Application code，选择 Upload your code，然后选择 Upload。对于 Source code origin，选择 Public S3 URL，然后将完整的 URL 键入到输出存储桶中的构建输出 ZIP 文件中。然后选择下载。
4. 在 Elastic Beanstalk 将构建输出部署至环境后，您可以在 Web 浏览器中查看结果。转到实例的环境 URL (例如，<http://my-environment-name.random-string.region-ID.elasticbeanstalk.com>)。Web 浏览器将显示文字 Hello World!。

方案 B：使用 AWS CodePipeline 运行 AWS CodeBuild 并部署到 Elastic Beanstalk 中

在此方案中，您将手动准备并上传要生成的源代码。然后，您将使用 AWS CodePipeline 控制台创建管道以及 Elastic Beanstalk 应用程序和环境。在您创建管道之后，AWS CodePipeline 会自动构建源代码，并将构建输出部署到环境中。

步骤 B1：将构建规范文件添加至源代码

在此步骤中，您将创建构建规范文件并将其添加到您在 [创建源代码](#) (p. 79) 中创建的代码中。然后，您要将源代码上传至 Amazon S3 输入存储桶或者 AWS CodeCommit 或 GitHub 存储库。

1. 使用以下内容创建名为 buildspec.yml 的文件。将此文件存储在 [\(root directory name\)](#)/my-web-app 目录内。

```
version: 0.2

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

2. 您的文件结构现在应如下所示。

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

3. 将 my-web-app 目录的此内容上传到 Amazon S3 输入存储桶或者上传到 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请勿上传 *(root directory name)* 或 *(root directory name)/my-web-app*，仅上传 *(root directory name)/my-web-app* 内的目录和文件。

如果您使用的是 Amazon S3 输入存储桶，请确保创建一个包含目录结构和文件的 ZIP 文件，然后将其上传至输入存储桶。请勿将 *(root directory name)* 或 *(root directory name)/my-web-app* 添加到 ZIP 文件，仅添加 *(root directory name)/my-web-app* 内的目录和文件。

步骤 B2：创建管道并部署

在此步骤中，您将使用 AWS CodePipeline 和 Elastic Beanstalk 控制台创建管道、应用程序和环境。在您创建管道并运行之后，AWS CodePipeline 将使用 AWS CodeBuild 构建源代码，然后它将使用 Elastic Beanstalk 把构建输出部署到环境中。

1. 创建或标识 AWS CodePipeline、AWS CodeBuild 和 Elastic Beanstalk 可用来代表您工作的服务角色。有关更多信息，请参阅 [先决条件 \(p. 136\)](#)。
2. 通过以下网址打开 AWS CodePipeline 控制台：<https://console.aws.amazon.com/codepipeline/>。

使用 AWS 区域选择器选择一个支持 AWS CodeBuild 的区域，如果您将源代码存储在 Amazon S3 输入存储桶中，请选择与您存储输入存储桶的区域相匹配的区域。

3. 创建管道。有关信息，请参阅 [创建使用了 AWS CodeBuild 的管道 \(AWS CodePipeline 控制台\) \(p. 137\)](#)。除这些设置以外，将所有设置保留为默认值。
 - 在 Step 3: Build 中，对于 Configure your project，选择 Create a new build project。在 Environment: How to build 中：
 - 对于 Environment image，选择 Use an image managed by AWS CodeBuild。
 - 对于 Operating system，选择 Ubuntu。
 - 对于 Runtime，选择 Java。

- 对于 Version，选择 `aws/codebuild/java:openjdk-8`。
 - 在 Step 4: Beta 中，对于 Deployment provider，选择 AWS Elastic Beanstalk。
 - 对于应用程序，选择 `create a new one in Elastic Beanstalk` 链接。此操作将打开 Elastic Beanstalk 控制台。有关更多信息，请参阅[管理和配置 AWS Elastic Beanstalk 应用程序](#)。在您创建应用程序之后，请返回到 AWS CodePipeline 控制台，然后选择您刚刚创建的应用程序。
 - 对于环境，选择 `create a new one in Elastic Beanstalk` 链接。此操作将打开 Elastic Beanstalk 控制台。有关更多信息，请参阅[新建环境向导](#)。除以下设置外，将所有其他设置保留为默认值：对于 Platform，选择 Tomcat。在您创建环境之后，请返回到 AWS CodePipeline 控制台，然后选择您刚刚创建的环境。
4. 在管道成功运行之后，您可以在 Web 浏览器中查看结果。转到实例的环境 URL (例如，`http://my-environment-name.random-string.region-ID.elasticbeanstalk.com`)。Web 浏览器将显示文字 `Hello World!`。

现在，只要您更改源代码并将这些更改上传到原始 Amazon S3 输入存储桶或者上传到 AWS CodeCommit、GitHub 或 Bitbucket 存储库时，AWS CodePipeline 就会检测更改并再次运行管道。这将导致 AWS CodeBuild 自动重新构建代码，然后 Elastic Beanstalk 会将重新构建输出自动部署到环境中。

方案 C：使用 Elastic Beanstalk 命令行界面 (EB CLI) 运行 AWS CodeBuild 并将其部署到 Elastic Beanstalk 环境中

在此方案中，您将手动准备并上传要生成的源代码。然后，您将运行 EB CLI 创建 Elastic Beanstalk 应用程序和环境，使用 AWS CodeBuild 构建源代码，并将构建输出部署到环境中。有关更多信息，请参阅 AWS Elastic Beanstalk 开发人员指南 中的[通过 AWS CodeBuild 使用 EB CLI](#)。

步骤 C1：将文件添加至源代码

在此步骤中，您将把 Elastic Beanstalk 配置文件和构建规范文件添加到您在[创建源代码 \(p. 79\)](#)中创建的代码中。您还将创建或标识适用于构建规范文件的服务角色。

1. 创建或标识 Elastic Beanstalk 和 EB CLI 可以代表您使用的服务角色。有关信息，请参阅[创建 AWS CodeBuild 服务角色 \(p. 202\)](#)。
2. 在 `(root directory name)/my-web-app` 目录内创建一个名为 `.ebextensions` 的子目录。在 `.ebextensions` 子目录中，使用此内容创建一个名为 `fix-path.config` 的文件。

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

3. 使用以下内容创建名为 `buildspec.yml` 的文件。将此文件存储在 `(root directory name)/my-web-app` 目录内。

```
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
```

在前面的代码中，将 `my-service-role-name` 替换为您之前创建或标识的服务角色的名称。

4. 您的文件结构现在应如下所示。

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

步骤 C2：安装并运行 EB CLI

1. 如果您尚未完成此操作，请在您创建源代码的同一计算机或实例上安装和配置 EB CLI。有关信息，请参阅 [安装 Elastic Beanstalk 命令行界面 \(EB CLI\)](#) 和 [配置 EB CLI](#)。
2. 从您的计算机或实例的命令行或终端中，运行 `cd` 命令或类似命令以切换到 `(root directory name)/my-web-app` 目录。运行 `eb init` 命令配置 EB CLI。

```
eb init
```

出现提示时：

- 选择一个支持 AWS CodeBuild 且与您要在其中创建 Elastic Beanstalk 应用程序和环境的区域相匹配的 AWS 区域。
 - 创建 Elastic Beanstalk 应用程序，然后键入该应用程序的名称。
 - 选择 Tomcat 平台。
 - 选择 Tomcat 8 Java 8 版本。
 - 选择是否要使用 SSH 设置对您的环境实例的访问。
3. 从同一目录中，运行 `eb create` 命令创建 Elastic Beanstalk 环境。

```
eb create
```

出现提示时：

- 键入新环境的名称，或者接受建议的名称。
 - 键入该环境的 DNS 别名记录前缀，或者接受建议值。
 - 对于此示例，接受 Classic 负载均衡器类型。
4. 在您运行 `eb create` 命令之后，EB CLI 将执行以下操作：
 1. 从源代码中创建 ZIP 文件，然后将此 ZIP 文件上传至您账户中的 Amazon S3 存储桶。
 2. 创建 Elastic Beanstalk 应用程序和应用程序版本。
 3. 创建 AWS CodeBuild 项目。
 4. 基于新项目运行构建。

5. 构建完成后删除该项目。
6. 创建 Elastic Beanstalk 环境。
7. 将构建输出部署到环境中。
5. 在 EB CLI 将构建输出部署到环境中后，您可以在 Web 浏览器中查看结果。转到实例的环境 URL (例如，<http://my-environment-name.random-string.region-ID.elasticbeanstalk.com>)。Web 浏览器将显示文字 Hello World!。

如果需要，您可以更改源代码，然后从同一目录运行 `eb deploy` 命令。EB CLI 与 `eb create` 命令执行的步骤相同，但它将构建输出部署到现有环境中，而不是创建新的环境。

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

基于 AWS CodeBuild 代码的示例

请参阅这些基于代码的示例以试验 AWS CodeBuild：

名称	描述
C++ 示例 (p. 85)	使用 C++ 输出单个 .out 文件。
Go 示例 (p. 87)	使用 Go 输出单个二进制文件。
Maven 示例 (p. 89)	使用 Apache Maven 生成单个 JAR 文件。
Node.js 示例 (p. 92)	使用 Mocha 测试代码中的内部变量是否包含特定字符串值。生成单个 .js 文件。
Python 示例 (p. 94)	使用 Python 测试代码中的内部变量是否已设置为特定字符串值。生成单个 .py 文件。
Ruby 示例 (p. 96)	使用 RSpec 测试代码中的内部变量是否已设置为特定字符串值。生成单个 .rb 文件。
Scala 示例 (p. 98)	使用 sbt 生成单个 JAR 文件。
Java 示例 (p. 101)	使用 Apache Maven 生成单个 WAR 文件。
Linux 中的 .NET 核心示例 (p. 103)	使用 .NET 核心通过用 C# 编写的代码构建可执行文件。

适用于 AWS CodeBuild 的 C++ Hello World 示例

该 C++ 示例会生成一个名为 `hello.out` 的二进制文件作为构建输出。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多

信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

主题

- [运行示例 \(p. 86\)](#)
- [目录结构 \(p. 86\)](#)
- [文件 \(p. 87\)](#)
- [相关资源 \(p. 87\)](#)

运行示例

要运行此示例，请：

1. 按照本主题的“目录结构”和“文件”中部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
4. 将 *CPlusPlusOutputArtifact.zip* 文件下载到您的本地计算机或实例，然后提取文件的内容。在提取出来的内容中，获取 hello.out 文件。

目录结构

此示例假定有这样一个目录结构。

(root directory name)

```
|-- buildspec.yml
|-- hello.cpp
```

文件

此示例将使用这些文件。

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

```
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 (在 *(root directory name)*)

```
#include <iostream>

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

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门](#) (p. 4)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除](#) (p. 225)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制](#) (p. 234)。

适用于 AWS CodeBuild 的 Go Hello World 示例

该 Go 示例会生成一个名为 hello 的二进制文件作为构建输出。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

Note

您以前是否知道可使用 AWS Cloud9 处理本主题中的代码？AWS Cloud9 是一个在线的、基于云的集成开发环境 (IDE)，可用于编写、运行、调试和部署代码 (只需使用已连接 Internet 的计算机中

的浏览器)。AWS Cloud9 包含代码编辑器、调试程序、终端和基本工具，例如 AWS CLI 和 Git。在许多情况下，您不需要安装文件或配置您的开发计算机即可开始使用代码。有关更多信息，请参阅[AWS Cloud9 User Guide](#)。

主题

- [运行示例 \(p. 88\)](#)
- [目录结构 \(p. 88\)](#)
- [文件 \(p. 89\)](#)
- [相关资源 \(p. 89\)](#)

运行示例

要运行此示例，请：

1. 按照本主题的“目录结构”和“文件”中部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 (*root directory name*)，而只上传 (*root directory name*) 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 (*root directory name*) 添加到 ZIP 文件中，而只添加 (*root directory name*) 中的文件。

2. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
4. 将 *GoOutputArtifact.zip* 文件下载到您的本地计算机或实例，然后提取文件的内容。在提取出来的内容中，获取 hello 文件。

目录结构

此示例假定有这样一个目录结构。

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

文件

此示例将使用这些文件。

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

```
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 (在 *(root directory name)*)

```
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)
}
```

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门](#) (p. 4)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除](#) (p. 225)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制](#) (p. 234)。

用于 AWS CodeBuild 的 5 分钟学会 Maven 示例

此 Maven 示例将生成一个名为 my-app-1.0-SNAPSHOT.jar 的单个 JAR 文件作为构建输出。该示例以 Apache Maven 网站上的 [5 分钟学会 Maven](#) 主题为基础。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

运行示例

要运行此示例，请：

1. 下载并安装 Maven。有关更多信息，请参阅 Apache Maven 网站上的[下载 Apache Maven](#) 和[安装 Apache Maven](#)。
2. 切换到您的本地计算机或实例上的空目录，然后运行此 Maven 命令。

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

如果成功，将创建此目录结构和文件。

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

3. 使用此内容创建文件。将文件命名为 `buildspec.yml`，然后将其添加到 `(root directory name)/my-app` 目录。

```
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
```

完成后，您的目录结构和文件应该如下所示。

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

```

|-- test                                |-- App.java
|-- java
|-- com
|-- mycompany
|-- app
-- AppTest.java

```

4. 将 my-app 目录的此内容上传到 Amazon S3 输入存储桶或者上传到 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请勿上传 *(root directory name)* 或 *(root directory name)/my-app*，仅上传 *(root directory name)/my-app* 内的目录和文件。
如果您使用的是 Amazon S3 输入存储桶，请确保创建一个包含目录结构和文件的 ZIP 文件，然后将其上传至输入存储桶。请勿将 *(root directory name)* 或 *(root directory name)/my-app* 添加到 ZIP 文件，仅添加 *(root directory name)/my-app* 内的目录和文件。

5. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```

{
  "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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
7. 将 *MavenIn5MinutesOutputArtifact.zip* 文件下载到您的本地计算机或实例，然后提取 *MavenIn5MinutesOutputArtifact.zip* 文件的内容。在提取出来的内容中，打开 target 文件夹，获取 my-app-1.0-SNAPSHOT.jar 文件。

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

适用于 AWS CodeBuild 的 Node.js Hello World 示例

该 Node.js 示例用于测试代码中的内部变量是否以字符串 `Hello` 开头。它将生成一个名为 `HelloWorld.js` 的单个文件作为构建输出。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

Note

您以前是否知道可使用 AWS Cloud9 处理本主题中的代码？AWS Cloud9 是一个在线的、基于云的集成开发环境 (IDE)，可用于编写、运行、调试和部署代码 (只需使用已连接 Internet 的计算机中的浏览器)。AWS Cloud9 包含代码编辑器、调试程序、终端和基本工具，例如 AWS CLI 和 Git。在许多情况下，您不需要安装文件或配置您的开发计算机即可开始使用代码。有关更多信息，请参阅 [AWS Cloud9 User Guide](#)。

主题

- [运行示例 \(p. 92\)](#)
- [目录结构 \(p. 93\)](#)
- [文件 \(p. 93\)](#)
- [相关资源 \(p. 94\)](#)

运行示例

要运行此示例，请：

1. 在本地计算机或实例上，按照本主题的“目录结构和文件”部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 `start-build` 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
4. 将 *NodeJSOutputArtifact.zip* 文件下载到您的本地计算机或实例，然后提取 文件的内容。在提取出来的内容中，获取 HelloWorld.js 文件。

目录结构

此示例假定有这样一个目录结构。

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

文件

此示例将使用这些文件。

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

```
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 (在 (*root directory name*))

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

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

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

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门](#) (p. 4)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除](#) (p. 225)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制](#) (p. 234)。

适用于 AWS CodeBuild 的 Python Hello World 示例

此 Python 示例将测试代码中的内部变量是否包含字符串 `Hello world!`。它将生成一个名为 `HelloWorld.py` 的单个文件作为构建输出。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

Note

您以前是否知道可使用 AWS Cloud9 处理本主题中的代码？AWS Cloud9 是一个在线的、基于云的集成开发环境 (IDE)，可用于编写、运行、调试和部署代码 (只需使用已连接 Internet 的计算机中的浏览器)。AWS Cloud9 包含代码编辑器、调试程序、终端和基本工具，例如 AWS CLI 和 Git。在许多情况下，您不需要安装文件或配置您的开发计算机即可开始使用代码。有关更多信息，请参阅 [AWS Cloud9 User Guide](#)。

主题

- [运行示例](#) (p. 94)
- [目录结构](#) (p. 95)
- [文件](#) (p. 95)
- [相关资源](#) (p. 96)

运行示例

要运行此示例，请：

1. 按照本主题的“目录结构”和“文件”中部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 请按照 [直接运行 AWS CodeBuild](#) (p. 124) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 `create-project` 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
4. 将 *PythonOutputArtifact*.zip 文件下载到您的本地计算机或实例，然后提取 文件的内容。在提取出来的内容中，获取 HelloWorld.py 文件。

目录结构

此示例假定有这样一个目录结构。

```

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

```

文件

此示例将使用这些文件。

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

```

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 (在 (*root directory name*))

```

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

```

HelloWorld_tst.py (在 (*root directory name*))

```

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()
```

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

适用于 AWS CodeBuild 的 Ruby Hello World 示例

此 Ruby 示例将测试代码中的内部变量是否包含字符串 Hello, world!。它将生成一个名为 HelloWorld.rb 的单个文件作为构建输出。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

Note

您以前是否知道可使用 AWS Cloud9 处理本主题中的代码？AWS Cloud9 是一个在线的、基于云的集成开发环境 (IDE)，可用于编写、运行、调试和部署代码 (只需使用已连接 Internet 的计算机中的浏览器)。AWS Cloud9 包含代码编辑器、调试程序、终端和基本工具，例如 AWS CLI 和 Git。在许多情况下，您不需要安装文件或配置您的开发计算机即可开始使用代码。有关更多信息，请参阅 [AWS Cloud9 User Guide](#)。

主题

- [运行示例 \(p. 96\)](#)
- [目录结构 \(p. 97\)](#)
- [文件 \(p. 97\)](#)
- [相关资源 \(p. 98\)](#)

运行示例

要运行此示例，请：

1. 按照本主题的“目录结构”和“文件”中部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 (*root directory name*)，而只上传 (*root directory name*) 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 (*root directory name*) 添加到 ZIP 文件中，而只添加 (*root directory name*) 中的文件。

2. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 `create-project` 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
4. 将 *RubyOutputArtifact.zip* 文件下载到您的本地计算机或实例，然后提取文件的内容。在提取出来的内容中，获取 `HelloWorld.rb` 文件。

目录结构

此示例假定有这样一个目录结构。

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

文件

此示例将使用这些文件。

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

```
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 (在 *(root directory name)*)

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

HelloWorld_spec.rb (在 *(root directory name)*)

```
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
```

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门](#) (p. 4)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除](#) (p. 225)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制](#) (p. 234)。

用于 AWS CodeBuild 的 Scala Hello World 示例

本 Scala 示例将生成一个名为 `core_2.11-1.0.0.jar` 的单个 JAR 文件作为构建输出。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

主题

- [运行示例](#) (p. 98)
- [目录结构](#) (p. 99)
- [文件](#) (p. 100)
- [相关资源](#) (p. 101)

运行示例

要运行此示例，请：

1. 识别 Docker 镜像，其中包含一种用于 Scala 项目的构建工具 – sbt。要查找兼容的 Docker 镜像，请在 [Docker Hub](#) 中搜索 **sbt**。
2. 按照本主题的“目录结构和文件”部分的说明创建目录结构和文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的目录和文件。

如果您使用的是 Amazon S3 输入存储桶，请确保创建一个包含目录结构和文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的目录和文件。

3. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
5. 将 *ScalaOutputArtifact.zip* 文件下载到您的本地计算机或实例，然后提取文件的内容。在提取出来的内容中，打开 core/target/scala-2.11 文件夹，获取 core_2.11-1.0.0.jar 文件。

目录结构

此示例假定有这样一个目录结构。

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



```
`-- build.properties
```

文件

此示例将使用这些文件。

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

```
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 (在 *(root directory name)/core/src/main/scala*)

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

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

```
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 (在 *(root directory name)/project*)

```
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 (在 *(root directory name)/project*)

```
sbt.version=0.13.7
```

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门](#) (p. 4)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除](#) (p. 225)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制](#) (p. 234)。

适用于 AWS CodeBuild 的 WAR Hello World 示例

该 Maven 示例会生成一个名为 my-web-app.war 的 Web 应用程序存档 (WAR) 文件作为构建输出。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

运行示例

要运行此示例，请：

1. 下载并安装 Maven。有关更多信息，请参阅 Apache Maven 网站上的[下载 Apache Maven](#) 和[安装 Apache Maven](#)。
2. 切换到您的本地计算机或实例上的空目录，然后运行此 Maven 命令。

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

如果成功，将创建此目录结构和文件。

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

3. 使用此内容创建文件。将文件命名为 `buildspec.yml`，然后将其添加到 `(root directory name)/my-web-app` 目录。

```
version: 0.2

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

完成后，您的目录结构和文件应该如下所示。

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

4. 将 `my-web-app` 目录的内容上传到 Amazon S3 输入存储桶或者上传到 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请勿上传 `(root directory name)` 或 `(root directory name)/my-web-app`，仅上传 `(root directory name)/my-web-app` 内的目录和文件。

如果您使用的是 Amazon S3 输入存储桶，请确保创建一个包含目录结构和文件的 ZIP 文件，然后将其上传至输入存储桶。请勿将 `(root directory name)` 或 `(root directory name)/my-web-app` 添加到 ZIP 文件，仅添加 `(root directory name)/my-web-app` 内的目录和文件。例如，ZIP 文件应包含以下目录和文件：

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

5. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 `create-project` 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要获取构建输出项目，请打开您的 Amazon S3 输出存储桶。
7. 将 `WebArchiveHelloWorldOutputArtifact.zip` 文件下载到您的本地计算机或实例，然后提取文件的内容。在提取出来的内容中，打开 `target` 文件夹，获取 `my-web-app.war` 文件。

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

适用于 AWS CodeBuild 的 Linux 中的 .NET 核心示例

此示例使用运行 .NET 核心的 AWS CodeBuild 构建环境来通过用 C# 编写的代码构建可执行文件。

Important

运行此示例可能会导致您的 AWS 账户产生相关费用。这些费用包括可能针对 AWS CodeBuild 和 AWS 资源以及与 Amazon S3、AWS KMS 和 CloudWatch Logs 相关的操作收取的费用。有关更多信息，请参阅 [AWS CodeBuild 定价](#)、[Amazon S3 定价](#)、[AWS Key Management Service 定价](#) 和 [Amazon CloudWatch 定价](#)。

运行示例

要运行此示例，请：

1. 按照本主题的“目录结构”和“文件”中部分的说明创建文件，然后将其上传到 Amazon S3 输入存储桶或者 AWS CodeCommit、GitHub 或 Bitbucket 存储库。

Important

请不要上传 *(root directory name)*，而只上传 *(root directory name)* 中的文件。如果您使用的是 Amazon S3 输入存储桶，请务必创建一个包含文件的 ZIP 文件，然后将其上传至输入存储桶。请不要将 *(root directory name)* 添加到 ZIP 文件中，而只添加 *(root directory name)* 中的文件。

2. 请按照 [直接运行 AWS CodeBuild \(p. 124\)](#) 中的步骤创建构建项目、运行构建并查看相关构建信息。

如果您使用 AWS CLI 创建构建项目，则 create-project 命令的 JSON 格式输入可能与此类似。(请将占位符替换为您自己的值。)

```
{
  "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. 要获取构建输出项目，请在您的 Amazon S3 输出存储桶中，将 *dot-net-core-output-artifact.zip* 文件下载到您的本地计算机或实例。提取内容以访问可执行文件 HelloWorldSample.dll，该文件可在 bin\Debug\netcoreapp2.0 目录中找到。

目录结构

此示例假定有这样一个目录结构。

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

文件

此示例将使用这些文件。

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

```
version: 0.2

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

HelloWorldSample.csproj (在 *(root directory name)*)

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

Program.cs (在 *(root directory name)*)

```
using System;

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

相关资源

- 有关 AWS CodeBuild 入门的更多信息，请参阅 [AWS CodeBuild 入门 \(p. 4\)](#)。
- 有关解决与 AWS CodeBuild 相关的问题的更多信息，请参阅 [AWS CodeBuild 故障排除 \(p. 225\)](#)。
- 有关 AWS CodeBuild 中的限制的更多信息，请参阅 [AWS CodeBuild 的限制 \(p. 234\)](#)。

为 AWS CodeBuild 规划构建

在使用 AWS CodeBuild 运行构建之前，您必须回答以下问题：

1. 源代码位于什么位置？AWS CodeBuild 目前支持通过以下源代码存储库提供商进行构建。源代码必须包含构建规范文件，或构建规范必须声明为构建项目定义的一部分。生成规范 是生成命令和相关设置的集合，采用 YAML 格式，由 AWS CodeBuild 用来运行生成任务。

存储库提供商	必需	Documentation
AWS CodeCommit	存储库名称。 (可选) 与源代码关联的提交 ID。	请参阅 AWS CodeCommit 用户指南中的以下主题： 创建 AWS CodeCommit 存储库 在 AWS CodeCommit 中创建提交
Amazon S3	输入存储桶名称。 与包含源代码的构建输入 ZIP 文件对应的对象名称。 (可选) 与构建输入 ZIP 文件相关联的版本 ID。	请参阅 Amazon S3 入门指南中的以下主题： 创建存储桶 向存储桶添加对象
GitHub	存储库名称。 (可选) 与源代码关联的提交 ID。	请参阅 GitHub 帮助网站上的以下主题： 创建存储库
Bitbucket	存储库名称。 (可选) 与源代码关联的提交 ID。	请参阅 Bitbucket 云文档网站上的以下主题： 创建存储库

2. 您需要运行哪些构建命令，按照什么顺序运行？默认情况下，AWS CodeBuild 会从您指定的提供商处下载构建输入，然后将构建输出上传到您指定的存储桶。您可以使用构建规范作为指导，将下载的构建输入转换为预期的构建输出。有关更多信息，请参见 [构建规范参考](#) (p. 107)。
3. 运行构建需要哪些运行时和工具？例如，您是否是为 Java、Ruby、Python 或 Node.js 构建的？构建是否需要 Maven 或 Ant？或者是否需要适用于 Java、Ruby 或 Python 的编译器？构建是否需要 Git、AWS CLI 或其他工具？

AWS CodeBuild 在使用 Docker 映像的构建环境中运行构建。这些 Docker 映像必须存储在 AWS CodeBuild 支持的存储库类型中。这些存储库包括 AWS CodeBuild Docker 映像存储库、Docker Hub 和 Amazon Elastic Container Registry (Amazon ECR)。有关 AWS CodeBuild Docker 映像存储库的更多信息，请参见 [AWS CodeBuild 提供的 Docker 映像](#) (p. 114)。

4. 是否需要 AWS CodeBuild 不会自动提供的 AWS 资源？如果需要，这些资源又需要哪些安全策略呢？例如，您可能需要修改 AWS CodeBuild 服务角色来允许 AWS CodeBuild 使用这些资源。
5. 是否要将 AWS CodeBuild 与您的 VPC 结合使用？如果是，您需要您的 VPC 配置的 VPC ID、子网 ID 和安全组 ID。有关更多信息，请参见 [将 AWS CodeBuild 与 Amazon Virtual Private Cloud 结合使用](#) (p. 125)。

回答完这些问题后，您应该已经具备了成功运行构建所需的设置和资源。要运行构建，您可以：

- 使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包。有关更多信息，请参阅 [直接运行 AWS CodeBuild \(p. 124\)](#)。
- 在 AWS CodePipeline 中创建或标识管道，然后添加构建或测试操作来指示 AWS CodeBuild 自动测试代码、运行构建，或同时执行两者。有关更多信息，请参阅 [将 AWS CodePipeline 与 AWS CodeBuild 结合使用 \(p. 136\)](#)。

适用于 AWS CodeBuild 的构建规范参考

本主题提供有关构建规范的重要参考信息。生成规范 是生成命令和相关设置的集合，采用 YAML 格式，由 AWS CodeBuild 用来运行生成任务。您可以将构建规范作为源代码的一部分，也可以在创建构建项目时定义构建规范。有关构建规范工作原理的信息，请参阅 [AWS CodeBuild 如何工作 \(p. 2\)](#)。

主题

- [构建规范文件名称和存储位置 \(p. 107\)](#)
- [构建规范语法 \(p. 107\)](#)
- [构建规范示例 \(p. 112\)](#)
- [构建规范版本 \(p. 113\)](#)

构建规范文件名称和存储位置

如果您在源代码中包含构建规范，默认情况下，构建规范文件必须名为 `buildspec.yml` 且放置在源目录的根目录中。

您还可以覆盖默认的构建规范文件名和位置。例如，您可以：

- 对同一存储库中的不同构建使用不同的构建规范文件，如 `buildspec_debug.yml` 和 `buildspec_release.yml`。
- 将构建规范文件存储在源目录的根目录之外的位置，如 `config/buildspec.yml`。

您可以只为构建项目指定一个构建规范，而不管构建规范文件的名称如何。

要覆盖默认构建规范文件名、默认位置或这两者，执行下列操作之一：

- 运行 AWS CLI `create-project` 或 `update-project` 命令，将 `buildspec` 值设置为备用构建规范文件 (与内置环境变量 `CODEBUILD_SRC_DIR` 的值相关) 的路径。您还可以在 AWS 开发工具包中执行与“创建项目”操作等效的操作。有关更多信息，请参阅 [创建构建项目 \(p. 154\)](#) 或 [更改构建项目的设置 \(p. 174\)](#)。
- 运行 AWS CLI `start-build` 命令，将 `buildspecOverride` 值设置为至备用构建规范文件 (与内置环境变量 `CODEBUILD_SRC_DIR` 的值相关) 的路径。您还可以在 AWS 开发工具包中执行与“启动构建”操作等效的操作。有关更多信息，请参阅 [运行构建项目 \(p. 183\)](#)。
- 在 AWS CloudFormation 模板中，将类型为 `AWS::CodeBuild::Project` 的资源的 `Source` 的 `BuildSpec` 属性设置为至备用规范文件 (与内置环境变量 `CODEBUILD_SRC_DIR` 的值相关) 的路径。有关更多信息，请参阅 AWS CloudFormation 用户指南的 [AWS CodeBuild 项目源](#) 中的“BuildSpec”属性。

构建规范语法

构建规范必须以 [YAML](#) 格式表示。

构建规范的语法如下：


```
version: 0.2

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

phases:
  install:
    commands:
      - command
      - command
    finally:
      - command
      - command
  pre_build:
    commands:
      - command
      - command
    finally:
      - command
      - command
  build:
    commands:
      - command
      - command
    finally:
      - command
      - command
  post_build:
    commands:
      - command
      - command
    finally:
      - command
      - command
artifacts:
  files:
    - location
    - location
  discard-paths: yes
  base-directory: location
cache:
  paths:
    - path
    - path
```

构建规范包含以下内容：

- **version**：必需的映射。表示构建规范版本。我们建议您使用 0.2。

Note

版本 0.1 仍受支持。但是，我们建议您尽可能使用版本 0.2。有关更多信息，请参阅 [构建规范版本 \(p. 113\)](#)。

- **env**：可选的序列。表示一个或多个自定义环境变量的信息。
 - **variables**：在指定了 **env** 并且您希望定义纯文本格式的自定义环境变量时必需。包含 **key/value** 标量的映射，其中每个映射表示一个纯文本格式的自定义环境变量。**key** 是自定义环境变量的名称，而 **value** 是该变量的值。

Important

我们强烈建议不要使用环境变量存储敏感值，尤其是 AWS 访问密钥 ID 和秘密访问密钥。使用 AWS CodeBuild 控制台和 AWS CLI 等工具能够以纯文本格式显示环境变量。对于敏感值，我们建议您改用 `parameter-store` 映射，如本节后面所述。

您设置的任何环境变量都将替换现有的环境变量。例如，如果 Docker 镜像已经包含一个名为 `MY_VAR` 的环境变量，其值为 `my_value`，且您设置了一个名为 `MY_VAR` 的环境变量，其值为 `other_value`，那么，`my_value` 将替换为 `other_value`。同样，如果 Docker 镜像已经包含一个名为 `PATH` 的环境变量，其值为 `/usr/local/sbin:/usr/local/bin`，且您设置了一个名为 `PATH` 的环境变量，其值为 `$PATH:/usr/share/ant/bin`，那么，`/usr/local/sbin:/usr/local/bin` 将替换为文本值 `$PATH:/usr/share/ant/bin`。

请勿使用以 `CODEBUILD_` 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。

如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
 - 构建项目定义中的值优先级处于其次。
 - 构建规范声明中的值优先级最低。
- `parameter-store`：在指定了 `env` 并且您要检索存储在 Amazon EC2 Systems Manager Parameter Store 中的自定义环境变量时必需。包含 **key/value** 标量的映射，其中每个映射表示一个存储在 Amazon EC2 Systems Manager Parameter Store 中的自定义环境变量。**key** 是您之后将在构建命令中用于表示此自定义环境变量的名称，而 **value** 是存储在 Amazon EC2 Systems Manager Parameter Store 中的自定义环境变量的名称。要存储敏感值，请参阅 Amazon EC2 Systems Manager 用户指南中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store 控制台演练](#)。

Important

要允许 AWS CodeBuild 检索存储在 Amazon EC2 Systems Manager Parameter Store 中的自定义环境变量，您必须将 `ssm:GetParameters` 操作添加到您的 AWS CodeBuild 服务角色。有关更多信息，请参阅 [创建 AWS CodeBuild 服务角色 \(p. 202\)](#)。

您从 Amazon EC2 Systems Manager Parameter Store 检索到的任何环境变量都将替换现有环境变量。例如，如果 Docker 映像已经包含一个名为 `MY_VAR`、值为 `my_value` 的环境变量，而您检索到一个名为 `MY_VAR`、值为 `other_value` 的环境变量，则 `my_value` 将替换为 `other_value`。同样，如果 Docker 映像已经包含一个名为 `PATH`、值为 `/usr/local/sbin:/usr/local/bin` 的环境变量，而您检索到一个名为 `PATH`、值为 `$PATH:/usr/share/ant/bin` 的环境变量，则 `/usr/local/sbin:/usr/local/bin` 将替换为文本值 `$PATH:/usr/share/ant/bin`。

请勿存储名称以 `CODEBUILD_` 开头的任何环境变量。此前缀是专为内部使用预留的。

如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
 - 构建项目定义中的值优先级处于其次。
 - 构建规范声明中的值优先级最低。
- `phases`：必需的序列。表示 AWS CodeBuild 在构建的各个阶段将运行的命令。

Note

在构建规范版本 0.1 中，AWS CodeBuild 将在构建环境中的默认 Shell 的单独实例中运行每条命令。这表示各个命令独立于其他所有命令而运行。因此，在默认情况下，您无法运行依赖所有上一个命令状态的单个命令（如更改目录或设置环境变量）。要绕开此限制，我们建议您使用版本 0.2 来解决此问题。如果您出于某种原因必须使用构建规范版本 0.1，我们推荐[构建环境中的 Shell 和命令 \(p. 120\)](#)中的方法。

允许的构建阶段名称是：

- `install`：可选的序列。表示 AWS CodeBuild 在安装过程中将运行的命令（如果有）。我们建议您使用只适用于构建环境中安装软件包的 `install` 阶段。例如，您可以使用此阶段来安装代码测试框架（如 Mocha 或 RSpec）。

- **commands** : 如果指定 `install` , 则为必需的序列。包含一系列标量, 其中各个标量表示 AWS CodeBuild 将在安装过程中运行的单个命令。AWS CodeBuild 按照列出的顺序从开始到结束一次运行一个命令。
- **pre_build** : 可选的序列。表示 AWS CodeBuild 在构建之前会运行的命令 (如果有)。例如, 您可以使用此阶段登录 Amazon ECR, 还可以安装 npm 依赖项。
 - **commands** : 如果指定 `pre_build` , 则为必需的序列。包含一系列标量, 其中各个标量表示 AWS CodeBuild 将在构建前运行的单个命令。AWS CodeBuild 按照列出的顺序从开始到结束一次运行一个命令。
- **build** : 可选的序列。表示 AWS CodeBuild 在构建过程中会运行的命令 (如果有)。例如, 您可以使用此阶段运行 Mocha、RSpec 或 sbt。
 - **commands** : 在指定 `build` 时是必需的。包含一系列标量, 其中各个标量表示 AWS CodeBuild 将在构建过程中运行的单个命令。AWS CodeBuild 按照列出的顺序从开始到结束一次运行一个命令。
- **post_build** : 可选的序列。表示 AWS CodeBuild 在构建后会运行的命令 (如果有)。例如, 您可以使用 Maven 将构建项目打包成 JAR 或 WAR 文件, 还可以将 Docker 镜像推入到 Amazon ECR 中。然后, 您可以通过 Amazon SNS 发送构建通知。
 - **commands** : 在指定 `post_build` 时是必需的。包含一系列标量, 其中各个标量表示 AWS CodeBuild 将在构建后运行的单个命令。AWS CodeBuild 按照列出的顺序从开始到结束一次运行一个命令。

Important

如果早期构建阶段中的命令失败, 则一些构建阶段中的命令可能无法运行。例如, 如果 `install` 阶段过程中的命令失败, 则 `pre_build`、`build` 和 `post_build` 阶段中的命令都将无法在构建的生命周期内运行。有关更多信息, 请参阅 [构建阶段过渡 \(p. 191\)](#)。

- **finally** : 可选语句块。在 `finally` 语句块中指定的命令在 `commands` 语句块命令之后执行。即便 `commands` 语句块命令失败, 仍会执行 `finally` 语句块命令。例如, 假设 `commands` 语句块包含三条命令, 而第一条命令失败了, 则 AWS CodeBuild 会跳过后面的两条命令来运行 `finally` 语句块中的任何命令。当 `commands` 和 `finally` 语句块中的所有命令都成功运行后, 此阶段才算成功。如果某个阶段有任何命令失败, 则该阶段失败。
- **artifacts** : 可选的序列。表示有关 AWS CodeBuild 可在何处查找构建输出以及 AWS CodeBuild 如何进行准备以便将其上传到 Amazon S3 输出存储桶的信息。如果出现以下情况则不需要序列, 例如, 您正在构建或将 Docker 镜像推入到 Amazon ECR, 或者正在运行源代码上的单元测试但却并未构建。
- **files** : 必需的序列。表示包含构建环境中的构建输出项目的位置。包含一系列标量, 其中每个标量表示一个相对于原始构建位置或基本目录 (如果已设置) 的单独位置, AWS CodeBuild 可在此处查找构建输出项目。位置可包含以下内容:
 - 单个文件 (如 `my-file.jar`)。
 - 子目录中的单个文件 (例如, `my-subdirectory/my-file.jar` 或 `my-parent-subdirectory/my-subdirectory/my-file.jar`)。
 - `/**/*` 表示所有文件均采用递归方式。
 - `my-subdirectory/*` 表示名为 `my-subdirectory` 的子目录中的所有文件。
 - `my-subdirectory/**/*` 表示采用递归方式的所有文件都从名为 `my-subdirectory` 的子目录开始。

您在指定构建输出项目位置时, AWS CodeBuild 可在构建环境中查找原始构建位置。您无需将包含路径的构建项目输出位置放在原始构建位置的前面, 或指定 `./` 或类似。如果您需要了解此位置的路径, 则可以在构建过程中运行命令 `echo $CODEBUILD_SRC_DIR`。各个构建环境的位置可能略有不同。

- **discard-paths** : 可选的映射。表示是否放弃构建输出项目中的文件的路径。如果丢弃路径, 则为 `yes`; 否则为 `no` 或未指定 (默认)。例如, 如果构建输出项目中的文件路径为 `com/mycompany/app/HelloWorld.java`, 则指定 `yes` 可能会将此路径缩短为 `HelloWorld.java`。
- **base-directory** : 可选的映射。表示相对于原始构建位置的一个或多个顶级目录, AWS CodeBuild 使用这些目录确定在构建输出项目中包含哪些文件和子目录。有效值包括:
 - 单个顶级目录 (如 `my-directory`)。
 - `'my-directory*'` 表示名称以 `my-directory` 为开头的所有顶级目录。

构建输出项目中不包含映射顶级目录，只包含其文件和子目录。

您可以使用 `files` 和 `discard-paths` 进一步限制包含哪些文件和子目录。例如，对于以下目录结构：

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

对于以下 `artifacts` 序列：

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

以下子目录和文件应包含在构建输出项目中：

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

对于以下 `artifacts` 序列：

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

以下文件应包含在构建输出项目中：

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

- `cache`：可选的序列。表示 AWS CodeBuild 可在何处准备用于将缓存上传到 Amazon S3 缓存存储桶的文件的相关信息。如果项目的缓存类型为 `No Cache`，则不需要此序列。
- `paths`：必需的序列。表示缓存的位置。包含一系列标量，其中每个标量表示一个相对于原始构建位置或基本目录（如果已设置）的单独位置，AWS CodeBuild 可在此处查找构建输出项目。位置可包含以下内容：
 - 单个文件（如 `my-file.jar`）。
 - 子目录中的单个文件（例如，`my-subdirectory/my-file.jar` 或 `my-parent-subdirectory/my-subdirectory/my-file.jar`）。
 - `'**/*'` 表示所有文件均采用递归方式。
 - `my-subdirectory/*` 表示名为 `my-subdirectory` 的子目录中的所有文件。
 - `my-subdirectory/**/*` 表示采用递归方式的所有文件都从名为 `my-subdirectory` 的子目录开始。

Important

因为构建规范声明必须为有效的 YAML，所以构建规范声明中的间隔至关重要。如果构建规范声明中的间隔数量无效，则构建可能会立即失败。您可以使用 YAML 验证程序测试构建规范声明是否为有效的 YAML。

如果您在创建或更新构建项目时使用 AWS CLI 或 AWS 开发工具包声明构建规范，则构建规范必须是以 YAML 格式表示的单个字符串，包括必需的空格和换行转义字符。将在下一部分中提供示例。如果您使用 AWS CodeBuild 或 AWS CodePipeline 控制台而不是 buildspec.yml 文件，则您只能插入适合 build 阶段的命令。您不要使用前一语法，而是要使用单行列出需要在构建阶段使用的所有命令。对于多个命令，使用 && 分开各个命令 (例如 mvn test && mvn package)。

您可以使用 AWS CodeBuild 或 AWS CodePipeline 控制台而不是 buildspec.yml 文件指定构建环境中构建输出项目的位置。您不要使用前一语法，而是要使用单行列出所有位置。对于多个位置，使用逗号分开各个位置 (appspec.yml, target/my-app.jar)。

构建规范示例

以下是 buildspec.yml 文件的示例。

```
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
    finally:
      - echo This always runs even if the update or install command fails
  pre_build:
    commands:
      - echo Entered the pre_build phase...
      - docker login -u User -p $LOGIN_PASSWORD
    finally:
      - echo This always runs even if the login command fails
  build:
    commands:
      - echo Entered the build phase...
      - echo Build started on `date`
      - mvn install
    finally:
      - echo This always runs even if the install command fails
  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/**/*'
```

以下是上一构建规范的示例，此规范与 AWS CLI 或 AWS 开发工具包一起使用并表示为单个字符串。

```
"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 Entered 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`\n\nartifacts:\n  files:\n    - target/messageUtil-1.0.jar\n  discard-paths: yes"
```

以下是 build 阶段中命令的示例，与 AWS CodeBuild 或 AWS CodePipeline 控制台一起使用。

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

在这些示例中：

- 将设置密钥为 `JAVA_HOME`，值为 `/usr/lib/jvm/java-8-openjdk-amd64` 的纯文本格式的自定义环境变量。
- 随后，将通过使用密钥 `LOGIN_PASSWORD` 在构建命令中引用您在 Amazon EC2 Systems Manager Parameter Store 中存储的一个名为 `dockerLoginPassword` 的自定义环境变量。
- 您无法更改这些构建阶段名称。将在此示例中运行的命令是 `apt-get update -y`、`apt-get install -y maven`（用于安装 Apache Maven）、`mvn install`（用于编译和测试源代码并将源代码打包到构建输出项目中，以及执行其他操作（例如构建输出项目的内部存储库中安装该项目））、`docker login`（用于使用与您在 Amazon EC2 Systems Manager Parameter Store 中设置的自定义环境变量 `dockerLoginPassword` 的值对应的密码登录 Docker）以及若干 `echo` 命令。此处包含的 `echo` 命令用于显示 AWS CodeBuild 运行命令的方式以及运行命令的顺序。
- `files` 表示要上传到构建输出位置的文件。在本示例中，AWS CodeBuild 将上传单个文件 `messageUtil-1.0.jar`。在构建环境中名为 `messageUtil-1.0.jar` 的相对目录中，可找到 `target` 文件。由于指定了 `discard-paths: yes`，因此将直接上传 `messageUtil-1.0.jar`（而不上传到中间 `target` 目录）。文件名称 `messageUtil-1.0.jar` 和 `target` 的相对目录名称均基于 Apache Maven 如何创建并存储构建输出项目（仅针对此示例）。在您自己的方案中，这些文件名称和目录会有所不同。

构建规范版本

下表列出了构建规范版本以及版本间的变化。

版本	更改
0.2	<p><code>environment_variables</code> 已重命名为 <code>env</code>。<code>plaintext</code> 已重命名为 <code>variables</code>。</p> <p>在版本 0.1 中，AWS CodeBuild 将在构建环境内默认 Shell 的单独实例中运行每个构建命令。在版本 0.2 中，此问题已得到解决；AWS CodeBuild 将在构建环境内默认 Shell 的同一实例中运行所有构建命令。</p>
0.1	这是构建规范格式的初始定义。

AWS CodeBuild 构建环境参考

当您调用 AWS CodeBuild 运行构建时，您必须提供有关 AWS CodeBuild 将使用的构建环境的信息。生成环境是由操作系统、编程语言运行时和 AWS CodeBuild 用于运行生成任务的工具组成的。有关构建环境的工作方式的信息，请参阅 [AWS CodeBuild 如何工作](#) (p. 2)。

构建环境包含 Docker 映像。有关信息，请参阅 Docker 文档网站上的 [Docker 词汇表：映像](#)。

当您向 AWS CodeBuild 提供有关构建环境的信息时，您可以在支持的存储库类型中指定 Docker 映像的标识符。其中包括 AWS CodeBuild Docker 映像存储库 (Docker Hub 中公开可用的映像) 和 AWS 账户中的 Amazon Elastic Container Registry (Amazon ECR) 存储库：

- 我们建议您使用 AWS CodeBuild Docker 映像存储库中存储的 Docker 映像，因为它们已经过优化以用于此服务。有关更多信息，请参阅 [AWS CodeBuild 提供的 Docker 映像 \(p. 114\)](#)。
- 要获取 Docker Hub 中存储的公开可用的 Docker 映像的标识符，请参阅 Docker 文档网站上的 [搜索映像](#)。
- 要了解如何在您的 AWS 账户中使用 Amazon ECR 存储库中存储的 Docker 映像，请参阅我们的 [Amazon ECR 示例 \(p. 38\)](#)。

除了 Docker 映像标识符，您还可指定构建环境将使用的一组计算资源。有关更多信息，请参阅 [构建环境计算类型 \(p. 119\)](#)。

主题

- [AWS CodeBuild 提供的 Docker 映像 \(p. 114\)](#)
- [构建环境计算类型 \(p. 119\)](#)
- [构建环境中的 Shell 和命令 \(p. 120\)](#)
- [构建环境中的环境变量 \(p. 120\)](#)
- [构建环境中的后台任务 \(p. 122\)](#)

AWS CodeBuild 提供的 Docker 映像

AWS CodeBuild 管理 AWS CodeBuild 和 AWS CodePipeline 控制台中可用的以下 Docker 映像。

Note

如果您在此页中没有找到您的映像，那它很可能包含了供应商不再提供支持的组件。AWS CodeBuild 控制台或 AWS CodeBuild 软件开发工具包中不提供包含有一个或多个不受支持组件的映像。这些映像可能在 CLI 中仍然提供，但它们不受支持，并且不会更新。

平台	编程语言或框架	运行时版本	映像标识符	定义
Ubuntu 14.04	(基本映像)		aws/codebuild/ubuntu-base:14.04	ubuntu/ubuntu-base/14.04
Ubuntu 14.04	Android	26.1.1	aws/codebuild/android-java-8:26.1.1	ubuntu/android-java-8/26.1.1
Ubuntu 14.04	Android	24.4.1	aws/codebuild/android-java-8:24.4.1	ubuntu/android-java-8/24.4.1
Ubuntu 14.04	Docker	17.09.0	aws/codebuild/docker:17.09.0	ubuntu/docker/17.09.0
Ubuntu 14.04	Golang	1.10	aws/codebuild/golang:1.10	ubuntu/golang/1.10
Ubuntu 14.04	Java	8	aws/codebuild/java:openjdk-8	ubuntu/java/openjdk-8

平台	编程语言或框架	运行时版本	映像标识符	定义
Ubuntu 14.04	Java	9	aws/codebuild/ java:openjdk-9	ubuntu/java/ openjdk-9
Ubuntu 14.04	Node.js	10.1.0	aws/codebuild/ nodejs:10.1.0	ubuntu/ nodejs/10.1.0
Ubuntu 14.04	Node.js	8.11.0	aws/codebuild/ nodejs:8.11.0	ubuntu/ nodejs/8.11.0
Ubuntu 14.04	Node.js	6.3.1	aws/codebuild/ nodejs:6.3.1	ubuntu/ nodejs/6.3.1
Ubuntu 14.04	PHP	5.6	aws/codebuild/ php:5.6	ubuntu/php/5.6
Ubuntu 14.04	PHP	7.0	aws/codebuild/ php:7.0	ubuntu/php/7.0
Ubuntu 14.04	Python	3.6.5	aws/codebuild/ python:3.6.5	ubuntu/ python/3.6.5
Ubuntu 14.04	Python	3.5.2	aws/codebuild/ python:3.5.2	ubuntu/ python/3.5.2
Ubuntu 14.04	Python	3.4.5	aws/codebuild/ python:3.4.5	ubuntu/ python/3.4.5
Ubuntu 14.04	Python	3.3.6	aws/codebuild/ python:3.3.6	ubuntu/ python/3.3.6
Ubuntu 14.04	Python	2.7.12	aws/codebuild/ python:2.7.12	ubuntu/ python/2.7.12
Ubuntu 14.04	Ruby	2.5.1	aws/codebuild/ ruby:2.5.1	ubuntu/ruby/2.5.1
Ubuntu 14.04	Ruby	2.3.1	aws/codebuild/ ruby:2.3.1	ubuntu/ruby/2.3.1
Ubuntu 14.04	Ruby	2.2.5	aws/codebuild/ ruby:2.2.5	ubuntu/ruby/2.2.5
Ubuntu 14.04	.NET 核心	1.1	aws/codebuild/ dot-net:core-1	ubuntu/dot-net/ core-1
Ubuntu 14.04	.NET 核心	2.0	aws/ codebuild/dot- net:core-2.0	ubuntu/dot-net/ core-2
Ubuntu 14.04	.NET 核心	2.1	aws/ codebuild/dot- net:core-2.1	ubuntu/dot-net/ core-2.1
Windows Server Core 2016	(基本映像)		aws/codebuild/ windows- base:1.0	

AWS CodeBuild 还管理 AWS CodeBuild 和 AWS CodePipeline 控制台中未包含的以下 Docker 映像。

平台	编程语言或框架	运行时版本	其他组件	映像标识符
Amazon Linux 2016.03 , 64 位 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 位 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 位 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 位 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
Amazon Linux 2016.03 , 64 位 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 位 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 位 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 位 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 位 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 位 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 位 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 位 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 位 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 位 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

平台	编程语言或框架	运行时版本	其他组件	映像标识符
Amazon Linux 2016.03 , 64 位 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 位 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 位 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
Amazon Linux 2016.03 , 64 位 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 位 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 位 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 位 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 位 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 位 v2.3.2	Ruby	2.3	Bundler、RubyGems	aws/codebuild/eb-ruby-2.3-amazonlinux-64:2.3.2
Amazon Linux 2016.03 , 64 位 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 位 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 位 v2.3.2	Ruby	2.2	Bundler、RubyGems	aws/codebuild/eb-ruby-2.2-amazonlinux-64:2.3.2
Amazon Linux 2016.03 , 64 位 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 位 v2.1.3	Ruby	2.1.9	Bundler、RubyGems	aws/codebuild/eb-ruby-2.1-amazonlinux-64:2.1.3

平台	编程语言或框架	运行时版本	其他组件	映像标识符
Amazon Linux 2016.03, 64 位 v2.3.2	Ruby	2.1	Bundler、RubyGems	aws/codebuild/eb-ruby-2.1-amazonlinux-64:2.3.2
Amazon Linux 2016.03, 64 位 v2.3.2	Ruby	2.0	Bundler、RubyGems	aws/codebuild/eb-ruby-2.0-amazonlinux-64:2.3.2
Amazon Linux 2016.03, 64 位 v2.1.6	Ruby	2.0.0	Bundler、RubyGems	aws/codebuild/eb-ruby-2.0-amazonlinux-64:2.1.6
Amazon Linux 2016.03, 64 位 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 位 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 位 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 位 v2.3.2	Ruby	1.9	Bundler、RubyGems	aws/codebuild/eb-ruby-1.9-amazonlinux-64:2.3.2
有关标识符中包含 eb- 的 Docker 映像的更多信息, 请参阅 AWS Elastic Beanstalk 开发人员指南 中的 支持的平台 和 平台历史记录 。标识符中包含 eb- 的 Docker 映像可在 Elastic Beanstalk 中可用, 但在 AWS CodeBuild 和 AWS CodePipeline 控制台中不可用。				

在 install 构建阶段, 您可以使用构建规范来安装其他组件 (例如, AWS CLI、Apache Maven、Apache Ant、Mocha、RSpec 或类似组件)。有关更多信息, 请参阅 [构建规范示例](#) (p. 112)。

AWS CodeBuild 频繁更新 Docker 映像的列表。要获取最新列表, 执行下列操作之一:

- 在 AWS CodeBuild 控制台中的 Create project 向导或 Update project 页面中, 对于 Environment image, 选择 Use an image managed by AWS CodeBuild。从 Operating system、Runtime 和 Version 下拉列表中选择。有关更多信息, 请参阅 [创建构建项目 \(控制台\)](#) (p. 154) 或 [更改构建项目的设置 \(控制台\)](#) (p. 174)。
- 在 AWS CodePipeline 控制台中, 在 Create pipeline 向导中的 Step 3: Build 页面上, 或在 Add action 或 Edit action 窗格的 AWS CodeBuild 部分中, 选择 Create a new build project。在 Environment: How to build 中, 对于 Environment image, 选择 Use an image managed by AWS CodeBuild。从 Operating system、Runtime 和 Version 下拉列表中选择。有关更多信息, 请参阅 [创建使用了 AWS CodeBuild 的管道 \(AWS CodePipeline 控制台\)](#) (p. 137) 或 [将 AWS CodeBuild 构建操作添加到管道 \(AWS CodePipeline 控制台\)](#) (p. 144)。
- 对于 AWS CLI, 请运行 list-curated-environment-images 命令:

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

- 对于 AWS 软件开发工具包, 请为您的目标编程语言调用 ListCuratedEnvironmentImages 操作。有关更多信息, 请参见 [AWS 开发工具包和工具参考](#) (p. 208)。

要确认 Docker 映像上安装的组件版本，您可以在构建过程中运行命令。输出中将显示组件的版本号。例如，您的构建规范中包含以下一个或多个命令：

- 对于 Microsoft .NET Framework，请运行 `reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP"`。
- 对于 .NET 核心，请运行 `dotnet --version`。
- 对于 Apache Ant，请运行 `ant -version`。
- 对于 Apache Maven，请运行 `mvn -version`。
- 对于 AWS CLI，请运行 `aws --version`。
- 对于 Bundler，请运行 `bundle version`。
- 对于 Git，请运行 `git --version`。
- 对于 Gradle，请运行 `gradle --version`。
- 对于 Java，请运行 `java -version`。
- 对于 NPM，请运行 `npm --version`。
- 对于 PHP，请运行 `php --version`。
- 对于 pip，请运行 `pip --version`。
- 对于 Python，请运行 `python --version`。
- 对于 RubyGems，请运行 `gem --version`。
- 对于 setuptools，请运行 `easy_install --version`。

以下构建命令 (通过 AWS CodeBuild 或 AWS CodePipeline 控制台作为构建项目设置的一部分输入) 可返回安装了这些组件的 Docker 映像上的 AWS CLI、Git、pip 和 Python 版本：`aws --version && git --version && pip --version && python --version`。

构建环境计算类型

AWS CodeBuild 为构建环境提供了以下可用内存、vCPU 和可用磁盘空间：

计算类型	computeType 值	内存	vCPU	磁盘空间
build.general1.small	BUILD_GENERAL1_SMALL	2GB	2	64GB
build.general1.medium	BUILD_GENERAL1_MEDIUM	4GB	4	128GB
build.general1.large	BUILD_GENERAL1_LARGE	8GB	8	128GB

Note

对于自定义构建环境映像，AWS CodeBuild 支持高达 10 GB 的未压缩的 Docker 映像，无论计算类型如何。要检查构建映像的大小，请使用 Docker 运行 `docker images REPOSITORY:TAG` 命令。

选择其中一个计算类型：

- 在 AWS CodeBuild 控制台中，在 Create Project 向导或 Update project 页面中，展开 Show advanced settings，然后从 Compute type 中选择一个选项。有关更多信息，请参阅 [创建构建项目 \(控制台\)](#) (p. 154) 或 [更改构建项目的设置 \(控制台\)](#) (p. 174)。
- 在 AWS CodePipeline 控制台中，在 Create Pipeline 向导的 Step 3: Build 页面上，或在 Add action 或 Edit action 窗格中，选择 Create a new build project，展开 Advanced，然后在 Compute type 中选择一个

选项。有关更多信息，请参阅 [创建使用了 AWS CodeBuild 的管道 \(AWS CodePipeline 控制台\)](#) (p. 137) 或 [将 AWS CodeBuild 构建操作添加到管道 \(AWS CodePipeline 控制台\)](#) (p. 144)。

- 对于 AWS CLI，请运行 `create-project` 或 `update-project` 命令，指定 `environment` 对象的 `computeType` 值。有关更多信息，请参阅 [创建构建项目 \(AWS CLI\)](#) (p. 162) 或 [更改构建项目的设置 \(AWS CLI\)](#) (p. 181)。
- 对于 AWS 软件开发工具包，请为您的目标编程语言调用等效于 `CreateProject` 或 `UpdateProject` 的操作，指定 `environment` 对象的 `computeType` 等效值。有关更多信息，请参见 [AWS 开发工具包和工具参考](#) (p. 208)。

构建环境中的 Shell 和命令

您为 AWS CodeBuild 提供了一组命令，用于在构建的生命周期期间 (例如，安装构建依赖项、测试和编译您的源代码) 在构建环境中运行。可通过多种方式指定这些命令：

- 创建构建规范文件并将其包含在您的源代码中。在此文件中，指定您要在构建生命周期的每个阶段运行的命令。有关更多信息，请参见 [适用于 AWS CodeBuild 的构建规范参考](#) (p. 107)。
- 使用 AWS CodeBuild 或 AWS CodePipeline 控制台创建构建项目。在 Insert build commands 中，对于 Build command，指定您要在 build 阶段运行的命令。有关更多信息，请参阅 [创建构建项目 \(控制台\)](#) (p. 154) 或 [将 AWS CodePipeline 与 AWS CodeBuild 结合使用](#) (p. 136) 中的 Insert build commands 的描述。
- 使用 AWS CodeBuild 控制台更改构建项目的设置。在 Insert build commands 中，对于 Build command，指定您要在 build 阶段运行的命令。有关更多信息，请参阅 [更改构建项目的设置 \(控制台\)](#) (p. 174) 中的 Insert build commands 的描述。
- 使用 AWS CLI 或 AWS 开发工具包创建构建项目或更改构建项目的设置。参考包含构建规范文件以及您的命令的源代码，或指定一个包含同等构建规范文件的内容的字符串。有关更多信息，请参阅 [创建构建项目](#) (p. 154) 或 [更改构建项目的设置](#) (p. 174) 中 `buildspec` 值的描述。
- 使用 AWS CLI 或 AWS 开发工具包开始构建，指定构建规范文件或一个包含同等构建规范文件内容的字符串。有关更多信息，请参阅 [运行构建项目](#) (p. 183) 中 `buildspecOverride` 值的描述。

您可以指定构建环境的默认 Shell 所支持的任何命令 (`-sh` 是辅助映像中的默认 Shell)。在构建规范版本 0.1 中，AWS CodeBuild 将在构建环境中的默认 Shell 的单独实例中运行每条命令。这表示各个命令独立于其他所有命令而运行。因此，在默认情况下，您无法运行依赖所有上一个命令状态的单个命令 (如更改目录或设置环境变量)。要绕开此限制，我们建议您使用版本 0.2 来解决此问题。如果您出于某个原因必须使用版本 0.1，我们建议使用以下方法：

- 在包含您要在默认 Shell 的单个实例中运行的命令的源代码中包含一个 Shell 脚本。例如，您可以在包含 `cd MyDir; mkdir -p mySubDir; cd mySubDir; pwd;` 等命令的源代码中包含一个名为 `my-script.sh` 的文件。然后，在您的构建规范文件中，指定命令 `./my-script.sh`。
- 在您的构建规范文件中，或对于仅针对 build 阶段的控制台的 Build command 设置，指定一条包含您想在默认 Shell 的单个实例中运行的所有命令的命令 (例如，`cd MyDir && mkdir -p mySubDir && cd mySubDir && pwd`)。

如果 AWS CodeBuild 遇到错误，那么与在默认 Shell 的实例中运行单个命令相比，错误更难排除。

构建环境中的环境变量

AWS CodeBuild 提供了您可以在构建命令中使用的多个环境变量：

- `AWS_DEFAULT_REGION`：构建运行的 AWS 区域 (例如 `us-east-1`)。此环境变量主要由 AWS CLI 使用。
- `AWS_REGION`：构建运行的 AWS 区域 (例如 `us-east-1`)。此环境变量主要由 AWS 软件开发工具包使用。

- `CODEBUILD_BUILD_ARN` : 构建的 Amazon 资源名称 (ARN) (例如, `arn:aws:codebuild:region-ID:account-ID:build/codebuild-demo-project:b1e6661e-e4f2-4156-9ab9-82a19EXAMPLE`)。
- `CODEBUILD_BUILD_ID` : 构建的 AWS CodeBuild ID (例如, `codebuild-demo-project:b1e6661e-e4f2-4156-9ab9-82a19EXAMPLE`)。
- `CODEBUILD_BUILD_IMAGE` : AWS CodeBuild 构建映像标识符 (例如, `aws/codebuild/java:openjdk-8`)。
- `CODEBUILD_BUILD_SUCCEEDING` : 无论当前构建是否成功。如果构建失败, 设置为 0 ; 如果构建成功, 设置为 1。
- `CODEBUILD_INITIATOR` : 启动构建的实体。如果 AWS CodePipeline 启动了构建, 那么这就是管道的名称, 例如 `codepipeline/my-demo-pipeline`。如果 IAM 用户启动了构建, 那么这就是用户的名称, 例如 `MyUserName`。如果适用于 AWS CodeBuild 的 Jenkins 插件启动了构建, 那么这就是字符串 `CodeBuild-Jenkins-Plugin`。
- `CODEBUILD_KMS_KEY_ID` : AWS CodeBuild 用于加密构建输出项目的 AWS KMS 密钥的标识符 (例如, `arn:aws:kms:region-ID:account-ID:key/key-ID` 或 `alias/key-alias`)。
- `CODEBUILD_LOG_PATH` : CloudWatch Logs 中有关构建的日志流名称。
- `CODEBUILD_RESOLVED_SOURCE_VERSION` : 对于 AWS CodePipeline 运行的构建, 是指要构建的源代码的提交 ID 或 Amazon S3 版本 ID。请注意, 此值仅在管道相关的源操作以 Amazon S3、AWS CodeCommit 或 GitHub 存储库为基础时可用。
- `CODEBUILD_SOURCE_REPO_URL` : 输入项目或源代码存储库的 URL。对于 Amazon S3, 这是 `s3://`, 后跟存储桶名称和输入项目的路径。对于 AWS CodeCommit 和 GitHub, 这是存储库的克隆 URL。
- `CODEBUILD_SOURCE_VERSION` : 对于 Amazon S3, 是指与输入项目相关联的版本 ID。对于 AWS CodeCommit, 是指与要生成的源代码的版本相关联的提交 ID 或分支名称。对于 GitHub, 是指与要生成的源代码的版本相关联的提交 ID、分支名称或标签名称。
- `CODEBUILD_SRC_DIR` : AWS CodeBuild 用于构建的目录路径 (例如, `/tmp/src123456789/src`)。
- `CODEBUILD_START_TIME` : 构建的开始时间。
- `HOME` : 此环境变量始终设置为 `/root`。

您也可以为构建环境提供您自己的环境变量。有关更多信息, 请参阅以下主题 :

- [将 AWS CodePipeline 与 AWS CodeBuild 结合使用 \(p. 136\)](#)
- [创建构建项目 \(p. 154\)](#)
- [更改构建项目的设置 \(p. 174\)](#)
- [运行构建项目 \(p. 183\)](#)
- [构建规范参考 \(p. 107\)](#)

要列出构建环境中的所有可用环境变量, 在构建期间, 您可以运行 `printenv` 命令 (针对基于 Linux 的构建环境) 或 `"Get-ChildItem Env:"` (针对基于 Windows 的构建环境)。除之前列出的这些变量之外, 以 `CODEBUILD_` 开始的环境变量供 AWS CodeBuild 内部使用。它们不应用于您的构建命令。

Important

我们强烈建议不要使用环境变量存储敏感值, 尤其是 AWS 访问密钥 ID 和秘密访问密钥。使用 AWS CodeBuild 控制台和 AWS CLI 等工具能够以纯文本格式显示环境变量。

对于敏感值, 我们建议您将其存储在 Amazon EC2 Systems Manager Parameter Store 中, 然后从您的构建规范中检索它们。要存储敏感值, 请参阅 Amazon EC2 Systems Manager 用户指南中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store 控制台演练](#)。要检索它们, 请参阅 [构建规范语法 \(p. 107\)](#) 中的 `parameter-store` 映射。

构建环境中的后台任务

您可以在构建环境中运行后台任务。要执行此操作，请在您的构建规范中，使用 `nohup` 命令将命令作为后台中的任务运行，即使构建过程已退出 Shell 也是如此。使用 `disown` 命令强制停止正在运行的后台任务。

示例：

- 启动后台进程并等待其稍后完成：

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

- 启动后台进程，但不等待其完成：

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

- 启动后台进程并稍后将其终止：

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

使用 AWS CodeBuild 代理在本地测试和调试

本主题提供了有关如何运行 AWS CodeBuild 代理和订阅代理新版本通知的信息。

使用 AWS CodeBuild 代理在本地计算机上测试和调试

您可以使用 AWS CodeBuild 代理在本地计算机上测试和调试构建。要使用此代理，运行以下命令：

```
docker run -it -v /var/run/docker.sock:/var/run/docker.sock -e \
    "IMAGE_NAME=amazon/aws-codebuild-local" -e \
    "ARTIFACTS=absolute-path-to-your-artifact-output-directory" -e \
    "SOURCE=absolute-path-to-your-source-directory name-of-the-agent" -e \
    "BUILDSPEC=absolute-or-relative-path-to-your-buildspec-file"
```

可从 <https://hub.docker.com/r/amazon/aws-codebuild-local/> 使用 AWS CodeBuild 代理。它的安全散列算法 (SHA) 签名为 `f030a118a72c67e263c5d252c2484664ab91bdc7e221c10f6a725dd67eaed710`。您可以通过此签名识别代理的版本。要查看代理的 SHA 签名，运行以下命令：

```
docker inspect amazon/aws-codebuild-local
```

接收有关新的 AWS CodeBuild 代理版本的通知

当有新的 AWS CodeBuild 代理发布时，Amazon SNS 会通知您。使用以下过程订阅这些通知。

订阅 AWS CodeBuild 代理通知：

- 打开 <https://console.aws.amazon.com/sns/v2/home> 中的 Amazon SNS 控制台。
- 在导航栏中，将区域改为 美国东部（弗吉尼亚北部）（如果尚未选中）。您必须选择此区域，因为您订阅的 SNS 通知就是在此区域中创建的。

3. 在导航窗格中，选择 Subscriptions。
4. 选择 Create subscription。
5. 在创建订阅对话框中：
 - a. 对于 Topic ARN，请使用以下 Amazon 资源名称 (ARN)：

```
arn:aws:sns:us-east-1:850632864840:AWS-CodeBuild-Local-Agent-Updates
```

- b. 对于协议，选择电子邮件或 SMS。
 - c. 对于终端节点，选择要接收通知的位置：
 - 如果选择电子邮件，则键入电子邮件地址。
 - 如果选择 SMS，则键入电话号码，包括区号。
 - d. 选择 Create subscription。
6. 如果选择电子邮件，则您会收到一封要求确认订阅的电子邮件。按照电子邮件中的指示完成订阅。

当有新版本的 AWS CodeBuild 代理发布时，订户会收到通知。如果您不希望再收到这些通知，请通过以下步骤取消订阅。

取消订阅 AWS CodeBuild 代理通知：

1. 打开 <https://console.aws.amazon.com/sns/v2/home> 中的 Amazon SNS 控制台。
2. 在导航窗格中，选择订阅。
3. 选择订阅，然后从操作中，选择删除订阅。请在提示您进行确认时选择 Delete。

直接运行 AWS CodeBuild

要直接借助 AWS CodeBuild 设置、运行和监控构建，您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包。

不是您要找的内容？要使用 AWS CodePipeline 运行 AWS CodeBuild，请参阅 [将 AWS CodePipeline 与 AWS CodeBuild 结合使用](#) (p. 136)。

主题

- [先决条件](#) (p. 124)
- [直接运行 AWS CodeBuild \(控制台\)](#) (p. 124)
- [直接运行 AWS CodeBuild \(AWS CLI\)](#) (p. 124)

先决条件

回答 [规划构建](#) (p. 106) 中的问题。

直接运行 AWS CodeBuild (控制台)

1. 创建构建项目。有关信息，请参阅 [创建构建项目 \(控制台\)](#) (p. 154)。
2. 运行构建。有关信息，请参阅 [运行构建项目 \(控制台\)](#) (p. 183)。
3. 获取有关构建的信息。有关信息，请参阅 [查看构建详细信息 \(控制台\)](#) (p. 190)。

直接运行 AWS CodeBuild (AWS CLI)

有关将 AWS CLI 与 AWS CodeBuild 结合使用的更多信息，请参阅 [命令行参考](#) (p. 207)。

1. 创建构建项目。有关信息，请参阅 [创建构建项目 \(AWS CLI\)](#) (p. 162)。
2. 运行构建。有关信息，请参阅 [运行构建项目 \(AWS CLI\)](#) (p. 186)。
3. 获取有关构建的信息。有关信息，请参阅 [查看构建详细信息 \(AWS CLI\)](#) (p. 190)。

将 AWS CodeBuild 与 Amazon Virtual Private Cloud 结合使用

通常，AWS CodeBuild 无法访问 VPC 中的资源。要支持访问，您必须提供额外的 VPC 特定配置信息以作为 AWS CodeBuild 项目配置的一部分。这包括 VPC ID、VPC 子网 ID 和 VPC 安全组 ID。支持 VPC 的构建随后就可以访问 VPC 内的资源。有关在 Amazon VPC 中设置 VPC 的更多信息，请参阅 [VPC 用户指南](#)。

主题

- [使用案例 \(p. 125\)](#)
- [在您的 AWS CodeBuild 项目中启用 Amazon VPC 访问 \(p. 125\)](#)
- [VPC 的最佳实践 \(p. 126\)](#)
- [VPC 设置问题排查 \(p. 127\)](#)
- [使用 VPC 终端节点 \(p. 127\)](#)
- [AWS CloudFormation VPC 模板 \(p. 128\)](#)

使用案例

来自 AWS CodeBuild 构建的 VPC 连接使以下操作成为可能：

- 通过您的构建对私有子网上隔离的 Amazon RDS 数据库中的数据运行集成测试。
- 直接通过测试查询 Amazon ElastiCache 集群中的数据。
- 与托管于 Amazon EC2、Amazon ECS 或使用内部 Elastic Load Balancing 的服务上的内部 Web 服务交互。
- 从自托管的内部项目存储库 (如适用于 Python 的 PyPI、适用于 Java 的 Maven 和适用于 Node.js 的 npm) 检索依赖项。
- 访问 Amazon S3 存储桶中配置为仅允许通过 Amazon VPC 终端节点访问的对象。
- 通过与您的子网关联的 NAT 网关或 NAT 实例的弹性 IP 地址查询需要固定 IP 地址的外部 Web 服务。

您的构建可以访问您的 VPC 中托管的任何资源。

在您的 AWS CodeBuild 项目中启用 Amazon VPC 访问

在您的 VPC 配置中包含以下设置：

- 对于 VPC ID，选择 AWS CodeBuild 使用的 VPC ID。
- 对于 Subnets，选择包含 AWS CodeBuild 使用的资源的子网。
- 对于 Security Groups，选择 AWS CodeBuild 用来支持对 VPC 中资源的访问的安全组。

创建构建项目 (控制台)

有关创建构建项目的信息，请参阅[创建构建项目 \(控制台\)](#) (p. 154)。当您创建或更改 AWS CodeBuild 项目时，请在 VPC 中，选择您的 VPC ID、子网和安全组。

创建构建项目 (AWS CLI)

有关创建构建项目的信息，请参阅[创建构建项目 \(AWS CLI\)](#) (p. 162)。如果您要将 AWS CLI 与 AWS CodeBuild 结合使用，则 AWS CodeBuild 用来代表 IAM 用户与服务交互的服务器角色必须已附加以下策略：[允许 AWS CodeBuild 访问创建 VPC 网络接口时所需的 AWS 服务](#) (p. 220)。

`vpcConfig` 对象应包含您的 `vpcId`、`securityGroupIds` 和 `subnets`。

- `vpcId`：必填值。AWS CodeBuild 使用的 VPC ID。要获取您的区域中所有 Amazon VPC ID 的列表，请运行以下命令：

```
aws ec2 describe-vpcs
```

- `subnets`：必填值。包含 AWS CodeBuild 使用的资源的子网 ID。要获取这些 ID，请运行以下命令：

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

Note

将 us-east-1 替换为您的区域。

- `securityGroupIds`：必填值。AWS CodeBuild 用来支持对 VPC 中的资源的访问的安全组 ID。要获取这些 ID，请运行以下命令：

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

Note

将 us-east-1 替换为您的区域。

VPC 的最佳实践

在设置 VPC 以与 AWS CodeBuild 结合使用时，请使用此核对清单。

- 设置具有公有和私有子网以及一个 NAT 网关的 VPC。有关更多信息，请参阅[具有公有和私有子网 \(NAT\) 的 VPC](#)。

Important

您需要一个 NAT 网关或 NAT 实例以便将 AWS CodeBuild 与您的 Amazon VPC 结合使用，从而使 AWS CodeBuild 能够访问公有终端节点 (例如，在运行构建时执行 CLI 命令)。您不能使用 Internet 网关代替 NAT 网关或 NAT 实例，因为 AWS CodeBuild 不支持将弹性 IP 地址分配给其创建的网络接口，并且 Amazon EC2 不支持为在 Amazon EC2 实例启动之外创建的任何网络接口自动分配公有 IP 地址。

- 将多个可用区包含在您的 VPC 中。
- 确保您的安全组不允许入站 (入口) 流量流至您的构建。有关更多信息，请参阅[安全组规则](#)。
- 为您的构建设置单独的子网。
- 当您设置 AWS CodeBuild 项目以访问 VPC 时，请仅选择私有子网。

有关在 Amazon VPC 中设置 VPC 的更多信息，请参阅[Amazon VPC 用户指南](#)。

有关借助 AWS CloudFormation 将 Amazon VPC 配置为使用 AWS CodeBuild VPC 功能的更多信息，请参阅[AWS CloudFormation VPC 模板](#) (p. 128)。

VPC 设置问题排查

当您排查 VPC 问题时，请使用错误消息中显示的信息帮助您发现、诊断和解决问题。

下面是一些帮助您排查常见的 AWS CodeBuild VPC 错误“Build does not have internet connectivity.Please check subnet network configuration”的准则。

1. 确保您的 [Internet 网关](#) 已连接到 VPC。
2. 确保您的公有子网的路由表指向 [Internet 网关](#)。
3. 确保您的网络 ACL 允许流量流动。
4. 确保您的安全组允许流量流动。
5. 对 NAT 网关进行故障排除。
6. 确保私有子网的路由表指向 NAT 网关。
7. 确保 AWS CodeBuild 用来代表 IAM 用户与服务交互的服务器角色具有[此策略](#)中的权限。有关更多信息，请参阅 [创建 AWS CodeBuild 服务角色 \(p. 202\)](#)。

如果 AWS CodeBuild 缺少权限，您可能会收到如下错误：“Unexpected EC2 error: UnauthorizedOperation”(意外的 EC2 错误：UnauthorizedOperation)。如果 AWS CodeBuild 没有使用 Amazon VPC 时所需的 Amazon EC2 权限，则会出现此错误。

使用 VPC 终端节点

您可以通过将 AWS CodeBuild 配置为使用接口 VPC 终端节点来提高构建的安全性。接口终端节点由 PrivateLink 提供技术支持，该技术使您能够通过使用私有 IP 地址私下访问 Amazon EC2 和 AWS CodeBuild。PrivateLink 将托管实例、AWS CodeBuild 和 Amazon EC2 之间的所有网络流量限制在 Amazon 网络内 (托管实例无法访问 Internet)。而且，您无需 Internet 网关、NAT 设备或虚拟专用网关。不要求您配置 PrivateLink，但推荐进行配置。有关 PrivateLink 和 VPC 终端节点的更多信息，请参阅[通过 PrivateLink 访问 AWS 服务](#)。

在您创建 VPC 终端节点前

在配置 AWS CodeBuild 的 VPC 端点之前，请注意以下限制。

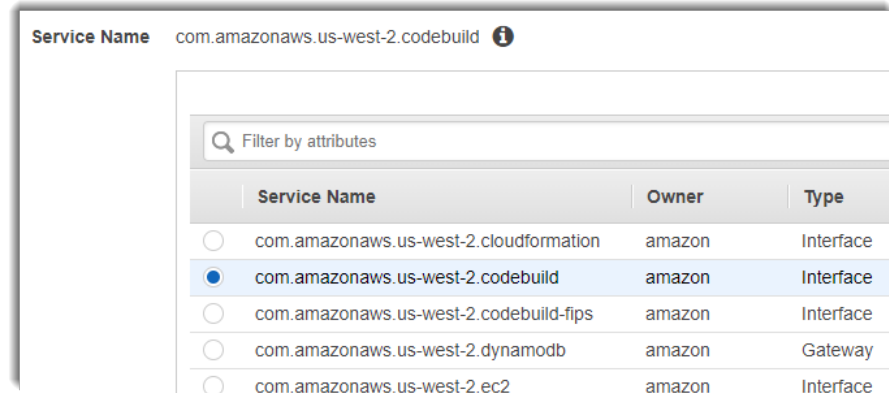
Note

以下服务必须与 Internet 进行通信。利用 [Amazon VPC NAT 网关](#)，您可以将 VPC 终端节点与 AWS CodeBuild 以及这些服务结合使用。

- AWS CodeCommit，它可能是一个源存储库。
- Amazon ECR，它可用于自定义 Docker 映像。
- Active Directory.
- Amazon CloudWatch Events 和 Amazon CloudWatch Logs。
- VPC 终端节点仅通过 Amazon Route 53 支持 Amazon 提供的 DNS。如果您希望使用自己的 DNS，可以使用条件 DNS 转发。有关更多信息，请参阅“Amazon VPC 用户指南”中的 [DHCP 选项集](#)。
- VPC 终端节点当前不支持跨区域请求。确保您在存储构建输入和输出的任何 Amazon S3 存储桶所在的区域内创建终端节点。您可以使用 Amazon S3 控制台或 [get-bucket-location](#) 命令来查找存储桶的位置。使用区域特定的 Amazon S3 终端节点访问存储桶；例如，mybucket.s3-us-west-2.amazonaws.com。有关 Amazon S3 的区域特定的终端节点的更多信息，请参阅 Amazon Web Services 一般参考 中的 [Amazon Simple Storage Service \(Amazon S3\)](#)。如果您使用 AWS CLI 向 Amazon S3 发起请求，请将默认区域设置为您的存储桶所在的区域，或在请求中使用 `--region` 参数。

创建 AWS CodeBuild 的 VPC 终端节点

使用 [创建接口终端节点](#) 来创建终端节点 `com.amazonaws.region.codebuild`。这是 AWS CodeBuild 服务的 VPC 终端节点。



region 表示 AWS CodeBuild 支持的 AWS 区域的区域标识符，例如美国东部（俄亥俄州）区域的 `us-east-2`。有关支持的 **region** 值的列表，请参阅 AWS 一般参考 中的 [区域和终端节点的 AWS CodeBuild 表](#) 中的“区域”列。使用您在登录到 AWS 时指定的区域来预填充终端节点。如果您更改自己的区域，则 VPC 终端节点将使用新区域进行更新。

AWS CloudFormation VPC 模板

AWS CloudFormation 使您能够预见性地反复创建和配置 AWS 基础设施部署，方式是使用模板文件批量创建和删除一系列资源的集合（视为一个堆栈）。有关更多信息，请参阅 [AWS CloudFormation 用户指南](#)。

下面是用于配置 Amazon VPC 以使用 AWS CodeBuild VPC 功能的 AWS CloudFormation YAML 模板。它可从 https://s3.amazonaws.com/codebuild-cloudformation-templates-public/vpc_cloudformation_template.yml 下载。

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

将 AWS CodePipeline 与 AWS CodeBuild 结合使用以测试代码和运行构建

通过使用 AWS CodePipeline 测试您的代码并借助 AWS CodeBuild 运行构建，您可以自动执行发布流程。

下表列出了可用于执行这些操作的任务和方法。本主题不介绍如何使用 AWS 开发工具包完成这些任务。

任务	可用方法	本主题中介绍的方法
借助 AWS CodePipeline 创建可使用 AWS CodeBuild 自动运行构建的持续交付 (CD) 管道	<ul style="list-style-type: none">• AWS CodePipeline 控制台• AWS CLI• AWS 软件开发工具包	<ul style="list-style-type: none">• 使用 AWS CodePipeline 控制台 (p. 137)• 使用 AWS CLI (p. 141)• 您可以调整本主题中的信息以使用 AWS 开发工具包。有关更多信息，请参阅适用于 Amazon Web Services 的工具内软件开发工具包部分中您的编程语言对应的创建管道操作文档，或参阅 AWS CodePipeline API 参考 中的 CreatePipeline。
将借助 AWS CodeBuild 实现的测试和构建自动化添加到 AWS CodePipeline 中现有的管道	<ul style="list-style-type: none">• AWS CodePipeline 控制台• AWS CLI• AWS 软件开发工具包	<ul style="list-style-type: none">• 通过使用 AWS CodePipeline 控制台添加构建自动化 (p. 144)• 通过使用 AWS CodePipeline 控制台添加测试自动化 (p. 148)• 对于 AWS CLI，您可以调整本主题中的信息，以创建包含 AWS CodeBuild 构建操作或测试操作的管道。有关更多信息，请参阅 AWS CodePipeline 用户指南 中的编辑管道 (AWS CLI) 和 AWS CodePipeline 管道结构参考。• 您可以调整本主题中的信息以使用 AWS 开发工具包管道。有关更多信息，请参阅适用于 Amazon Web Services 的工具内软件开发工具包部分中您的编程语言对应的更新管道操作文档，或参阅 AWS CodePipeline API 参考 中的 UpdatePipeline。

主题

- [先决条件 \(p. 136\)](#)
- [创建使用了 AWS CodeBuild 的管道 \(AWS CodePipeline 控制台\) \(p. 137\)](#)
- [创建使用 AWS CodeBuild 的管道 \(AWS CLI\) \(p. 141\)](#)
- [将 AWS CodeBuild 构建操作添加到管道 \(AWS CodePipeline 控制台\) \(p. 144\)](#)
- [将 AWS CodeBuild 测试操作添加到管道 \(AWS CodePipeline 控制台\) \(p. 148\)](#)

先决条件

1. 回答 [规划构建 \(p. 106\)](#) 中的问题。
2. 如果您通过 IAM 用户 (而不是 AWS 根账户或管理员 IAM 用户) 访问 AWS CodePipeline，请向该用户 (或该用户所属的 IAM 组) 附加名为 `AWSCodePipelineFullAccess` 的托管策略。(不建议使用 AWS 根账户。)这样，用户即可在 AWS CodePipeline 中创建管道。有关更多信息，请参阅 IAM 用户指南中的[附加托管策略](#)。

Note

向该用户 (或该用户所属的 IAM 组) 附加策略的 IAM 实体在 IAM 中必须拥有附加策略的权限。
有关更多信息, 请参阅 IAM 用户指南中的 [委派权限来管理 IAM 用户、组和凭证](#)。

- 如果您的 AWS 账户中还没有 AWS CodePipeline 服务角色, 请创建一个。借助此服务角色, AWS CodePipeline 可代表您与其他 AWS 服务进行交互, 包括 AWS CodeBuild。例如, 要通过使用 AWS CLI 创建 AWS CodePipeline 服务角色, 请运行 `iam create-role` 命令:

对于 Linux, macOS, or Unix :

```
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"}]}'
```

对于 Windows :

```
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

创建此 AWS CodePipeline 服务角色的 IAM 实体必须拥有在 IAM 中创建服务角色的权限。

- 在您创建一个 AWS CodePipeline 服务角色或识别现有服务角色后, 您必须向其添加策略声明。按 AWS CodePipeline 用户指南中的 [查看默认 AWS CodePipeline 服务角色策略](#) 中所述, 将默认 AWS CodePipeline 服务角色策略添加到服务角色。

Note

添加此 AWS CodePipeline 服务角色策略的 IAM 实体必须拥有在 IAM 中将服务角色策略添加到服务角色的权限。

- 创建源代码并将其上传到 AWS CodeBuild 和 AWS CodePipeline 支持的存储库类型, 如 AWS CodeCommit、Amazon S3 或 GitHub。(AWS CodePipeline 目前不支持 Bitbucket。)确保源代码包含构建规范文件 (或者, 在本主题后面定义构建项目时, 您可以声明构建规范), 它提供了用于构建源代码的说明。有关更多信息, 请参见 [构建规范参考](#) (p. 107)。

Important

如果您计划使用管道来部署已构建的源代码, 那么构建输出项目必须与您将使用的部署系统兼容。

- 有关 AWS CodeDeploy, 请参阅本指南中的 [AWS CodeDeploy 示例](#) (p. 75), 并参阅 AWS CodeDeploy 用户指南中的 [准备 AWS CodeDeploy 的修订](#)。
- 对于 AWS Elastic Beanstalk, 请参阅本指南中的 [Elastic Beanstalk 示例](#) (p. 79), 并参阅 AWS Elastic Beanstalk 开发人员指南中的 [创建应用程序源包](#)。
- 对于 AWS OpsWorks, 请参阅 AWS OpsWorks 用户指南中的 [应用程序源](#)和 [将 AWS CodePipeline 与 AWS OpsWorks 结合使用](#)。

创建使用了 AWS CodeBuild 的管道 (AWS CodePipeline 控制台)

执行以下步骤, 创建使用 AWS CodeBuild 来构建和部署源代码的管道。

要创建仅测试源代码的管道，您的选项包括：

- 执行以下步骤来创建管道，然后从管道中删除构建和测试阶段。然后使用本主题中的 [将 AWS CodeBuild 测试操作添加到管道 \(AWS CodePipeline 控制台\) \(p. 148\)](#) 步骤，将使用 AWS CodeBuild 的测试操作添加到管道。
- 使用本主题中的其他步骤之一来创建管道，然后使用本主题中的 [将 AWS CodeBuild 测试操作添加到管道 \(AWS CodePipeline 控制台\) \(p. 148\)](#) 步骤，将使用 AWS CodeBuild 的测试操作添加到管道。

使用 AWS CodePipeline 中的创建管道向导来创建使用 AWS CodeBuild 的管道

1. 完成 [先决条件 \(p. 136\)](#) 中的步骤。
2. 通过以下网址打开 AWS CodePipeline 控制台：<https://console.aws.amazon.com/codepipeline>。

您需要通过使用以下方式之一登录 AWS 管理控制台：

- 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅 IAM 用户指南中的[账户根用户](#)。
- AWS 账户中的 IAM 管理员用户。有关更多信息，请参阅 IAM 用户指南中的[创建您的第一个 IAM 管理员用户和组](#)。
- 您的 AWS 账户中的 IAM 用户有权使用以下最低程度的操作：

```
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. 在 AWS 区域选择器中，选择您的管道和相关的 AWS 资源所在的区域。此区域还必须支持 AWS CodeBuild。有关更多信息，请参阅 Amazon Web Services 一般参考 中的“区域和终端节点”主题中的 [AWS CodeBuild](#)。
4. 创建如下所示的管道：

如果显示欢迎页面，请选择 Get started (开始使用)。

如果显示 All Pipelines 页面，请选择 Create pipeline。

5. 在 Step 1: Name 页面上，对于 Pipeline name，键入管道的名称；例如，**CodeBuildDemoPipeline**。如果您选择了其他名称，请用该名称替换掉整个过程中对应的名称。选择下一步。
6. 在 Step 2: Source 页面上，对于 Source provider，执行下列操作之一：
 - 如果您的源代码存储在 Amazon S3 存储桶中，请选择 Amazon S3。对于 Amazon S3 location，请使用格式 `s3://bucket-name/path/to/file-name.zip` 键入源代码的路径。选择下一步。

- 如果您的源代码存储在 AWS CodeCommit 存储库中，请选择 AWS CodeCommit。对于 Repository name，请选择包含源代码的存储库的名称。对于 Branch name，请选择表示要构建的源代码版本的分支名称。选择下一步。
 - 如果您的源代码存储在 GitHub 存储库中，请选择 GitHub。选择 Connect to GitHub，然后按照说明进行操作以通过 GitHub 进行身份验证。对于 Repository，请选择包含源代码的存储库的名称。对于 Branch，请选择表示您要构建的源代码版本的分支名称。选择下一步。
7. 在 Step 3: Build 页面上，对于 Build provider，选择 AWS CodeBuild。
 8. 如果您已拥有要使用的构建项目，请选择 Select an existing build project。对于 Project name，请选择构建项目的名称，然后前往本过程的步骤 17。

Note

如果您选择一个现有的构建项目，那么它必须具有已定义的构建输出项目设置（即使 AWS CodePipeline 将覆盖它们）。有关更多信息，请参阅[创建构建项目 \(控制台\) \(p. 154\)](#) 或 [更改构建项目的设置 \(控制台\) \(p. 174\)](#) 中的 Artifacts: Where to put the artifacts from this build project 的描述。

Important

如果您为 AWS CodeBuild 项目启用 Webhook，并且该项目用作 AWS CodePipeline 中的构建步骤，则将为每次提交创建两个相同的构建。一个构建通过 Webhook 触发，另一个构建通过 AWS CodePipeline 触发。由于账单基于每个构建，因此您需要为这两个构建付费。因此，如果您使用的是 AWS CodePipeline，建议您在 CodeBuild 中禁用 Webhook。在 AWS CodeBuild 控制台中，清除 Webhook 框。有关更多信息，请参阅[更改构建项目的设置 \(控制台\) \(p. 174\)](#) 中的步骤 9。

9. 选择 Create a new build project。
10. 对于 Project name，键入此构建项目的名称。构建项目名称在您的各个 AWS 账户内必须是唯一的。
11. (可选) 在 Description 框中键入描述。
12. 对于 Environment image，执行下列操作之一：
 - 要使用基于由 AWS CodeBuild 管理的 Docker 映像的构建环境，请选择 Use an image managed by AWS CodeBuild。从 Operating system、Runtime 和 Version 下拉列表中进行选择。有关更多信息，请参阅 [AWS CodeBuild 提供的 Docker 映像 \(p. 114\)](#)。
 - 要使用基于 AWS 账户内 Amazon ECR 存储库中的 Docker 映像的构建环境，请选择 Specify a Docker image。对于 Custom image type，选择 Amazon ECR。使用 Amazon ECR repository 和 Amazon ECR image 下拉列表指定所需的 Amazon ECR 存储库以及该存储库中的 Docker 映像。
 - 要使用基于 Docker Hub 中公开可用的 Docker 映像的构建环境，请选择 Specify a Docker image。对于 Custom image type，选择 Other。在 Custom image ID 框中，使用格式 `docker-repo-name/docker-image-name:tag` 键入 Docker 映像 ID。
13. 对于 Build specification，执行下列操作之一：
 - 如果您的源代码包含构建规范文件，请选择 Use the buildspec.yml in the source code root directory (在源代码根目录中使用 buildspec.yml)。
 - 如果您的源代码不包含构建规范文件，请选择 Insert build commands。对于 Build command (构建命令)，请键入要在构建阶段在构建环境中运行的命令；对于多条命令，针对基于 Linux 的构建环境，请使用 && 将每条命令分隔开；针对基于 Windows 的构建环境，请使用 ; 将每条命令分隔开。对于 Output files，请键入要发送给 AWS CodePipeline 的构建环境中的构建输出文件路径；对于多个文件，请使用逗号将每个文件路径分隔开。有关更多信息，请参阅控制台中的工具提示。
14. 对于 AWS CodeBuild service role，执行下列操作之一：
 - 如果您的 AWS 账户中没有 AWS CodeBuild 服务角色，请选择 Create a service role in your account。在 Role name 框中，键入服务角色的名称，或保留建议的名称。(服务角色名称在您的 AWS 账户内必须是唯一的。)

Note

如果您使用控制台创建 AWS CodeBuild 服务角色，那么默认情况下，这个服务角色仅能与此构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，那么此角色将更新，以便与关联的构建项目配合使用。单个 AWS CodeBuild 服务角色最多可与十个构建项目配合使用。

- 如果您的 AWS 账户中有 AWS CodeBuild 服务角色，请选择 Choose an existing service role from your account。在 Role name 框中，选择服务角色的名称。

15. 展开 Advanced。

要指定 60 分钟 (默认值) 以外的构建超时时间，请使用 hours 和 minutes 框设置一个介于 5 和 480 分钟 (8 小时) 之间的超时时间。

仅当您计划使用此构建项目来构建 Docker 映像，并且您选择的构建环境映像不是由支持 Docker 的 AWS CodeBuild 提供的映像时，选中 Privileged 复选框。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。请注意，您还必须启动 Docker 守护程序，以便您的构建能够根据需要与其交互。执行此操作的一种方法是通过运行以下构建命令在您的构建规范的 install 阶段初始化 Docker 守护程序。(如果选择了由支持 Docker 的 AWS CodeBuild 提供的构建环境映像，请不要运行以下构建命令。)

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --  
host=tcp://127.0.0.1:2375 --storage-driver=overlay&  
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

对于 Compute，请选择一个可用选项。

对于 Environment variables，请使用 Name 和 Value 为要使用的构建环境指定任何可选的环境变量。要添加更多环境变量，请选择 Add row。

Important

我们强烈建议不要使用环境变量存储敏感值，尤其是 AWS 访问密钥 ID 和秘密访问密钥。使用 AWS CodeBuild 控制台和 AWS CLI 等工具能够以纯文本格式显示环境变量。

要存储和检索敏感值，我们建议您的构建命令使用 AWS CLI 来与 Amazon EC2 Systems Manager Parameter Store 进行交互。AWS CLI 已在由 AWS CodeBuild 提供的所有构建环境上预安装和预配置。有关更多信息，请参阅 Amazon EC2 Systems Manager 用户指南 中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store CLI 演练](#)。

16. 选择 Save build project。

17. 在保存构建项目后，请选择 Next step。

18. 在 Step 4: Deploy 页面上，执行下列操作之一：

- 如果您不想部署构建输出项目，对于 Deployment provider，请选择 No Deployment。
- 如果您要部署构建输出项目，对于 Deployment provider，请选择部署提供程序，然后在系统提示时指定设置。

19. 选择下一步。

20. 在 Step 5: Service Role 页面上，对于 Role name，请选择您作为本主题的先决条件的一部分创建或标识的 AWS CodePipeline 服务角色。

请勿使用此页面创建 AWS CodePipeline 服务角色。如果您这样做，服务角色将不会拥有与 AWS CodeBuild 配合使用所需的权限。

21. 选择下一步。

22. 在 Step 6: Review 页面上，选择 Create pipeline。

23. 管道成功运行后，您可以获取构建输出项目。管道在 AWS CodePipeline 控制台中显示后，在 Build 操作中，将鼠标指针停留在工具提示上。记下 Output artifact 的值 (例如，MyAppBuild)。

Note

您还可以通过在 AWS CodeBuild 控制台的构建详细信息页面上选择 Build artifacts 链接来获取构建输出项目。要前往此页面，请跳过此过程中的剩余步骤，并参阅 [查看构建详细信息 \(控制台\)](#) (p. 190)。

24. 通过以下网址打开 Amazon S3 控制台：<https://console.aws.amazon.com/s3/>。
25. 在存储桶列表中，请打开管道使用的存储桶。此存储桶的名称应遵循格式 `codepipeline-region-ID-random-number`。您可以使用 AWS CLI 运行 AWS CodePipeline `get-pipeline` 命令，以获取存储桶的名称，其中，*my-pipeline-name* 是管道的显示名称：

```
aws codepipeline get-pipeline --name my-pipeline-name
```

在输出中，该 `pipeline` 对象包含一个 `artifactStore` 对象，其中包含带有存储桶名称的 `location` 值。

26. 打开与您的管道名称匹配的文件夹 (根据管道名称的长度，文件夹名称可能被截断)，然后打开与您在此过程的步骤 23 中记录的 Output artifact 的值匹配的文件夹。
27. 提取文件内容。如果该文件夹中有多个文件，请提取具有最新 Last Modified 时间戳的文件的内容。(您可能需要为文件提供 `.zip` 扩展名，这样，您可以将其用于您系统内的 ZIP 实用工具。)构建输出项目将位于文件的提取内容中。
28. 如果您指示 AWS CodePipeline 部署构建输出项目，请使用部署提供商的说明，获取部署目标上的构建输出项目。

创建使用 AWS CodeBuild 的管道 (AWS CLI)

执行以下步骤，创建使用 AWS CodeBuild 来构建源代码的管道。

要使用 AWS CLI 创建可部署已生成的源代码或仅测试源代码的管道，您可以调整 AWS CodePipeline 用户指南中的 [编辑管道 \(AWS CLI\)](#) 和 [AWS CodePipeline 管道结构参考](#)。

1. 完成 [先决条件](#) (p. 136) 中的步骤。
2. 创建或标识 AWS CodeBuild 中的构建项目。有关更多信息，请参阅 [创建构建项目](#) (p. 154)。

Important

构建项目必须定义构建输出项目设置 (即使 AWS CodePipeline 将覆盖它们)。有关更多信息，请参阅 [创建构建项目 \(AWS CLI\)](#) (p. 162) 中 `artifacts` 的描述。

3. 请确保您已使用与本主题中所述的 IAM 实体之一对应的 AWS 访问密钥和 AWS 秘密访问密钥配置 AWS CLI。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的 [开始设置 AWS Command Line Interface](#)。
4. 创建代表管道结构的 JSON 格式的文件。将文件命名为 `create-pipeline.json` 或类似名称。例如，此 JSON 格式的结构借助引用了 Amazon S3 输入存储桶的源操作和使用 AWS CodeBuild 的构建操作创建了管道：

```
{
  "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
}
}
```

在此 JSON 格式的数据中：

- `roleArn` 的值必须与您作为先决条件的一部分创建或标识的 AWS CodePipeline 服务角色 ARN 相匹配。
- `configuration` 中 `S3Bucket` 和 `S3ObjectKey` 的值假定源代码存储在 Amazon S3 存储桶中。有关其他源代码存储库类型的设置，请参阅 AWS CodePipeline 用户指南中的 [AWS CodePipeline 管道结构参考](#)。
- `ProjectName` 的值是您之前在此过程中创建的 AWS CodeBuild 构建项目的名称。

- `location` 的值是此管道所用的 Amazon S3 存储桶的名称。有关更多信息，请参阅 AWS CodePipeline 用户指南中的[为用作 AWS CodePipeline 项目存储的 Amazon S3 存储桶创建策略](#)。
- `name` 的值是此管道的名称。所有管道名称对您的账户都必须是唯一的。

尽管此数据仅介绍源操作和构建操作，但您可以为与测试、部署构建输出项目和调用 AWS Lambda 函数等相关的活动添加操作。有关更多信息，请参阅 AWS CodePipeline 用户指南中的[AWS CodePipeline 管道结构参考](#)。

5. 切换到包含 JSON 文件的文件夹，然后运行 AWS CodePipeline `create-pipeline` 命令，并指定文件名：

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

Note

您必须在支持 AWS CodeBuild 的 AWS 区域中创建管道。有关更多信息，请参阅 Amazon Web Services 一般参考 中的“区域和终端节点”主题中的[AWS CodeBuild](#)。

输出中将显示 JSON 格式的数据，并且 AWS CodePipeline 会创建管道。

6. 要获取有关管道状态的信息，请运行 AWS CodePipeline `get-pipeline-state` 命令，指定管道名称：

```
aws codepipeline get-pipeline-state --name my-pipeline-name
```

在输出中，查找确认构建成功的信息。省略号 (...) 用于显示为简洁起见而省略的数据。

```
{
  ...
  "stageStates": [
    ...
    {
      "actionStates": [
        {
          "actionName": "AWS CodeBuild",
          "latestExecution": {
            "status": "SUCCEEDED",
            ...
          },
          ...
        }
      ]
    }
  ]
}
```

如果您过早运行此命令，您可能不会看到有关构建操作的信息。您可能需要多次运行此命令，直到管道已完成构建操作的运行。

7. 成功构建后，请按照以下说明操作，获取构建输出项目。通过以下网址打开 Amazon S3 控制台：<https://console.aws.amazon.com/s3/>。

Note

您还可以通过在 AWS CodeBuild 控制台的相关构建详细信息页面上选择 Build artifacts 链接来获取构建输出项目。要前往此页面，请跳过此过程中的剩余步骤，并参阅[查看构建详细信息 \(控制台\)](#) (p. 190)。

- 在存储桶列表中，请打开管道使用的存储桶。此存储桶的名称应遵循格式 `codepipeline-region-ID-random-number`。您可以从 `create-pipeline.json` 文件中获取存储桶的名称，或您可以通过运行 AWS CodePipeline `get-pipeline` 命令获取该存储桶的名称。

```
aws codepipeline get-pipeline --name my-pipeline-name
```

在输出中，该 `pipeline` 对象包含一个 `artifactStore` 对象，其中包含带有存储桶名称的 `location` 值。

- 打开与您的管道名称相匹配的文件夹 (例如，`my-pipeline-name`)。
- 在该文件夹中，打开名为 `default` 的文件夹。
- 提取文件内容。如果该文件夹中有多个文件，请提取具有最新 Last Modified 时间戳的文件的内容。(您可能需要为文件提供 `.zip` 扩展名，这样，您可以将其用于您系统内的 ZIP 实用工具。)构建输出项目将位于文件的提取内容中。

将 AWS CodeBuild 构建操作添加到管道 (AWS CodePipeline 控制台)

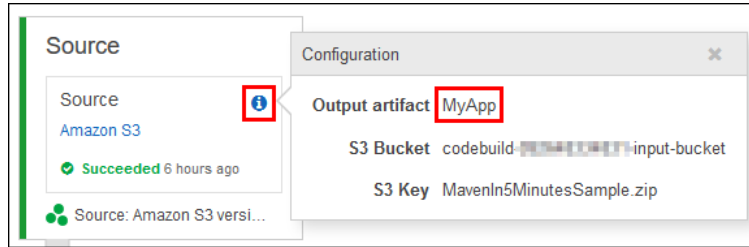
- 通过以下网址打开 AWS CodePipeline 控制台：<https://console.aws.amazon.com/codepipeline>。

您应该已使用以下任一身份登录到 AWS 管理控制台：

- 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅 IAM 用户指南中的[账户根用户](#)。
- AWS 账户中的 IAM 管理员用户。有关更多信息，请参阅 IAM 用户指南中的[创建您的第一个 IAM 管理员用户和组](#)。
- AWS 账户中的 IAM 用户，具有执行以下最基本操作的权限：

```
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
```

- 在 AWS 区域选择器中，请选择管道所在的区域。此区域还必须支持 AWS CodeBuild。有关更多信息，请参阅 Amazon Web Services 一般参考 中的“区域和终端节点”主题中的 [AWS CodeBuild](#)。
- 在 All Pipelines 页面上，选择管道名称。
- 在管道详细信息页面上，在 Source 操作中，请将鼠标指针停留在工具提示上。记下 Output artifact 的值 (例如，MyApp)：



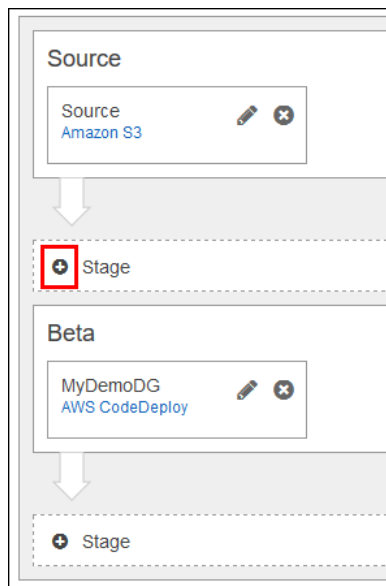
Note

此过程假定您要将构建操作添加到 Source 和 Beta 阶段之间的构建阶段内。如果您要在其他位置添加构建操作，请将鼠标指针停留在您要添加构建操作的位置之前，并记下 Output artifact 的值。

5. 选择 Edit。
6. 在 Source 和 Beta 阶段之间，选择 Stage 旁的添加符号 (+)。

Note

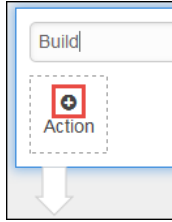
此过程假设您要向您的管道添加新的构建阶段。要将构建操作添加到现有的阶段，请选择现有阶段中的编辑 (铅笔) 图标，然后跳到此过程的步骤 8。
此过程还假定您要在 Source 和 Beta 阶段之间添加构建阶段。要在其他位置添加构建阶段，请在所需位置选择添加符号。



7. 对于 Enter stage name，键入构建阶段的名称 (例如，**Build**)。如果您选择了其他名称，请在整个过程中使用该名称。
8. 在选定阶段内，选择 Action 旁的添加符号 (+)。

Note

此过程假定您要在构建阶段内添加构建操作。要在其他位置添加构建操作，请在所需位置选择添加符号。您可能需要先在您要添加构建操作的现有阶段内选择编辑 (铅笔) 图标。



9. 在 Add action 窗格中，对于 Action category，选择 Build。
10. 在 Build actions 中，对于 Action name，键入操作的名称 (例如，**AWS CodeBuild**)。如果您选择了其他名称，请在整个过程中使用该名称。
11. 对于 Build provider，选择 AWS CodeBuild。
12. 如果您在 AWS CodeBuild 中有构建项目，请选择 Select an existing build project。对于 Project name，请选择构建项目的名称，然后向前跳至本过程的步骤 21。

Note

如果您选择一个现有的构建项目，那么它必须具有已定义的构建输出项目设置 (即使 AWS CodePipeline 将覆盖它们)。有关更多信息，请参阅[创建构建项目 \(控制台\) \(p. 154\)](#) 或[更改构建项目的设置 \(控制台\) \(p. 174\)](#) 中的 Artifacts: Where to put the artifacts from this build project 的描述。

Important

如果您为 AWS CodeBuild 项目启用 Webhook，并且该项目用作 AWS CodePipeline 中的构建步骤，则将为每次提交创建两个相同的构建。一个构建通过 Webhook 触发；另一个构建通过 AWS CodePipeline 触发。由于账单基于每个构建，因此您需要为这两个构建付费。因此，如果您使用的是 AWS CodePipeline，建议您在 CodeBuild 中禁用 Webhook。在 AWS CodeBuild 控制台中，取消选中 Webhook 框。有关更多信息，请参阅[更改构建项目的设置 \(控制台\) \(p. 174\)](#) 中的步骤 9。

13. 选择 Create a new build project。
14. 对于 Project name，键入此构建项目的名称。构建项目名称在您的各个 AWS 账户内必须是唯一的。
15. (可选) 在 Description 框中键入描述。
16. 对于 Environment image，执行下列操作之一：
 - 要使用基于由 AWS CodeBuild 管理的 Docker 映像的构建环境，请选择 Use an image managed by AWS CodeBuild。从 Operating system、Runtime 和 Version 下拉列表中进行选择。有关更多信息，请参阅[AWS CodeBuild 提供的 Docker 映像 \(p. 114\)](#)。
 - 要使用基于 AWS 账户内 Amazon ECR 存储库中的 Docker 映像的构建环境，请选择 Specify a Docker image。对于 Custom image type，选择 Amazon ECR。使用 Amazon ECR repository 和 Amazon ECR image 下拉列表指定所需的 Amazon ECR 存储库以及该存储库中的 Docker 映像。
 - 要使用基于 Docker Hub 中的 Docker 映像的构建环境，请选择 Specify a Docker image。对于 Custom image type，选择 Other。在 Custom image ID 框中，使用格式 `docker-repo-name/docker-image-name:tag` 键入 Docker 映像 ID。
17. 对于 Build specification，执行下列操作之一：
 - 如果您的源代码包含构建规范文件，请选择 Use the buildspec.yml in the source code root directory (在源代码根目录中使用 buildspec.yml)。
 - 如果您的源代码不包含构建规范文件，请选择 Insert build commands。对于 Build command (构建命令)，请键入要在构建阶段在构建环境中运行的命令；对于多条命令，针对基于 Linux 的构建环境，请使用 `&&` 将每条命令分隔开；针对基于 Windows 的构建环境，请使用 `;` 将每条命令分隔开。对于 Output files，请键入要发送给 AWS CodePipeline 的构建环境中的构建输出文件路径；对于多个文件，请使用逗号将每个文件路径分隔开。有关更多信息，请参阅控制台中的工具提示。
18. 对于 AWS CodeBuild service role，执行下列操作之一：

- 如果您的 AWS 账户中没有 AWS CodeBuild 服务角色，请选择 Create a service role in your account。在 Role name 框中，键入服务角色的名称，或保留建议的名称。(服务角色名称在您的 AWS 账户内必须是唯一的。)

Note

如果您使用控制台创建 AWS CodeBuild 服务角色，那么默认情况下，这个角色仅能与此构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，那么此角色将更新，以便与关联的构建项目配合使用。单个 AWS CodeBuild 服务角色最多可与十个构建项目配合使用。

- 如果您的 AWS 账户中有 AWS CodeBuild 服务角色，请选择 Choose an existing service role from your account。在 Role name 框中，选择服务角色的名称。

19. 展开 Advanced。

要指定 60 分钟 (默认值) 以外的构建超时时间，请使用 hours 和 minutes 框指定一个介于 5 和 480 分钟 (8 小时) 之间的超时时间。

对于 Compute，请选择一个可用选项。

仅当您计划使用此构建项目来构建 Docker 映像，并且您选择的构建环境映像不是由支持 Docker 的 AWS CodeBuild 提供的映像时，选中 Privileged 复选框。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。请注意，您还必须启动 Docker 守护程序，以便您的构建能够根据需要与其交互。执行此操作的一种方法是通过运行以下构建命令在您的构建规范的 install 阶段初始化 Docker 守护程序。(如果选择了由支持 Docker 的 AWS CodeBuild 提供的构建环境映像，请不要运行以下构建命令。)

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --  
host=tcp://127.0.0.1:2375 --storage-driver=overlay&  
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

对于 Environment variables，请使用 Name 和 Value 为要使用的构建环境指定任何可选的环境变量。要添加更多环境变量，请选择 Add row。

Important

我们强烈建议不要使用环境变量存储敏感值，尤其是 AWS 访问密钥 ID 和秘密访问密钥。使用 AWS CodeBuild 控制台和 AWS CLI 等工具能够以纯文本格式显示环境变量。

要存储和检索敏感值，我们建议您的构建命令使用 AWS CLI 来与 Amazon EC2 Systems Manager Parameter Store 进行交互。AWS CLI 已在由 AWS CodeBuild 提供的所有构建环境上预安装和预配置。有关更多信息，请参阅 Amazon EC2 Systems Manager 用户指南 中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store CLI 演练](#)。

20. 选择 Save build project。

21. 对于 Input artifact #1，请键入您在此过程的步骤 4 中记录的 Output artifact 值。

22. 对于 Output artifact #1，请键入输出项目的名称 (例如，**MyAppBuild**)。

23. 选择 Add Action。

24. 选择 Save pipeline changes，然后选择 Save and continue。

25. 选择 Release change。

26. 管道成功运行后，您可以获取构建输出项目。管道在 AWS CodePipeline 控制台中显示后，在 Build 操作中，将鼠标指针停留在工具提示上。记下 Output artifact 的值 (例如，MyAppBuild)。

Note

您还可以通过在 AWS CodeBuild 控制台的构建详细信息页面上选择 Build artifacts 链接来获取构建输出项目。要访问此页面，请参阅 [查看构建详细信息 \(控制台\)](#) (p. 190)，然后跳到此过程的步骤 31。

27. 通过以下网址打开 Amazon S3 控制台：<https://console.aws.amazon.com/s3/>。

28. 在存储桶列表中，请打开管道使用的存储桶。此存储桶的名称应遵循格式 `codepipeline-region-ID-random-number`。您可以使用 AWS CLI 运行 `AWS CodePipeline get-pipeline` 命令，获取存储桶的名称：

```
aws codepipeline get-pipeline --name my-pipeline-name
```

在输出中，该 `pipeline` 对象包含一个 `artifactStore` 对象，其中包含带有存储桶名称的 `location` 值。

29. 打开与您的管道名称匹配的文件夹（根据管道名称的长度，文件夹名称可能被截断），然后打开与您在此过程的步骤 26 中记录的 Output artifact 的值匹配的文件夹。
30. 提取文件内容。如果该文件夹中有多个文件，请提取具有最新 Last Modified 时间戳的文件的内容。（您可能需要为文件提供 `.zip` 扩展名，这样，您可以将其用于您系统内的 ZIP 实用工具。）构建输出项目将位于文件的提取内容中。
31. 如果您指示 AWS CodePipeline 部署构建输出项目，请使用部署提供商的说明，获取部署目标上的构建输出项目。

将 AWS CodeBuild 测试操作添加到管道 (AWS CodePipeline 控制台)

1. 通过以下网址打开 AWS CodePipeline 控制台：<https://console.aws.amazon.com/codepipeline>。

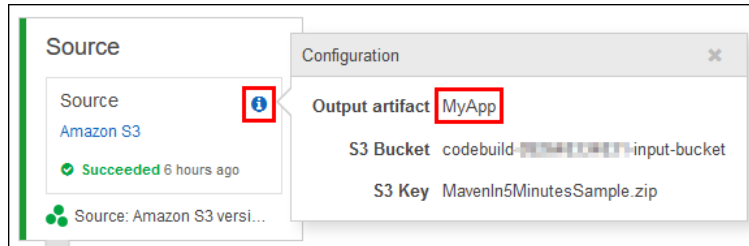
您应该已使用以下任一身份登录到 AWS 管理控制台：

- 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅 IAM 用户指南中的[账户根用户](#)。
- AWS 账户中的 IAM 管理员用户。有关更多信息，请参阅 IAM 用户指南中的[创建您的第一个 IAM 管理员用户和组](#)。
- AWS 账户中的 IAM 用户，具有执行以下最基本操作的权限：

```
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. 在 AWS 区域选择器中，请选择管道所在的区域。此区域还必须支持 AWS CodeBuild。有关更多信息，请参阅 Amazon Web Services 一般参考 中的“区域和终端节点”主题中的 [AWS CodeBuild](#)。
3. 在 All Pipelines 页面上，选择管道名称。

- 在管道详细信息页面上，在 Source 操作中，请将鼠标指针停留在工具提示上。记下 Output artifact 的值 (例如，MyApp)：



Note

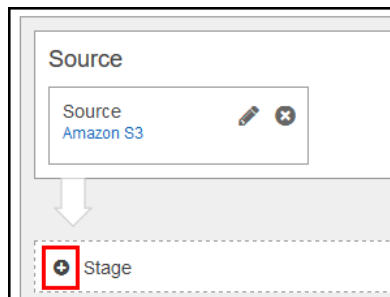
此过程假定您要将测试操作添加到 Source 和 Beta 阶段之间的测试阶段内。如果您要在其他位置添加测试操作，请将鼠标指针停留在之前的操作上，然后记下 Output artifact 的值。

- 选择 Edit。
- 紧接着 Source 阶段，选择 Stage 旁的添加 (+)。

Note

此过程假定您要将测试阶段添加到您的管道。要将测试操作添加到现有的阶段，请选择现有阶段中的编辑 (铅笔) 图标，然后跳到此过程的步骤 8。

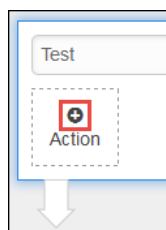
此过程还假定您要紧接着 Source 阶段添加测试阶段。要在其他位置添加测试阶段，请在所需位置选择添加符号。



- 对于 Enter stage name，键入测试阶段的名称 (例如，**Test**)。如果您选择了其他名称，请在整个过程中使用该名称。
- 在选定阶段内，选择 Action 旁的添加 (+)。

Note

此过程假定您要在测试阶段内添加测试操作。要在其他位置添加测试操作，请在所需位置选择添加符号。您可能需要先在您要添加测试操作的现有阶段内选择编辑 (铅笔) 图标。



- 在 Add action 窗格中，对于 Action category，选择 Test。
- 在 Test actions 中，对于 Action name，键入操作的名称 (例如，**Test**)。如果您选择了其他名称，请在整个过程中使用该名称。
- 对于 Test provider，选择 AWS CodeBuild。

12. 如果您在 AWS CodeBuild 中有构建项目，请选择 Select an existing build project。对于 Project name，请选择构建项目的名称，然后向前跳至本过程的步骤 21。

Important

如果您为 AWS CodeBuild 项目启用 Webhook，并且该项目用作 AWS CodePipeline 中的构建步骤，则将为每次提交创建两个相同的构建。一个构建通过 Webhook 触发；另一个构建通过 AWS CodePipeline 触发。由于账单基于每个构建，因此您需要为这两个构建付费。因此，如果您使用的是 AWS CodePipeline，建议您在 CodeBuild 中禁用 Webhook。在 AWS CodeBuild 控制台中，清除 Webhook 框。有关更多信息，请参阅[更改构建项目的设置 \(控制台\)](#) (p. 174) 中的步骤 9。

13. 选择 Create a new build project。
14. 对于 Project name，键入此构建项目的名称。构建项目名称在您的各个 AWS 账户内必须是唯一的。
15. (可选) 在 Description 框中键入描述。
16. 对于 Environment image，执行下列操作之一：
- 要使用基于由 AWS CodeBuild 管理的 Docker 映像的构建环境，请选择 Use an image managed by AWS CodeBuild。从 Operating system、Runtime 和 Version 下拉列表中进行选择。有关更多信息，请参阅 [AWS CodeBuild 提供的 Docker 映像](#) (p. 114)。
 - 要使用基于 AWS 账户内 Amazon ECR 存储库中的 Docker 映像的构建环境，请选择 Specify a Docker image。对于 Custom image type，选择 Amazon ECR。使用 Amazon ECR repository 和 Amazon ECR image 下拉列表指定所需的 Amazon ECR 存储库以及该存储库中的 Docker 映像。
 - 要使用基于 Docker Hub 中的 Docker 映像的构建环境，请选择 Specify a Docker image。对于 Custom image type，选择 Other。在 Custom image ID 框中，使用格式 `docker-repo-name/docker-image-name:tag` 键入 Docker 映像 ID。
17. 对于 Build specification，执行下列操作之一：
- 如果您的源代码包含构建规范文件，请选择 Use the buildspec.yml in the source code root directory (在源代码根目录中使用 buildspec.yml)。
 - 如果您的源代码不包含构建规范文件，请选择 Insert build commands。对于 Build command，请在构建环境中键入您要在构建阶段运行的命令。对于多条命令，针对基于 Linux 的构建环境，请使用 `&&` 将每条命令分隔开；针对基于 Windows 的构建环境，请使用 `;` 将每条命令分隔开。对于 Output files，请键入您要在构建环境中发送给 AWS CodePipeline 的构建输出文件的路径。对于多个文件，请用逗号分隔各个文件路径。有关更多信息，请参阅控制台中的工具提示。
18. 对于 AWS CodeBuild service role，执行下列操作之一：
- 如果您的 AWS 账户中没有 AWS CodeBuild 服务角色，请选择 Create a service role in your account。在 Role name 框中，键入服务角色的名称，或保留建议的名称。(服务角色名称在您的 AWS 账户内必须是唯一的。)

Note

如果您使用控制台创建 AWS CodeBuild 服务角色，那么默认情况下，这个服务角色仅能与此构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，那么此角色将更新，以便与关联的构建项目配合使用。单个 AWS CodeBuild 服务角色最多可与十个构建项目配合使用。

- 如果您的 AWS 账户中有 AWS CodeBuild 服务角色，请选择 Choose an existing service role from your account。在 Role name 框中，选择服务角色的名称。
19. (可选) 展开 Advanced。

要指定 60 分钟 (默认值) 以外的构建超时时间，请使用 hours 和 minutes 框指定一个介于 5 和 480 分钟 (8 小时) 之间的超时时间。

仅当您计划使用此构建项目来构建 Docker 映像，并且您选择的构建环境映像不是由支持 Docker 的 AWS CodeBuild 提供的映像时，选中 Privileged 复选框。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。请注意，您还必须启动 Docker 守护程序，以便您的构建能够根据需要与其交互。

执行此操作的一种方法是通过运行以下构建命令在您的构建规范的 `install` 阶段初始化 Docker 守护程序。(如果选择了由支持 Docker 的 AWS CodeBuild 提供的构建环境映像，请不要运行以下构建命令。)

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --  
host=tcp://127.0.0.1:2375 --storage-driver=overlay&  
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

对于 Compute，请选择一个可用选项。

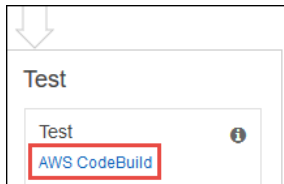
对于 Environment variables，请使用 Name 和 Value 为要使用的构建环境指定任何可选的环境变量。要添加更多环境变量，请选择 Add row。

Important

我们强烈建议不要使用环境变量存储敏感值，尤其是 AWS 访问密钥 ID 和秘密访问密钥。使用 AWS CodeBuild 控制台和 AWS CLI 等工具能够以纯文本格式显示环境变量。

要存储和检索敏感值，我们建议您的构建命令使用 AWS CLI 来与 Amazon EC2 Systems Manager Parameter Store 进行交互。AWS CLI 已在由 AWS CodeBuild 提供的所有构建环境上预安装和预配置。有关更多信息，请参阅 Amazon EC2 Systems Manager 用户指南 中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store CLI 演练](#)。

20. 选择 Save build project。
21. 对于 Input artifacts #1，请键入在此过程的步骤 4 中记录的 Output artifact 值。
22. (可选) 如果您希望测试操作来生成输出项目，并且相应地设置构建规范，那么对于 Output artifact #1，请键入您要分配给输出项目的值。
23. 选择 Add Action。
24. 选择 Save pipeline changes，然后选择 Save and continue。
25. 选择 Release change。
26. 管道成功运行后，您可以获取测试结果。在管道的 Test 阶段中，选择 AWS CodeBuild 超链接以在 AWS CodeBuild 控制台中打开相关的构建项目页面。



27. 在构建项目页面上，在 Build history 区域中，选择相关的 Build run 超链接。
28. 在构建运行页面上，在 Build logs 区域中，选择 View entire log 超链接以在 Amazon CloudWatch 控制台中打开相关的构建日志。
29. 滚动浏览构建日志，查看测试结果。

将 AWS CodeBuild 与 Jenkins 结合使用

借助 AWS CodeBuild Jenkins 插件，您可以将 AWS CodeBuild 与您的 Jenkins 构建任务相集成。您可以使用插件将您的构建作业发送给 AWS CodeBuild，而不是发送给 Jenkins 构建节点。这样一来，便无需预置、配置和管理 Jenkins 构建节点。

设置 Jenkins

有关使用 AWS CodeBuild 插件设置 Jenkins 的信息，请参阅 AWS DevOps 博客上的[利用 AWS CodeBuild 简化 Jenkins 构建任务](#)博客文章。您可以从 <https://github.com/awslabs/aws-codebuild-jenkins-plugin> 下载 AWS CodeBuild Jenkins。

安装插件

如果您已设置 Jenkins 并希望仅安装 AWS CodeBuild 插件，请在您的 Jenkins 实例上的插件管理器中搜索“AWS CodeBuild Plugin for Jenkins”。

使用插件

将 AWS CodeBuild 与 Amazon VPC 外部的源结合使用

1. 在 AWS CodeBuild 控制台中创建项目。有关更多信息，请参阅 [创建构建项目 \(控制台\)](#) (p. 154)。
 - 选择要在其中运行构建任务的区域。
 - (可选) 设置 Amazon VPC 配置以允许 AWS CodeBuild 构建容器访问您的 Amazon VPC 中的资源。
 - 记下您的项目的名称。您在步骤 3 中需要它。
 - (可选) 如果 AWS CodeBuild 本机不支持您的源存储库，可以将 Amazon S3 设置为您的项目的输入源类型。
2. 在 IAM 控制台中，创建一个 IAM 用户以供 Jenkins 插件使用。
 - 当您为该用户创建凭证时，请选择编程访问。
 - 创建如下所示的策略，然后将该策略附加到您的用户。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": [ "arn:aws:logs:{{region}}:{{awsAccountId}}:log-group:/aws/codebuild/{{projectName}}:*" ],
      "Action": [ "logs:GetLogEvents" ]
    },
    {
      "Effect": "Allow",
```

```

        "Resource": ["arn:aws:s3:::{inputBucket}"],
        "Action": ["s3:GetBucketVersioning"]
    },
    {
        "Effect": "Allow",
        "Resource": ["arn:aws:s3:::{inputBucket}/{inputObject}"],
        "Action": ["s3:PutObject"]
    },
    {
        "Effect": "Allow",
        "Resource": ["arn:aws:s3:::{outputBucket}/*"],
        "Action": ["s3:GetObject"]
    },
    {
        "Effect": "Allow",
        "Resource": ["arn:aws:codebuild:{region}::{awsAccountId}:project/
        {{projectName}}"],
        "Action": ["codebuild:StartBuild",
        "codebuild:BatchGetBuilds",
        "codebuild:BatchGetProjects"]
    }
}

```

3. 在 Jenkins 中创建一个自由式项目。

- 在 Configure (配置) 页面上，选择 Add build step (添加构建步骤)，然后选择 Run build on AWS CodeBuild (在 AWS CodeBuild 上运行构建任务)。
- 配置您的构建步骤。
 - 为 Region (区域)、Credentials (凭证) 和 Project Name (项目名称) 提供值。
 - 选择 Use Project source (使用项目源)。
 - 保存配置并从 Jenkins 运行构建任务。

4. 对于 Source Code Management (源代码管理)，选择您希望如何检索您的源。您可能需要在 Jenkins 服务器上安装 GitHub 插件 (或您的源存储库提供商的 Jenkins 插件)。

- 在 Configure (配置) 页面上，选择 Add build step (添加构建步骤)，然后选择 Run build on AWS CodeBuild (在 AWS CodeBuild 上运行构建任务)。
- 配置您的构建步骤。
 - 为 Region (区域)、Credentials (凭证) 和 Project Name (项目名称) 提供值。
 - 选择 Use Jenkins source (使用 Jenkins 源)。
 - 保存配置并从 Jenkins 运行构建任务。

将 AWS CodeBuild 插件与 Jenkins 管道插件结合使用

- 在您的 Jenkins 管道项目页面上，使用代码段生成器来生成将 AWS CodeBuild 作为管道中的步骤添加的管道脚本。它应生成如下所示的脚本：

```

awsCodeBuild projectName: 'project', credentialsType: 'keys', region: 'us-west-2',
sourceControlType: 'jenkins'

```

使用 AWS CodeBuild 中的构建项目和构建

要开始使用，请执行[创建构建项目 \(p. 154\)](#)中的步骤，然后执行[运行构建项目 \(p. 183\)](#)中的步骤。有关构建项目和构建的更多信息，请参阅以下主题。

主题

- [使用构建项目 \(p. 154\)](#)
- [使用 AWS CodeBuild 中的构建 \(p. 183\)](#)

使用构建项目

生成项目 定义 AWS CodeBuild 如何运行生成任务。它包括的信息有源代码获取位置、要使用的生成环境、要运行的生成命令和存储生成输出的位置等。

在使用构建项目时，您可以执行以下任务：

主题

- [在 AWS CodeBuild 中创建构建项目 \(p. 154\)](#)
- [查看 AWS CodeBuild 中构建项目名称的列表 \(p. 168\)](#)
- [查看 AWS CodeBuild 中构建项目的详细信息 \(p. 169\)](#)
- [创建 AWS CodeBuild 触发器 \(p. 171\)](#)
- [编辑 AWS CodeBuild 触发器 \(p. 172\)](#)
- [更改 AWS CodeBuild 中构建项目的设置 \(p. 174\)](#)
- [删除 AWS CodeBuild 中的构建项目 \(p. 182\)](#)

在 AWS CodeBuild 中创建构建项目

您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包创建构建项目。

主题

- [先决条件 \(p. 154\)](#)
- [创建构建项目 \(控制台\) \(p. 154\)](#)
- [创建构建项目 \(AWS CLI\) \(p. 162\)](#)
- [创建构建项目 \(AWS 软件开发工具包\) \(p. 168\)](#)
- [创建构建项目 \(AWS CloudFormation\) \(p. 168\)](#)

先决条件

回答 [规划构建 \(p. 106\)](#) 中的问题。

创建构建项目 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 如果显示欢迎页面，请选择 Get started (开始使用)。

如果未显示欢迎页面，则在导航窗格中选择 Build projects，然后选择 Create project。

3. 在 Configure your project 页面上，对于 Project name，键入此构建项目的名称。构建项目名称在您的各个 AWS 账户内必须是唯一的。
4. (可选) 选择 Add description，然后在 Description 框中键入说明。
5. 在 Source: What to build (源代码: 要构建的内容) 中，对于 Source provider (源提供商)，选择源提供商类型。使用下表为您的源提供商选择适当的选项：

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
For Bucket, choose the name of the input bucket that contains the source code.	X				
For S3 object key, type the name of the ZIP file that contains the source code.	X				
Choose Connect to Bitbucket and follow the instructions to connect (or reconnect) with Bitbucket.			X		
Choose Connect to GitHub and follow the instructions to connect (or reconnect) with GitHub and authorize access to AWS CodeBuild.				X	
For Personal Access token, see GitHub Enterprise 示例 (p. 46) for information about how to copy a personal					X

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
<p>access token to your clipboard. Paste the token in the text field, and then choose Save Token.</p> <p>Note</p> <p>You only need to enter and save the personal access token once. All future AWS CodeBuild projects will use this token.</p>					
From Repository, choose the repository you want to use.		X			
For Repository, choose whether to use a public repository or a repository in your account.			X	X	
Use Repository URL only if you use a public repository. Enter its URL.			X	X	

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
Use Choose a repository only if you use a repository in your account.			X	X	
Choose Git clone depth to create a shallow clone with a history truncated to the specified number of commits. If you want a full clone, choose Full.		X	X	X	X
Choose Webhook if you want AWS CodeBuild to build the source code every time a code change is pushed to this repository.				X	X
If you chose Webhook, in Branch filter, enter a regular expression filter to specify which branches are built. If the name of a branch matches the branch filter, the branch is built. If no filter is specified, all branches are built.				X	X

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
<p>Choose Report build status, if you want the status of your build's start and completion reported to your source provider.</p> <p>Note</p> <p>The status of a build triggered by a webhook is always reported to your source provider.</p>				X	
<p>Choose Insecure SSL to ignore SSL warnings while connecting to your GitHub Enterprise project repository.</p>					X
<p>Choose Build Badge to make your project's build status visible and embeddable. For more information, see 构建徽章示例 (p. 54).</p>		X	X	X	X

6. 在 Environment: How to build 中：

对于 Environment image，执行下列操作之一：

- 要使用由 AWS CodeBuild 托管的 Docker 映像，请选择 Use an image managed by AWS CodeBuild，然后从 Operating system、Runtime 和 Version 中进行相应选择。
- 要使用其他 Docker 映像，请选择 Specify a Docker image。对于 Custom image type，选择 Other 或 Amazon ECR。如果选择 Other (其他)，则对于 Custom image ID (自定义映像 ID)，键入 Docker Hub 中 Docker 映像的名称和标签。使用以下格式：`repository-name/image-name: image-tag`。如果您选择 Amazon ECR，请使用 Amazon ECR repository 和 Amazon ECR image 在您的 AWS 账户中选择 Docker 映像。

对于 Build specification，执行下列操作之一：

- 如果您的源代码包含构建规范文件，请选择 Use the buildspec.yml in the source code root directory (在源代码根目录中使用 buildspec.yml)。
- 如果您的源代码不包含构建规范文件，或者您要运行的构建命令与源代码根目录的 buildspec.yml 文件中为 build 阶段指定的构建命令不同，请选择 Insert build commands (插入构建命令)。对于 Build command，请键入您要在 build 阶段运行的命令。对于多个命令，使用 && 分开各个命令 (例如 `mvn test && mvn package`)。要在其他阶段运行命令，或者，如果 build 阶段对应的命令列表特别长，请将 buildspec.yml 文件添加到源代码根目录，将命令添加到该文件中，然后选择 Use the buildspec.yml in the source code root directory。

有关更多信息，请参见 [构建规范参考 \(p. 107\)](#)。

7. 在 Artifacts: Where to put the artifacts from this build project 中，对于 Artifacts type，执行下列操作之一：
 - 如果您不想创建任何构建输出项目，请选择 No artifacts。如果您只运行构建测试，或者您要将 Docker 映像推送到 Amazon ECR 存储库，建议执行此操作。
 - 要将构建输出存储在 Amazon S3 存储桶中，请选择 Amazon S3，然后执行以下操作：
 - 如果您要将项目名称用于构建输出 ZIP 文件或文件夹，请将 Artifacts name 留空。否则，请在 Artifacts name (构件名称) 中键入名称。(如果您要输出 ZIP 文件，并且要让 ZIP 文件包含文件扩展名，请务必在 ZIP 文件名之后添加扩展名。)
 - 对于 Bucket name，请选择输出存储桶的名称。
 - 如果您在此过程的前面部分选择了 Insert build commands，那么对于 Output files，请键入您要放到构建输出 ZIP 文件或文件夹的构建中的文件位置。对于多个位置，使用逗号分开各个位置 (appspect.yml, target/my-app.jar)。有关更多信息，请参阅 [构建规范语法 \(p. 107\)](#) 中 files 的描述。
8. 在 Cache 中，执行下列操作之一：
 - 如果您不想使用缓存，请选择 No cache。
 - 要使用缓存，请选择 Amazon S3，然后执行以下操作：
 - 对于 Bucket，选择存储缓存的 Amazon S3 存储桶的名称。
 - (可选) 对于 Path prefix，键入一个 Amazon S3 路径前缀。Path prefix (路径前缀) 值类似于目录名称。它使您能够在存储桶的同一目录下存储缓存。

Important

请不要在路径前缀末尾附加“/”。

使用缓存可节省大量构建时间，因为构建环境的可重用部分被存储在缓存中，并且可跨构建使用。有关在构建规范文件中指定缓存的信息，请参阅 [构建规范语法 \(p. 107\)](#)。

9. 在 Service role 中，执行下列操作之一：
 - 如果您没有 AWS CodeBuild 服务角色，请选择 Create a service role in your account。在 Role name 中，接受默认名称或键入您自己的名称。

- 如果您有 AWS CodeBuild 服务角色，请选择 Choose an service existing role from your account。在 Role name 中，选择服务角色。

Note

当您使用控制台来创建或更新构建项目时，您可以同时创建 AWS CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

10. 在 VPC 中，执行下列操作之一：

- 如果您没有将 VPC 用于项目，请选择 No VPC。
- 如果要 AWS CodeBuild 与您的 VPC 结合使用：
 - 对于 VPC，选择 AWS CodeBuild 使用的 VPC ID。
 - 对于 Subnets，选择包含 AWS CodeBuild 使用的资源的子网。
 - 对于 Security Groups，选择 AWS CodeBuild 用来支持对 VPC 中资源的访问的安全组。

有关更多信息，请参阅 [将 AWS CodeBuild 与 Amazon Virtual Private Cloud 结合使用 \(p. 125\)](#)。

11. 展开 Show advanced settings。

Note

如果您从欢迎页面中选择 Get started 后进入此页面，那么 Show advanced settings 部分将不会显示。跳到本过程的步骤 20。有关更改默认设置的信息，请参阅 [更改构建项目的设置 \(控制台\) \(p. 174\)](#)。

12. (可选) 对于 Timeout，请指定 5 到 480 分钟 (8 个小时) 之间的一个值，在此时间后，如果构建未完成，AWS CodeBuild 会将其停止。如果 hours 和 minutes 都留空，则将使用 60 分钟的默认值。
13. (可选) 对于 Encryption key，执行下列操作之一：
 - 要使用您的账户中适用于 Amazon S3 的 AWS 托管客户主密钥 (CMK) 加密构建输出项目，请将 Encryption key 留空。这是默认值。
 - 要使用客户托管 CMK 加密构建输出项目，请在 Encryption key 中键入 CMK 的 ARN。采用格式 `arn:aws:kms:region-ID:account-ID:key/key-ID`。
14. (可选) 仅当您计划使用此构建项目来构建 Docker 映像且您选择的构建环境映像不是由具有 Docker 支持的 AWS CodeBuild 提供时，才选择 Privileged。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。您还必须启动 Docker 守护程序，以便您的构建与其交互。执行此操作的一种方法是通过运行以下构建命令在您的构建规范的 install 阶段初始化 Docker 守护程序。如果您选择了由具有 Docker 支持的 AWS CodeBuild 提供的构建环境映像，请不要运行这些命令。

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --  
host=tcp://127.0.0.1:2375 --storage-driver=overlay&  
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

15. (可选) 如果您在此过程的前面部分为 Artifacts type 选择了 Amazon S3，那么对于 Artifacts packaging，执行下列操作之一：
 - 要让 AWS CodeBuild 创建一个包含构建输出的 ZIP 文件，请选择 Zip。
 - 要让 AWS CodeBuild 创建一个包含构建输出的文件夹，请选择 None (无)。(这是默认值。)
16. 对于 Compute type，选择一个可用选项。
17. 对于 Environment variables，键入每个环境变量的名称、值和类型以供构建使用。使用 Add row 添加环境变量。

Note

AWS CodeBuild 会自动为您的 AWS 区域设置环境变量。如果您不将它们添加到您 buildspec.yml，则必须设置以下环境变量：

- AWS_ACCOUNT_ID
- IMAGE_REPO_NAME
- IMAGE_TAG

其他人可以使用 AWS CodeBuild 控制台和 AWS CLI 查看环境变量。如果您不担心环境变量的可见性，请设置 Name 和 Value 字段，然后将 Type 设置为 Plaintext。

我们建议您将具有敏感值 (例如 AWS 访问密钥 ID、AWS 秘密访问密钥或密码) 的环境变量作为参数存储在 Amazon EC2 Systems Manager Parameter Store 中。对于 Type，选择 Parameter Store。对于 Name，键入一个标识符以供 AWS CodeBuild 引用。对于 Value，按照 Amazon EC2 Systems Manager Parameter Store 中存储的参数名称键入参数名称。使用名为 /CodeBuild/dockerLoginPassword 的参数作为示例，对于 Type，选择 Parameter Store。对于 Name，键入 LOGIN_PASSWORD。对于 Value，键入 /CodeBuild/dockerLoginPassword。

Important

我们建议您将名称以 /CodeBuild/ 开头的参数 (例如，/CodeBuild/dockerLoginPassword) 存储在 Amazon EC2 Systems Manager Parameter Store 中。可以使用 AWS CodeBuild 控制台在 Amazon EC2 Systems Manager 中创建参数。选择 Create a parameter，然后按照对话框中的说明操作。(在该对话框中，对于 KMS key，您可以选择指定您的账户中的 AWS KMS 密钥的 ARN。Amazon EC2 Systems Manager 使用此密钥在存储过程中加密参数的值并在检索过程中解密参数的值。)如果您使用 AWS CodeBuild 控制台创建了一个参数，控制台将在参数名称被存储时以 /CodeBuild/ 作为它的开头。有关更多信息，请参阅 Amazon EC2 Systems Manager 用户指南 中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store 控制台演练](#)。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数，则构建项目的服务角色必须允许 ssm:GetParameters 操作。如果您之前选择了 Create a service role in your account，则 AWS CodeBuild 将自动在您的构建项目的默认服务角色中包含此操作。但是，如果您选择了 Choose an existing service role from your account，则必须将此操作单独包含在您的服务角色中。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的名称不以 /CodeBuild/ 开头的参数，且您选择了 Create a service role in your account，则您必须更新该服务角色以允许访问不以 /CodeBuild/ 开头的参数名称。这是因为该服务角色仅允许访问以 /CodeBuild/ 开头的参数名称。

您设置的环境变量将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 MY_VAR 的环境变量 (值为 my_value)，并且您设置了一个名为 MY_VAR 的环境变量 (值为 other_value)，那么 my_value 将被替换为 other_value。同样，如果 Docker 映像已经包含一个名为 PATH 的环境变量 (值为 /usr/local/sbin:/usr/local/bin)，并且您设置了一个名为 PATH 的环境变量 (值为 \$PATH:/usr/share/ant/bin)，那么 /usr/local/sbin:/usr/local/bin 将被替换为文本值 \$PATH:/usr/share/ant/bin。

请勿使用以 CODEBUILD_ 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。

如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
- 构建项目定义中的值优先级处于其次。
- 构建规范声明中的值优先级最低。

18. (可选) 对于 Tags (标签)，键入您希望支持 AWS 服务使用的任何标签的名称和值。使用 Add row 添加标签。您最多可以添加 50 个标签。

19. 选择 Continue (继续)。

20. 在 Review 页面上，执行下列操作之一：

- 要运行构建，请选择 Save and build。
- 要在不运行构建的情况下完成构建项目的创建，请选择 Save。

创建构建项目 (AWS CLI)

有关将 AWS CLI 与 AWS CodeBuild 结合使用的信息，请参阅[命令行参考 \(p. 207\)](#)。

1. 运行 create-project 命令：

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

输出中将显示 JSON 格式的数据。将数据复制到本地计算机上或安装 AWS CLI 的实例上某位置处的文件 (如 `create-project.json`) 中。按照下面所示修改复制的数据，并保存您的结果。

```
{
  "name": "project-name",
  "description": "description",
  "source": {
    "type": "source-type",
    "location": "source-location",
    "gitCloneDepth": "gitCloneDepth",
    "buildspec": "buildspec",
    "insecureSsl": "InsecureSsl",
    "reportBuildStatus": "reportBuildStatus",
    "auth": {
      "type": "auth-type",
      "resource": "resource"
    }
  },
  "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"
  },
  "badgeEnabled": "badgeEnabled"
}

```

替换以下内容：

- **project-name**：必填值。此构建项目的名称。此名称在您的 AWS 账户的所有构建项目中必须是唯一的。
- **description**：可选值。此构建项目的描述。
- 对于必填的 **source** 对象，则为有关此构建项目的源代码设置的信息。这些设置包括：
 - **source-type**：必填值。包含要构建的源代码的存储库的类型。有效值包括 CODECOMMIT、CODEPIPELINE、GITHUB、GITHUB_ENTERPRISE、BITBUCKET 和 S3。
 - **source-location**：必填值 (除非您将 **source-type** 设置为 CODEPIPELINE)。指定存储库类型的源代码的位置。
 - 对于 AWS CodeCommit，则为指向包含源代码和构建规范的存储库的 HTTPS 克隆 URL (例如，`https://git-codecommit.region-id.amazonaws.com/v1/repos/repo-name`)。
 - 对于 Amazon S3，则为构建输入存储桶名称，后跟正斜杠 (/)，再加上包含源代码和构建规范的 ZIP 文件的名称 (例如，`bucket-name/object-name.zip`)。此时假定 ZIP 文件在构建输入存储桶的根目录中。(如果 ZIP 文件在存储桶内的文件夹中，请改用 `bucket-name/path/to/object-name.zip`。)
 - 对于 GitHub，则为指向包含源代码和构建规范的存储库的 HTTPS 克隆 URL。您必须将您的 AWS 账户连接到您的 GitHub 账户。为此，请使用 AWS CodeBuild 控制台创建构建项目。当您使用控制台与 GitHub 连接 (或重新连接) 时，在 GitHub Authorize application 页面上，对于 Organization access，选择您希望 AWS CodeBuild 能够访问的每个存储库旁边的 Request access。选择 Authorize application。(在连接到您的 GitHub 账户后，您不需要完成构建项目的创建。可以关闭 AWS CodeBuild 控制台。)要指示 AWS CodeBuild 使用此连接，请在 **source** 对象中，将 **auth** 对象的 **type** 值设置为 OAUTH。
 - 对于 GitHub Enterprise，则为指向包含源代码和构建规范的存储库的 HTTP 或 HTTPS 克隆 URL。您还必须将您的 AWS 账户连接到您的 GitHub Enterprise 账户。为此，请使用 AWS CodeBuild 控制台创建构建项目。

首先，在 GitHub Enterprise 中创建个人访问令牌。将此令牌复制到您的剪贴板，以便在创建 AWS CodeBuild 项目时使用。有关更多信息，请参阅 GitHub 帮助网站上的[在 GitHub Enterprise 中创建个人访问令牌](#)。如果您使用控制台创建 AWS CodeBuild 项目，请在 Source: What to build 中针对 Source provider 选择 GitHub Enterprise。对于 Personal Access Token，请粘贴已复制到剪贴板中的令牌。选择 Save Token。您的 AWS CodeBuild 账户现在已与您的 GitHub Enterprise 账户连接。

- 对于 Bitbucket，则为指向包含源代码和构建规范的存储库的 HTTPS 克隆 URL。您还必须将您的 AWS 账户连接到您的 Bitbucket 账户。为此，请使用 AWS CodeBuild 控制台创建构建项目。当您使用控制台与 Bitbucket 连接 (或重新连接) 时，在 Bitbucket Confirm access to your account 页面上，选择 Grant access。(在连接到您的 Bitbucket 账户后，您不需要完成构建项目的创建。可以关闭 AWS CodeBuild 控制台。)要指示 AWS CodeBuild 使用此连接，请在 **source** 对象中，将 **auth** 对象的 **type** 值设置为 OAUTH。

- 对于 AWS CodePipeline，请勿为 `source` 指定 `location` 值。AWS CodePipeline 会忽略该值，因为当您在 AWS CodePipeline 中创建管道时，您会在管道的源阶段指定源代码位置。
- `gitCloneDepth`：可选值。要下载的历史记录深度。最小值为 0。如果此值为 0、大于 25 或未提供，则会下载每个构建项目的完整历史记录。如果您的源类型是 Amazon S3，则不支持此值。
- `buildspec`：可选值。要使用的构建规范定义或文件。如果设置了该值，则它可以是内联构建规范定义，也可以是指向相对于内置 `CODEBUILD_SRC_DIR` 环境变量的值的替代构建规范文件的路径。如果此值未提供，或设置为空字符串，则源代码必须在其根目录中添加 `buildspec.yml` 文件。有关更多信息，请参阅 [构建规范文件名称和存储位置](#) (p. 107)。
- `auth`：此对象仅由 AWS CodeBuild 控制台使用。不要为 `auth-type`（除非 `source-type` 设置为 `GITHUB`）或 `resource` 指定值。
- `reportBuildStatus`：可选值。此值指定是否向源提供商发送构建的开始和完成状态。如果用源提供商而非 GitHub 设置此项，则会引发 `invalidInputException`。
- `InsecureSsl`：可选值。此值仅适用于 GitHub Enterprise。将此值设为 `true`，将在连接到您的 GitHub Enterprise 项目存储库时忽略 SSL 警告。默认值为 `false`。`InsecureSsl` 应当仅用于测试目的。它不应在生产环境中使用。
- 对于必填的 `artifacts` 对象，则为有关此构建项目的输出项目设置的信息。这些设置包括：
 - `artifacts-type`：必填值。生成输出项目的类型。有效值包括 `CODEPIPELINE`、`NO_ARTIFACTS` 和 `S3`。
 - `artifacts-location`：必填值（除非您将 `artifacts-type` 设置为 `CODEPIPELINE` 或 `NO_ARTIFACTS`）。构建输出项目的位置：
 - 如果您为 `artifacts-type` 指定了 `CODEPIPELINE`，请勿为 `artifacts` 指定 `location`。
 - 如果您为 `artifacts-type` 指定了 `NO_ARTIFACTS`，请勿为 `artifacts` 指定 `location`。
 - 如果您为 `artifacts-type` 指定了 `S3`，那么这就是您在先决条件中创建或标识的输出存储桶的名称。
 - `path`：可选值。构建输出 ZIP 文件或文件夹的路径和名称：
 - 如果您为 `artifacts-type` 指定了 `CODEPIPELINE`，那么请勿为 `artifacts` 指定 `path`。
 - 如果您为 `artifacts-type` 指定了 `NO_ARTIFACTS`，请勿为 `artifacts` 指定 `path`。
 - 如果您为 `artifacts-type` 指定了 `S3`，那么这就是 `artifacts-location` 内指向构建输出 ZIP 文件或文件夹的路径。如果您没有为 `path` 指定值，那么 AWS CodeBuild 将使用 `namespaceType`（如果指定）和 `artifacts-name` 来决定构建输出 ZIP 文件或文件夹的路径和名称。例如，如果您为 `path` 指定 `MyPath`，并为 `artifacts-name` 指定 `MyArtifact.zip`，那么路径和名称将为 `MyPath/MyArtifact.zip`。
 - `namespaceType`：可选值。构建输出 ZIP 文件或文件夹的路径和名称：
 - 如果您为 `artifacts-type` 指定了 `CODEPIPELINE`，请勿为 `artifacts` 指定 `namespaceType`。
 - 如果您为 `artifacts-type` 指定了 `NO_ARTIFACTS`，请勿为 `artifacts` 指定 `namespaceType`。
 - 如果您为 `artifacts-type` 指定了 `S3`，则有效值包括 `BUILD_ID` 和 `NONE`。使用 `BUILD_ID` 将构建 ID 插入到构建输出 ZIP 文件或文件夹的路径中。否则，请使用 `NONE`。如果您没有为 `namespaceType` 指定值，那么 AWS CodeBuild 将使用 `path`（如果指定）和 `artifacts-name` 来决定构建输出 ZIP 文件或文件夹的路径和名称。例如，如果您为 `path` 指定 `MyPath`，为 `namespaceType` 指定 `BUILD_ID`，并为 `artifacts-name` 指定 `MyArtifact.zip`，那么路径和名称将为 `MyPath/build-ID/MyArtifact.zip`。
 - `artifacts-name`：必填值（除非您将 `artifacts-type` 设置为 `CODEPIPELINE` 或 `NO_ARTIFACTS`）。构建输出 ZIP 文件或文件夹的路径和名称：
 - 如果您为 `artifacts-type` 指定了 `CODEPIPELINE`，请勿为 `artifacts` 指定 `name`。
 - 如果您为 `artifacts-type` 指定了 `NO_ARTIFACTS`，请勿为 `artifacts` 指定 `name`。
 - 如果您为 `artifacts-type` 指定了 `S3`，那么这就是 `artifacts-location` 内构建输出 ZIP 文件或文件夹的名称。例如，如果您为 `path` 指定 `MyPath`，并为 `artifacts-name` 指定 `MyArtifact.zip`，那么路径和名称将为 `MyPath/MyArtifact.zip`。
 - `packaging`：可选值。要创建的构建输出项目的类型：

- 如果您为 `artifacts-type` 指定了 CODEPIPELINE，请勿为 artifacts 指定 packaging。
- 如果您为 `artifacts-type` 指定了 NO_ARTIFACTS，请勿为 artifacts 指定 packaging。
- 如果您为 `artifacts-type` 指定了 S3，则有效值包括 ZIP 和 NONE。要创建包含构建输出的 ZIP 文件，请使用 ZIP。要创建包含构建输出的文件夹，请使用 NONE。默认值为 NONE。
- 对于必填的 `cache` 对象，则为有关此构建项目的缓存设置的信息。这些设置包括：
 - `CacheType`：必填值。有效值为 S3 或 NO_CACHE。
 - `CacheLocation`：必填值（除非您将 `CacheType` 设置为 NONE）。如果您为 `CacheType` 指定了 S3，则这是 S3 存储桶的 ARN 以及路径前缀。例如，如果您的 Amazon S3 存储桶名称为 my-bucket，且您的路径前缀为 build-cache，则您的 `CacheLocation` 可接受的格式为 my-bucket/build-cache 或 aws:s3::my-bucket/build-cache。
- `serviceRole`：必填值。AWS CodeBuild 代表 IAM 用户用来与服务进行交互的服务角色 ARN（例如，arn:aws:iam::account-id:role/role-name）。
- 对于可选的 `vpcConfig` 对象，则为有关 VPC 配置的信息。这些设置包括：
 - `vpcId`：必填值。AWS CodeBuild 使用的 VPC ID。运行以下命令获取您的区域中所有 Amazon VPC ID 的列表：

```
aws ec2 describe-vpcs
```

- `subnets`：必填值。包含 AWS CodeBuild 使用的资源的子网 ID。运行以下命令获取这些 ID：

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

Note

如果您使用的是 us-east-1 以外的区域，请确保在运行命令时使用它。

- `securityGroupIds`：必填值。AWS CodeBuild 用来支持对 VPC 中的资源的访问的安全组 ID。运行以下命令获取这些 ID：

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

Note

如果您使用的是 us-east-1 以外的区域，请确保在运行命令时使用它。

- 对于必填的 `environment` 对象，则为有关此项目的构建环境设置的信息。这些设置包括：
 - `environment-type`：必填值。构建环境的类型。有效值为 LINUX_CONTAINER 和 WINDOWS_CONTAINER。
 - `image`：必填值。此构建环境使用的 Docker 映像标识符。通常，此标识符以 `image-name:tag` 的形式表示。例如，在 AWS CodeBuild 用来管理其 Docker 镜像的 Docker 存储库中，这可能是 aws/codebuild/java:openjdk-8。在 Docker Hub 中，为 maven:3.3.9-jdk-8。在 Amazon ECR 中，为 `account-id.dkr.ecr.region-id.amazonaws.com/your-Amazon-ECR-repo-name:tag`。有关更多信息，请参阅 [AWS CodeBuild 提供的 Docker 映像 \(p. 114\)](#)。
 - `computeType`：必填值。与此构建环境使用的 CPU 内核数和内存容量相对应的类别。允许的值包括 BUILD_GENERAL1_SMALL、BUILD_GENERAL1_MEDIUM 和 BUILD_GENERAL1_LARGE。
 - `certificate`：可选值。S3 存储桶的 ARN、路径前缀和包含 PEM 编码证书的对象键。对象键可以仅为 .pem 文件，也可以为包含 pem 编码证书的 .zip 文件。例如，如果您的 Amazon S3 存储桶名称是 my-bucket，路径前缀是 cert，对象键名称是 certificate.pem，则您的 ## 的可接受格式为 my-bucket/cert/certificate.pem 或 arn:aws:s3::my-bucket/cert/certificate.pem。
- 对于可选 `environmentVariables` 数组，则为有关您要为此构建环境指定的任何环境变量的信息。每个环境变量都以包含 `environmentVariable-name`、`environmentVariable-value` 和 `environmentVariable-type` 的对象形式表示。

其他人可以使用 AWS CodeBuild 控制台和 AWS CLI 查看环境变量。如果您不担心环境变量的可见性，请设置 `environmentVariable-name` 和 `environmentVariable-value`，然后将 `environmentVariable-type` 设置为 `PLAINTEXT`。

我们建议您将具有敏感值 (例如 AWS 访问密钥 ID、AWS 秘密访问密钥或密码) 的环境变量作为参数存储在 Amazon EC2 Systems Manager Parameter Store 中。对于 `environmentVariable-name`，请为该存储的参数设置一个标识符以供 AWS CodeBuild 引用。对于 `environmentVariable-value`，按照 Amazon EC2 Systems Manager Parameter Store 中存储的参数名称设置参数名称。将 `environmentVariable-type` 设置为 `PARAMETER_STORE`。使用名为 `/CodeBuild/dockerLoginPassword` 的参数作为示例，将 `environmentVariable-name` 设置为 `LOGIN_PASSWORD`。将 `environmentVariable-value` 设置为 `/CodeBuild/dockerLoginPassword`。将 `environmentVariable-type` 设置为 `PARAMETER_STORE`。

Important

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数，则构建项目的服务角色必须允许 `ssm:GetParameters` 操作。如果您之前选择了 `Create a service role in your account`，则 AWS CodeBuild 将自动在您的构建项目的默认服务角色中包含此操作。但是，如果您选择了 `Choose an existing service role from your account`，则必须将此操作单独包含在您的服务角色中。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的名称不以 `/CodeBuild/` 开头的参数，且您选择了 `Create a service role in your account`，则您必须更新该服务角色以允许访问不以 `/CodeBuild/` 开头的参数名称。这是因为该服务角色仅允许访问以 `/CodeBuild/` 开头的参数名称。

您设置的任何环境变量都将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 `MY_VAR` 的环境变量 (值为 `my_value`)，并且您设置了一个名为 `MY_VAR` 的环境变量 (值为 `other_value`)，那么 `my_value` 将被替换为 `other_value`。同样，如果 Docker 映像已经包含一个名为 `PATH` 的环境变量 (值为 `/usr/local/sbin:/usr/local/bin`)，并且您设置了一个名为 `PATH` 的环境变量 (值为 `$PATH:/usr/share/ant/bin`)，那么 `/usr/local/sbin:/usr/local/bin` 将被替换为文本值 `$PATH:/usr/share/ant/bin`。

请勿使用以 `CODEBUILD_` 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
 - 构建项目定义中的值优先级处于其次。
 - 构建规范声明中的值优先级最低。
- 仅当您计划使用此构建项目构建 Docker 映像且您指定的构建环境映像不由具有 Docker 支持的 AWS CodeBuild 提供时，您才必须指定值为 `true` 的 `privilegedMode`。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。您还必须启动 Docker 守护程序，以便您的构建与其交互。执行此操作的一种方法是通过运行以下构建命令在您的构建规范的 `install` 阶段初始化 Docker 守护程序。如果您指定了由具有 Docker 支持的 AWS CodeBuild 提供的构建环境映像，请不要运行这些命令。

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --  
host=tcp://127.0.0.1:2375 --storage-driver=overlay&  
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

- `badgeEnabled`：可选值。要在您的 AWS CodeBuild 项目中包含构建徽章，您必须指定值为 `true` 的 `badgeEnabled`。有关更多信息，请参阅 [使用 AWS CodeBuild 构建徽章示例 \(p. 54\)](#)。
- `timeoutInMinutes`：可选值。5 到 480 分钟 (8 个小时) 之间的一个分钟数，在此时间后，如果构建未完成，AWS CodeBuild 会将其停止。如果未指定，则使用默认值 60。要确定 AWS CodeBuild 是否以及何时因超时而停止构建，请运行 `batch-get-builds` 命令。要确定构建是否已停止，请在输出中查看 `buildStatus` 的值是否为 `FAILED`。要确定构建何时超时，请在输出中查看与 `TIMED_OUT` 的 `phaseStatus` 值关联的 `endTime` 值。

- **encryptionKey**：可选值。AWS CodeBuild 用来加密构建输出的 AWS KMS 客户主密钥 (CMK) 的别名或 ARN。如果您指定别名，请使用格式 `arn:aws:kms:region-ID:account-ID:key/key-ID`，或者，如果存在别名，请使用格式 `alias/key-alias`。如果未指定，则会使用适用于 Amazon S3 的 AWS 托管 CMK。
- 对于可选的 **tags** 数组，则为有关您希望与此构建项目关联的任何标签的信息。您最多可指定 50 个标签。这些标签可由支持 AWS CodeBuild 构建项目标签的任何 AWS 服务使用。每个标签都以包含 **tag-key** 和 **tag-value** 的 key 和 value 值的对象形式表示。

有关示例，请参阅[创建构建项目 \(AWS CLI\)](#) (p. 10)。

2. 切换到包含您刚才保存的文件的目录，然后再次运行 `create-project` 命令：

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

3. 如果成功，输出中将显示与以下内容类似的数据：

```
{
  "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` 对象包含有关新构建项目的信息：
 - `lastModified` 值表示有关构建项目的信息上次更改的时间，采用 Unix 时间格式。
 - `created` 值表示构建项目的创建时间，采用 Unix 时间格式。
 - `arn` 值表示构建项目的 ARN。

Note

您稍后可以更改构建项目的任何设置，但构建项目名称除外。有关更多信息，请参阅 [更改构建项目的设置 \(AWS CLI\)](#) (p. 181)。

要开始运行构建，请参阅 [运行构建项目 \(AWS CLI\)](#) (p. 186)。

如果您的源代码存储在 GitHub 存储库中，并且您希望 AWS CodeBuild 在每次代码更改被推送到存储库时重建源代码，请参阅[开始自动运行构建 \(AWS CLI\)](#) (p. 189)。

创建构建项目 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 开发工具包结合使用的信息，请参阅 [AWS 开发工具包和工具参考](#) (p. 208)。

创建构建项目 (AWS CloudFormation)

有关将 AWS CodeBuild 与 AWS CloudFormation 结合使用的信息，请参阅 AWS CloudFormation 用户指南中的 [AWS CodeBuild 的 AWS CloudFormation 模板](#)。

查看 AWS CodeBuild 中构建项目名称的列表

要查看 AWS CodeBuild 中构建项目的列表，您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包。

主题

- [查看构建项目名称的列表 \(控制台\)](#) (p. 168)
- [查看构建项目名称的列表 \(AWS CLI\)](#) (p. 168)
- [查看构建项目名称的列表 \(AWS 软件开发工具包\)](#) (p. 169)

查看构建项目名称的列表 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 在导航窗格中，选择 Build projects。

Note

默认情况下，仅显示十个最新的构建项目。要查看更多构建项目，请为 Projects per page 选择其他值，或者为 Viewing projects 选择向后和向前箭头。

查看构建项目名称的列表 (AWS CLI)

运行 `list-projects` 命令：

```
aws codebuild list-projects --sort-by sort-by --sort-order sort-order --next-token next-token
```


替换上一命令中的以下占位符：

- **sort-by**：可选字符串。要用于列出构建项目名称的标准。有效值包括：
 - `CREATED_TIME`：根据每个构建项目的创建时间列出构建项目名称。
 - `LAST_MODIFIED_TIME`：根据每个构建项目信息上次更改的时间列出构建项目名称。
 - `NAME`：根据每个构建项目的名称列出构建项目名称。
- **sort-order**：可选字符串。根据 **sort-by** 列出构建项目的顺序。有效值包括 `ASCENDING` 和 `DESCENDING`。
- **next-token**：可选字符串。在上次运行时，如果列表中有 100 个以上的项目，则只能返回前 100 个项目，以及名为下一个令牌的唯一字符串。要获取列表中的下一批项目，请再次运行此命令，将下一个令牌添加到调用中。要获取列表中的所有项目，请利用每个后续的下一个令牌运行此命令，直到不再返回下一个令牌。

例如，如果您运行此命令：

```
aws codebuild list-projects --sort-by NAME --sort-order ASCENDING
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "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"
  ]
}
```

如果您再次运行此命令：

```
aws codebuild list-projects --sort-by NAME --sort-order ASCENDING --next-token
Ci33ACF6...The full token has been omitted for brevity...U+AkMx8=
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "projects": [
    "codebuild-demo-project100",
    "codebuild-demo-project101",
    ... The full list of build project names has been omitted for brevity ...
    "codebuild-demo-project122"
  ]
}
```

查看构建项目名称的列表 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 软件开发工具包结合使用的更多信息，请参阅 [AWS 开发工具包和工具参考](#) (p. 208)。

查看 AWS CodeBuild 中构建项目的详细信息

要查看 AWS CodeBuild 中构建项目的详细信息，您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包。

主题

- [查看构建项目的详细信息 \(控制台\) \(p. 170\)](#)
- [查看构建项目的详细信息 \(AWS CLI\) \(p. 170\)](#)
- [查看构建项目的详细信息 \(AWS 软件开发工具包\) \(p. 171\)](#)

查看构建项目的详细信息 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 在导航窗格中，选择 Build projects。

Note

默认情况下，仅显示十个最新的构建项目。要查看更多构建项目，请为 Projects per page 选择其他值，或者为 Viewing projects 选择向后和向前箭头。

3. 在构建项目列表中的 Project 列中，选择与构建项目对应的链接。
4. 在 Build project: **project-name** 页面上，展开 Project details。

查看构建项目的详细信息 (AWS CLI)

有关将 AWS CLI 与 AWS CodeBuild 结合使用的更多信息，请参阅 [命令行参考 \(p. 207\)](#)。

运行 batch-get-projects 命令：

```
aws codebuild batch-get-projects --names names
```

在上述命令中，替换以下占位符：

- **names**：必填字符串。要查看其详细信息的一个或多个构建项目名称。要指定多个构建项目，请用空格分隔各个构建项目的名称。您最多可以指定 100 个构建项目名称。要获取构建项目的列表，请参阅 [查看构建项目名称的列表 \(AWS CLI\) \(p. 168\)](#)。

例如，如果您运行此命令：

```
aws codebuild batch-get-projects --names codebuild-demo-project codebuild-demo-project2 my-other-demo-project
```

与以下内容类似的结果可能会出现在输出中。为简洁起见，用省略号 (...) 表示省略的数据。

```
{
  "projectsNotFound": [
    "my-other-demo-project"
  ],
  "projects": [
    {
      ...
      "name": codebuild-demo-project,
      ...
    },
    {
      ...
      "name": codebuild-demo-project2",
      ...
    }
  ]
}
```

在前面的输出中，`projectsNotFound` 数组列出了指定的但却找不到任何相关信息的所有构建项目名称。`projects` 数组列出了可找到相关信息的所有构建项目的详细信息。为简洁起见，前面的输出中省略了构建项目的详细信息。有关更多信息，请参阅 [创建构建项目 \(AWS CLI\) \(p. 162\)](#) 的输出。

查看构建项目的详细信息 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 软件开发工具包结合使用的更多信息，请参阅 [AWS 开发工具包和工具参考 \(p. 208\)](#)。

创建 AWS CodeBuild 触发器

您可以在项目上创建触发器以安排每小时、每天或每周进行一次构建。您也可以将自定义规则与 Amazon CloudWatch cron 表达式结合使用来创建触发器。例如，通过使用 cron 表达式，您可以安排在每个工作日的特定时间进行构建。

创建项目后，您可以创建触发器。

创建触发器

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 在导航窗格中，选择 Build projects.
3. 选择要将触发器添加到的构建项目的链接，然后选择 Build triggers (构建触发器) 选项卡。

Note

默认情况下，将显示 100 个最新的构建项目。要查看更多构建项目，请为 Projects per page 选择其他值，或者为 Viewing projects 选择向后和向前箭头。

4. 选择 Create trigger.
5. 在触发器名称中键入名称。
6. 从 Frequency (频率) 下拉列表中，选择触发器的频率。如果要使用 Cron 表达式创建频率，请选择 Custom (自定义)。
7. 为触发器的频率指定参数。您可以在文本框中键入您的选项的前几个字符以筛选下拉菜单项。

Note

Start hours and minutes are zero-based. The start minute is a number between zero and 59. The start hour is a number between zero and 23. For example, a daily trigger that starts every day at 12:15 P.M. has a start hour of 12 and a start minute of 15. A daily trigger that starts every day at midnight has a start hour of zero and a start minute of zero. A daily trigger that starts every day at 11:59 P.M. has a start hour of 23 and a start minute of 59.

Frequency	Required Parameters	Details
Hourly	<ul style="list-style-type: none">• Start minute	<ul style="list-style-type: none">• Use the Start minute drop-down menu.
Daily	<ul style="list-style-type: none">• Start minute• Start hour	<ul style="list-style-type: none">• Use the Start minute drop-down menu.• Use the Start hour drop-down menu.
Weekly	<ul style="list-style-type: none">• Start minute• Start hour• Start day	<ul style="list-style-type: none">• Use the Start minute drop-down menu.• Use the Start hour drop-down menu.

Frequency	Required Parameters	Details
		<ul style="list-style-type: none"> Use the Start day drop-down menu.
Custom	<ul style="list-style-type: none"> Cron expression 	<p>Type a cron expression in Cron expression. A cron expression has six required fields that are separated by white space. The fields specify a start value for minute, hour, day of month, month, day of week, and year. You can use wildcards to specify a range, additional values, and more. For example, the cron expression 0 9 ? * MON-FRI * schedules a build every weekday at 9:00 A.M. For more information, see Cron Expressions.</p>

- 选择 Enable this trigger (启用此触发器) 以启用触发器。
- (可选) 在源版本中，键入源的版本。
 - 对于 Amazon S3，键入与您要生成的输入项目的版本相对应的版本 ID。如果 Source version (源版本) 留空，则使用最新版本。
 - 对于 AWS CodeCommit，键入一个提交 ID。如果 Source version (源版本) 留空，则使用默认分支的 HEAD 提交 ID。
 - 对于 GitHub 或 GitHub Enterprise，请键入提交 ID、拉取请求 ID、分支名称或与您要构建的源代码版本对应的标签名称。如果您要指定拉取请求 ID，则必须使用格式 `pr/pull-request-ID` (例如，`pr/25`)。如果您要指定分支名称，则将使用分支的 HEAD 提交 ID。如果 Source version 留空，则将使用默认分支的 HEAD 提交 ID。
 - 对于 Bitbucket，键入提交 ID、分支名称或与您要构建的源代码版本对应的标签名称。如果您要指定分支名称，则将使用分支的 HEAD 提交 ID。如果 Source version 留空，则将使用默认分支的 HEAD 提交 ID。
- (可选) 指定介于 5 分钟和 480 分钟 (8 小时) 之间的超时。此值指定 AWS CodeBuild 在停止前尝试构建的时间长度。如果小时和分钟保留为空，则使用项目中指定的默认超时值。

编辑 AWS CodeBuild 触发器

您可以在项目上编辑触发器以安排每小时、每天或每周进行一次构建。您也可以编辑触发器以将自定义规则与 Amazon CloudWatch cron 表达式结合使用。例如，通过使用 cron 表达式，您可以安排在每个工作日的特定时间进行构建。有关创建触发器的信息，请参阅[创建 AWS CodeBuild 触发器 \(p. 171\)](#)。

要编辑触发器，请执行以下操作：

- Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
- 在导航窗格中，选择 Build projects。
- 选择您要更改的构建项目的链接，然后选择 Edit project。

Note

默认情况下，将显示 100 个最新的构建项目。要查看更多构建项目，请为 Projects per page 选择其他值，或者为 Viewing projects 选择向后和向前箭头。

- 执行以下任一操作：

- 选择要更改的触发器旁边的单选按钮，选择操作，然后选择编辑。
 - 选择要更改的触发器的链接。
5. 从 Frequency (频率) 下拉列表中，选择触发器的频率。如果要使用 Cron 表达式创建频率，请选择 Custom (自定义)。
 6. 为触发器的频率指定参数。您可以在文本框中键入您的选项的前几个字符以筛选下拉菜单项。

Note

Start hours and minutes are zero-based. The start minute is a number between zero and 59. The start hour is a number between zero and 23. For example, a daily trigger that starts every day at 12:15 P.M. has a start hour of 12 and a start minute of 15. A daily trigger that starts every day at midnight has a start hour of zero and a start minute of zero. A daily trigger that starts every day at 11:59 P.M. has a start hour of 23 and a start minute of 59.

Frequency	Required Parameters	Details
Hourly	<ul style="list-style-type: none"> Start minute 	<ul style="list-style-type: none"> Use the Start minute drop-down menu.
Daily	<ul style="list-style-type: none"> Start minute Start hour 	<ul style="list-style-type: none"> Use the Start minute drop-down menu. Use the Start hour drop-down menu.
Weekly	<ul style="list-style-type: none"> Start minute Start hour Start day 	<ul style="list-style-type: none"> Use the Start minute drop-down menu. Use the Start hour drop-down menu. Use the Start day drop-down menu.
Custom	<ul style="list-style-type: none"> Cron expression 	<p>Type a cron expression in Cron expression. A cron expression has six required fields that are separated by white space. The fields specify a start value for minute, hour, day of month, month, day of week, and year. You can use wildcards to specify a range, additional values, and more. For example, the cron expression <code>0 9 ? * MON-FRI *</code> schedules a build every weekday at 9:00 A.M. For more information, see Cron Expressions.</p>

7. 选择 Enable this trigger (启用此触发器) 以启用触发器。

Note

您可以使用 <https://console.aws.amazon.com/cloudwatch/> 上的 Amazon CloudWatch 控制台来编辑源版本、超时以及 AWS CodeBuild 中不可用的其他选项。

更改 AWS CodeBuild 中构建项目的设置

要更改 AWS CodeBuild 中构建项目的设置，您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包。

主题

- [更改构建项目的设置 \(控制台\)](#) (p. 174)
- [更改构建项目的设置 \(AWS CLI\)](#) (p. 181)
- [更改构建项目的设置 \(AWS 软件开发工具包\)](#) (p. 182)

更改构建项目的设置 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 在导航窗格中，选择 Build projects。
3. 执行以下任一操作：
 - 选择您要更改的构建项目旁边的单选按钮，选择 Actions，然后选择 Update。
 - 选择您要更改的构建项目的链接，然后选择 Edit project。

Note

默认情况下，将显示 100 个最新的构建项目。要查看更多构建项目，请为 Projects per page 选择其他值，或者为 Viewing projects 选择向后和向前箭头。

4. 在项目详细信息页面的 Description 中键入描述。

有关此过程中引用的设置的更多信息，请参阅[创建构建项目 \(控制台\)](#) (p. 154)。

5. 要更改有关源代码位置的信息，请在 Source: What to build 区域中，选择 Update source。使用下表针对您的源提供商进行相应的选择：

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
For Bucket, choose the name of the input bucket that contains the source code.	X				
For S3 object key, type the name of the ZIP file that contains the source code.	X				
Choose Connect to Bitbucket and follow the instructions to connect (or			X		

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
reconnect) with Bitbucket.					
Choose Connect to GitHub and follow the instructions to connect (or reconnect) with GitHub and authorize access to AWS CodeBuild.				X	

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
<p>For Personal Access token, see GitHub Enterprise 示例 (p. 46) for information about how to copy a personal access token to your clipboard. Paste the token in the text field, and then choose Save Token.</p> <p>Note</p> <p>You only need to enter and save the personal access token once. All future AWS CodeBuild projects will use this token.</p>					X
From Repository, choose the repository you want to use.		X			

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
For Repository, choose whether to use a public repository or a repository in your account.			X	X	
Use Repository URL only if you use a public repository. Enter its URL.			X	X	
Use Choose a repository only if you use a repository in your account.			X	X	
Choose Git clone depth to create a shallow clone with a history truncated to the specified number of commits. If you want a full clone, choose Full.		X	X	X	X
Choose Webhook if you want AWS CodeBuild to build the source code every time a code change is pushed to this repository.				X	X

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
If you chose Webhook, in Branch filter, enter a regular expression filter to specify which branches are built. If the name of a branch matches the branch filter, the branch is built. If no filter is specified, all branches are built.				X	X
<p>Choose Report build status, if you want the status of your build's start and completion reported to your source provider.</p> <p>Note</p> <p>The status of a build triggered by a webhook is always reported to your source provider.</p>				X	

源提供商	Amazon S3	AWS CodeCommit	Bitbucket	GitHub	GitHub Enterprise
If you chose Webhook, choose Rotate webhook secret key if you want GitHub to rotate your secret key every time a code change triggers a build.				X	X
Choose Insecure SSL to ignore SSL warnings while connecting to your GitHub Enterprise project repository.					X
Choose Build Badge to make your project's build status visible and embeddable. For more information, see 构建徽章示例 (p. 54) .		X	X	X	X

6. 要更改有关构建环境的信息，请在 Environment: How to build 中选择 Update image。对构建环境类型 (例如，Environment image、Operating system、Runtime、Version、Custom image type、Custom image ID、Amazon ECR repository 或 Amazon ECR image) 进行适当的更改。
7. 执行以下任一操作：
 - 如果您的源代码之前不包含 buildspec.yml 文件但现在包含，请选择 Update build specification，然后选择 Use buildspec.yml from source code。
 - 如果您的源代码以前包含 buildspec.yml 文件但现在不包含此文件，请选择 Update build specification，然后选择 Insert build commands，再在 Build commands 中键入命令。
8. 要更改有关构建输出项目位置和名称的信息，请在 Artifacts: Where to put the artifacts from this build project 中更改 Artifacts type、Artifact name、Bucket name 或 Output files 的值。
9. 要更改有关缓存的信息，请在 Cache 中执行以下操作之一：
 - 如果您之前选择了缓存但现在不想使用缓存，请选择 No cache。
 - 如果您之前选择了 No cache 但现在想使用缓存，请选择 Amazon S3，然后执行以下操作：
 - 对于 Bucket，选择存储缓存的 Amazon S3 存储桶的名称。

- (可选) 对于 Path prefix，键入一个 Amazon S3 路径前缀。Path prefix 值类似于支持您将缓存存储在存储桶内同一目录下的目录名称。

Important

请不要在 Path prefix 的末尾附加“/”。

使用缓存可节省大量构建时间，因为构建环境的可重用部分被存储在缓存中，并且可跨构建使用。有关在构建规范文件中指定缓存的信息，请参阅[构建规范语法 \(p. 107\)](#)。

10. 要更改有关 AWS CodeBuild 服务角色的信息，请在 Service role 中更改 Create a role、Choose an existing service role from your account 或 Role name 的值。

Note

当您使用控制台来创建或更新构建项目时，您可以同时创建 AWS CodeBuild 服务角色。默认情况下，这个角色仅能与该构建项目配合使用。如果您使用控制台将此服务角色与另一个构建项目关联，则此角色将更新以便与关联的构建项目结合使用。一个服务角色最多可与 10 个构建项目结合使用。

11. 在 VPC 中，执行下列操作之一：

- 如果您没有将 VPC 用于项目，请选择 No VPC。
- 如果要使用 AWS CodeBuild 与您的 VPC 结合使用：
 - 对于 VPC，选择 AWS CodeBuild 使用的 VPC ID。
 - 对于 Subnets，选择包含 AWS CodeBuild 使用的资源的子网。
 - 对于 Security Groups，选择 AWS CodeBuild 用来支持对 VPC 中资源的访问的安全组。

有关更多信息，请参阅[将 AWS CodeBuild 与 Amazon Virtual Private Cloud 结合使用 \(p. 125\)](#)。

12. 要更改有关构建超时的信息，请在 Show advanced settings 中为 Timeout 更改 hours 和 minutes 的值。如果 hours 和 minutes 都留空，则默认值为 60 分钟。
13. 要更改有关 AWS KMS 客户主密钥 (CMK) 的信息，请在 Show advanced settings 中，更改 Encryption key 的值。

Important

如果您将 Encryption key 留空，则 AWS CodeBuild 会将 AWS 托管 CMK 用于您 AWS 账户中的 Amazon S3。

14. 如果您计划使用此构建项目来构建 Docker 映像且指定的构建环境不由具有 Docker 支持的 AWS CodeBuild 提供，则在 Show advanced settings 中，选择 Privileged。否则，尝试与 Docker 守护程序交互的所有关联的构建都将失败。您还必须启动 Docker 守护程序，以便您的构建可根据需要与之交互。执行此操作的一种方法是通过运行以下构建命令在您的构建规范的 install 阶段初始化 Docker 守护程序。(如果指定的构建环境映像由具有 Docker 支持的 AWS CodeBuild 提供，请不要运行以下构建命令。)

```
- nohup /usr/local/bin/dockerd --host=unix:///var/run/docker.sock --  
host=tcp://127.0.0.1:2375 --storage-driver=overlay&  
- timeout -t 15 sh -c "until docker info; do echo .; sleep 1; done"
```

15. 要更改有关构建输出项目存储方式的信息，请在 Show advanced settings 中，更改 Artifacts packaging 的值。
16. 要更改用于运行构建的内存量和 vCPU 量，请在 Show advanced settings 中，更改 Compute type 的值。
17. 要更改有关您希望构建使用的环境变量的信息，请在 Show advanced settings 中为 Environment variables 更改 Name、Value 和 Type 的值。使用 Add row 添加环境变量。选择您不想再使用的环境变量旁边的删除 (X) 按钮。

其他人可以使用 AWS CodeBuild 控制台和 AWS CLI 查看环境变量。如果您不担心环境变量的可见性，请设置 Name 和 Value 字段，然后将 Type 设置为 Plaintext。

我们建议您将具有敏感值 (例如 AWS 访问密钥 ID、AWS 秘密访问密钥或密码) 的环境变量作为参数存储在 Amazon EC2 Systems Manager Parameter Store 中。对于 Type，选择 Parameter Store。对于 Name，键入一个标识符以供 AWS CodeBuild 引用。对于 Value，按照 Amazon EC2 Systems Manager Parameter Store 中存储的参数名称键入参数名称。使用名为 /CodeBuild/dockerLoginPassword 的参数作为示例，对于 Type，选择 Parameter Store。对于 Name，键入 LOGIN_PASSWORD。对于 Value，键入 /CodeBuild/dockerLoginPassword。

Important

我们建议您将名称以 /CodeBuild/ 开头的参数 (例如，/CodeBuild/dockerLoginPassword) 存储在 Amazon EC2 Systems Manager Parameter Store 中。可以使用 AWS CodeBuild 控制台在 Amazon EC2 Systems Manager 中创建参数。选择 Create a parameter，然后按照对话框中的说明操作。(在该对话框中，对于 KMS key，您可以选择指定您的账户中的 AWS KMS 密钥的 ARN。Amazon EC2 Systems Manager 使用此密钥在存储过程中加密参数的值并在检索过程中解密参数的值。)如果您使用 AWS CodeBuild 控制台创建了一个参数，控制台将在参数名称被存储时以 /CodeBuild/ 作为它的开头。有关更多信息，请参阅 Amazon EC2 Systems Manager 用户指南 中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store 控制台演练](#)。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数，则构建项目的服务角色必须允许 ssm:GetParameters 操作。如果您之前选择了 Create a service role in your account，则 AWS CodeBuild 将自动在您的构建项目的默认服务角色中包含此操作。但是，如果您选择了 Choose an existing service role from your account，则必须将此操作单独包含在您的服务角色中。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的名称不以 /CodeBuild/ 开头的参数，且您选择了 Create a service role in your account，则您必须更新该服务角色以允许访问不以 /CodeBuild/ 开头的参数名称。这是因为该服务角色仅允许访问以 /CodeBuild/ 开头的参数名称。

您设置的环境变量将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 MY_VAR 的环境变量 (值为 my_value)，并且您设置了一个名为 MY_VAR 的环境变量 (值为 other_value)，那么 my_value 将被替换为 other_value。同样，如果 Docker 映像已经包含一个名为 PATH 的环境变量 (值为 /usr/local/sbin:/usr/local/bin)，并且您设置了一个名为 PATH 的环境变量 (值为 \$PATH:/usr/share/ant/bin)，那么 /usr/local/sbin:/usr/local/bin 将被替换为文本值 \$PATH:/usr/share/ant/bin。

请勿使用以 CODEBUILD_ 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。

如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
- 构建项目定义中的值优先级处于其次。
- 构建规范声明中的值优先级最低。

18. 要更改有关此构建项目标签的信息，请在 Show advanced settings 中，对于 Tags，更改 Name 和 Value 的值。使用 Add row 添加标签。您最多可以添加 50 个标签。选择您不想再使用的标签旁边的删除 (X) 图标。

19. 选择 Update。

更改构建项目的设置 (AWS CLI)

有关将 AWS CLI 与 AWS CodeBuild 结合使用的更多信息，请参阅 [命令行参考](#) (p. 207)。

1. 运行 update-project 命令，如下所示：

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

输出中将显示 JSON 格式的数据。将数据复制到本地计算机上或安装 AWS CLI 的实例上某位置处的文件 (如 `update-project.json`) 中。然后按照 [创建构建项目 \(AWS CLI\)](#) (p. 162) 中的说明修改复制的数据，并保存您的结果。

Note

在 JSON 格式的数据中，您必须提供要更改其设置的构建项目的名称。所有其他设置都是可选的。您无法更改构建项目的名称，但可以更改它的任何其他设置。

2. 切换到包含您刚刚保存的文件的目录，然后再次运行 `update-project` 命令。

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

3. 如果成功，与[创建构建项目 \(AWS CLI\)](#) (p. 162) 中所述内容类似的数据将出现在输出中。

更改构建项目的设置 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 开发工具包结合使用的信息，请参阅 [AWS 开发工具包和工具参考](#) (p. 208)。

删除 AWS CodeBuild 中的构建项目

您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包删除 AWS CodeBuild 中的构建项目。

Warning

如果您删除某个构建项目，则它无法恢复。有关构建的所有信息也将被删除，并且无法恢复。

主题

- [删除构建项目 \(控制台\)](#) (p. 182)
- [删除构建项目 \(AWS CLI\)](#) (p. 182)
- [删除构建项目 \(AWS 软件开发工具包\)](#) (p. 183)

删除构建项目 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 在导航窗格中，选择 Build projects.
3. 执行以下任一操作：
 - 选择您要删除的构建项目旁边的单选按钮，选择 Actions，然后选择 Delete.
 - 选择您要删除的构建项目的链接，然后选择 Delete.

Note

默认情况下，仅显示 10 个最新的构建项目。要查看更多构建项目，请为 Projects per page 选择其他值，或者为 Viewing projects 选择向后和向前箭头。

删除构建项目 (AWS CLI)

有关将 AWS CLI 与 AWS CodeBuild 结合使用的更多信息，请参阅 [命令行参考](#) (p. 207)。

1. 运行 `delete-project` 命令：

```
aws codebuild delete-project --name name
```

替换以下占位符：

- *name*：必需的字符串。要删除的构建项目的名称。要获取可用构建项目的列表，请运行 `list-projects` 命令。有关更多信息，请参阅 [查看构建项目名称的列表 \(AWS CLI\)](#) (p. 168)。

2. 如果成功，则输出中不会出现任何数据和错误。

删除构建项目 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 软件开发工具包结合使用的更多信息，请参阅 [AWS 开发工具包和工具参考](#) (p. 208)。

使用 AWS CodeBuild 中的构建

一个构建代表一组由 AWS CodeBuild 执行的操作，目的是基于一组输入项目 (例如，一系列 Java 类文件) 创建输出项目 (例如，JAR 文件)。

在使用构建时，您可以执行以下任务：

主题

- [在 AWS CodeBuild 中运行构建项目](#) (p. 183)
- [查看 AWS CodeBuild 中的构建详细信息](#) (p. 190)
- [查看 AWS CodeBuild 中的构建 ID 的列表](#) (p. 192)
- [查看 AWS CodeBuild 中构建项目的构建 ID 列表](#) (p. 194)
- [在 AWS CodeBuild 停止构建](#) (p. 195)
- [在 AWS CodeBuild 中删除构建](#) (p. 196)

在 AWS CodeBuild 中运行构建项目

您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包运行 AWS CodeBuild 中的构建项目。

主题

- [运行构建项目 \(控制台\)](#) (p. 183)
- [运行构建项目 \(AWS CLI\)](#) (p. 186)
- [开始自动运行构建 \(AWS CLI\)](#) (p. 189)
- [停止自动运行构建 \(AWS CLI\)](#) (p. 189)
- [运行构建项目 \(AWS 软件开发工具包\)](#) (p. 190)

运行构建项目 (控制台)

要使用 AWS CodePipeline 运行 AWS CodeBuild 中的构建项目，可跳过这些步骤并按照[将 AWS CodePipeline 与 AWS CodeBuild 结合使用](#) (p. 136)中的说明操作。

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.

2. 执行以下任一操作：

- 如果您刚创建完构建项目，则应显示 Build project: **project-name** 页面。选择 Start build。
- 如果您之前创建了构建项目，则应在导航窗格中选择 Build projects。选择构建项目，然后选择 Start build。

3. 在 Start new build 页面上，执行下列操作之一：

- 对于 Amazon S3，请键入与您需要构建的输入项目的版本相对应的版本 ID 作为可选 Source version 值。如果 Source version 留空，则将使用最新版本。
- 对于 AWS CodeCommit，请选择包含您需要构建的源代码版本的分支名称作为可选 Source version 值的 Branch。对于 Source version，请接受显示的 HEAD 提交 ID 或键入另一个 ID。如果 Source version 留空，则将使用默认分支的 HEAD 提交 ID。您无法为 Source version 键入标签名称。要指定一个标签，请键入该标签的提交 ID。更改 Git clone depth 的值。这会创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择 Full。
- 对于 GitHub 或 GitHub Enterprise，请键入提交 ID、拉取请求 ID、分支名称或您要构建的源代码版本的相应标签名称，作为可选 Source version 值。如果您要指定拉取请求 ID，则必须使用格式 `pr/pull-request-ID` (例如，`pr/25`)。如果您要指定分支名称，则将使用分支的 HEAD 提交 ID。如果 Source version 留空，则将使用默认分支的 HEAD 提交 ID。更改 Git clone depth 的值。这会创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择 Full。
- 对于 Bitbucket，请键入提交 ID、分支名称或您要构建的源代码版本的相应标签名称作为可选 Source version 值。如果您要指定分支名称，则将使用分支的 HEAD 提交 ID。如果 Source version 留空，则将使用默认分支的 HEAD 提交 ID。更改 Git clone depth 的值。这会创建一个浅克隆，其历史记录会截断至指定数量的提交。如果您需要完整克隆，请选择 Full。
- 选择 Override source (覆盖源) 仅对此构建使用不同的源提供商。有关源提供商选项和设置的更多信息，请参阅 [Choose source provider](#)。

4. 展开 Override build project settings (覆盖构建项目设置)。

在这里，您可以仅为构建更改设置。此部分中的设置是可选的。

在 Environment: How to build (环境: 如何构建) 下，您可以：

- 选择 Override image (覆盖映像) 来选择不同的映像。
- 选中或清除 Privileged (特权) 来更改特权设置。
- 选择 Override build specification (覆盖构建规范) 来选择不同的构建规范。
- 从 Certificate (证书) 中选择一种不同设置。

在 Artifacts: Where to put the artifacts from this build project (构件: 要将此构建项目中的构件放置到的位置) 下，您可以：

- 从 Type (类型) 中，选择不同的构件类型。
- 在 Name (名称) 中，键入不同的输出构件名称。
- 在 Path (路径) 中，键入不同的输出构件路径。
- 在 Namespace type (命名空间类型) 中，选择不同的类型。选择 Build ID (构建 ID) 将构建 ID 插入构建输出文件的路径 (例如，`My-Path/Build-ID/My-Artifact.zip`)。否则，选择 None (无)。
- 从 Bucket name (存储桶名称) 中，为您的输出构件选择不同的 Amazon S3 存储桶。
- 从 Artifacts packaging (构件打包) 中，选择 Zip，将构建构件文件放在一个压缩文件中。要将构建构件文件分别放入指定 Amazon S3 存储桶中 (不压缩)，则选择 None (无)。

在 Cache (缓存) 下，从 Type (类型) 中，选择不同的缓存设置。

在 Service role (服务角色) 下，您可以更改 AWS CodeBuild 用来调用从属 AWS 服务的角色。选择 Create a role (创建角色)，让 AWS CodeBuild 为您创建一个服务角色。

5. 展开 Show advanced options。

- 如果要仅为构建更改超时，则更改 Timeout (超时) 中的值。
- 如果要更改 Compute type (计算类型)，请选择其中一个可用选项。

6. 展开 Environment variables。

如果您需要更改环境变量 (仅针对此构建项目)，请更改 Name、Value 和 Type 的值。使用 Add row 添加新环境变量 (仅针对此构建项目)。如果您不需要在此构建项目中使用环境变量，请选择环境变量旁边的删除 (X) 按钮。

其他人可以使用 AWS CodeBuild 控制台和 AWS CLI 查看环境变量。如果您不担心环境变量的可见性，请设置 Name 和 Value 字段，然后将 Type 设置为 Plaintext。

我们建议您将具有敏感值 (例如 AWS 访问密钥 ID、AWS 秘密访问密钥或密码) 的环境变量作为参数存储在 Amazon EC2 Systems Manager Parameter Store 中。对于 Type，选择 Parameter Store。对于 Name，键入一个标识符以供 AWS CodeBuild 引用。对于 Value，按照 Amazon EC2 Systems Manager Parameter Store 中存储的参数名称键入参数名称。使用名为 /CodeBuild/dockerLoginPassword 的参数作为示例，对于 Type，选择 Parameter Store。对于 Name，键入 LOGIN_PASSWORD。对于 Value，键入 /CodeBuild/dockerLoginPassword。

Important

我们建议您将名称以 /CodeBuild/ 开头的参数 (例如，/CodeBuild/dockerLoginPassword) 存储在 Amazon EC2 Systems Manager Parameter Store 中。可以使用 AWS CodeBuild 控制台在 Amazon EC2 Systems Manager 中创建参数。选择 Create a parameter，然后按照对话框中的说明操作。(在该对话框中，对于 KMS key，您可以选择指定您的账户中的 AWS KMS 密钥的 ARN。Amazon EC2 Systems Manager 使用此密钥在存储过程中加密参数的值并在检索过程中解密参数的值。)如果您使用 AWS CodeBuild 控制台创建参数，控制台将在参数被存储时以 /CodeBuild/ 作为它的开头。有关更多信息，请参阅 Amazon EC2 Systems Manager 用户指南 中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store 控制台演练](#)。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数，则构建项目的服务角色必须允许 ssm:GetParameters 操作。如果您之前选择了 Create a service role in your account，则 AWS CodeBuild 将自动在您的构建项目的默认服务角色中包含此操作。但是，如果您选择了 Choose an existing service role from your account，则必须将此操作单独包含在您的服务角色中。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的名称不以 /CodeBuild/ 开头的参数，且您选择了 Create a service role in your account，则您必须更新该服务角色以允许访问不以 /CodeBuild/ 开头的参数名称。这是因为该服务角色仅允许访问以 /CodeBuild/ 开头的参数名称。

您设置的任何环境变量都将替换现有的环境变量。例如，如果 Docker 映像已经包含一个名为 MY_VAR 的环境变量 (值为 my_value)，并且您设置了一个名为 MY_VAR 的环境变量 (值为 other_value)，那么 my_value 将被替换为 other_value。同样，如果 Docker 映像已经包含一个名为 PATH 的环境变量 (值为 /usr/local/sbin:/usr/local/bin)，并且您设置了一个名为 PATH 的环境变量 (值为 \$PATH:/usr/share/ant/bin)，那么 /usr/local/sbin:/usr/local/bin 将被替换为文本值 \$PATH:/usr/share/ant/bin。

请勿使用以 CODEBUILD_ 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。

如果具有相同名称的环境变量在多处都有定义，则应按照如下方式确定其值：

- 构建操作调用开始时的值优先级最高。
- 构建项目定义中的值优先级处于其次。
- 构建规范声明中的值优先级最低。

7. 选择 Start build。

有关此构建项目的详细信息，请参阅[查看构建详细信息 \(控制台\)](#) (p. 190)。

运行构建项目 (AWS CLI)

Note

要使用 AWS CodePipeline 运行 AWS CodeBuild 中的构建项目，可跳过这些步骤并按照[创建使用 AWS CodeBuild 的管道 \(AWS CLI\) \(p. 141\)](#)中的说明操作。

有关将 AWS CLI 与 AWS CodeBuild 结合使用的更多信息，请参阅[命令行参考 \(p. 207\)](#)。

1. 使用以下方法之一运行 start-build 命令：

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

如果您要运行的构建项目使用的是最新版本的构建输入项目和构建项目现有设置，请使用此方法。

```
aws codebuild start-build --generate-cli-skeleton
```

如果您要运行的构建项目具有早期版本的构建输入项目，或者如果您要覆盖构建输出项目、环境变量、构建规范或默认构建超时期限的设置，请使用此方法。

2. 如果您运行具有 --project-name 选项的 start-build 命令，请将 *project-name* 替换为构建项目的名称，然后跳至此过程中的第 6 步。要获取构建项目的列表，请参阅[查看构建项目名称的列表 \(p. 168\)](#)。
3. 如果您运行带 --idempotency-token 选项的 start-build 命令，则 start-build 请求将附带唯一的区分大小写的标识符或令牌。令牌在发出请求后的 12 小时内有效。如果您重复发出带相同令牌的 start-build 请求，但更改了参数，则 AWS CodeBuild 会返回“参数不匹配”错误。
4. 如果您运行具有 --generate-cli-skeleton 选项的 start-build 命令，则采用 JSON 格式的数据将出现在输出中。将数据复制到本地计算机上或安装 AWS CLI 的实例上某位置处的文件 (如 *start-build.json*) 中。修改所复制的数据，使其符合以下格式，然后保存结果：

```
{
  "projectName": "projectName",
  "sourceVersion": "sourceVersion",
  "artifactsOverride": {
    "type": "type",
    "location": "location",
    "path": "path",
    "namespaceType": "namespaceType",
    "name": "artifactsOverride-name",
    "packaging": "packaging"
  },
  "buildspecOverride": "buildspecOverride",
  "cacheOverride": {
    "location": "cacheOverride-location",
    "type": "cacheOverride-type"
  },
  "certificateOverride": "certificateOverride",
  "computeTypeOverride": "computeTypeOverride",
  "environmentTypeOverride": "environmentTypeOverride",
  "environmentVariablesOverride": {
    "name": "environmentVariablesOverride-name",
    "value": "environmentVariablesValue",
    "type": "environmentVariablesOverride-type"
  },
  "gitCloneDepthOverride": "gitCloneDepthOverride",
  "imageOverride": "imageOverride",
  "idempotencyToken": "idempotencyToken",
  "insecureSslOverride": "insecureSslOverride",
  "privilegedModeOverride": "privilegedModeOverride",
  "reportBuildStatusOverride": "reportBuildStatusOverride",
  "timeoutInMinutesOverride": "timeoutInMinutesOverride",
```

```
"sourceAuthOverride": "sourceAuthOverride",  
"sourceLocationOverride": "sourceLocationOverride",  
"serviceRoleOverride": "serviceRoleOverride",  
"sourceTypeOverride": "sourceTypeOverride"  
}
```

替换以下占位符：

- **projectName**：必需的字符串。用于此构建项目的构建项目名称。
- **sourceVersion**：可选字符串。要构建的源代码版本，如下所示：
 - 对于 Amazon S3，与您需要构建的输入 ZIP 文件的版本相对应的版本 ID。如果未指定 **sourceVersion**，则将使用最新版本。
 - 对于 AWS CodeCommit，与您需要构建的源代码版本相对应的提交 ID。如果未指定 **sourceVersion**，则将使用分支的 HEAD 提交 ID。（您无法指定 **sourceVersion** 标签名称，但您可以指定标签提交 ID。）
 - 对于 GitHub，指定提交 ID、拉取请求 ID、分支名称或与您要构建的源代码版本相对应的标签名称。如果指定了拉取请求 ID，则必须使用格式 **pr/pull-request-ID**（例如，pr/25）。如果指定了分支名称，则将使用分支的 HEAD 提交 ID。如果未指定 **sourceVersion**，则将使用分支的 HEAD 提交 ID。
 - 对于 Bitbucket，指定提交 ID、分支名称或与您要构建的源代码版本相对应的标签名称。如果指定了分支名称，则将使用分支的 HEAD 提交 ID。如果未指定 **sourceVersion**，则将使用分支的 HEAD 提交 ID。
- 以下占位符适用于 **artifactsOverride**。
 - **type**：可选字符串。构建项目中定义覆盖此构建项目的构建输出项目类型。
 - **location**：可选字符串。构建项目中定义覆盖此构建项目的构建输出项目位置。
 - **path**：可选字符串。构建项目中定义覆盖此构建项目的构建输出项目路径。
 - **namespaceType**：可选字符串。构建项目中定义覆盖此构建项目的构建输出项目路径类型。
 - **name**：可选字符串。构建项目中定义覆盖此构建项目的构建输出项目名称。
 - **packaging**：可选字符串。构建项目中定义覆盖此构建项目的构建输出项目打包类型。
- **buildspecOverride**：可选字符串。构建项目中定义覆盖此构建项目的构建规范声明。如果设置了该值，则它可以是内联构建规范定义，也可以是指向相对于内置 **CODEBUILD_SRC_DIR** 环境变量的值的替代构建规范文件的路径。
- 以下占位符适用于 **cacheOverride**。
 - **cacheOverride-location**：可选字符串。此构建的 ProjectCache 对象的位置，该对象将覆盖构建项目中指定的 ProjectCache 对象。**cacheOverride** 是可选的，它采用 ProjectCache 对象。ProjectCache 对象需要 **location**。
 - **cacheOverride-type**：可选字符串。此构建的 ProjectCache 对象的类型，该对象将覆盖构建项目中指定的 ProjectCache 对象。**cacheOverride** 是可选的，它采用 ProjectCache 对象。ProjectCache 对象需要 **type**。
- **certificateOverride**：可选字符串。此构建的证书的名称，该证书将覆盖构建项目中指定的证书。
- **environmentTypeOverride**：可选字符串。此构建的容器类型，该容器类型将覆盖构建项目中指定的容器类型。当前的有效字符串为 **LINUX_CONTAINER**。
- 以下占位符适用于 **environmentVariablesOverride**。
 - **environmentVariablesOverride-name**：可选字符串。构建项目中的环境变量名称，其值将会覆盖此构建项目中的相应值。
 - **environmentVariablesOverride-type**：可选字符串。构建项目中的环境变量类型，其值将会覆盖此构建项目中的相应值。
 - **environmentVariablesValue**：可选字符串。构建项目中定义的环境变量值，其值将会覆盖此构建项目中的相应值。
- **gitCloneDepthOverride**：可选字符串。构建项目中 Git clone depth 的值，您希望此构建项目会覆盖其值。如果您的源类型是 Amazon S3，则不支持此值。

- **imageOverride** : 可选字符串。此构建的映像的名称, 该映像将覆盖构建项目中指定的映像。
- **idempotencyToken** : 可选字符串。一个字符串, 该字符串用作令牌来指定构建请求是幂等的。您可以选择任何包含 64 个或更少字符的字符串。令牌在发出 start-build 请求后的 12 小时内有效。如果您重复发出带相同令牌的 start-build 请求, 但更改了参数, 则 AWS CodeBuild 会返回“参数不匹配”错误。
- **insecureSslOverride** : 可选的布尔值, 该值指定是否覆盖构建项目中指定的不安全的 SSL 设置。不安全的 SSL 设置确定是否忽略 SSL 警告, 并连接到项目源代码。此覆盖仅在构建的源为 GitHub Enterprise 时适用。
- **privilegedModeOverride** : 可选的布尔值。如果设置为 true, 则构建将覆盖构建项目中的特权模式。
- **reportBuildStatusOverride** : 可选布尔值, 指定是否向源提供商发送构建的开始和完成状态。如果用源提供商而非 GitHub 设置此项, 则会引发 `invalidInputException`。
- **sourceAuthOverride** : 可选字符串。此构建的授权类型, 该授权类型将覆盖构建项目中定义的授权类型。此覆盖仅在构建项目的源为 BitBucket 或 GitHub 时适用。
- **sourceLocationOverride** : 可选字符串。此构建的源位置, 该源位置将覆盖构建项目中定义的源位置。
- **serviceRoleOverride** : 可选字符串。此构建的服务角色的名称, 该角色将覆盖构建项目中指定的角色。
- **sourceTypeOverride** : 可选字符串。此构建的源输入类型, 该源输入将覆盖构建项目中定义的源输入。有效字符串包括: NO_SOURCE、CODECOMMIT、CODEPIPELINE、GITHUB、S3、BITBUCKET 和 GITHUB_ENTERPRISE。
- **timeoutInMinutesOverride** : 可选的编号。构建项目中定义覆盖此构建项目的构建超时分钟数。

Important

我们建议您将具有敏感值 (例如 AWS 访问密钥 ID、AWS 秘密访问密钥或密码) 的环境变量作为参数存储在 Amazon EC2 Systems Manager Parameter Store 中。如果 Amazon EC2 Systems Manager Parameter Store 中存储的参数的名称以 `/CodeBuild/` 开头 (例如, `/CodeBuild/dockerLoginPassword`), 则 AWS CodeBuild 可以使用该参数。可以使用 AWS CodeBuild 控制台在 Amazon EC2 Systems Manager 中创建参数。选择 `Create a parameter`, 然后按照对话框中的说明操作。(在该对话框中, 对于 KMS key, 您可以选择指定您的账户中的 AWS KMS 密钥的 ARN。Amazon EC2 Systems Manager 使用此密钥在存储过程中加密参数的值并在检索过程中解密参数的值。)如果您使用 AWS CodeBuild 控制台创建参数, 控制台将在参数被存储时以 `/CodeBuild/` 作为它的开头。但是, 如果您使用 Amazon EC2 Systems Manager Parameter Store 控制台创建参数, 则必须使用以 `/CodeBuild/` 开头的参数名称, 且必须将 Type 设置为 `Secure String`。有关更多信息, 请参阅 Amazon EC2 Systems Manager 用户指南 中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store 控制台演练](#)。

如果您的构建项目引用了 Amazon EC2 Systems Manager Parameter Store 中存储的参数, 则构建项目的服务角色必须允许 `ssm:GetParameters` 操作。如果您之前选择了 `Create a new service role in your account`, 则 AWS CodeBuild 将自动在您的构建项目的默认服务角色中包含此操作。但是, 如果您选择了 `Choose an existing service role from your account`, 则必须将此操作单独包含在您的服务角色中。

您设置的环境变量将替换现有的环境变量。例如, 如果 Docker 映像已经包含一个名为 `MY_VAR` 的环境变量 (值为 `my_value`), 并且您设置了一个名为 `MY_VAR` 的环境变量 (值为 `other_value`), 那么 `my_value` 将被替换为 `other_value`。同样, 如果 Docker 映像已经包含一个名为 `PATH` 的环境变量 (值为 `/usr/local/sbin:/usr/local/bin`), 并且您设置了一个名为 `PATH` 的环境变量 (值为 `$PATH:/usr/share/ant/bin`), 那么 `/usr/local/sbin:/usr/local/bin` 将被替换为文本值 `$PATH:/usr/share/ant/bin`。

请勿使用以 `CODEBUILD_` 打头的名称设置任何环境变量。此前缀是专为内部使用预留的。

如果具有相同名称的环境变量在多处都有定义, 则将按照如下方式确定环境变量的值:

- 构建操作调用开始时的值优先级最高。

5. 切换到包含您刚才保存的文件的目录，然后再次运行 `start-build` 命令。

```
aws codebuild start-build --cli-input-json file://start-build.json
```

开始自动运行构建 (AWS CLI)

运行 `create-webhook` 命令，如下所示：

```
aws codebuild create-webhook --project-name
```

```
{
  "webhook": {
    "url": "url"
  }
}
```

[illegible]

停止自动运行构建 (AWS CLI)

如果您已启用了此行为，则可以通过运行 `delete-webhook` 命令将其关闭，如下所示：

```
aws codebuild delete-webhook --project-name
```

- 其中，`project-name` 是包含要重建的源代码的构建项目的名称。

如果此命令成功，则输出中不会出现任何信息和错误。

Note

仅会从您的 AWS CodeBuild 项目中删除 Webhook。您还应该从 GitHub 或 GitHub Enterprise 存储库中删除 Webhook。

运行构建项目 (AWS 软件开发工具包)

要使用 AWS CodePipeline 运行 AWS CodeBuild 中的构建项目，可跳过这些步骤并按照[将 AWS CodePipeline 与 AWS CodeBuild 结合使用以测试代码和运行构建](#) (p. 136) 中的说明操作。

有关将 AWS CodeBuild 与 AWS 开发工具包结合使用的信息，请参阅[AWS 开发工具包和工具参考](#) (p. 208)。

查看 AWS CodeBuild 中的构建详细信息

要查看有关由 AWS CodeBuild 管理的构建的详细信息，您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包。

主题

- [查看构建详细信息 \(控制台\)](#) (p. 190)
- [查看构建详细信息 \(AWS CLI\)](#) (p. 190)
- [查看构建详细信息 \(AWS 软件开发工具包\)](#) (p. 191)
- [构建阶段过渡](#) (p. 191)

查看构建详细信息 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 执行以下任一操作：
 - 在导航窗格中，选择 Build history。在构建列表中的 Build run 列中，选择与构建对应的链接。
 - 在导航窗格中，选择 Build projects。在构建项目列表中的 Project 列中，选择与构建项目名称对应的链接。然后，在构建列表中的 Build run 列中，选择与构建对应的链接。

Note

默认情况下，仅显示十个最新的构建或构建项目。要查看更多构建或构建项目，请为 Builds per page 或 Projects per page 选择其他值，或者为 Viewing builds 或 Viewing projects 选择向后和向前箭头。

查看构建详细信息 (AWS CLI)

有关将 AWS CLI 与 AWS CodeBuild 结合使用的更多信息，请参阅[命令行参考](#) (p. 207)。

运行 `batch-get-builds` 命令：

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

替换以下占位符：

- *ids*：必填字符串。要查看其详细信息的一个或多个构建 ID。要指定多个构建 ID，请用空格分隔各个构建 ID。您最多可以指定 100 个构建 ID。要获取构建 ID 的列表，请参阅以下一个或多个主题：
 - [查看构建 ID 的列表 \(AWS CLI\) \(p. 193\)](#)
 - [查看构建项目的构建 ID 列表 \(AWS CLI\) \(p. 194\)](#)

例如，如果您运行此命令：

```
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
```

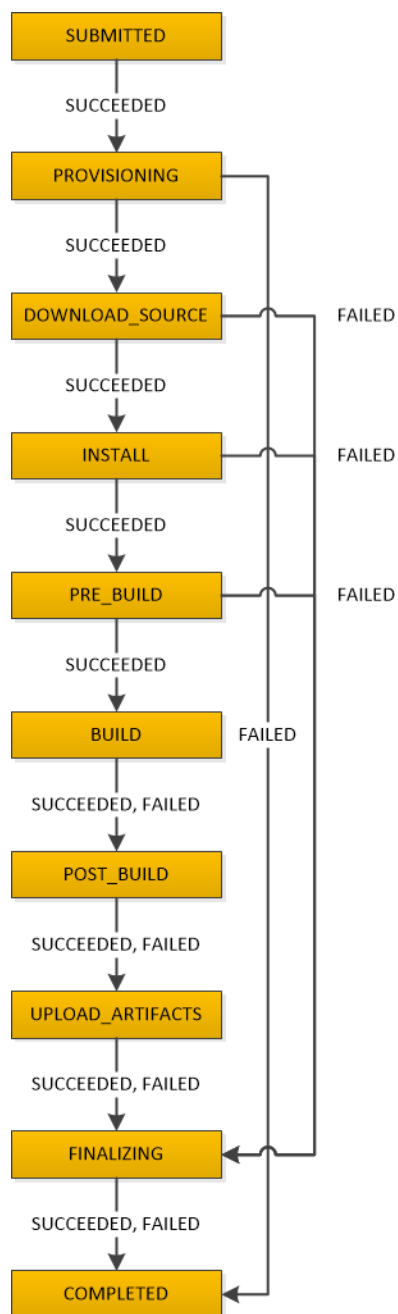
如果该命令成功运行，与“入门”中的 [查看汇总的构建信息 \(AWS CLI\) \(p. 14\)](#) 程序中所述内容类似的数据将出现在输出中。

查看构建详细信息 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 软件开发工具包结合使用的更多信息，请参阅 [AWS 开发工具包和工具参考 \(p. 208\)](#)。

构建阶段过渡

AWS CodeBuild 中的构建分阶段进行：



此处有一点需要格外注意：即使 **BUILD** 阶段失败，也会始终尝试 **UPLOAD_ARTIFACTS** 阶段。

查看 AWS CodeBuild 中的构建 ID 的列表

要查看由 AWS CodeBuild 管理的构建的构建 ID 列表，您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包。

主题

- [查看构建 ID 的列表 \(控制台\) \(p. 193\)](#)
- [查看构建 ID 的列表 \(AWS CLI\) \(p. 193\)](#)

- [查看构建 ID 的列表 \(AWS 软件开发工具包\) \(p. 194\)](#)

查看构建 ID 的列表 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 在导航窗格中，选择 Build history。

Note

默认情况下，仅显示十个最新构建。要查看更多构建，请为 Builds per page 选择不同值，或为 Viewing builds 选择向后和向前箭头。

查看构建 ID 的列表 (AWS CLI)

有关将 AWS CLI 与 AWS CodeBuild 结合使用的更多信息，请参阅 [命令行参考 \(p. 207\)](#)。

- 运行 list-builds 命令：

```
aws codebuild list-builds --sort-order sort-order --next-token next-token
```

替换上一命令中的以下占位符：

- *sort-order*：可选字符串。如何列出构建 ID。有效值包括 ASCENDING 和 DESCENDING。
- *next-token*：可选字符串。在上次运行时，如果列表中有 100 个以上的项目，则只能返回前 100 个项目，以及名为下一个令牌的唯一字符串。要获取列表中的下一批项目，请再次运行此命令，将下一个令牌添加到调用中。要获取列表中的所有项目，请利用每个后续的下一个令牌运行此命令，直到不再返回下一个令牌。

例如，如果您运行此命令：

```
aws codebuild list-builds --sort-order ASCENDING
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "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"
  ]
}
```

如果您再次运行此命令：

```
aws codebuild list-builds --sort-order ASCENDING --next-token 4AEA6u7J...The full token
has been omitted for brevity...MzY2OA==
```

与以下内容类似的结果可能会出现在输出中：

```
{
  "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"  
]  
}
```

查看构建 ID 的列表 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 软件开发工具包结合使用的更多信息，请参阅 [AWS 开发工具包和工具参考](#) (p. 208)。

查看 AWS CodeBuild 中构建项目的构建 ID 列表

您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包来查看 AWS CodeBuild 中构建项目的构建 ID 列表。

主题

- [查看构建项目的构建 ID 列表 \(控制台\)](#) (p. 194)
- [查看构建项目的构建 ID 列表 \(AWS CLI\)](#) (p. 194)
- [查看构建项目的构建 ID 列表 \(AWS 软件开发工具包\)](#) (p. 195)

查看构建项目的构建 ID 列表 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 在导航窗格中，选择 Build projects。在构建项目列表中的 Project 列中，选择构建项目。

Note

默认情况下，仅显示十个最新的构建或构建项目。要查看更多构建或构建项目，请为 Builds per page 或 Projects per page 选择其他值，或者为 Viewing builds 或 Viewing projects 选择向后和向前箭头。

查看构建项目的构建 ID 列表 (AWS CLI)

有关将 AWS CLI 与 AWS CodeBuild 结合使用的更多信息，请参阅 [命令行参考](#) (p. 207)。

运行 list-builds-for-project 命令，如下所示：

```
aws codebuild list-builds-for-project --project-name project-name --sort-order sort-order  
--next-token next-token
```

替换上一命令中的以下占位符：

- **project-name**：必需的字符串。要列出其构建 ID 的构建项目的名称。要获取构建项目的列表，请参阅 [查看构建项目名称的列表 \(AWS CLI\)](#) (p. 168)。
- **sort-order**：可选字符串。如何列出构建 ID。有效值包括 ASCENDING 和 DESCENDING。
- **next-token**：可选字符串。在上次运行时，如果列表中有 100 个以上的项目，则只能返回前 100 个项目，以及名为下一个令牌的唯一字符串。要获取列表中的下一批项目，请再次运行此命令，将下一个令牌添加到调用中。要获取列表中的所有项目，请借助随后返回的每个后续令牌不断运行此命令，直到不再返回后续令牌。

例如，如果您按照类似以下的方式运行此命令：

```
aws codebuild list-builds-for-project --project-name codebuild-demo-project --sort-order
ASCENDING
```

输出中可能会显示类似于以下内容的结果：

```
{
  "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"
  ]
}
```

如果您再次运行此命令：

```
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==
```

输出结果可能与以下内容类似：

```
{
  "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"
  ]
}
```

查看构建项目的构建 ID 列表 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 软件开发工具包结合使用的更多信息，请参阅 [AWS 开发工具包和工具参考](#) (p. 208)。

在 AWS CodeBuild 停止构建

要在 AWS CodeBuild 中停止构建，您可以使用 AWS CodeBuild 控制台、AWS CLI 或 AWS 开发工具包。

主题

- [停止构建 \(控制台\)](#) (p. 195)
- [停止构建 \(AWS CLI\)](#) (p. 196)
- [停止构建 \(AWS 软件开发工具包\)](#) (p. 196)

停止构建 (控制台)

1. Open the AWS CodeBuild console at <https://console.aws.amazon.com/codebuild/>.
2. 执行以下任一操作：
 - 如果显示了 **build-project-name:build-ID** 页面，请选择 Stop。
 - 在导航窗格中，选择 Build history。在构建列表中，选中与构建对应的框，然后选择 Stop。

- 在导航窗格中，选择 Build projects。在构建项目列表中的 Project 列中，选择与构建项目的名称对应的链接。在构建列表中，选中与构建对应的框，然后选择 Stop。

Note

默认情况下，仅显示最新的 10 个构建或构建项目。要查看更多构建或构建项目，请为 Builds per page 或 Projects per page 选择其他值，或者为 Viewing builds 或 Viewing projects 选择向后和向前箭头。

如果 AWS CodeBuild 无法成功停止构建 (例如，构建过程已完成)，则 Stop 按钮将禁用，也可能完全消失。

停止构建 (AWS CLI)

- 运行 stop-build 命令：

```
aws codebuild stop-build --id id
```

在上述命令中，替换以下占位符：

- id*：必填字符串。要停止的构建的 ID。要获取构建 ID 的列表，请参阅以下主题：
 - [查看构建 ID 的列表 \(AWS CLI\) \(p. 193\)](#)
 - [查看构建项目的构建 ID 列表 \(AWS CLI\) \(p. 194\)](#)

如果 AWS CodeBuild 成功停止构建，则输出中 build 对象的 buildStatus 值将为 STOPPED。

如果 AWS CodeBuild 无法成功停止构建 (例如，构建已完成)，则输出中 build 对象的 buildStatus 值将为最终构建状态 (例如，SUCCEEDED)。

停止构建 (AWS 软件开发工具包)

有关将 AWS CodeBuild 与 AWS 软件开发工具包结合使用的更多信息，请参阅 [AWS 开发工具包和工具参考 \(p. 208\)](#)。

在 AWS CodeBuild 中删除构建

要在 AWS CodeBuild 中删除构建，您可以使用 AWS CLI 或 AWS 开发工具包。

删除构建 (AWS CLI)

运行 batch-delete-builds 命令：

```
aws codebuild batch-delete-builds --ids ids
```

在上述命令中，替换以下占位符：

- ids*：必填字符串。要删除的构建的 ID。要指定多个构建，请用空格将每个构建 ID 分隔开。要获取构建 ID 的列表，请参阅以下主题：
 - [查看构建 ID 的列表 \(AWS CLI\) \(p. 193\)](#)
 - [查看构建项目的构建 ID 列表 \(AWS CLI\) \(p. 194\)](#)

如果成功，buildsDeleted 数组将显示在输出中，其中包含已成功删除的每个构建的 Amazon 资源名称 (ARN)。有关未成功删除的构建的信息将显示在输出中的 buildsNotDeleted 数组中。

例如，如果您运行此命令：

```
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
```

与以下内容类似的信息将显示在输出中：

```
{
  "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"
  ]
}
```

删除构建 (AWS 开发工具包)

有关将 AWS CodeBuild 与 AWS 开发工具包结合使用的信息，请参阅 [AWS 开发工具包和工具参考 \(p. 208\)](#)。

高级主题

此部分包含几个高级主题，这些主题对于经验更丰富的 AWS CodeBuild 用户很有用。

主题

- [高级设置 \(p. 198\)](#)
- [AWS CodeBuild 的命令行参考 \(p. 207\)](#)
- [适用于 AWS CodeBuild 的 AWS 开发工具包和工具参考 \(p. 208\)](#)
- [指定 AWS CodeBuild 终端节点 \(p. 209\)](#)
- [AWS CodeBuild 的身份验证和访问控制 \(p. 211\)](#)
- [使用 AWS CloudTrail 记录 AWS CodeBuild API 调用 \(p. 222\)](#)

高级设置

如果您第一次遵循 [入门 \(p. 4\)](#) 中的步骤访问 AWS CodeBuild，那么您可能不需要参考本主题中的信息。但是，随着您继续使用 AWS CodeBuild，您会需要执行这些操作：例如，为您所在组织内的 IAM 组 and 用户提供 AWS CodeBuild 的访问权限、修改 IAM 中的现有服务角色或 AWS KMS 中的客户主密钥以访问 AWS CodeBuild，或者跨您所在组织的工作站设置 AWS CLI 以访问 AWS CodeBuild。本主题将介绍如何完成相关的设置步骤。

我们假定您已经有一个 AWS 账户。但是，如果您还没有账户，请转到 <http://aws.amazon.com>，选择 Sign In to the Console，然后按照在线说明进行操作。

主题

- [为 IAM 组或 IAM 用户添加 AWS CodeBuild 访问权限 \(p. 198\)](#)
- [创建 AWS CodeBuild 服务角色 \(p. 202\)](#)
- [为 AWS CodeBuild 创建和配置 AWS KMS CMK \(p. 205\)](#)
- [安装和配置 AWS CLI \(p. 207\)](#)

为 IAM 组或 IAM 用户添加 AWS CodeBuild 访问权限

要以 IAM 组或 IAM 用户身份访问 AWS CodeBuild，您必须添加访问权限。本部分介绍了如何使用 IAM 控制台或 AWS CLI 完成此操作。

如果以 AWS 根账户 (不推荐) 或 AWS 账户中的管理员 IAM 用户身份访问 AWS CodeBuild，则无需遵循这些说明。

有关 AWS 根账户和管理员 IAM 用户的信息，请参阅 IAM 用户指南中的[账户根用户](#)和[创建您的第一个 IAM 管理员用户和组](#)。

为 IAM 组或 IAM 用户添加 AWS CodeBuild 访问权限 (控制台)

1. 通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。

您应该已使用以下任一身份登录到 AWS 管理控制台：

- 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅 IAM 用户指南中的[账户根用户](#)。
- AWS 账户中的 IAM 管理员用户。有关更多信息，请参阅 IAM 用户指南中的[创建您的第一个 IAM 管理员用户和组](#)。
- AWS 账户中的 IAM 用户，具有执行以下最基本操作的权限：

```
iam:AttachGroupPolicy
iam:AttachUserPolicy
iam:CreatePolicy
iam:ListAttachedGroupPolicies
iam:ListAttachedUserPolicies
iam:ListGroups
iam:ListPolicies
iam:ListUsers
```

有关更多信息，请参阅 IAM 用户指南中的 [IAM 策略概述](#)。

2. 在导航窗格中，选择 Policies。
3. 要为 IAM 组或 IAM 用户添加一组自定义的 AWS CodeBuild 访问权限，请向前跳到此过程的第 4 步。

要向 IAM 组或 IAM 用户添加一组默认的 AWS CodeBuild 访问权限，请依次选择 Policy Type 和 AWS Managed，然后执行以下操作：

- 要添加对 AWS CodeBuild 的完全访问权限，请选中名为 AWSCodeBuildAdminAccess 的框。然后，选择 Policy Actions 和 Attach。选中目标 IAM 组或 IAM 用户旁的框，然后选择 Attach Policy。对名为 AmazonS3ReadOnlyAccess 和 IAMFullAccess 的策略重复执行此操作。
- 要为除构建项目管理之外的所有内容添加对 AWS CodeBuild 的访问权限，请选中名为 AWSCodeBuildDeveloperAccess 的框。然后，选择 Policy Actions 和 Attach。选中目标 IAM 组或 IAM 用户旁的框，然后选择 Attach Policy。对名为 AmazonS3ReadOnlyAccess 的策略重复执行此操作。
- 要添加对 AWS CodeBuild 的只读访问权限，请选中名为 AWSCodeBuildReadOnlyAccess 的框。选中目标 IAM 组或 IAM 用户旁的框，然后选择 Attach Policy。对名为 AmazonS3ReadOnlyAccess 的策略重复执行此操作。

现在，您已经为 IAM 组或 IAM 用户添加了一组默认的 AWS CodeBuild 访问权限。跳过此过程中的其余步骤。

4. 选择 Create Policy。
5. 在 Create Policy 页面上的 Create Your Own Policy 旁，选择 Select。
6. 在 Review Policy 页面上，对于 Policy Name，键入策略的名称（例如，**CodeBuildAccessPolicy**）。如果您使用了其他名称，请在整个过程中使用此名称进行替换。
7. 对于 Policy Document，键入以下内容，然后选择 Create Policy。

```
{
  "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

此策略允许访问所有 AWS CodeBuild 操作以及潜在的大量 AWS 资源。要限制对特定 AWS CodeBuild 操作的访问权限，请在 AWS CodeBuild 策略声明中更改 `codebuild:*` 的值。有关更多信息，请参阅 [身份验证和访问控制 \(p. 211\)](#)。要限制对特定 AWS 资源的访问权限，请更改 `Resource` 对象的值。有关更多信息，请参阅 [身份验证和访问控制 \(p. 211\)](#)。

8. 在导航窗格中，选择 Groups 或 Users。
9. 在组或用户列表中，选择要向其添加 AWS CodeBuild 访问权限的 IAM 组或 IAM 用户的名称。
10. 对于组，在组设置页面上的 Permissions 选项卡上，展开 Managed Policies，然后选择 Attach Policy。

对于用户，在用户设置页面上的 Permissions 选项卡上，选择 Add permissions。

11. 对于组，在 Attach Policy 页面上，选择 CodeBuildAccessPolicy，然后选择 Attach Policy。

对于用户，在 Add permissions 页面上，选择 Attach existing policies directly。依次选择 CodeBuildAccessPolicy、Next: Review 和 Add permissions。

为 IAM 组或 IAM 用户添加 AWS CodeBuild 访问权限 (AWS CLI)

1. 如前面的过程所述，确保您已为 AWS CLI 配置了与某个 IAM 实体相对应的 AWS 访问密钥和 AWS 秘密访问密钥。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的[开始设置 AWS Command Line Interface](#)。
2. 要为 IAM 组或 IAM 用户添加一组自定义的 AWS CodeBuild 访问权限，请向前跳到此过程的第 3 步。

要为 IAM 组或 IAM 用户添加一组默认的 AWS CodeBuild 访问权限，请执行以下操作：

运行以下任一命令，具体取决于您是否要为 IAM 组或 IAM 用户添加权限：

```
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
```

您必须运行该命令三次，将 **group-name** 或 **user-name** 替换为 IAM 组名称或 IAM 用户名称，然后为下面每个策略 Amazon 资源名称 (ARN) 替换 **policy-arn** 一次：

- 要添加对 AWS CodeBuild 的完全访问权限，请使用以下策略 ARN：
 - `arn:aws:iam::aws:policy/AWSCodeBuildAdminAccess`
 - `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`

- `arn:aws:iam::aws:policy/IAMFullAccess`
- 要添加除构建项目管理以外的所有 AWS CodeBuild 访问权限，请使用以下策略 ARN：
 - `arn:aws:iam::aws:policy/AWSCodeBuildDeveloperAccess`
 - `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`
- 要添加对 AWS CodeBuild 的只读访问权限，请使用以下策略 ARN：
 - `arn:aws:iam::aws:policy/AWSCodeBuildReadOnlyAccess`
 - `arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess`

现在，您已经为 IAM 组或 IAM 用户添加了一组默认的 AWS CodeBuild 访问权限。跳过此过程中的其余步骤。

3. 在安装 AWS CLI 的本地工作站或实例上的空目录中，创建名为 `put-group-policy.json` 或 `put-user-policy.json` 的文件。如果您使用了其他文件名称，请在整个过程中使用此名称进行替换。

```
{
  "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

此策略允许访问所有 AWS CodeBuild 操作以及潜在的大量 AWS 资源。要限制对特定 AWS CodeBuild 操作的访问权限，请在 AWS CodeBuild 策略声明中更改 `codebuild:*` 的值。有关更多信息，请参阅 [身份验证和访问控制 \(p. 211\)](#)。要限制对特定 AWS 资源的访问权限，请更改相关 Resource 对象的值。有关更多信息，请参阅 [身份验证和访问控制 \(p. 211\)](#) 或特定 AWS 服务的安全文档。

4. 切换到您保存该文件的目录，然后运行以下任一命令。您可以为 `CodeBuildGroupAccessPolicy` 和 `CodeBuildUserAccessPolicy` 使用不同的值。如果您使用了不同的值，请在此处使用这些值进行替换。

对于 IAM 组：

```
aws iam put-group-policy --group-name group-name --policy-name  
CodeBuildGroupAccessPolicy --policy-document file://put-group-policy.json
```

对于 IAM 用户：

```
aws iam put-user-policy --user-name user-name --policy-name CodeBuildUserAccessPolicy  
--policy-document file://put-user-policy.json
```

在前面的命令中，将 *group-name* 或 *user-name* 替换为目标 IAM 组或 IAM 用户的名称。

创建 AWS CodeBuild 服务角色

您需要一个 AWS CodeBuild 服务角色，以便 AWS CodeBuild 能代表您与相关 AWS 服务进行交互。您可以使用 AWS CodeBuild 或 AWS CodePipeline 控制台创建一个 AWS CodeBuild 服务角色。想要了解有关信息，请参阅：

- [创建构建项目 \(控制台\) \(p. 154\)](#)
- [创建使用了 AWS CodeBuild 的管道 \(AWS CodePipeline 控制台\) \(p. 137\)](#)
- [将 AWS CodeBuild 构建操作添加到管道 \(AWS CodePipeline 控制台\) \(p. 144\)](#)
- [更改构建项目的设置 \(控制台\) \(p. 174\)](#)

如果您不打算使用这些控制台，本部分介绍了如何使用 IAM 控制台或 AWS CLI 创建 AWS CodeBuild 服务角色。

Note

此页上描述的服务角色包含一项策略，可授予使用 AWS CodeBuild 时所需的最低权限。您可能需要根据使用案例添加额外的权限。例如，如果您希望将 AWS CodeBuild 与 Amazon Virtual Private Cloud 配合使用，则您创建的服务角色需要以下策略中的权限：[创建 AWS CodeBuild 服务角色 \(p. 202\)](#)。

创建 AWS CodeBuild 服务角色 (控制台)

1. 通过以下网址打开 IAM 控制台：<https://console.aws.amazon.com/iam/>。

您应该已使用以下任一身份登录到控制台：

- 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅 IAM 用户指南中的[账户根用户](#)。
- AWS 账户中的 IAM 管理员用户。有关更多信息，请参阅 IAM 用户指南中的[创建您的第一个 IAM 管理员用户和组](#)。
- AWS 账户中的 IAM 用户，具有执行以下最基本操作的权限：

```
iam:AddRoleToInstanceProfile  
iam:AttachRolePolicy  
iam:CreateInstanceProfile  
iam:CreatePolicy  
iam:CreateRole  
iam:GetRole  
iam:ListAttachedRolePolicies  
iam:ListPolicies  
iam:ListRoles  
iam:PassRole  
iam:PutRolePolicy
```

```
iam:UpdateAssumeRolePolicy
```

有关更多信息，请参阅 IAM 用户指南中的 [IAM 策略概述](#)。

2. 在导航窗格中，选择 Policies。
3. 选择 Create Policy。
4. 在 Create Policy 页面上，选择 JSON。
5. 对于 JSON 策略，键入以下内容，然后选择 Review Policy：

```
{
  "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

此策略包含允许访问潜在的大量 AWS 资源的语句。要限制 AWS CodeBuild 访问特定的 AWS 资源，请更改 Resource 数组的值。有关更多信息，请参阅有关 AWS 服务的安全文档。

- 在 Review Policy 页面上，对于 Policy Name，键入策略的名称 (例如，**CodeBuildServiceRolePolicy**)，然后选择 Create policy。

Note

如果您使用了其他名称，请在整个过程中使用此名称进行替换。

- 在导航窗格中，选择 Roles。
- 选择 Create role。
- 在 Create role 页面上，在已选择 AWS Service 的情况下，选择 CodeBuild (因为该服务将使用此角色)，然后选择 Next:Permissions。
- 在 Attach permissions policies 页面上，选择 CodeBuildServiceRolePolicy，然后选择 Next: Review。
- 在 Create role and review 页面上，对于 Role name，键入角色的名称 (例如，**CodeBuildServiceRole**)，然后选择 Create role。

创建 AWS CodeBuild 服务角色 (AWS CLI)

- 如前面的过程所述，确保您已为 AWS CLI 配置了与某个 IAM 实体相对应的 AWS 访问密钥和 AWS 秘密访问密钥。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的[开始设置 AWS Command Line Interface](#)。
- 在安装了 AWS CLI 的本地工作站或实例的空目录中，创建名为 create-role.json 和 put-role-policy.json 的两个文件。如果您选择了其他文件名称，请在整个过程中使用此名称进行替换。

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

此策略包含允许访问潜在的大量 AWS 资源的语句。要限制 AWS CodeBuild 访问特定的 AWS 资源，请更改 Resource 数组的值。有关更多信息，请参阅有关 AWS 服务的安全文档。

3. 切换到您保存上述文件的目录，然后按照这个顺序运行以下两个命令，一次运行一个。您可以为 CodeBuildServiceRole 和 CodeBuildServiceRolePolicy 使用不同的值，但请务必在此处替换它们。

```
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
```

为 AWS CodeBuild 创建和配置 AWS KMS CMK

要使 AWS CodeBuild 加密其构建输出项目，需要具备对 AWS KMS 客户主密钥 (CMK) 的访问权限。默认情况下，AWS CodeBuild 使用您 AWS 账户中适用于 Amazon S3 的 AWS 托管的 CMK。

如果您不想使用此 CMK，则必须自行创建并配置一个客户托管的 CMK。本部分介绍了如何通过 IAM 控制台执行此操作。

有关 CMK 的信息，请参阅 AWS KMS 开发人员指南中的 [AWS Key Management Service 概念](#)和[创建密钥](#)。

要配置由 AWS CodeBuild 使用的 CMK，请遵循 AWS KMS 开发人员指南中[修改密钥策略](#)的“如何修改密钥策略”部分中的说明。然后将以下语句 (在 **### BEGIN ADDING STATEMENTS HERE ###** 和 **### END ADDING STATEMENTS HERE ###** 之间) 添加到密钥策略中。为了简洁起见，也为了帮您查找添加语句的位置，此处使用了省略号 (...)。请勿删除任何语句，也不要将这些省略号键入密钥策略中。

```
{
  "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** 表示与 AWS CodeBuild 关联的 Amazon S3 存储桶所在的 AWS 区域的 ID (例如, us-east-1)。

- `account-ID` 表示拥有 CMK 的 AWS 账户的 ID。
- `CodeBuild-service-role` 表示您之前在此主题中创建或标识的 AWS CodeBuild 服务角色的名称。

Note

要通过 IAM 控制台创建或配置 CMK，您必须先使用以下任一身份登录该 AWS 管理控制台：

- 您的 AWS 根账户。我们不建议这么做。有关更多信息，请参阅 IAM 用户指南中的[账户根用户](#)。
- AWS 账户中的 IAM 管理员用户。有关更多信息，请参阅 IAM 用户指南中的[创建您的第一个 IAM 管理员用户和组](#)。
- AWS 账户中具有创建或修改 CMK 权限的 IAM 用户。有关更多信息，请参阅 AWS KMS 开发人员指南中的[使用 AWS KMS 控制台所需的权限](#)。

安装和配置 AWS CLI

要访问 AWS CodeBuild，您可以将 AWS CLI 与 AWS CodeBuild 控制台、AWS CodePipeline 控制台或 AWS 开发工具包结合使用，或者改用后者。要安装和配置 AWS CLI，请参阅 AWS Command Line Interface 用户指南中的[开始设置 AWS Command Line Interface](#)。

1. 运行以下命令以确认您安装的 AWS CLI 是否支持 AWS CodeBuild：

```
aws codebuild list-builds
```

如果成功，将在输出中显示与以下内容类似的信息：

```
{
  "ids": []
}
```

空方括号表示您尚未运行任何构建。

2. 如果输出一个错误，您必须卸载当前版本的 AWS CLI，然后安装最新版本。有关更多信息，请参阅 AWS Command Line Interface 用户指南中的[卸载 AWS CLI](#)和[安装 AWS Command Line Interface](#)。

AWS CodeBuild 的命令行参考

AWS CLI 可提供命令来自动化 AWS CodeBuild。使用本主题中的信息作为 [AWS Command Line Interface 用户指南](#) 和 [AWS CLI Reference for AWS CodeBuild](#) 的补充。

不是您要找的内容？如果要使用 AWS 开发工具包调用 AWS CodeBuild，请参阅 [AWS 开发工具包和工具参考 \(p. 208\)](#)。

要使用本主题中的信息，您应该已经安装了 AWS CLI，并已按照 [安装和配置 AWS CLI \(p. 207\)](#) 中的说明将其配置为与 AWS CodeBuild 配合使用。

要使用 AWS CLI 指定 AWS CodeBuild 的终端节点，请参阅[指定 AWS CodeBuild 终端节点 \(AWS CLI\) \(p. 209\)](#)。

运行此命令可获取 AWS CodeBuild 命令的列表。

```
aws codebuild help
```

运行此命令可获取有关 AWS CodeBuild 命令的信息，其中 `command-name` 是命令的名称。

```
aws codebuild command-name help
```

AWS CodeBuild 命令包括：

- `batch-delete-builds`：删除 AWS CodeBuild 中的一个或多个构建。有关更多信息，请参阅 [删除构建 \(AWS CLI\) \(p. 196\)](#)。
- `batch-get-builds`：获取有关 AWS CodeBuild 中多个构建的信息。有关更多信息，请参阅 [查看构建详细信息 \(AWS CLI\) \(p. 190\)](#)。
- `batch-get-projects`：获取有关一个或多个指定构建项目的信息。有关更多信息，请参阅 [查看构建项目的详细信息 \(AWS CLI\) \(p. 170\)](#)。
- `create-project`：创建构建项目。有关更多信息，请参阅 [创建构建项目 \(AWS CLI\) \(p. 162\)](#)。
- `delete-project`：删除构建项目。有关更多信息，请参阅 [删除构建项目 \(AWS CLI\) \(p. 182\)](#)。
- `list-builds`：列出 AWS CodeBuild 中的构建的 Amazon 资源名称 (ARN)。有关更多信息，请参阅 [查看构建 ID 的列表 \(AWS CLI\) \(p. 193\)](#)。
- `list-builds-for-project`：获取与指定构建项目相关联的构建 ID 的列表。有关更多信息，请参阅 [查看构建项目的构建 ID 列表 \(AWS CLI\) \(p. 194\)](#)。
- `list-curated-environment-images`：获取由 AWS CodeBuild 管理的可用于构建的 Docker 映像的列表。有关更多信息，请参阅 [AWS CodeBuild 提供的 Docker 映像 \(p. 114\)](#)。
- `list-projects`：获取构建项目名称的列表。有关更多信息，请参阅 [查看构建项目名称的列表 \(AWS CLI\) \(p. 168\)](#)。
- `start-build`：开始运行构建。有关更多信息，请参阅 [运行构建项目 \(AWS CLI\) \(p. 186\)](#)。
- `stop-build`：尝试停止运行指定的构建。有关更多信息，请参阅 [停止构建 \(AWS CLI\) \(p. 196\)](#)。
- `update-project`：更改指定构建项目的信息。有关更多信息，请参阅 [更改构建项目的设置 \(AWS CLI\) \(p. 181\)](#)。

适用于 AWS CodeBuild 的 AWS 开发工具包和工具参考

要使用一种 AWS 开发工具包或工具来自动化 AWS CodeBuild，请参阅以下资源。

如果要使用 AWS CLI 运行 AWS CodeBuild，请参阅 [命令行参考 \(p. 207\)](#)。

适用于 AWS CodeBuild 的受支持的 AWS 开发工具包和工具

以下 AWS 开发工具包和工具支持 AWS CodeBuild：

- [适用于 C++ 的 AWS 软件开发工具包](#)。有关更多信息，请参阅适用于 C++ 的 AWS 软件开发工具包 API 参考的 `Aws::CodeBuild` 命名空间部分。
- [适用于 Go 的 AWS 软件开发工具包](#)。有关更多信息，请参阅适用于 Go 的 AWS 软件开发工具包 API 参考的 `codebuild` 部分。
- [适用于 Java 的 AWS 软件开发工具包](#)。有关更多信息，请参阅适用于 Java 的 AWS 软件开发工具包 API 参考的 `com.amazonaws.services.codebuild` 和 `com.amazonaws.services.codebuild.model` 部分。
- [适用于浏览器中 JavaScript 的 AWS 软件开发工具包](#)和[适用于 Node.js 中 JavaScript 的 AWS 软件开发工具包](#)。有关更多信息，请参阅类：[AWS.AWS CodeBuild](#) 部分 (位于适用于 JavaScript 的 AWS 软件开发工具包 API 参考中)。

- 适用于 .NET 的 AWS 软件开发工具包。有关更多信息，请参阅 [Amazon.CodeBuild](#) 和 [Amazon.CodeBuild.Model](#) 命名空间部分 (位于适用于 .NET 的 AWS 软件开发工具包 API 参考中)。
- 适用于 PHP 的 AWS 软件开发工具包。有关更多信息，请参阅适用于 PHP 的 AWS 软件开发工具包 API 参考的命名空间 [Aws\CodeBuild](#) 部分。
- 适用于 Python 的 AWS 软件开发工具包 (Boto3)。有关更多信息，请参阅 Boto 3 文档的 [CodeBuild](#) 部分。
- 适用于 Ruby 的 AWS 软件开发工具包。有关更多信息，请参阅适用于 Ruby 的 AWS 软件开发工具包 API 参考的模块：[Aws::CodeBuild](#) 部分。
- 适用于 PowerShell 的 AWS 工具。有关更多信息，请参阅适用于 PowerShell 的 AWS 工具 Cmdlet 参考的 [AWS CodeBuild](#) 部分。

指定 AWS CodeBuild 终端节点

您可以使用 AWS Command Line Interface (AWS CLI) 或 AWS 开发工具包之一指定由 AWS CodeBuild 使用的终端节点。AWS CodeBuild 可用的每个区域都有一个终端节点。除了一个区域终端节点之外，四个区域还有联邦信息处理标准 (FIPS) 终端节点。有关 FIPS 终端节点的更多信息，请参阅 [FIPS 140-2 概述](#)。

可以选择指定终端节点。如果您未明确告知 AWS CodeBuild 要使用哪个终端节点，该服务将使用与您的 AWS 账户所用区域关联的终端节点。AWS CodeBuild 从不默认使用 FIPS 终端节点。如果您希望使用 FIPS 终端节点，则必须使用以下方法之一将 AWS CodeBuild 与其关联。

Note

您可以使用 AWS 开发工具包，通过别名或区域名称指定终端节点。如果使用的是 AWS CLI，则您必须使用终端节点的完整名称。

有关可用于 AWS CodeBuild 的终端节点，请参阅 [AWS CodeBuild 区域和终端节点](#)。

主题

- [指定 AWS CodeBuild 终端节点 \(AWS CLI\) \(p. 209\)](#)
- [指定 AWS CodeBuild 终端节点 \(AWS 开发工具包\) \(p. 209\)](#)

指定 AWS CodeBuild 终端节点 (AWS CLI)

您可以在任意 AWS CodeBuild 命令中使用 `--endpoint-url` 参数，通过 AWS CLI 指定访问 AWS CodeBuild 时所用的终端节点。例如，运行此命令以获取在美国东部（弗吉尼亚北部）地区中使用联邦信息处理标准 (FIPS) 终端节点的项目生成名称列表：

```
aws codebuild list-projects --endpoint-url https://codebuild-fips.us-east-1.amazonaws.com
```

在终端节点的开头包括 `https://`。

`--endpoint-url` AWS CLI 参数可供所有 AWS 服务使用。有关此参数和其他 AWS CLI 参数的更多信息，请参阅 [AWS CLI Command Reference](#)。

指定 AWS CodeBuild 终端节点 (AWS 开发工具包)

您可以使用 AWS 开发工具包指定访问 AWS CodeBuild 时使用的终端节点。虽然此示例使用 [适用于 Java 的 AWS 开发工具包](#)，不过您可以指定具有其他 AWS 开发工具包的终端节点。

构造 `AWSCodeBuild` 客户端时使用 `withEndpointConfiguration` 方法。下面是使用的格式：

```
AWSCodeBuild awsCodeBuild = AWSCodeBuildClientBuilder.standard().
    withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration("endpoint",
    "region")).
    withCredentials(new AWSStaticCredentialsProvider(sessionCredentials)).
    build();
```

有关 `AWSCodeBuildClientBuilder` 的信息，请参阅类 [AWSCodeBuildClientBuilder](#)。

在 `withCredentials` 中使用的凭证的类型必须为 `AWSCredentialsProvider`。有关更多信息，请参阅 [使用 AWS 凭证](#)。

不要在终端节点的开头包括 `https://`。

如果您希望指定非 FIPS 终端节点，则可以使用区域而非实际终端节点。例如，要在美国东部（弗吉尼亚北部）区域中指定终端节点，您可以使用 `us-east-1` 而不是完整的终端节点名称 `codebuild.us-east-1.amazonaws.com`。

如果您要指定 FIPS 终端节点，可以使用别名来简化代码。只有 FIPS 终端节点有别名。其他终端节点必须使用其区域或完整名称指定。

下表列出了四个可用 FIPS 终端节点的各自的别名。

区域名称	区域	终端节点	别名
美国东部 (弗吉尼亚北部)	us-east-1	codebuild-fips.us-east-1.amazonaws.com	us-east-1-fips
美国东部 (俄亥俄州)	us-east-2	codebuild-fips.us-east-2.amazonaws.com	us-east-2-fips
美国西部 (加利福尼亚北部)	us-west-1	codebuild-fips.us-west-1.amazonaws.com	us-west-1-fips
美国西部 (俄勒冈)	us-west-2	codebuild-fips.us-west-2.amazonaws.com	us-west-2-fips

要指定使用美国西部（俄勒冈）区域中的 FIPS 终端节点，请使用别名：

```
AWSCodeBuild awsCodeBuild = AWSCodeBuildClientBuilder.standard().
    withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration("us-west-2-fips",
    "us-west-2")).
    withCredentials(new AWSStaticCredentialsProvider(sessionCredentials)).
    build();
```

指定使用美国东部（弗吉尼亚北部）区域中的非 FIPS 终端节点：

```
AWSCodeBuild awsCodeBuild = AWSCodeBuildClientBuilder.standard().
    withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration("us-east-1", "us-
    east-1")).
    withCredentials(new AWSStaticCredentialsProvider(sessionCredentials)).
```



```
build();
```

指定使用亚太地区（孟买）区域中的非 FIPS 终端节点：

```
AWSCodeBuild awsCodeBuild = AWSCodeBuildClientBuilder.standard().  
    withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration("ap-south-1", "ap-  
south-1")).  
    withCredentials(new AWSStaticCredentialsProvider(sessionCredentials)).  
    build();
```

AWS CodeBuild 的身份验证和访问控制

访问 AWS CodeBuild 需要凭证。这些凭证必须有权访问 AWS 资源，如存储和检索 Amazon S3 存储桶中的构建项目并查看构建项目的 Amazon CloudWatch Logs。下面几节介绍如何使用[AWS Identity and Access Management \(IAM\)](#) 和 AWS CodeBuild 帮助保护对您的资源的访问：

- [身份验证 \(p. 211\)](#)
- [访问控制 \(p. 212\)](#)

身份验证

您可以以下面任一类型的身份访问 AWS：

- **AWS 账户根用户** – 注册 AWS 时，您需要提供与您的 AWS 账户关联的电子邮件地址和密码。这些是您的根凭证，它们提供对您所有 AWS 资源的完全访问权限。

Important

出于安全考虑，我们建议您仅使用根凭证创建管理员用户，该用户是对您的 AWS 账户具有完全访问权的 IAM 用户。随后，您可以使用此管理员用户来创建具有有限权限的其他 IAM 用户和角色。有关更多信息，请参阅《IAM 用户指南 指南》中的 [IAM 最佳实践](#)和[创建管理员用户和组](#)。

- **IAM 用户** – [IAM 用户](#)就是您的 AWS 账户中的一种身份，它具有自定义权限（例如，用于在 AWS CodeBuild 中创建构建项目的权限）。您可以使用 IAM 用户名和密码来登录以保护 AWS 网页，如 [AWS 管理控制台](#)、[AWS 开发论坛](#)或 [AWS Support Center](#)。

除了用户名和密码之外，您还可以为每个用户生成[访问密钥](#)。在通过 [AWS 软件开发工具包之一](#)或使用 [AWS Command Line Interface \(AWS CLI\)](#) 以编程方式访问 AWS 服务时，可以使用这些密钥。AWS 软件开发工具包和 AWS CLI 工具使用访问密钥对您的请求进行加密签名。如果您不使用 AWS 工具，则必须自行对请求签名。AWS CodeBuild 支持签名版本 4，这是用于对入站 API 请求进行身份验证的协议。有关对请求进行身份验证的更多信息，请参阅 AWS General Reference 中的[签名版本 4 签名流程](#)。

- **IAM 角色** – [IAM 角色](#)类似于 IAM 用户，但不与特定人员关联。利用 IAM 角色，您可以获得可用于访问 AWS 服务和资源的临时访问密钥。具有临时凭证的 IAM 角色在以下情况下很有用：
 - **联合身份用户访问** – 您可以不创建 IAM 用户，而是使用来自 AWS Directory Service、您的企业用户目录或 Web 身份提供商的既有用户身份。他们被称为联合身份用户。在通过[身份提供商](#)请求访问权限时，AWS 将为联合用户分配角色。有关联合身份用户的更多信息，请参阅《IAM 用户指南 指南》中的[联合身份用户和角色](#)。
 - **跨账户访问** – 可以使用您账户中的 IAM 角色向另一个 AWS 账户授予对您账户的资源的访问权限。有关示例，请参阅《IAM 用户指南 指南》中的[教程：使用 IAM 角色委派跨 AWS 账户的访问权限](#)。
 - **AWS 服务访问** – 您可以使用您账户中的 IAM 角色向 AWS 服务授予对您账户的资源进行访问的权限。例如，您可以创建一个角色，此角色允许 Amazon Redshift 代表您访问 Amazon S3 存储桶，然后将

存储在该存储桶中的数据加载到 Amazon Redshift 群集中。有关更多信息，请参阅《IAM 用户指南 指南》中的[创建向 AWS 服务委派权限的角色](#)。

- 在 Amazon EC2 上运行的应用程序 – 您不用将访问密钥存储在 Amazon EC2 实例中以供实例上运行的应用程序使用并发出 AWS API 请求，而是可以使用 IAM 角色管理这些应用程序的临时凭证。要将 AWS 角色分配给 Amazon EC2 实例并使其对该实例的所有应用程序均可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 Amazon EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅《IAM 用户指南 指南》中的[对 Amazon EC2 上的应用程序使用角色](#)。

访问控制

您可以使用有效的凭证来对自己的请求进行身份验证，但您还必须拥有权限才能创建或访问 AWS CodeBuild 资源。例如，您必须拥有权限才能创建、查看或删除构建项目并启动、停止或查看构建项目。

下面几节介绍如何管理 AWS CodeBuild 的权限。我们建议您先阅读概述。

- [管理您的 AWS CodeBuild 资源的访问权限概述 \(p. 212\)](#)
- [为 AWS CodeBuild 使用基于身份的策略 \(IAM 策略\) \(p. 214\)](#)
- [AWS CodeBuild 权限参考 \(p. 220\)](#)

管理您的 AWS CodeBuild 资源的访问权限概述

每个 AWS 资源都归某个 AWS 账户所有，创建和访问资源的权限由权限策略进行管理。账户管理员可以向 IAM 身份 (即：用户、组和角色) 挂载权限策略。

Note

账户管理员 (或管理员用户) 是具有管理员权限的用户。有关更多信息，请参阅《IAM 用户指南 指南》中的[IAM 最佳实践](#)。

在您授予权限时，您要决定谁将获得权限、这些人可以访问的资源以及可以对这些资源执行的操作。

主题

- [AWS CodeBuild 资源和操作 \(p. 212\)](#)
- [了解资源所有权 \(p. 213\)](#)
- [管理对资源的访问 \(p. 213\)](#)
- [指定策略元素：操作、效果和委托人 \(p. 214\)](#)

AWS CodeBuild 资源和操作

在 AWS CodeBuild 中，构建项目是主要资源。在策略中，您可以使用 Amazon 资源名称 (ARN) 标识策略应用到的资源。构建项目也是资源，且具有相关联的 ARN。有关更多信息，请参阅 Amazon Web Services 一般参考 中的[Amazon 资源名称 \(ARN\)](#) 和 [AWS 服务命名空间](#)。

资源类型	ARN 格式
构建项目	arn:aws:codebuild: <i>region-ID</i> : <i>account-ID</i> :project/ <i>project-name</i>
生成	arn:aws:codebuild: <i>region-ID</i> : <i>account-ID</i> :build/ <i>build-ID</i>
所有 AWS CodeBuild 资源	arn:aws:codebuild:*

资源类型	ARN 格式
指定账户在指定地区拥有的所有 AWS CodeBuild 资源	<code>arn:aws:codebuild:<i>region-ID</i>:<i>account-ID</i>:*</code>

Note

大多数 AWS 服务将 ARN 中的冒号 (:) 或正斜杠 (/) 视为同一个字符。不过，AWS CodeBuild 在资源模式和规则中使用精确匹配。请务必在创建事件模式时使用正确的字符，以使其与资源中的 ARN 语法匹配。

例如，您可以在语句中使用特定构建项目 (*myBuildProject*) 的 ARN 来指示它，如下所示：

```
"Resource": "arn:aws:codebuild:us-east-2:123456789012:project/myBuildProject"
```

要指定所有资源，或者如果 API 操作不支持 ARN，请在 Resource 元素中使用通配符 (*)，如下所示：

```
"Resource": "*"
```

有些 AWS CodeBuild API 操作接受多个资源 (例如，BatchGetProjects)。要在单个语句中指定多种资源，请使用逗号将它们隔开，如下所示：

```
"Resource": [  
  "arn:aws:codebuild:us-east-2:123456789012:project/myBuildProject",  
  "arn:aws:codebuild:us-east-2:123456789012:project/myOtherBuildProject"  
]
```

AWS CodeBuild 提供一组操作来处理 AWS CodeBuild 资源。有关列表，请参阅[AWS CodeBuild 权限参考](#) (p. 220)。

了解资源所有权

AWS 账户对在该账户下创建的资源具有所有权，而无论创建资源的人员是谁。具体而言，资源所有者是对资源创建请求进行身份验证的[委托人实体](#)（即根账户、IAM 用户或 IAM 角色）的 AWS 账户。以下示例说明了它的工作原理：

- 如果您使用 AWS 账户的根账户凭证创建规则，则您的 AWS 账户即为该 AWS CodeBuild 资源的所有者。
- 如果您在您的 AWS 账户中创建 IAM 用户并授予该用户创建 AWS CodeBuild 资源的权限，则该用户可以创建 AWS CodeBuild 资源。但是，该用户所属的 AWS 账户拥有这些 AWS CodeBuild 资源。
- 如果您在 AWS 账户中创建一个具有 AWS CodeBuild 资源创建权限的 IAM 角色，则任何能够担任该角色的人都可以创建 AWS CodeBuild 资源。该角色所属的 AWS 账户拥有这些 AWS CodeBuild 资源。

管理对资源的访问

权限策略规定谁可以访问哪些资源。

Note

本节讨论如何在 AWS CodeBuild 中使用 IAM。它不提供有关 IAM 服务的详细信息。有关完整的 IAM 文档，请参阅[什么是 IAM？](#)（位于《IAM 用户指南 指南》中）。有关 IAM 策略语法和说明的信息，请参阅《IAM 用户指南 指南》中的[AWS IAM 策略参考](#)。

附加到 IAM 身份的策略称作基于身份的策略 (IAM 策略)。附加到资源的策略称作基于资源的策略。AWS CodeBuild 只支持基于身份的策略 (IAM 策略)。

基于身份的策略 (IAM 策略)

您可以向 IAM 身份挂载策略。

- 将权限策略挂载到您的账户中的用户或组 – 要授予用户在 AWS CodeBuild 控制台中查看构建项目和其他 AWS CodeBuild 资源的权限，您可以将权限策略挂载到用户或用户所属的组。
- 向角色挂载权限策略（授予跨账户权限） - 您可以向 IAM 角色挂载基于身份的权限策略，以授予跨账户的权限。例如，账户 A 中的管理员可以创建一个角色，以向其他 AWS 账户（如账户 B）或某项 AWS 服务授予跨账户权限，如下所述：
 1. 账户 A 管理员创建一个 IAM 角色，向该角色挂载授权其访问账户 A 中资源的权限策略。
 2. 账户 A 管理员可以向将账户 B 标识为能够担任该角色的委托人的角色挂载信任策略。
 3. 之后，账户 B 管理员可以委派权限，以指派账户 B 中的任何用户担任该角色。这样，账户 B 中的用户便可以创建或访问账户 A 中的资源。如果您需要授予 AWS 服务相应的权限以担任该角色，则信任策略中的委托人还必须是 AWS 服务委托人。

有关使用 IAM 委派权限的更多信息，请参阅《IAM 用户指南 指南》中的[访问权限管理](#)。

在 AWS CodeBuild 中，基于身份的策略用于管理对与部署过程相关的资源的权限。例如，您可以控制对构建项目的访问。

您可以创建 IAM 策略来限制您账户中的用户有权访问的调用和资源，然后将这些策略挂载到 IAM 用户。有关创建 IAM 角色和探索 AWS CodeBuild 的示例 IAM 策略语句的更多信息，请参阅[管理您的 AWS CodeBuild 资源的访问权限概述 \(p. 212\)](#)。

指定策略元素：操作、效果和委托人

对于每种 AWS CodeBuild 资源，该服务都定义了一组 API 操作。为授予这些 API 操作的权限，AWS CodeBuild 定义了一组您可以在策略中指定的操作。某些 API 操作可能需要多个操作的权限才能执行 API 操作。有关更多信息，请参阅[AWS CodeBuild 资源和操作 \(p. 212\)](#)和[AWS CodeBuild 权限参考 \(p. 220\)](#)。

以下是基本的策略元素：

- Resource - 您使用 Amazon 资源名称 (ARN) 来标识策略应用到的资源。
- Action - 您可以使用操作关键字来标识要允许或拒绝的资源操作。例如，codebuild:CreateProject 权限授予用户执行 CreateProject 操作的权限。
- Effect - 用于指定当用户请求操作时的效果（可以是允许或拒绝）。如果没有显式授予（允许）对资源的访问权限，则隐式拒绝访问。您也可显式拒绝访问。您可以执行此操作，以确保用户无法访问资源，即使有其他策略授予了访问权限也是如此。
- Principal - 在基于身份的策略 (IAM 策略) 中，挂载了策略的用户是隐式委托人。对于基于资源的策略，您可以指定您希望将权限授予的用户、账户、服务或其他实体。

有关 IAM 策略语法和说明的更多信息，请参阅《IAM 用户指南 指南》中的[AWS IAM 策略参考](#)。

有关显示所有 AWS CodeBuild API 操作及其适用资源的表，请参阅[AWS CodeBuild 权限参考 \(p. 220\)](#)。

为 AWS CodeBuild 使用基于身份的策略 (IAM 策略)

本主题提供了基于身份的策略的示例，这些示例展示了账户管理员如何将权限策略挂载到 IAM 身份（即用户、组和角色），从而授予对 AWS CodeBuild 资源执行操作的权限。

Important

我们建议您首先阅读以下介绍性主题，这些主题介绍了可用于管理 AWS CodeBuild 资源访问的基本概念和选项。有关更多信息，请参阅[管理您的 AWS CodeBuild 资源的访问权限概述 \(p. 212\)](#)。

主题

- [使用 AWS CodeBuild 控制台所需要的权限 \(p. 215\)](#)
- [AWS CodeBuild 控制台连接到使用 OAuth 的源提供程序所需的权限 \(p. 215\)](#)
- [适用于 AWS CodeBuild 的 AWS 托管 \(预定义\) 策略 \(p. 216\)](#)
- [客户托管的策略示例 \(p. 216\)](#)

以下是一个权限策略示例，仅允许用户在 123456789012 账户的 us-east-2 区域中获取任何以 my 名称开头的构建项目的相关信息：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchGetProjects",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

使用 AWS CodeBuild 控制台所需要的权限

使用 AWS CodeBuild 控制台的用户必须拥有一组最低权限，这些权限允许用户描述 AWS 账户的其他 AWS 资源。您必须拥有来自以下服务的权限：

- AWS CodeBuild
- Amazon CloudWatch
- AWS CodeCommit (如果您要将源代码存储在 AWS CodeCommit 存储库中)
- Amazon Elastic Container Registry (Amazon ECR) (如果您要使用依赖于 Amazon ECR 存储库中 Docker 镜像的构建环境)
- Amazon Elastic Container Service (Amazon ECS) (如果您要使用依赖于 Amazon ECR 存储库中 Docker 镜像的构建环境)
- AWS Identity and Access Management (IAM)
- AWS Key Management Service (AWS KMS)
- Amazon Simple Storage Service (Amazon S3)

如果您创建比必需的最低权限更为严格的 IAM 策略，控制台将无法按预期正常运行。

AWS CodeBuild 控制台连接到使用 OAuth 的源提供程序所需的权限

AWS CodeBuild 控制台使用以下 API 操作连接到使用 OAuth 的源提供程序 (例如，GitHub 存储库)。

- `codebuild:ListConnectedOAuthAccounts`
- `codebuild:ListRepositories`
- `codebuild:PersistOAuthToken`

要将使用 OAuth 的源提供程序 (如 GitHub 存储库) 与您的构建项目关联，您必须使用 AWS CodeBuild 控制台。为此，您必须先将上述 API 操作添加到与用于访问 AWS CodeBuild 控制台的 IAM 用户关联的 IAM 访问策略。

这些 API 操作不应由您的代码调用。因此，这些 API 操作未包含在 AWS CLI 和 AWS 开发工具包中。

适用于 AWS CodeBuild 的 AWS 托管 (预定义) 策略

AWS 通过提供由 AWS 创建和管理的独立 IAM 策略满足许多常用案例的要求。这些 AWS 托管策略可授予常用案例的必要权限，因此，您可以免去调查都需要哪些权限的工作。有关更多信息，请参阅 IAM 用户指南 中的 [AWS 托管策略](#)。

以下 AWS 托管策略 (您可以将其挂载到您账户中的用户) 是特定于 AWS CodeBuild 的。

- [AWSCodeBuildAdminAccess](#) – 提供对 AWS CodeBuild 的完全访问权限 (包括管理 AWS CodeBuild 构建项目的权限)。
- [AWSCodeBuildDeveloperAccess](#) – 提供对 AWS CodeBuild 的访问权限，但不允许管理构建项目。
- [AWSCodeBuildReadOnlyAccess](#) – 提供对 AWS CodeBuild 的只读访问权限。

要访问 AWS CodeBuild 创建的构建输出项目，您还必须附加名为 [AmazonS3ReadOnlyAccess](#) 的 AWS 托管策略。

要创建和管理 AWS CodeBuild 服务角色，您还必须附加名为 [IAMFullAccess](#) 的 AWS 托管策略。

此外，您还可以创建自己的自定义 IAM 策略，以授予 AWS CodeBuild 操作和资源的相关权限。您可以将这些自定义策略挂载到需要这些权限的 IAM 用户或组。

客户托管的策略示例

本节的用户策略示例介绍如何授予执行 AWS CodeBuild 操作的权限。当您使用 AWS CodeBuild API、AWS 软件开发工具包或 AWS CLI 时，可以使用这些策略。当您使用控制台时，您必须授予特定于控制台的其他权限。有关信息，请参阅 [使用 AWS CodeBuild 控制台所需要的权限 \(p. 215\)](#)。

您可以使用以下示例 IAM 策略来限制 IAM 用户和角色对 AWS CodeBuild 的访问。

主题

- [允许用户获取有关构建项目的信息 \(p. 216\)](#)
- [允许用户创建构建项目 \(p. 217\)](#)
- [允许用户删除构建项目 \(p. 217\)](#)
- [允许用户获取构建项目名称的列表 \(p. 217\)](#)
- [允许用户更改有关构建项目的信息 \(p. 218\)](#)
- [允许用户获取有关构建项目的信息 \(p. 218\)](#)
- [允许用户获取构建项目的构建 ID 列表 \(p. 218\)](#)
- [允许用户获取构建 ID 的列表 \(p. 219\)](#)
- [允许用户开始运行构建项目 \(p. 219\)](#)
- [允许用户尝试停止构建项目 \(p. 219\)](#)
- [允许用户尝试删除构建 \(p. 219\)](#)
- [允许用户获取有关由 AWS CodeBuild 管理的 Docker 镜像的信息 \(p. 220\)](#)
- [允许 AWS CodeBuild 访问创建 VPC 网络接口时所需的 AWS 服务 \(p. 220\)](#)

允许用户获取有关构建项目的信息

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中获取任何以名称 my 开头的构建项目的信息：

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Action": "codebuild:BatchGetProjects",
    "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
  }
]
```

允许用户创建构建项目

以下示例策略语句允许用户创建使用任何名称的构建项目，但只能在 123456789012 账户的 us-east-2 区域中创建，并且只能使用指定的 AWS CodeBuild 服务角色：

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

允许用户删除构建项目

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中删除任何以名称 my 开头的构建项目：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:DeleteProject",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

允许用户获取构建项目名称的列表

以下示例策略语句允许用户获取同一账户的构建项目名称的列表：

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

```
}
```

允许用户更改有关构建项目的信息

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中更改有关使用任何名称的构建项目的信息，并且只能使用指定的 AWS CodeBuild 服务角色：

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

允许用户获取有关构建项目的信息

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中获取名为 my-build-project 和 my-other-build-project 的构建项目的信息：

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

允许用户获取构建项目的构建 ID 列表

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中获取名为 my-build-project 和 my-other-build-project 的构建项目的构建 ID：

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


允许用户获取构建 ID 的列表

以下示例策略语句允许用户获取同一账户的所有构建 ID 的列表：

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

允许用户开始运行构建项目

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中运行任何以名称 my 开头的构建项目：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:StartBuild",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

允许用户尝试停止构建项目

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中尝试停止任何以名称 my 开头的运行中构建项目：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:StopBuild",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```

允许用户尝试删除构建

以下示例策略语句仅允许用户在 123456789012 账户的 us-east-2 区域中尝试删除任何以名称 my 开头的构建项目：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "codebuild:BatchDeleteBuilds",
      "Resource": "arn:aws:codebuild:us-east-2:123456789012:project/my*"
    }
  ]
}
```



```
]
}
```

允许用户获取有关由 AWS CodeBuild 管理的 Docker 镜像的信息

以下示例策略语句允许用户获取有关由 AWS CodeBuild 管理的所有 Docker 镜像的信息：

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

允许 AWS CodeBuild 访问创建 VPC 网络接口时所需的 AWS 服务

以下示例策略语句授予 AWS CodeBuild 权限以在 Amazon VPC 中创建网络接口：

```
{
  "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 权限参考

在设置 [访问控制](#) (p. 212) 和编写您可挂载到 IAM 身份的权限策略 (基于身份的策略) 时，可以使用下表作为参考。

您可以在 AWS CodeBuild 策略中使用 AWS 级条件键来表示条件。有关列表，请参阅 IAM 用户指南 中的[可用键](#)。

请在策略的 Action 字段中指定这些操作。要指定操作，请在 API 操作名称之前使用 codebuild: 前缀 (例如，codebuild:CreateProject 和 codebuild:StartBuild)。要在单个语句中指定多项操作，请使用逗号将它们隔开 (例如，"Action": ["codebuild:CreateProject", "codebuild:StartBuild"])。

使用通配符

您可以在策略的 Resource 字段中指定带或不带通配符 (*) 的 ARN 作为资源值。您可以使用通配符指定多个操作或资源。例如，codebuild:* 指定所有 AWS CodeBuild 操作，codebuild:Batch* 指定以单词 Batch 开头的所有 AWS CodeBuild 操作。以下示例授予对以 my 名称开头的所有构建项目的访问权限：

```
arn:aws:codebuild:us-east-2:123456789012:project/my*
```

AWS CodeBuild API 操作和必需的操作权限

BatchDeleteBuilds

操作 : codebuild:BatchDeleteBuilds

删除构建所必需的。

资源 : arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

BatchGetBuilds

操作 : codebuild:BatchGetBuilds

获取有关构建项目的信息所必需的。

资源 : arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

BatchGetProjects

操作 : codebuild:BatchGetProjects

获取有关构建项目的信息所必需的。

资源 : arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

CreateProject

操作: codebuild:CreateProject, iam:PassRole

是创建构建项目所必需的。

Resources: arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*,
arn:aws:iam:*account-ID*:role/*role-name*

DeleteProject

操作 : codebuild>DeleteProject

是删除构建项目所必需的。

资源 : arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

ListBuilds

操作 : codebuild>ListBuilds

是获取构建 ID 的列表所必需的。

资源：*

ListBuildsForProject

操作：codebuild:ListBuildsForProject

是获取构建项目的构建 ID 列表所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

ListCuratedEnvironmentImages

操作：codebuild:ListCuratedEnvironmentImages

是获取由 AWS CodeBuild 管理的所有 Docker 镜像的相关信息所必需的。

资源：* (必需，但不引用可寻址的 AWS 资源)

ListProjects

操作：codebuild:ListProjects

是获取构建项目名称的列表所必需的。

资源：*

StartBuild

操作：codebuild:StartBuild

是开始运行构建项目所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

StopBuild

操作：codebuild:StopBuild

是尝试停止运行中构建项目所必需的。

资源：arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*

UpdateProject

操作: codebuild:UpdateProject, iam:PassRole

是更改构建项目的相关信息所必需的。

Resources: arn:aws:codebuild:*region-ID*:*account-ID*:project/*project-name*,
arn:aws:iam:*account-ID*:role/*role-name*

使用 AWS CloudTrail 记录 AWS CodeBuild API 调用

AWS CodeBuild 与 CloudTrail 集成在一起，后者是一种服务，它在 AWS 账户中捕获由 AWS CodeBuild 或代表它发出的 API 调用，并将日志文件传输到您指定的 Amazon S3 存储桶。CloudTrail 捕获来自 AWS CodeBuild 控制台、AWS CLI 和 AWS 开发工具包的 API 调用。通过 CloudTrail 收集的信息，您可以确定向 AWS CodeBuild 发出了什么请求、发出请求的源 IP 地址、何人发出的请求以及发出请求的时间等。要了解有关 CloudTrail 的更多信息，包括如何对其进行配置和启用，请参阅 [AWS CloudTrail User Guide](#)

CloudTrail 中的 AWS CodeBuild 信息

在您的 AWS 账户中启用 CloudTrail 日志记录后，将在日志文件中跟踪对 AWS CodeBuild 操作执行的调用。AWS CodeBuild 记录会与其他 AWS 服务记录一起写入日志文件。CloudTrail 基于时间段和文件大小来确定何时创建新文件并向其写入内容。

所有 AWS CodeBuild 操作都会记录并正式记载到 [命令行参考 \(p. 207\)](#) 中。例如，对创建构建项目和运行构建的调用会在 CloudTrail 日志文件中生成条目。

每个日志条目都包含有关生成请求的人员的信息。日志中的用户身份信息有助于确定发出的请求是否具有根或 IAM 用户证书，是否具有角色或联合用户的临时安全证书，或者是否是由其他 AWS 服务发出的。有关更多信息，请参阅 [userIdentityCloudTrail 事件参考](#) 中的 [字段](#)。

日志文件可以在存储桶中存储任意长时间，不过您也可以定义 Amazon S3 生命周期规则以自动存档或删除日志文件。默认情况下，将使用 Amazon S3 服务器端加密 (SSE) 对日志文件进行加密。

可选择让 CloudTrail 在传输新日志文件时发布 Amazon SNS 通知。有关更多信息，请参阅 [为 CloudTrail 配置 Amazon SNS 通知](#)。

您也可以将多个 AWS 区域和多个 AWS 账户中的 AWS CodeBuild 日志文件聚合到单个 Amazon S3 存储桶中。有关更多信息，请参阅 [接收来自多个区域的 CloudTrail 日志文件](#)。

了解 AWS CodeBuild 日志文件条目

CloudTrail 日志文件可包含一个或多个日志条目，每个条目由多个 JSON 格式的事件组成。一个日志条目表示来自任何源的一个请求，包括有关所请求的操作、所有参数以及操作的日期和时间等信息。日志条目不一定具有任何特定顺序。也即，它们不是公用调用的有序堆栈跟踪。

下面的示例显示了一个 CloudTrail 日志条目，该条目演示了如何在 AWS CodeBuild 中创建构建项目：

```
{
  "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"
      }
    },
    "sessionArn": "arn:aws:sts::account-ID:federated-user/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": {}
}
```

```
"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 故障排除

使用本主题中的信息来帮助您识别、诊断和解决问题。

主题

- [创建或更新构建项目时收到错误：“CodeBuild 无权执行：sts:AssumeRole”](#) (p. 225)
- [运行构建时收到错误：“必须使用指定的终端节点来寻址当前尝试访问的存储桶...”](#) (p. 226)
- [运行构建时收到错误：“无法上传项目：arn 无效”](#) (p. 226)
- [错误：“Unable to Locate Credentials”](#) (p. 226)
- [构建规范中的前期命令无法被后续命令识别](#) (p. 227)
- [来自错误存储库的 Apache Maven 构建参考项目](#) (p. 227)
- [默认情况下，构建命令以根用户身份运行](#) (p. 228)
- [Bourne 外壳 \(sh\) 必须存在于构建映像中](#) (p. 228)
- [运行构建时出现错误“AWS CodeBuild is experiencing an issue”](#) (p. 229)
- [使用自定义构建映像时出现错误“BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE”](#) (p. 229)
- [当文件名包含非美国英语字符时，构建可能失败](#) (p. 229)
- [当从 Amazon EC2 Parameter Store 获取参数时，构建可能失败](#) (p. 230)
- [尝试下载缓存时出现“Access denied”错误消息](#) (p. 230)
- [错误：“Unable to download cache: RequestError: send request failed caused by: x509: failed to load system roots and no roots provided”\(无法下载缓存: 请求错误: 发送请求失败原因: x509: 无法加载系统根, 没有提供任何根\)](#) (p. 231)
- [错误：“Unable to download certificate from S3. AccessDenied”](#) (p. 231)
- [错误：“Git Clone Failed: unable to access 'your-repository-URL': SSL certificate problem: self signed certificate”](#) (p. 232)
- [错误：“The policy's default version was not created by enhanced zero click role creation or was not the most recent version created by enhanced zero click role creation”\(策略的默认版本不是通过增强的零单击角色创建来创建的或不是通过增强的零单击角色创建来创建的最新版本\)。](#) (p. 232)

创建或更新构建项目时收到错误：“CodeBuild 无权执行：sts:AssumeRole”

问题：在尝试创建或更新构建项目时，您收到以下错误：“代码：InvalidInputException，消息：CodeBuild 无权在 `arn:aws:iam::account-ID:role/service-role-name` 上执行：sts:AssumeRole”。

可能的原因：

- AWS Security Token Service (AWS STS) 已在您尝试创建或更新构建项目的 AWS 区域停用。
- 与构建项目相关联的 AWS CodeBuild 服务角色不存在，或没有足够的权限来信任 AWS CodeBuild。

建议的解决方案：

- 确保 AWS STS 已经为您尝试创建或更新构建项目的 AWS 区域激活。有关更多信息，请参阅 IAM 用户指南中的[在 AWS 区域中激活和停用 AWS STS](#)。
- 确保您的 AWS 账户中存在目标 AWS CodeBuild 服务角色。如果您没有使用控制台，请确保在创建或更新构建项目时没有拼错服务角色的 Amazon 资源名称 (ARN)。

- 确保目标 AWS CodeBuild 服务角色具有足够的权限来信任 AWS CodeBuild。有关更多信息，请参阅 [创建 AWS CodeBuild 服务角色 \(p. 202\)](#) 中的信任关系策略声明。

运行构建时收到错误：“必须使用指定的终端节点来寻址当前尝试访问的存储桶...”

问题：在运行构建时，`DOWNLOAD_SOURCE` 构建阶段失败并显示以下错误：“必须使用指定的终端节点来寻址当前尝试访问的存储桶。请将未来的所有请求发送到此终端节点”。

可能的原因：您预先构建的源代码存储在 Amazon S3 存储桶中，而该存储桶与 AWS CodeBuild 构建项目位于不同的 AWS 区域。

建议的解决方案：更新构建项目的设置，以指向包含预构建源代码且与该构建项目位于同一区域的存储桶。

运行构建时收到错误：“无法上传项目：arn 无效”

问题：在运行构建时，`UPLOAD_ARTIFACTS` 构建阶段失败并显示以下错误：“无法上传项目：arn 无效”。

可能的原因：您的 Amazon S3 输出存储桶 (AWS CodeBuild 用于存储其构建输出的存储桶) 与 AWS CodeBuild 构建项目位于不同的 AWS 区域。

建议的解决方案：更新构建项目的设置，以指向与该构建项目位于同一区域的输出存储桶。

错误：“Unable to Locate Credentials”

问题：在尝试运行 AWS CLI、使用 AWS 软件开发工具包或调用其他类似组件作为构建的一部分时，您收到与 AWS CLI、AWS 软件开发工具包或组件直接相关的构建错误。例如，您可能会收到以下构建错误：“无法查找凭证”。

可能的原因：

- 构建环境中的 AWS CLI、AWS 开发工具包或组件的版本与 AWS CodeBuild 不兼容。
- 您将在使用 Docker 的构建环境内运行 Docker 容器，并且此 Docker 容器在默认情况下无权访问必需的 AWS 凭证。

建议的解决方案：

- 确保您的构建环境具有以下版本或更高版本的 AWS CLI、AWS 开发工具包或组件。
 - AWS CLI：1.10.47
 - 适用于 C++ 的 AWS 软件开发工具包：0.2.19
 - 适用于 Go 的 AWS 软件开发工具包：1.2.5
 - 适用于 Java 的 AWS 软件开发工具包：1.11.16
 - 适用于 JavaScript 的 AWS 软件开发工具包：2.4.7
 - 适用于 PHP 的 AWS 软件开发工具包：3.18.28
 - 适用于 Python (Boto3) 的 AWS 软件开发工具包：1.4.0
 - 适用于 Ruby 的 AWS 软件开发工具包：2.3.22
 - Botocore：1.4.37
 - CoreCLR：3.2.6-beta

- Node.js : 2.4.7
- 如果您需要在某个构建环境中运行某个 Docker 容器，而该容器需要 AWS 凭证，则必须将该凭证从该构建环境传递到该容器。在您的构建规范中，包含与下面类似的 Docker run 命令，在此示例中，该命令将使用 `aws s3 ls` 命令列出可用的 Amazon S3 存储桶。-e 选项将传递您的容器访问 AWS 凭证所需的环境变量。

```
docker run -e AWS_DEFAULT_REGION -e AWS_CONTAINER_CREDENTIALS_RELATIVE_URI your-image-tag
aws s3 ls
```

- 如果您要构建 Docker 映像并且该构建需要 AWS 凭证 (例如，从 Amazon S3 下载文件)，则必须将凭证从构建环境传递到 Docker 构建过程，如下所示。
 1. 在您的源代码的用于 Docker 映像的 Dockerfile 中，指定以下 ARG 指令。

```
ARG AWS_DEFAULT_REGION
ARG AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

2. 在您的构建规范中，包含类似于下面的 Docker build 命令。--build-arg 选项将为您的 Docker 构建过程设置访问 AWS 凭证所需的环境变量。

```
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 .
```

构建规范中的前期命令无法被后续命令识别

问题：构建规范中一个或多个命令的结果无法被同一构建规范中的后续命令识别。例如，某个命令可能会设置本地环境变量，但稍后运行的命令可能无法获取该本地环境变量的值。

可能的原因：在构建规范版本 0.1 中，AWS CodeBuild 将在构建环境内默认 Shell 的单独实例中运行每个命令。这表示各个命令独立于其他所有命令而运行。默认情况下，您无法运行依赖于任何先前命令的状态的单个命令。

建议的解决方案：我们建议您使用构建规范版本 0.2，它能解决此问题。如果您出于某种原因必须使用构建规范版本 0.1，我们建议使用 Shell 命令链接运算符 (例如，Linux 中的 `&&`) 将多个命令合并为一个命令。或者，您也可以在源代码中包括一个含有多个命令的 Shell 脚本，然后从构建规范中的单个命令调用该 Shell 脚本。有关更多信息，请参阅 [构建环境中的 Shell 和命令 \(p. 120\)](#) 和 [构建环境中的环境变量 \(p. 120\)](#)。

来自错误存储库的 Apache Maven 构建参考项目

问题：在将 Maven 与 AWS CodeBuild 提供的 Java 构建环境结合使用时，Maven 会从安全的 Maven 中央存储库 (网址为 <https://repo1.maven.org/maven2>) 中提取构建和插件依赖项。即使您构建项目的 `pom.xml` 文件明确声明会改用其他位置，也会发生这种情况。

可能的原因：AWS CodeBuild 提供的 Java 构建环境包含一个名为 `settings.xml` 的文件，该文件预先安装在构建环境的 `/root/.m2` 目录中。该 `settings.xml` 文件包含以下声明，这些声明将指示 Maven 始终从安全的 Maven 中央存储库 (网址为 <https://repo1.maven.org/maven2>) 中提取构建和插件依赖项。

```
<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>
```

建议的解决方案：执行以下操作：

1. 向源代码中添加 settings.xml 文件。
2. 在此 settings.xml 文件中，使用上述 settings.xml 格式作为指导，声明您希望 Maven 从哪些存储库中提取构建和插件依赖项。
3. 在构建项目的 install 阶段，指示 AWS CodeBuild 将您的 settings.xml 文件复制到构建环境的 /root/.m2 目录。例如，考虑说明此行为的 buildspec.yml 文件中的以下代码段。

```
version 0.2

phases:
  install:
    commands:
      - cp ./settings.xml /root/.m2/settings.xml
```

默认情况下，构建命令以根用户身份运行

问题：AWS CodeBuild 以根用户身份运行您的构建命令。即使您的相关构建映像的 Dockerfile 将 USER 指令设置为另一位用户，也会发生这种情况。

原因：默认情况下，AWS CodeBuild 将以根用户身份运行所有构建命令。

建议的解决方案：无。

Bourne 外壳 (sh) 必须存在于构建映像中

问题：您使用的不是 AWS CodeBuild 提供的构建映像，并且您的构建失败并返回消息“build container found dead before completing the build”。

可能的原因：Bourne 外壳 (sh) 未包含在您的构建映像中。AWS CodeBuild 需要 sh 才能运行构建命令和脚本。

建议的解决方案：如果您的构建映像中不存在 sh，请确保您在启动使用映像的任何其他构建前包含它。(AWS CodeBuild 已将 sh 包含在其构建映像中。)

运行构建时出现错误“AWS CodeBuild is experiencing an issue”

问题：当您尝试运行构建项目时，您将在构建的 `PROVISIONING` 阶段收到以下错误消息：“AWS CodeBuild is experiencing an issue”。

可能的原因：您的构建使用的环境变量对 AWS CodeBuild 过大。当所有环境变量的长度 (所有名称和值加在一起) 超出最大组合长度 (约 5,500 个字符) 时，AWS CodeBuild 可能引发错误。

建议的解决方案：使用 Amazon EC2 Systems Manager Parameter Store 存储大型环境变量，然后从您的构建规范中检索它们。Amazon EC2 Systems Manager Parameter Store 可以存储组合长度为 4,096 个字符或更少的字符的独立环境变量 (名称和值加在一起)。要存储大型环境变量，请参阅 Amazon EC2 Systems Manager 用户指南 中的 [Systems Manager Parameter Store](#) 和 [Systems Manager Parameter Store 控制台演练](#)。要检索它们，请参阅 [构建规范语法 \(p. 107\)](#) 中的 `parameter-store` 映射。

使用自定义构建映像时出现错误“BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE”

问题：当您尝试运行使用自定义构建映像的构建时，构建将失败并返回错误 `BUILD_CONTAINER_UNABLE_TO_PULL_IMAGE`。

可能的原因：

- 构建映像的整体未压缩大小大于构建环境计算类型的可用磁盘空间。要检查构建映像的大小，请使用 Docker 运行 `docker images REPOSITORY:TAG` 命令。有关按计算类型分类的可用磁盘空间的列表，请参阅 [构建环境计算类型 \(p. 119\)](#)。
- AWS CodeBuild 无权从您的 Amazon Elastic Container Registry (Amazon ECR) 中拉取构建映像。

建议的解决方案：

- 对较大的计算类型使用更多的可用磁盘空间，或者减小自定义构建映像的大小。
- 更新 Amazon ECR 中的存储库中的权限，以便 AWS CodeBuild 可以将自定义构建映像拉取到构建环境中。有关更多信息，请参见 [Amazon ECR 示例 \(p. 38\)](#)。

当文件名包含非美国英语字符时，构建可能失败

问题：当您运行的构建使用的文件的名称包含非美国英语字符 (例如，中文字符)，构建将失败。

可能的原因：由 AWS CodeBuild 提供的构建环境将其默认区域设置设置为 `POSIX`。`POSIX` 本地化设置不太兼容 AWS CodeBuild 以及包含非美国英语字符和可能导致相关构建失败的文件名。

建议的解决方案：将以下命令添加到构建规范的 `pre_build` 部分。这些命令使构建环境使用美国英语 UTF-8 作为其本地化设置，该格式与 AWS CodeBuild 和包含非美国英语字符的文件名更兼容。

对于基于 Ubuntu 的构建环境：

```
pre_build:
  commands:
    - export LC_ALL="en_US.UTF-8"
    - locale-gen en_US en_US.UTF-8
```

```
- dpkg-reconfigure locales
```

对于基于 Amazon Linux 的构建环境：

```
pre_build:
  commands:
    - export LC_ALL="en_US.utf8"
```

当从 Amazon EC2 Parameter Store 获取参数时，构建可能失败

问题：当构建尝试获取存储在 Amazon EC2 Parameter Store 中的一个或多个参数的值时，处于 DOWNLOAD_SOURCE 阶段的构建将失败并返回以下错误：“Parameter does not exist”。

可能的原因：构建项目依赖的服务角色无权调用 `ssm:GetParameters` 操作，或构建项目使用 AWS CodeBuild 生成的服务角色并允许调用 `ssm:GetParameters` 操作，但参数的名称不以 `/CodeBuild/` 开头。

建议的解决方案：

- 如果服务角色不是由 AWS CodeBuild 生成的，请将其定义更新为允许 AWS CodeBuild 调用 `ssm:GetParameters` 操作。例如，以下策略语句允许调用 `ssm:GetParameters` 操作以获取名称以 `/CodeBuild/` 开头的参数：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "ssm:GetParameters",
      "Effect": "Allow",
      "Resource": "arn:aws:ssm:REGION_ID:ACCOUNT_ID:parameter/CodeBuild/*"
    }
  ]
}
```

- 如果服务角色是由 AWS CodeBuild 生成的，请将其定义更新为允许 AWS CodeBuild 访问 Amazon EC2 Parameter Store 中名称并非以 `/CodeBuild/` 开头的参数。例如，以下策略语句允许调用 `ssm:GetParameters` 操作以获取具有指定名称的参数：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "ssm:GetParameters",
      "Effect": "Allow",
      "Resource": "arn:aws:ssm:REGION_ID:ACCOUNT_ID:parameter/PARAMETER_NAME"
    }
  ]
}
```

尝试下载缓存时出现“Access denied”错误消息

问题：当尝试下载已启用缓存的构建项目上的缓存时，您收到以下一般性错误消息：“Access denied”。

错误：“Unable to download cache: RequestError: send request failed caused by: x509: failed to load system roots and no roots provided”(无法下载缓存: 请求错误: 发送请求失败原因: x509: 无法加载系统根, 没有提供任何根)

可能的原因：

- 您刚刚已将缓存配置为您的构建项目的一部分。
- 最近已通过 `InvalidateProjectCache` API 使缓存失效。
- 正由 CodeBuild 使用的服务角色对包含缓存的 Amazon S3 存储桶没有 `s3:GetObject` 和 `s3:PutObject` 权限。

建议的解决方案：在首次使用时，在更新缓存配置后立即看到此错误是正常的。如果此错误持续存在，则您应该检查您的服务角色对包含缓存的 Amazon S3 存储桶是否具有 `s3:GetObject` 和 `s3:PutObject` 权限。有关更多信息，请参阅[指定 S3 权限](#)。

错误：“Unable to download cache: RequestError: send request failed caused by: x509: failed to load system roots and no roots provided”(无法下载缓存: 请求错误: 发送请求失败原因: x509: 无法加载系统根, 没有提供任何根)

问题：当您尝试运行构建项目时，构建失败，错误消息为：“RequestError: send request failed caused by: x509: failed to load system roots and no roots provided”(请求错误: 发送请求失败原因: x509: 无法加载系统根, 没有提供任何根)。

可能的原因：

- 您将缓存配置为您的构建项目的一部分并使用包含过期根证书的较旧 Docker 镜像。

建议的解决方案：

- 更新您的 AWS CodeBuild 项目中使用的 Docker 镜像。有关更多信息，请参阅 [AWS CodeBuild 提供的 Docker 映像 \(p. 114\)](#)。

错误：“Unable to download certificate from S3. AccessDenied”

问题：当您尝试运行构建项目时，构建失败，错误消息为“Unable to download certificate from S3. AccessDenied”。

可能的原因：

- 您选择了错误的证书 S3 存储桶。
- 您输入了错误的证书对象键。

建议的解决方案：

- 编辑您的项目。对于 Bucket of certificate，选择存储您的 SSL 证书的 S3 存储桶。
- 编辑您的项目。对于 Object key of certificate，键入您的 S3 对象键的名称。

错误：“Git Clone Failed: unable to access 'your-repository-URL': SSL certificate problem: self signed certificate”

问题：当您尝试运行构建项目时，构建失败，错误消息为“Git Clone Failed: unable to access 'your-repository-URL': SSL certificate problem: self signed certificate”(Git 克隆失败：无法访问 'your-repository-URL': SSL 证书问题：自签名证书)。

可能的原因：

- 您的源存储库具有一个自签名证书，但您在构建项目的过程中未选择从您的 S3 存储桶安装此证书。

建议的解决方案：

- 编辑您的项目。对于 Certificate，选择 Install certificate from S3。对于 Bucket of certificate，选择存储您的 SSL 证书的 S3 存储桶。对于 Object key of certificate，键入您的 S3 对象键的名称。
- 编辑您的项目。选择 Insecure SSL，在连接到您的 GitHub Enterprise 项目存储库时忽略 SSL 警告。

Note

建议您仅将 Insecure SSL 用于测试。它不应在生产环境中使用。

错误：“The policy's default version was not created by enhanced zero click role creation or was not the most recent version created by enhanced zero click role creation”(策略的默认版本不是通过增强的零单击角色创建来创建的或不是通过增强的零单击角色创建来创建的最新版本)。

问题：当您尝试在控制台中更新项目时，更新失败，错误消息为：“The policy's default version was not created by enhanced zero click role creation or was not the most recent version created by enhanced zero click role creation”(策略的默认版本不是通过增强的零单击角色创建来创建的或不是通过增强的零单击角色创建来创建的最新版本)。

可能的原因：

- 您已手动更新附加到目标 AWS CodeBuild 服务角色的策略。
- 您已选择附加到目标 AWS CodeBuild 服务角色的策略的以前版本。

建议的解决方案：

- 编辑您的 AWS CodeBuild 项目，并取消选择 Allow AWS CodeBuild to modify this service role so it can be used with this build project (允许 AWS CodeBuild 修改此服务角色以便它可用于此构建项目)。手动更新目标 AWS CodeBuild 服务角色以具有足够权限。有关更多信息，请参阅 [创建 AWS CodeBuild 服务角色 \(p. 202\)](#)。

错误：“The policy's default version was not created by enhanced zero click role creation or was not the most recent version created by enhanced zero click role creation”(策略的默认版本不是通过增强的零单击角色创建来创建的或不是通过增强的零单击角色创建来创建的最新版本)。

- 编辑您的 AWS CodeBuild 项目不是通过增强的零单击角色创建来创建的。

AWS CodeBuild 的限制

下表列出了 AWS CodeBuild 中的当前限制。这些限制适用于每个受支持的 AWS 区域中的每个 AWS 账户，除非另有规定。

构建项目

资源	默认限制
最大生成项目数	1000
构建项目名称的长度	2 到 255 个字符
构建项目名称中允许使用的字符	字母 A-Z 和 a-z、数字 0-9，以及特殊字符 - 和 _
构建项目描述的最大长度	255 个字符
构建项目描述中允许使用的字符	任何
可随时通过使用 AWS CLI 或 AWS 开发工具包请求其相关信息的构建项目的最大数目	100
可以与构建项目相关联的最大标签数	50
可以在构建项目中为所有相关构建的构建超时指定的分钟数	5 到 480 (8 个小时)
可以在 VPC 配置下添加的子网的数量	1 到 16
可以在 VPC 配置下添加的安全组的数量	1 到 5

构建

资源	默认限制
最大并行运行版本数 *	20
可随时通过使用 AWS CLI 和 AWS 开发工具包请求其相关信息的构建的最大数目	100
可以为单个构建的构建超时指定的分钟数	5 到 480 (8 个小时)

* 最大并发运行构建数的限制因计算类型的不同而有所不同。对于某些计算类型，默认值为 20。对于新账户，限制可以是 1 到 5 之间。如需请求更高的并发构建限制，或者如果您收到“账户不能有多于 x 个出处于活动状态的构建”错误，请联系 AWS 支持部门。

适用于 Windows 的 AWS CodeBuild – 第三方说明

在您使用适用于 Windows 的 AWS CodeBuild 生成时，可以选择使用一些第三方软件包/模块，使您可以生成运行在 Microsoft Windows 操作系统上并与一些第三方产品互操作的应用程序。以下列表包含在您使用指定的第三程序包/模块时，所需遵循的适用第三方法律条款。

主题

- 1) 基本 Docker 映像 – windowsservercore (p. 235)
- 2) 基于 Windows 的 Docker 映像 – Choco (p. 236)
- 3) 基于 Windows 的 Docker 映像 – git -- 版本 2.16.2 (p. 236)
- 4) 基于 Windows 的 Docker 映像 – microsoft-build-tools -- 版本 15.0.26320.2 (p. 236)
- 5) 基于 Windows 的 Docker 映像 – nuget.commandline -- 版本 4.5.1 (p. 236)
- 7) 基于 Windows 的 Docker 映像 – netfx-4.6.2-devpack (p. 236)
- 8) 基于 Windows 的 Docker 映像 – visualfsharp tools , v 4.0 (p. 236)
- 9) 基于 Windows 的 Docker 映像 – netfx-pcl-reference-assemblies-4.6 (p. 240)
- 10) 基于 Windows 的 Docker 映像 – visualcppbuildtools , v 14.0.25420.1 (p. 241)
- 11) 基于 Windows 的 Docker 映像 – microsoft-windows-netfx3-ondemand-package.cab (p. 243)
- 12) 基于 Windows 的 Docker 映像 – dotnet-sdk (p. 244)

1) 基本 Docker 映像 – windowsservercore

(许可条款位于 : <https://hub.docker.com/r/microsoft/windowsservercore/>)

许可：请求并使用此适用于 Windows 容器的容器操作系统映像即表明您承认、了解并同意以下补充许可条款：

MICROSOFT 软件补充许可条款

容器操作系统映像

Microsoft Corporation (或者您所在地的其附属公司) (简称为“我们”或“Microsoft”) 将此容器操作系统映像补充 (下称“补充”) 的许可授予您。根据授予的许可，您可以将此补充与基本主机操作系统软件 (下称“主机软件”) 一起使用，其目的仅用于帮助您在主机软件中运行容器功能。主机软件许可条款适用于您对补充的使用。如果您未获得主机软件的许可，就不能使用它。您可以将此补充用于主机软件的每个授予有效许可的副本。

其他许可要求和/或使用权利

您按照前述段落中的规定使用补充许可可能会导致创建或修改容器映像 (下称“容器映像”) ，而其中包含特定补充组件。为明确起见，容器映像是独立的，不同于虚拟机映像或虚拟设备映像。根据这些许可条款，在遵循下列条件的情况下，我们授予您重新分发此类补充组件的有限许可：

(i) 您仅可以在自己的容器映像中，将补充组件作为映像的一部分使用，

(ii) 只要您的容器映像中的重要主功能与补充在实质上是分离且不同的，您就可以在容器映像中使用此类补充；以及

(iii) 您同意在您的容器映像中包括这些许可条款（或者我们或托管商要求的类似条款），用于对您的最终用户可能使用补充组件正确授予许可。

我们保留未在此处明确授予的所有其他权限。

使用本补充内容即表示您接受这些条款。如果您不接受这些条款，请勿使用本补充内容。

作为适用于 Windows 容器的此容器操作系统映像补充许可条款的一部分，您还需要遵守基础 Windows Server 主机软件许可条款，该条款位于：<https://www.microsoft.com/en-us/useterms>。

2) 基于 Windows 的 Docker 映像 – Choco

（许可条款位于：<https://github.com/chocolatey/chocolatey.org/blob/master/LICENSE.txt>）

版权所有 – Present RealDimensions Software, LLC

根据 Apache 许可版本 2.0 授予许可（简称“许可”），如果不遵守许可，您不可使用这些文件。您可以在以下地址获取许可的副本

<http://www.apache.org/licenses/LICENSE-2.0>

除非适用的法律要求或以书面方式表示同意，否则，根据该许可分发的软件按“原样”分发，无任何明示或暗示的保证或条件。请参阅许可证以了解在许可证下特定语言的适用权限和限制。

3) 基于 Windows 的 Docker 映像 – git -- 版本 2.16.2

（许可条款位于：<https://chocolatey.org/packages/git/2.16.2>）

根据 GNU General Public License 版本 2 授予许可，该许可位于：<https://www.gnu.org/licenses/old-licenses/gpl-2.0.html>

4) 基于 Windows 的 Docker 映像 – microsoft-build-tools -- 版本 15.0.26320.2

（许可条款位于：<https://www.visualstudio.com/license-terms/mt171552/>）

MICROSOFT VISUAL STUDIO 2015 扩展、VISUAL STUDIO SHELL 和 C++ 可重新分发

这些许可条款是 Microsoft Corporation（或您所在地的其附属公司）与您达成的协议。它们适用于以上所述软件。这些条款还适用于软件的任意 Microsoft 服务或更新，除非另有附加条款。

如果您遵循这些许可条款，您将拥有以下权利。

1. 安装和使用权利。您可以安装和使用该软件任意数量的副本。
2. 特定组件的条款。
 - a. 实用程序。软件可能包含位于 <http://go.microsoft.com/fwlink/?LinkId=523763&clcid=0x409> 的实用程序列表上的一些项目。如果软件中包含，您可以复制并安装这些项目到您或其他第三方的计算机上，用于

调试和部署您使用软件开发的应用程序及数据库。请注意，实用程序设计用于临时使用，Microsoft 可能无法独立于软件的其余部分为实用程序打补丁或进行更新，并且一些实用程序在性质上会使得其他人可以访问安装该实用程序的计算机。因此，在您完成调试或部署应用程序及数据库之后，您应删除已安装的所有实用程序。对于任何第三方使用或访问您安装在任意计算机上的实用程序，Microsoft 不承担任何责任。

- b. Microsoft 平台。软件可能包括来自 Microsoft Windows、Microsoft Windows Server、Microsoft SQL Server、Microsoft Exchange、Microsoft Office 和 Microsoft SharePoint 的组件。这些组件受独立协议及自己的产品支持政策控制，如位于组件安装目录中或随软件提供的“Licenses”文件夹中的许可条款所述。
- c. 第三方组件。软件可能会包括第三方组件，这些组件具有单独的法律声明或受其他协议控制，如随软件提供的 ThirdPartyNotices 文件中所述。即使此类组件受其他协议控制，以下免责声明以及对损害赔偿的限制或排除仍适用。软件还包含在开源许可下授予许可的组件，具有源代码可用性义务。这些许可证的副本（如果适用）包含在 ThirdPartyNotices 文件中。您可以根据相关开源许可的要求，将 5.00 美元的汇票或支票发送到以下地址，从我们这里获取源代码：Source Code Compliance Team, Microsoft Corporation, 1 Microsoft Way, Redmond, WA 98052。在您付款的备注栏中，请注明以下列出的一个或多个组件的源代码：
 - Remote Tools for Visual Studio 2015；
 - Standalone Profiler for Visual Studio 2015；
 - IntelliTraceCollector for Visual Studio 2015；
 - Microsoft VC++ Redistributable 2015；
 - Multibyte MFC Library for Visual Studio 2015；
 - Microsoft Build Tools 2015；
 - Feedback Client；
 - Visual Studio 2015 Integrated Shell；或
 - Visual Studio 2015 Isolated Shell。

我们还在 <http://thirdpartysource.microsoft.com> 提供了源代码的副本。

3. 数据。软件可能会收集有关您以及您使用软件的信息，并将信息发送给 Microsoft。Microsoft 可能会使用此信息来提供服务和改进我们的产品及服务。您可以选择退出其中的多种情况，但并非全部，如产品文档中所述。此外，软件中还有一些功能，可能使您能够从应用程序的用户收集数据。如果您使用这些功能在应用程序中启用数据收集，则必须遵循适用的法律，包括向应用程序的用户提供相应的说明。您可在以下地址的帮助文档和隐私声明中了解有关数据收集和使用的更多信息：<http://go.microsoft.com/fwlink/?LinkId=528096&clcid=0x409>。您使用软件操作即表明您同意这些做法。
4. 许可范围。该软件授予许可，而非销售。本协议仅向您提供使用软件的一些权利。Microsoft 保留所有其他权利。除非适用法律在此限制之外赋予您更多的权利，否则您只能在本协议中明确允许的情况下使用该软件。在使用时，您必须遵守软件中仅允许您以特定方式使用它的任意技术限制。您不能
 - 规避软件中的任何技术限制；
 - 逆向工程、反编译或反汇编软件，或者尝试这样做（除非且仅限于在第三方许可条款要求的范围内，该条款规定软件中可能包含的特定开源组件的使用）
 - 删除、最小化、阻止或修改软件中 Microsoft 或其供应商的通知；
 - 以违反法律的方式使用软件；或者
 - 共享、发布、租借或租用软件，或者将软件独立托管为解决方案供其他人使用。
5. 出口限制。您必须遵守所有适用于该软件的国内和国际出口法律及法规，这包括对目的地、最终用户和最终用途的限制。有关出口限制的更多信息，请访问 (aka.ms/exporting)。
6. 支持服务。由于此软件“按原样”提供，我们可能不会为它提供支持服务。
7. 完整协议。本协议以及您使用的补充、更新、基于 Internet 的服务和支持服务的条款是本软件和支持服务的完整协议。
8. 适用的法律。如果您在美国购买本软件，华盛顿州法律管辖对本协议的解释以及违反协议的索赔，您居住州的法律适用于所有其他索赔。如果您在任何其他国家/地区购买本软件，则适用该国家/地区的法律。
9. 消费者权利；区域差异。本协议描述了特定法律权利。根据所在的州或国家/地区，您可能拥有其他权利，包括消费者权利。除了与 Microsoft 的关系之外，您还可能对与您购买该软件的一方的拥有相应权利。如

果您所在的州或国家/地区的法律不允许，本协议不会更改这些其他权利。例如，如果您在以下区域之一购买了本软件，或者有强制性国家/地区法律适用，则以下条款适用于您：

- a. 澳大利亚. 您在澳大利亚消费者法下获得了法定担保，本协议中的任何内容都无意影响这些权利。
 - b. 加拿大. 如果您在加拿大购买此软件，您可通过关闭自动更新功能、断开设备与 Internet 的连接（但是，在您重新连接到 Internet 时，软件将恢复检查和安全更新）或者卸载软件来停止接收更新。产品文档（如果有）可能会说明如何关闭特定设备或软件的更新。
 - c. 德国和奥地利。
 - i. 担保. 正确授予许可的软件，将基本按照随该软件所提供的任意 Microsoft 材料中所述执行。但是，Microsoft 不提供任何与所许可软件相关的合同担保。
 - ii. 责任限制. 在出现故意行为、重大过失、基于产品责任法的索赔时，以及出现死亡、人身伤害或物理伤害时，Microsoft 根据成文法律承担责任。根据前述条款 (ii)，Microsoft 仅在违背了实质性合同义务时，在承担责任有助于正当履行此协议时，违反许可会危害到此协议的目的时，以及符合当事人值得长期信任的情况下（称为“基本义务”），Microsoft 才会承担轻微过失责任。在其他轻微过失的情况下，Microsoft 不承担轻微过失的责任。
10. 免责声明. 本软件按“原样”授予许可。您自行承担使用它的风险。MICROSOFT 不提供任何明示的担保、保证或条件。在您当地法律允许的范围内，Microsoft 排除有关适销性、针对特定目的的适用性和不侵权的默示担保。
11. 对损害赔偿的限制或排除. 您可以从 MICROSOFT 及其供应商获得的直接损害赔偿为 5.00 美元。您无法获得任何其他损害赔偿，包括后果性损害、利润损失、间接损害或事故损害。此限制适用于 (a) 第三方 Internet 站点或第三方应用程序上与软件、服务、内容（包括代码）相关的任意内容；(b) 违反合同；违反担保、保证或条件；严格责任；疏忽；或法律允许情况下其他侵权行为的索赔。

即使 Microsoft 已知或者应该知道造成损害的可能性，此条款仍适用。由于您所在的国家/地区可能不允许排除或限制事故损害、后果性损害或其他损害，上述限制或排除可能不适用于您。

EULA ID : VS2015_Update3_ShellsRedist_<ENU>

5) 基于 Windows 的 Docker 映像 – nuget.commandline -- 版本 4.5.1

（许可条款位于：<https://github.com/NuGet/Home/blob/dev/LICENSE.txt>）

版权所有 (c) .NET Foundation。保留所有权利。

根据 Apache 许可版本 2.0 授予许可（简称“许可”），如果不遵守许可，您不可使用这些文件。您可以在以下地址获取许可的副本

<http://www.apache.org/licenses/LICENSE-2.0>

除非适用的法律要求或以书面方式表示同意，否则，根据该许可分发的软件按“原样”分发，无任何明示或暗示的保证或条件。请参阅许可证以了解在许可证下特定语言的适用权限和限制。

7) 基于 Windows 的 Docker 映像 – netfx-4.6.2-devpack

MICROSOFT 软件补充许可条款

用于 MICROSOFT WINDOWS 操作系统的 .NET FRAMEWORK 和相关语言包

Microsoft Corporation (或您所在地的其附属公司) 向您授予此补充的许可。如果您获得使用 Microsoft Windows 操作系统软件 (下称“软件”) 的许可, 则可以使用此补充。如果您未获得软件的许可, 就不能使用它。您可以将此补充用于软件的每个授予有效许可的副本。

以下许可条款描述了此补充的额外使用条款。软件的这些条款和许可条款适用于您对补充的使用。如果存在冲突, 这些补充许可条款适用。

使用本补充内容即表示您接受这些条款。如果您不接受这些条款, 请勿使用本补充内容。

如果您遵循这些许可条款, 您将拥有以下权利。

1. 可分发代码。此补充由可分发代码组成。“可分发代码”是在您遵守以下条款的情况下, 允许您在自己开发的程序中分发的代码。
 - a. 使用和分发权利。
 - 您可以复制和分发本补充的对象代码形式。
 - 第三方分发。您可以允许程序分发者复制和分发可分发代码作为这些程序的一部分。
 - b. 分发要求。对于您分发的任何可分发代码, 您必须
 - 在程序中向其添加重要主功能;
 - 对于文件扩展名为 .lib 的任意可分发代码, 仅分发通过您程序的链接器运行此类可分发代码的结果;
 - 仅在可分发代码未经修改的情况下, 将其作为安装程序的一部分分发;
 - 要求分发者和外部最终用户同意不低于本协议要求的保护条款;
 - 在您的程序上显示有效的版权声明; 以及
 - 对于与分发或使用您的程序相关的索赔, 为 Microsoft 辩护、补偿和使其免受损害, 包括律师费。
 - c. 分发限制。您不能
 - 更改可分发代码中的任何著作权、商标或专利通知;
 - 在您的程序名称中使用 Microsoft 商标, 或者以暗示程序来自 Microsoft 或者得到 Microsoft 认可的方式使用 Microsoft 商标;
 - 将可分发代码分发给在 Windows 平台之外的平台上运行;
 - 在可分发代码中包括恶意、欺骗性或者非法程序; 或者
 - 修改或分发任意可分发代码的源代码, 以使其任何部分都受限于排除许可。排除许可是针对使用、修改或分发的条件, 要求
 - 代码以源代码的形式公布或分发; 或
 - 其他人有权修改它。
2. 补充的支持服务。Microsoft 根据 www.support.microsoft.com/common/international.aspx 中所述向本软件提供支持服务。

8) 基于 Windows 的 Docker 映像 – visualfsharpools , v 4.0

(许可条款位于 : <https://raw.githubusercontent.com/Microsoft/visualfsharp/master/License.txt>)

版权所有 (c) Microsoft Corporation. 保留所有权利。

根据 Apache 许可版本 2.0 授予许可 (简称“许可”) , 如果不遵守许可, 您不可使用这些文件。您可以在以下地址获取许可的副本

<http://www.apache.org/licenses/LICENSE-2.0>

除非适用的法律要求或以书面方式表示同意, 否则, 根据该许可分发的软件按“原样”分发, 无任何明示或暗示的保证或条件。请参阅许可证以了解在许可证下特定语言的适用权限和限制。

9) 基于 Windows 的 Docker 映像 – netfx-pcl-reference-assemblies-4.6

MICROSOFT 软件许可条款

MICROSOFT .NET 可移植类库引用程序集 – 4.6

这些许可条款是 Microsoft Corporation (或您所在地的其附属公司) 与您达成的协议。请仔细阅读。它们适用于以上所述软件。该条款也适用于此软件的任意 Microsoft

- 更新,
- 补充、
- 基于 Internet 的服务和
- 支持服务 ,

除非随这些项目另有其他条款。如果是这样, 则那些条款也适用。

使用本软件即表示您接受这些条款。如果您不接受这些条款, 请勿使用本软件。

如果您遵循这些许可条款, 您将拥有以下永久权利。

1. 安装和使用权利。您可以安装和使用任何数量的软件副本来设计、开发和测试您的程序。
2. 其他许可要求和/或使用权利。
 - a. 可分发代码。您可以在您开发的开发人员工具程序中分发本软件, 以便您程序的客户可以开发可用于任何设备或操作系统的可移植库, 前提是您遵守以下条款。
 - i. 使用和分发权利。本软件是“可分发代码”。
 - 可分发代码。您可以复制和分发本软件的对象代码形式。
 - 第三方分发。您可以允许程序分发者复制和分发可分发代码作为这些程序的一部分。
 - ii. 分发要求。对于您分发的任何可分发代码, 您必须
 - 在程序中向其添加重要主功能;
 - 要求分发者和您的客户同意不低于本协议要求的保护条款;
 - 在您的程序上显示有效的版权声明; 以及
 - 对于与分发或使用您的程序相关的索赔, 为 Microsoft 辩护、补偿和使其免受损害, 包括律师费。
 - iii. 分发限制。您不能
 - 更改可分发代码中的任何著作权、商标或专利通知;
 - 在您的程序名称中使用 Microsoft 商标, 或者以暗示程序来自 Microsoft 或者得到 Microsoft 认可的方式使用 Microsoft 商标;
 - 在可分发代码中包括恶意、欺骗性或者非法程序; 或者
 - 修改或分发可分发代码, 以使其任何部分都受限排除许可。排除许可是针对使用、修改或分发的条件, 要求
 - 代码以源代码的形式公布或分发; 或
 - 其他人有权修改它。
 3. 许可范围。该软件授予许可, 而非销售。本协议仅向您提供使用软件的一些权利。Microsoft 保留所有其他权利。除非适用法律在此限制之外赋予您更多的权利, 否则您只能在本协议中明确允许的情况下使用该软件。在使用时, 您必须遵守软件中仅允许您以特定方式使用它的任意技术限制。您不能
 - 规避软件中的任何技术限制;

- 逆向工程、反编译或反汇编本软件，尽管有此限制，但在适用法律明确允许的范围内可以执行上述操作；
 - 发布本软件供其他人复制；或者
 - 出租、租赁或出借本软件。
4. 反馈。您可以提供关于本软件的反馈。如果您向 Microsoft 提供关于本软件的反馈，则表示您向 Microsoft 免费提供以任何方式和任何用途使用、共享和商业化您的反馈的权利。您还可以免费向第三方提供其产品、技术和服务所需的任何专利权，以便使用包含反馈的 Microsoft 软件或服务的任何特定部分或者与之进行交互。如果反馈受下面这样的许可证的约束，则您不能提供反馈：该许可证要求 Microsoft 向第三方提供其软件或文档的许可，因为我们在相应软件或文档中包含了您的反馈。这些权利不受本协议的约束。
 5. 转让给第三方。本软件的第一个用户可以将本软件和本协议直接转让给第三方。在转让之前，该第三方必须同意本协议适用于本软件的转让和使用。第一个用户在将本软件独立于设备进行转让之前，必须先卸载本软件。第一个用户不能保留任何副本。
 6. 出口限制。本软件受美国出口法律和法规的约束。您必须遵守适用于本软件的所有国内和国际出口法律和法规。这些法律包括对目的地、最终用户和最终用途的限制。有关更多信息，请参阅 www.microsoft.com/exporting。
 7. 支持服务。由于此软件“按原样”提供，我们可能不会为它提供支持服务。
 8. 完整协议。本协议以及您使用的补充、更新、基于 Internet 的服务和支持服务的条款是本软件和我们提供的任何支持服务的完整协议。
 9. 适用的法律。
 - a. 美国。如果您在美国购买本软件，华盛顿州法律管辖本协议的解释，并适用于违反本协议的索赔，无论法律原则是否冲突都是如此。您居住的州的法律管辖所有其他索赔，包括根据国家消费者保护法、不正当竞争法和侵权行为提起的索赔。
 - b. 在美国以外的国家或地区。如果您在任何其他国家/地区购买本软件，则适用该国家/地区的法律。
 10. 法律效力。本协议描述了特定法律权利。根据贵国法律，您可能拥有其他权利。您还可能拥有从其获得本软件的一方的相应权利。如果您所在的国家/地区的法律不允许，根据这些法律，本协议不会更改您的权利。
 11. 免责声明。本软件按“原样”授予许可。您自行承担使用它的风险。MICROSOFT 不提供任何明示的担保、保证或条件。根据所在地区的法律，您可能拥有其他本许可证无法更改的消费者权利或法定担保。在您当地法律允许的范围内，Microsoft 排除有关适销性、针对特定目的的适用性和不侵权的默示担保。
- 对于澳大利亚 – 您在澳大利亚消费者法下获得了法定担保，本协议中的任何条款都无意影响这些权利。
12. 对补救措施和损害赔偿的限制和排除。您可以从 Microsoft 及其供应商获得的直接损害赔偿为 5.00 美元。您无法获得任何其他损害赔偿，包括后果性损害、利润损失、间接损害或事故损害。

此限制适用于

- 与第三方 Internet 站点或第三程序中的软件、服务、内容（包括代码）相关的任何内容；以及
- 违反合同；违反担保、保证或条件；严格责任；疏忽；或适用法律允许情况下其他侵权行为的索赔。

即使 Microsoft 已知或者应该知道造成损害的可能性，此条款仍适用。由于您所在的国家/地区可能不允许排除或限制事故损害、后果性损害或其他损害，上述限制或排除可能不适用于您。

10) 基于 Windows 的 Docker 映像 – visualcppbuildtools , v 14.0.25420.1

(许可条款位于 : <https://www.visualstudio.com/license-terms/mt644918/>)

Microsoft Visual C++ Build Tools

Microsoft 软件许可条款

Microsoft Visual C++ Build Tools

这些许可条款是 Microsoft Corporation (或您所在地的其附属公司) 与您达成的协议。它们适用于以上所述软件。这些条款还适用于软件的任意 Microsoft 服务或更新, 除非另有其他条款。

如果您遵循这些许可条款, 您将拥有以下权利。

1. 安装和使用权利。

- a. 可以有一位用户使用本软件的副本来开发和测试他们的应用程序。

2. 数据。软件可能会收集有关您以及您使用软件的信息, 并将信息发送给 Microsoft。Microsoft 可能会使用此信息来提供服务和改进我们的产品及服务。您可以选择退出其中的多种情况, 但并非全部, 如产品文档中所述。此外, 软件中还有一些功能, 可能使您能够从应用程序的用户收集数据。如果您使用这些功能在应用程序中启用数据收集, 则必须遵循适用的法律, 包括向应用程序的用户提供相应的说明。您可在以下网址的帮助文档和隐私声明中了解有关数据收集和使用的更多信息: <http://go.microsoft.com/fwlink/?LinkID=528096>。您使用软件操作即表明您同意这些做法。

3. 特定组件的条款。

- a. 构建服务器。本软件可能包含 BuildServer.TXT 文件中列出的一些构建服务器组件, 和/或位于遵循此 Microsoft 软件许可条款的 BuildServer 列表中列出的任何文件。如果这些项目包含在本软件中, 您可以复制并安装到您的构建计算机。您和您组织中的其他人可以在您的构建计算机上使用这些项目, 仅用于编译、构建、验证和归档应用程序, 或者作为构建过程的一部分运行质量或性能测试。
- b. Microsoft 平台。软件可能包括来自 Microsoft Windows、Microsoft Windows Server、Microsoft SQL Server、Microsoft Exchange、Microsoft Office 和 Microsoft SharePoint 的组件。这些组件受独立协议及自己的产品支持政策控制, 如位于组件安装目录中或随软件提供的“Licenses”文件夹中的许可条款所述。
- c. 第三方组件。软件可能会包括第三方组件, 这些组件具有单独的法律声明或受其他协议控制, 如随软件提供的 ThirdPartyNotices 文件中所述。即使此类组件受其他协议控制, 以下免责声明以及对损害赔偿的限制或排除仍适用。
- d. 程序包管理器。本软件可能包含软件包管理器 (如 Nuget), 它允许您下载其他 Microsoft 和第三方软件包以供您的应用程序使用。这些软件包遵循它们自己的许可证, 而不是本协议。Microsoft 不会分发、许可任何第三方软件包或者为其提供任何担保。

4. 许可范围。该软件授予许可, 而非销售。本协议仅向您提供使用软件的一些权利。Microsoft 保留所有其他权利。除非适用法律在此限制之外赋予您更多的权利, 否则您只能在本协议中明确允许的情况下使用该软件。在使用时, 您必须遵守软件中仅允许您以特定方式使用它的任意技术限制。有关更多信息, 请参阅 <http://www.microsoft.com/licensing/userights>。您不能

- 规避软件中的任何技术限制;
- 逆向工程、反编译或反汇编软件, 或者尝试这样做, 除非且仅限于在第三方许可条款要求的范围内, 该条款规定本软件中可能包含的特定开源组件的使用;
- 删除、最小化、阻止或修改 Microsoft 或其供应商的任何通知;
- 以违反法律的方式使用软件; 或者
- 共享、发布、租借或租用软件, 或者将软件独立托管为解决方案供其他人使用。

5. 出口限制。您必须遵守所有适用于该软件的国内和国际出口法律及法规, 这包括对目的地、最终用户和最终用途的限制。有关出口限制的更多信息, 请访问 (aka.ms/exporting)。

6. 支持服务。由于此软件“按原样”提供, 我们可能不会为它提供支持服务。

7. 完整协议。本协议以及您使用的补充、更新、基于 Internet 的服务和支持服务的条款是本软件和支持服务的完整协议。

8. 适用的法律。如果您在美国购买本软件, 华盛顿州法律管辖对本协议的解释以及违反协议的索赔, 您居住州的法律适用于所有其他索赔。如果您在任何其他国家/地区购买本软件, 则适用该国家/地区的法律。

9. 消费者权利; 区域差异。本协议描述了特定法律权利。根据所在的州或国家/地区, 您可能拥有其他权利, 包括消费者权利。除了与 Microsoft 的关系之外, 您可能还对与您购买该软件的一方的拥有相应权利。如果您所在的州或国家/地区的法律不允许, 本协议不会更改这些其他权利。例如, 如果您在以下区域之一购买了本软件, 或者有强制性国家/地区法律适用, 则以下条款适用于您:

- 澳大利亚. 您在澳大利亚消费者法下获得了法定担保, 本协议中的任何内容都无意影响这些权利。
- 加拿大. 如果您在加拿大购买此软件, 您可通过关闭自动更新功能、断开设备与 Internet 的连接 (但是, 在您重新连接到 Internet 时, 软件将恢复检查和安全更新) 或者卸载软件来停止接收更新。产品文档 (如果有) 可能会说明如何关闭特定设备或软件的更新。
- 德国和奥地利。
 - 担保。正确授予许可的软件, 将基本按照随该软件所提供的任意 Microsoft 材料中所述执行。但是, Microsoft 不提供任何与所许可软件相关的合同担保。
 - 责任限制。如果发生故意行为、重大过失、基于“产品责任法案”的索赔, 以及在发生死亡或人身或身体伤害的情况下, Microsoft 将依照法定法律承担法律责任。

在遵守上述第 (ii) 条的前提下, 如果 Microsoft 违反此类重大合同义务, Microsoft 将仅对轻微过失承担责任, 履行这一条有助于本协议的正常履行, 违反这一条会危及本协议的用途以及一方可以不断信任的合规性 (所谓的“基本义务”)。在其他轻微过失的情况下, Microsoft 不承担轻微过失的责任。

- 10 法律效力。本协议描述了特定法律权利。根据您的州或国家/地区的法律, 您可能拥有其他权利。如果您所在州或国家/地区的法律不允许, 根据这些法律, 本协议不会更改您的权利。在不限制前述条款的情况下, 对于澳大利亚, 您在澳大利亚消费者法下获得了法定担保, 本协议中的任何条款都无意影响这些权利
- 11 免责声明。本软件按“原样”授予许可。您自行承担使用它的风险。MICROSOFT 不提供任何明示的担保、保证或条件。在您当地法律允许的范围内, Microsoft 排除有关适销性、针对特定目的的适用性和不侵权的默示担保。
- 12 对损害赔偿的限制或排除。您可以从 Microsoft 及其供应商获得的直接损害赔偿为 5.00 美元。您无法获得任何其他损害赔偿, 包括后果性损害、利润损失、间接损害或事故损害。

此限制适用于 (a) 第三方 Internet 站点或第三方应用程序上与软件、服务、内容 (包括代码) 相关的任意内容; (b) 违反合同; 违反担保、保证或条件; 严格责任; 疏忽; 或法律允许情况下其他侵权行为的索赔。

即使 Microsoft 已知或者应该知道造成损害的可能性, 此条款仍适用。由于您所在的国家/地区可能不允许排除或限制事故损害、后果性损害或其他损害, 上述限制或排除可能不适用于您。

11) 基于 Windows 的 Docker 映像 – microsoft-windows-netfx3-ondemand-package.cab

Microsoft 软件补充许可条款

Microsoft .NET Framework 3.5 SP1, 适用于 Microsoft Windows 操作系统

Microsoft Corporation (或您所在地的其附属公司) 向您授予此补充的许可。如果您获得使用 Microsoft Windows 操作系统软件 (本补充内容适用) (下称“软件”) 的许可, 则可以使用本补充内容。如果您未获得软件的许可, 就不能使用它。您可以将本补充内容的副本用于软件的每个授予有效许可的副本。

以下许可条款描述了此补充的额外使用条款。软件的这些条款和许可条款适用于您对补充的使用。如果存在冲突, 这些补充许可条款适用。

使用本补充内容即表示您接受这些条款。如果您不接受这些条款, 请勿使用本补充内容。

如果您遵循这些许可条款, 您将拥有以下权利。

1. 补充的支持服务。Microsoft 根据 www.support.microsoft.com/common/international.aspx 中所述向本软件提供支持服务。

2. Microsoft .NET 基准测试。本软件包括 Windows 操作系统 (.NET 组件) 的 .NET Framework、Windows Communication Foundation、Windows Presentation Foundation 和 Windows Workflow Foundation 组件。您可以对 .NET 组件执行内部基准测试。您可以披露 .NET 组件的任何基准测试的结果，前提是您必须遵守在以下网址建立的条件：<http://go.microsoft.com/fwlink/?LinkID=66406>。

尽管您可能与 Microsoft 达成任何其他协议，但如果您披露了此类基准测试结果，则 Microsoft 有权披露其针对与适用 .NET 组件竞争的产品进行基准测试的结果，前提是该组件符合以下网址建立的条件：<http://go.microsoft.com/fwlink/?LinkID=66406>。

12) 基于 Windows 的 Docker 映像 – dotnet-sdk

(网址：<https://github.com/dotnet/core/blob/master/LICENSE>)

MIT 许可证 (MIT)

版权所有 (c) Microsoft Corporation

特此免费授予任何人获得本软件及相关文档文件 (“软件”) 的副本，以无限制地处理本软件，包括但不限于使用、复制、修改、合并、发布、分发、再许可和/或销售本软件的副本，并允许向其提供了本软件上述权利的人员遵守以下条件：

上述版权声明和本许可声明应包含在本软件的所有副本或主要部分中。

此软件按“原样”提供，无任何明示或暗示的保证，包括但不限于有关适销性、针对特定目的的适用性和不侵权的保证。任何情况下，作者或版权所有者都不应承担任何索赔、损害赔偿或其他责任，无论是因软件或使用或其他软件处理引起的或与其相关的合同行为、侵权行为或其他行为。

AWS CodeBuild 用户指南文档历史记 录

下表描述了自 AWS CodeBuild 上一次发布以来对文档所做的重要更改。如需对此文档更新的通知，您可以订阅 RSS 源。

- 最新 API 版本：2016-10-06
- 文档最新更新时间：2018 年 7 月 10 日

update-history-change	update-history-description	update-history-date
支持报告构建的状态 (p. 245)	AWS CodeBuild 现在可以向您的源提供商报告构建的开始和完成状态。有关更多信息，请参阅在 AWS CodeBuild 中创建构建项目 。	July 10, 2018
添加到 AWS CodeBuild 文档的环境变量 (p. 245)	构建环境中的环境变量 页面使用 CODEBUILD_BUILD_ID、CODEBUILD_LOG_PATH 和 CODEBUILD_START_TIME 环境变量进行了更新。	July 9, 2018
支持构建规范文件中的 finally 语句块 (p. 245)	AWS CodeBuild 文档使用构建规范文件中的可选 finally 语句块的详细信息进行了更新。finally 语句块命令总是在其相应的命令语句块命令之后执行。有关更多信息，请参阅 构建规范语法 。	June 20, 2018
AWS CodeBuild 代理更新通知 (p. 245)	AWS CodeBuild 文档使用您通过 Amazon SNS 获得新版本 AWS CodeBuild 代理发布通知的方式的详细信息进行了更新。有关更多信息，请参阅 接收有关新 AWS CodeBuild 代理版本的通知 。	June 15, 2018

早期更新

下表描述了 2018 年 6 月之前的每个 AWS CodeBuild 用户指南版本中的重要更改。

更改	描述	日期
支持 Windows 构建	AWS CodeBuild 现在支持 Microsoft Windows Server 平台的构建，包括 Windows 上 .NET Core 2.0 的预打包构建环境。有关更多信息，请参阅 适用于 AWS CodeBuild 的 Microsoft Windows 示例 (p. 20) 。	2018 年 5 月 25 日

更改	描述	日期
支持构建幂等性	当您使用 AWS Command Line Interface (AWS CLI) 运行 <code>start-build</code> 命令时，可以指定构建为幂等性的。有关更多信息，请参阅 运行构建项目 (AWS CLI) (p. 186)。	2018 年 5 月 15 日
支持覆盖多个构建项目设置	现在，在创建构建时，可以覆盖多个构建项目设置。覆盖仅针对该构建。有关更多信息，请参阅 在 AWS CodeBuild 中运行构建项目 (p. 183)。	2018 年 5 月 15 日
VPC 终端节点支持	现在，您可以使用 VPC 终端节点来提高构建的安全性。有关更多信息，请参阅 使用 VPC 终端节点 (p. 127)。	2018 年 3 月 18 日
支持触发器	现在，您可以创建触发器，按固定频率安排构建。有关更多信息，请参阅 创建 AWS CodeBuild 触发器 (p. 171)。	2018 年 3 月 28 日
FIPS 终端节点文档	现在，您可以了解如何使用 AWS Command Line Interface (AWS CLI) 或 AWS 软件开发工具包来告知 AWS CodeBuild 使用四种联邦信息处理标准 (FIPS) 终端节点之一。有关更多信息，请参阅 指定 AWS CodeBuild 终端节点 (p. 209)。	2018 年 3 月 28 日
AWS CodeBuild 在 亚太地区（孟买）、欧洲（巴黎）和 南美洲（圣保罗）中可用	AWS CodeBuild 现已在 亚太地区（孟买）、欧洲（巴黎）和 南美洲（圣保罗）区域中可用。有关更多信息，请参阅 Amazon Web Services 一般参考 中的“AWS 区域和终端节点”主题的 AWS CodeBuild 部分。	2018 年 3 月 28 日
GitHub Enterprise 支持	AWS CodeBuild 现在可以从存储在 GitHub Enterprise 存储库中的源代码进行构建。有关更多信息，请参阅 GitHub Enterprise 示例 (p. 46)。	2018 年 1 月 25 日
Git clone 深度支持	AWS CodeBuild 现已支持创建一个浅克隆，其历史记录会截断至指定数量的提交。有关更多信息，请参阅 创建构建项目 (p. 154)。	2018 年 1 月 25 日
VPC 支持	支持 VPC 的构建现在能够访问 VPC 内的资源。有关更多信息，请参阅 VPC 支持 (p. 125)。	2017 年 11 月 27 日

更改	描述	日期
依赖项缓存支持	AWS CodeBuild 现在支持依赖项缓存。这允许 AWS CodeBuild 将构建环境的某些可重用部分保存在缓存中并在各个构建中使用它。	2017 年 11 月 27 日
构建徽章支持	AWS CodeBuild 现在支持使用构建徽章，该徽章提供一个动态生成的可嵌入映像 (徽章)，用以显示项目的最新构建状态。有关更多信息，请参阅 构建徽章示例 (p. 54) 。	2017 年 11 月 27 日
AWS Config 集成	AWS Config 现在支持 AWS CodeBuild 作为 AWS 资源，这表示该服务可以跟踪您的 AWS CodeBuild 项目。有关 AWS Config 的更多信息，请参阅 将 AWS Config 与 AWS CodeBuild 结合使用的示例 (p. 53) 。	2017 年 10 月 20 日
在 GitHub 存储库中自动重新构建更新的源代码	如果您的源代码存储在 GitHub 存储库中，则可让 AWS CodeBuild 在代码更改被推送到存储库时重新构建源代码。有关更多信息，请参阅 GitHub 拉取请求示例 (p. 51) 。	2017 年 9 月 21 日
在 Amazon EC2 Systems Manager Parameter Store 中存储和检索敏感或大型环境变量的新方法	您现在可以使用 AWS CodeBuild 控制台或 AWS CLI 来检索存储在 Amazon EC2 Systems Manager Parameter Store 中的敏感或大型环境变量。现在也可以使用 AWS CodeBuild 控制台将这些类型的环境变量存储在 Amazon EC2 Systems Manager Parameter Store 中。以前，您只能通过将这些类型的环境变量包含在构建规范中或运行构建命令以自动化 AWS CLI 来检索这些变量。您只能通过使用 Amazon EC2 Systems Manager Parameter Store 控制台来存储这些类型的环境变量。有关更多信息，请参阅 创建构建项目 (p. 154) 、 更改构建项目的设置 (p. 174) 和 运行构建项目 (p. 183) 。	2017 年 9 月 14 日
构建删除支持	您现在可以在 AWS CodeBuild 中删除构建。有关更多信息，请参阅 删除构建 (p. 196) 。	2017 年 8 月 31 日

更改	描述	日期
使用构建规范检索存储在 Amazon EC2 Systems Manager Parameter Store 中的敏感或大型环境变量的更新方法	AWS CodeBuild 现在可让您更轻松地使用构建规范来检索存储在 Amazon EC2 Systems Manager Parameter Store 中的敏感或大型环境变量。以前，您只能通过运行构建命令自动化 AWS CLI 来检索这些类型的环境变量。有关更多信息，请参阅 构建规范语法 (p. 107) 中的 parameter-store 映射。	2017 年 8 月 10 日
AWS CodeBuild 支持 Bitbucket	AWS CodeBuild 现在可以从存储在 Bitbucket 存储库中的源代码进行构建。有关更多信息，请参阅 创建构建项目 (p. 154) 和 运行构建项目 (p. 183) 。	2017 年 8 月 10 日
AWS CodeBuild 在美国西部 (加利福尼亚北部)、欧洲 (伦敦) 和加拿大 (中部) 中可用	AWS CodeBuild 现已在美国西部 (加利福尼亚北部)、欧洲 (伦敦) 和加拿大 (中部) 区域中可用。有关更多信息，请参阅 Amazon Web Services 一般参考中的“AWS 区域和终端节点”主题的 AWS CodeBuild 部分。	2017 年 6 月 29 日
Linux 中的 .NET 核心示例	已添加说明如何使用 AWS CodeBuild 和 .NET 核心通过用 C# 编写的代码构建可执行文件的示例。有关更多信息，请参见 Linux 中的 .NET 核心示例 (p. 103) 。	2017 年 6 月 29 日
已支持替代构建规范文件名称和位置	您现在可以指定构建规范文件的替代文件名称或位置以用于构建项目，而不是源代码根处的名为 buildspec.yml 的默认构建规范文件。有关更多信息，请参阅 构建规范文件名称和存储位置 (p. 107) 。	2017 年 6 月 27 日
更新了构建通知示例	AWS CodeBuild 现在通过 Amazon CloudWatch Events 和 Amazon Simple Notification Service (Amazon SNS) 提供对构建通知的内置支持。先前的 构建通知示例 (p. 56) 已更新为演示此新行为。	2017 年 6 月 22 日
添加了自定义映像中的 Docker 示例	已添加说明如何使用 AWS CodeBuild 和自定义 Docker 构建映像来构建和运行 Docker 映像的示例。有关更多信息，请参见 自定义映像示例中的 Docker (p. 73) 。	2017 年 6 月 7 日

更改	描述	日期
从 GitHub 提取源代码的拉取请求	当使用依赖于存储在 GitHub 存储库中的源代码的 AWS CodeBuild 运行构建时，您现在可以指定要构建的 GitHub 拉取请求 ID。您还可以改为指定提交 ID、分支名称或标签名称。有关更多信息，请参阅 运行构建项目 (控制台) (p. 183) 中的 Source version 值或 运行构建项目 (AWS CLI) (p. 186) 中的 sourceVersion 值。	2017 年 6 月 6 日
更新了构建规范版本	已发布新版本的构建规范格式。版本 0.2 可解决 AWS CodeBuild 在默认 Shell 的单独实例中运行每条构建命令的问题。此外，在版本 0.2 中，environment_variables 已重命名为 env，而且 plaintext 已重命名为 variables。有关更多信息，请参阅 适用于 AWS CodeBuild 的构建规范参考 (p. 107)。	2017 年 5 月 9 日
在 GitHub 中提供构建映像的 Dockerfile	AWS CodeBuild 提供的许多构建映像的定义在 GitHub 中作为 Dockerfile 提供。有关更多信息，请参阅 AWS CodeBuild 提供的 Docker 映像 (p. 114) 中表的“定义”列。	2017 年 5 月 2 日
AWS CodeBuild 在 欧洲 (法兰克福)、亚太区域 (新加坡)、亚太区域 (悉尼) 和 亚太区域 (东京) 可用	AWS CodeBuild 现已在 欧洲 (法兰克福)、亚太区域 (新加坡)、亚太区域 (悉尼) 和 亚太区域 (东京) 区域可用。有关更多信息，请参阅 Amazon Web Services 一般参考 中的“AWS 区域和终端节点”主题的 AWS CodeBuild 部分。	2017 年 3 月 21 日
AWS CodePipeline 对 AWS CodeBuild 的测试操作支持	现在，您可以将使用 AWS CodeBuild 的测试操作添加到 AWS CodePipeline 中的管道。有关更多信息，请参阅 将 AWS CodeBuild 测试操作添加到管道 (AWS CodePipeline 控制台) (p. 148)。	2017 年 3 月 8 日
构建规范支持从选定的顶级目录中获取构建输出	借助构建规范，您现在可以指定单独的顶级目录，指示 AWS CodeBuild 将其中的内容添加到构建输出项目中。为此，您可以使用 base-directory 映射。有关更多信息，请参阅 构建规范语法 (p. 107)。	2017 年 2 月 8 日

更改	描述	日期
内置环境变量	AWS CodeBuild 为您要使用的构建提供了额外的内置环境变量。这些包括描述启动了构建项目的实体的环境变量、指向源代码存储库的 URL 以及源代码的版本 ID 等。有关更多信息，请参阅 构建环境中的环境变量 (p. 120) 。	2017 年 1 月 30 日
AWS CodeBuild 在 美国东部 (俄亥俄州) 可用	AWS CodeBuild 现已在 美国东部 (俄亥俄州) 区域可用。有关更多信息，请参阅 Amazon Web Services 一般参考 中的“AWS 区域和终端节点”主题的 AWS CodeBuild 部分。	2017 年 1 月 19 日
AWS Lambda 示例	向示例添加了一条引用，说明了如何使用 AWS CodeBuild 以及 Lambda、AWS CloudFormation 和 AWS CodePipeline 来构建和部署遵循了 AWS 无服务器应用程序模型 (AWS SAM) 标准的无服务器应用程序。有关更多信息，请参见 AWS Lambda 示例 (p. 78) 。	2016 年 12 月 20 日
C++ 和 Go 示例	添加了说明如何使用 AWS CodeBuild 构建 C++ 和 Go 输出项目的示例。有关更多信息，请参阅 C++ 示例 (p. 85) 和 Go 示例 (p. 87) 。	2016 年 12 月 9 日
Shell 和命令行行为信息	AWS CodeBuild 在构建环境默认 Shell 的单独实例中运行您指定的每个命令。这种默认行为可对您的命令产生一些意想不到的副作用。我们推荐了一些方法，用于在需要时处理这种默认行为。有关更多信息，请参阅 构建环境中的 Shell 和命令 (p. 120) 。	2016 年 12 月 9 日
环境变量信息	AWS CodeBuild 提供了您可以在构建命令中使用的多个环境变量。您也可以定义自己的环境变量。有关更多信息，请参阅 构建环境中的环境变量 (p. 120) 。	2016 年 12 月 7 日
故障排除主题	现已提供故障排除信息。有关更多信息，请参阅 AWS CodeBuild 故障排除 (p. 225) 。	2016 年 12 月 5 日
Jenkins 插件初始版本	这是 AWS CodeBuild Jenkins 插件的初始版本。有关更多信息，请参阅 将 AWS CodeBuild 与 Jenkins 结合使用 (p. 152) 。	2016 年 12 月 5 日
用户指南初始版本	这是 AWS CodeBuild 用户指南的初始版本。	2016 年 12 月 1 日

AWS 词汇表

有关最新 AWS 术语，请参阅 AWS General Reference 中的 [AWS 词汇表](#)。