aws

用户指南

AWS AppConfig

AWS AppConfig: 用户指南

Table of Contents

什么是 AWS AppConfig？

AWS AppConfig功能标志和动态配置可帮助软件开发者快速、安全地调整生产环境中的应用程序行为，而无需部署全部代码。AWS AppConfig 加快了软件发布频率，提高了应用程序的弹性，并帮助您更快地解决突发问题。通过功能标志，您可以逐步向用户发布新功能，并在向所有用户全面部署新功能之前，衡量这些更改的影响。通过操作标志和动态配置，您可以更新阻止列表、允许列表、节流限制、日志记录冗余度，并执行其他操作调整，以快速应对生产环境中的问题。

> **Note**
>
> AWS AppConfig 是 AWS Systems Manager 的功能。

提高效率并更快地发布变更

使用带有新功能的功能标志可加快向生产环境发布变更的过程。功能标志使您能够使用基于主干的开发方法编写软件，而不是依赖需要在发布前进行复杂合并的长期开发分支。功能标志使您能够在 CI/CD 管道中安全地推出对用户隐藏的预发布代码。准备发布更改时，可以更新功能标志，而无需部署新代码。启动完成后，该标志仍可作为区块开关，禁用新功能或能力，而无需回滚代码部署。

借助内置安全功能，避免意外更改或故障

AWS AppConfig 提供了以下安全功能，可帮助您避免启用功能标志或更新可能导致应用程序故障的配置数据。

- 验证程序：在将更改部署到生产环境之前，验证程序确保配置数据在语法和语义上正确无误。
- 部署策略：部署策略使您能够在几分钟或几小时内缓慢地将更改发布到生产环境。
- 监控和自动回滚：AWS AppConfig与 Amazon CloudWatch 集成，可监控应用程序的更改。如果您的应用程序因错误的配置更改而变得不健康，并且该更改触发了 CloudWatch 中的警报，AWS AppConfig 会自动回滚更改，以尽量减少对应用程序用户的影响。

安全且可扩展的功能标志部署

AWS AppConfig 与 AWS Identity and Access Management (IAM) 集成，提供对服务的细粒度、基于角色的访问。AWS AppConfig 还与 AWS Key Management Service(AWS KMS) 集成以进行加密，与 AWS CloudTrail 集成以进行审计。在向外部客户发布之前，所有 AWS AppConfig 安全控制措施最初都是由大规模使用该服务的内部客户共同开发和验证的。

AWS AppConfig 使用案例

尽管应用程序的配置内容可能因应用程序而异，但 AWS AppConfig 支持以下用例，这些用例涵盖了广泛的客户需求：

- 功能标志和切换 — 在受控环境中安全地向客户发布新功能。如果您遇到问题，请立即回滚更改。
- 应用程序调整— 谨慎引入应用程序变更，同时在生产环境中测试这些变更对用户的影响。
- 允许列表或阻止列表— 无需部署新代码，即可控制高级功能的访问权限或即时阻止特定用户。
- 集中式配置存储 — 在所有工作负载中保持配置数据的有序性和一致性。您可以使用 AWS AppConfig 来部署存储在 AWS AppConfig 托管配置存储、AWS Secrets Manager、Systems Manager 参数存储或亚马逊 S3 中的配置数据。

使用 AWS AppConfig 的好处

AWS AppConfig 为您的组织提供了以下优势：

- 减少客户的意外停机时间

  AWS AppConfig 允许您创建规则以验证配置，从而减少应用程序停机时间。无法部署无效的配置。AWS AppConfig 提供了两种可选方法以验证配置：

  - 对于语法验证，您可以使用 JSON 架构。AWS AppConfig 使用 JSON 架构验证配置，以确保配置更改符合应用程序要求。
  - 对于语义验证，AWS AppConfig 可以调用您拥有的 AWS Lambda 函数来验证配置中的数据。
- 快速在一组目标中部署更改

  通过从中心位置部署配置更改，AWS AppConfig 简化了批量管理应用程序的过程。AWS AppConfig 支持在 AWS AppConfig 托管的配置商店、系统管理器参数商店、系统管理器 (SSM) 文档和 Amazon S3 存储的配置。您可以将 AWS AppConfig 与 EC2 实例上托管的应用程序、AWS Lambda、容器、移动应用程序或 IoT 设备一起使用。

  目标不需要配置 Systems Manager SSM Agent 或其他 Systems Manager 功能所需的 IAM 实例配置文件。这意味着，AWS AppConfig 适用于非托管实例。
- 更新应用程序而不会发生中断

  AWS AppConfig 在运行时将配置更改部署到目标，而无需执行繁重的生成过程或停止使用目标。
- 控制在应用程序中部署更改

在将配置更改部署到目标时，AWS AppConfig 允许您使用部署策略以最大限度降低风险。部署策略允许您缓慢地向您的设备群部署配置更改。如果在部署过程中遇到问题，您可以在配置更改影响到大多数主机之前将其回滚。

AWS AppConfig 的工作原理

本节简要介绍了 AWS AppConfig 的工作原理和入门方法。

1. 确定要在云端管理的代码中的配置值

   在开始创建 AWS AppConfig 构件之前，我们建议您在代码中确定要使用 AWS AppConfig 来动态管理的配置数据。好的例子包括功能标志或切换、允许和阻止列表、日志冗长度、服务限制和节流规则等。

   如果您的配置数据已经存在于云端，您可以利用 AWS AppConfig 验证、部署和扩展功能进一步简化配置数据管理。

2. 创建应用程序命名空间

   要创建命名空间，需要创建一个名为应用程序的 AWS AppConfig 构件。应用程序只是一个像文件夹一样的组织结构。

3. 创建环境

   对于每个 AWS AppConfig 应用程序，您可以定义一个或多个环境。环境是目标的逻辑分组，例如 `Beta` 或 `Production` 环境中的应用程序、AWS Lambda 函数或容器。您也可以为应用程序子组件定义环境，例如应用程序的 `Web`、`Mobile` 和 `Back-end`。

   您可以为每个环境配置 Amazon CloudWatch 警报。系统在部署配置期间监控警报。如果触发警报，系统将回滚配置。

4. 创建配置文件

   配置文件包括一个 URI 和一个配置文件类型，其中 URI 可使 AWS AppConfig 在其存储位置找到配置数据。 AWS AppConfig 支持两种配置文件类型：功能标志和自由格式配置。功能标志配置文件将其数据存储在 AWS AppConfig 托管的配置存储区中，URI 就是 `hosted`。对于自由格式配置文件，您可以将数据存储在 AWS AppConfig 托管配置存储区或任何与 AWS AppConfig 集成的 AWS 服务中，如 在中创建自由表单配置文件 AWS AppConfig 中所述。

配置文件还可能包含可选的验证程序，以确保配置数据在语法和语义上正确无误。在开始部署时，AWS AppConfig 使用验证程序以执行检查。如果检测到任何错误，部署将回滚到之前的配置数据。

5. 部署配置数据

   创建新配置时，应指定以下内容：

   - 应用程序 ID
   - 配置文件 ID
   - 配置版本
   - 要在其中部署配置数据的环境 ID
   - 部署策略 ID，可定义更改生效的速度

   调用 StartDeployment API 操作时，AWS AppConfig 会执行以下任务：

   1. 使用配置文件中的位置 URI 从底层数据存储中读取配置数据。
   2. 使用在创建配置文件时指定的验证程序，验证配置数据在语法和语义上是否正确。
   3. 缓存数据副本，以便应用程序随时检索。此缓存副本被称为已部署数据。

6. 检索配置

   您可以将 AWS AppConfig 代理配置为本地主机，并让代理轮询 AWS AppConfig 以获取配置更新。代理调用 StartConfigurationSession 和 GetLatestConfiguration API 操作，并在本地缓存配置数据。要检索数据，您的应用程序会向本地主机服务器发出 HTTP 调用。AWS AppConfig代理支持多种用例，如 简化的检索方法 中所述。

   如果您的用例不支持 AWS AppConfig 代理，您可以直接调用 StartConfigurationSession 和 GetLatestConfiguration API 操作，将应用程序配置为轮询 AWS AppConfig 以获取配置更新。

开始使用 AWS AppConfig

以下资源可以帮助您直接使用 AWS AppConfig。

在 Amazon Web Services YouTube 频道上观看更多 AWS 视频。

以下博客可以帮助您进一步了解 AWS AppConfig 及其功能：

- 使用AWS AppConfig功能标志
- 验证 AWS AppConfig 功能标志和配置数据的最佳实践

SDK

有关特定于 AWS AppConfig 语言的 SDK 的信息，请参阅以下资源：

- AWS Command Line Interface
- AWS SDK for .NET
- 适用于 C++ 的 AWS SDK
- 适用于 Go 的 AWS SDK
- 适用于 Java V2 的 AWS SDK
- 适用于 JavaScript 的 AWS SDK
- 适用于 PHP V3 的 AWS SDK
- 适用于 Python 的 AWS SDK
- 适用于 Ruby V3 的 AWS SDK

AWS AppConfig 定价

AWS AppConfig 的定价是基于配置数据和功能标志检索的按实际使用量付费。我们建议使用 AWS AppConfig 代理来帮助优化成本。有关更多信息，请参阅 AWS Systems Manager 定价。

AWS AppConfig 配额

有关 AWS AppConfig 端点和服务配额以及其他 Systems Manager 配额的信息，请参阅 Amazon Web Services 一般参考。

> **Note**
>
> 有关存储 AWS AppConfig 配置的服务的配额的信息，请参阅 关于配置存储配额和限制。

设置 AWS AppConfig

如果您尚未这样做，请注册 AWS 账户 并创建管理用户。

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

要注册 AWS 账户

1. 打开 https://portal.aws.amazon.com/billing/signup。
2. 按照屏幕上的说明进行操作。

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

   当您注册时 AWS 账户，就会创建AWS 账户根用户一个。根用户有权访问该账户中的所有 AWS 服务 和资源。作为安全最佳实践，请为用户分配管理访问权限，并且只使用根用户来执行需要根用户访问权限的任务。

AWS 注册过程完成后会向您发送一封确认电子邮件。在任何时候，您都可以通过转至 https://aws.amazon.com/ 并选择我的账户来查看当前的账户活动并管理您的账户。

创建具有管理访问权限的用户

注册后，请保护您的安全 AWS 账户 AWS 账户根用户 AWS IAM Identity Center，启用并创建管理用户，这样您就不会使用 root 用户执行日常任务。

保护你的 AWS 账户根用户

1. 选择 Root 用户并输入您的 AWS 账户 电子邮件地址，以账户所有者的身份登录。AWS Management Console在下一页上，输入您的密码。

   要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。
2. 为您的根用户启用多重身份验证 (MFA)。

   有关说明，请参阅 I A M 用户指南中的为 AWS 账户 根用户启用虚拟 MFA 设备（控制台）。

创建具有管理访问权限的用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。
2. 在 IAM Identity Center 中，为用户授予管理访问权限。

   有关使用 IAM Identity Center 目录 作为身份源的教程，请参阅《用户指南》 IAM Identity Center 目录中的使用默认设置配置AWS IAM Identity Center 用户访问权限。

以具有管理访问权限的用户身份登录

- 要使用您的 IAM Identity Center 用户身份登录，请使用您在创建 IAM Identity Center 用户时发送到您的电子邮件地址的登录网址。

  有关使用 IAM Identity Center 用户登录的帮助，请参阅AWS 登录 用户指南中的登录 AWS 访问门户。

将访问权限分配给其他用户

1. 在 IAM Identity Center 中，创建一个权限集，该权限集遵循应用最低权限的最佳做法。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。
2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的添加组。

授权以编程方式访问

如果用户想在 AWS 外部进行交互，则需要编程访问权限 AWS Management Console。授予编程访问权限的方式取决于正在访问的用户类型 AWS。

要向用户授予编程式访问权限，请选择以下选项之一。

哪个用户需要编程式访问权限？	目的	方式
人力身份 （在 IAM Identity Center 中管理的用户）	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关的 AWS CLI，请参阅《AWS Command Line Interface 用户指南》 AWS IAM Identity Center中的配置 AWS CLI 以使用。 • 有关 AWS 软件开发工具包、工具和 AWS API，请参阅《软件开发工具包和AWS 工具参考指南》中的 IAM 身份中心身份验证。
IAM	使用临时证书签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照 IAM 用户指南中的将临时证书与 AWS 资源配合使用中的说明进行操作。
IAM	（不推荐使用） 使用长期凭证签署向 AWS CLI、 AWS 软件开发工具包或 AWS API 发出的编程请求。	按照您希望使用的界面的说明进行操作。 • 有关信息 AWS CLI，请参阅用户指南中的使用 IAM 用户证书进行身份验证。AWS Command Line Interface • 有关 AWS SDK 和工具，请参阅 S AWS DK 和工具参考指南中的使用长期凭证进行身份验证。 • 有关 AWS API，请参阅 IAM 用户指南中的管理 IAM 用户的访问密钥。

（可选）根据警报配置回滚权限 CloudWatch

您可以配置 AWS AppConfig 为回滚到配置的先前版本，以响应一个或多个 Amazon CloudWatch 警报。在配置部署以响应 CloudWatch 警报时，您可以指定一个 AWS Identity and Access Management (IAM) 角色。 AWS AppConfig 需要此角色才能监视 CloudWatch 警报。

> **Note**
>
> IAM 角色必须属于当前账户。默认情况下， AWS AppConfig 只能监控当前账户拥有的警报。如果要配置为回滚部署 AWS AppConfig 以响应来自其他账户的指标，则必须配置跨账户警报。有关更多信息，请参阅 A mazon CloudWatch 用户指南中的跨账户跨区域 CloudWatch 控制台。

使用以下过程创建允许根据 CloudWatch 警报 AWS AppConfig 进行回滚的 IAM 角色。本节包括以下过程。

1. 步骤 1：根据警报创建回滚权限策略 CloudWatch
2. 步骤 2：根据警报创建用于回滚的 IAM 角色 CloudWatch
3. 第 3 步：添加信任关系

步骤 1：根据警报创建回滚权限策略 CloudWatch

使用以下过程创建授予调用 `DescribeAlarms` API 操作 AWS AppConfig 权限的 IAM 策略。

根据警报创建用于回滚的 IAM 权限策略 CloudWatch

1. 访问：https://console.aws.amazon.com/iam/，打开 IAM 控制台。
2. 在导航窗格中，选择 Policies (策略)，然后选择 Create policy (创建策略)。
3. 在创建策略页面上，选择 JSON 选项卡。
4. 将 JSON 选项卡上的默认内容替换以下权限策略，然后选择 Next: Tags。

> **Note**
>
> 要返回有关 CloudWatch 复合警报的信息，必须为 DescribeAlarmsAPI 操作分配*权限，如下所示。如果`DescribeAlarms`范围较窄，则无法返回有关复合警报的信息。

```
{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "cloudwatch:DescribeAlarms"
                ],
                "Resource": "*"
            }
        ]
    }
```

5. 为该角色输入标签，然后选择 Next: Review。
6. 在查看页面上，将 **SSMCloudWatchAlarmDiscoveryPolicy** 输入到 名称 字段中。
7. 选择 创建策略。系统将让您返回到 Policies 页面。

步骤 2：根据警报创建用于回滚的 IAM 角色 CloudWatch

使用以下过程创建 IAM 角色并向其分配您在上一过程中创建的策略。

根据警报创建用于回滚的 IAM 角色 CloudWatch

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择 Roles (角色)，然后选择 Create role (创建角色)。
3. 在 Select type of trusted entity (选择受信任实体的类型) 下，选择 AWS service (Amazon Web Services 服务)。
4. 在紧靠选择将使用此角色的服务下面，选择 EC2：允许 EC2 实例调用 AWS 服务，并代表您 ，然后选择 下一步: 权限。
5. 在附加权限策略页面上，搜索 SSM CloudWatchAlarmDiscoveryPolicy。
6. 选择此策略，然后选择 Next: Tags。
7. 为该角色输入标签，然后选择 Next: Review。
8. 在 创建角色 页面上，将 **SSMCloudWatchAlarmDiscoveryRole** 输入到 角色名称 字段中，然后选择 创建角色。
9. 在 Roles 页面上，选择您刚刚创建的角色。此时将打开摘要页面。

第 3 步：添加信任关系

使用以下过程将您刚刚创建的角色配置为信任 AWS AppConfig。

为添加信任关系 AWS AppConfig

1. 在刚刚创建的角色的摘要页面上，选择信任关系选项卡，然后选择编辑信任关系。
2. 编辑策略以仅包含“appconfig.amazonaws.com”，如以下示例中所示：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 选择更新信任策略。

在中创建功能标志和自由格式配置数据 AWS AppConfig

本节中的主题可帮助您完成中的以下任务 AWS AppConfig。这些任务用于创建部署配置数据所需的重要构件。

1. 创建应用程序命名空间

   要创建应用程序命名空间，您需要创建一个名为应用程序的 AWS AppConfig 构件。应用程序只是一个像文件夹一样的组织构造。

2. 创建环境

   您可以为每个 AWS AppConfig 应用程序定义一个或多个环境。环境是由 AWS AppConfig 目标组成的逻辑部署组，例如`Beta`或`Production`环境中的应用程序。您也可以为应用程序子组件定义环境，例如 `AWS Lambda functions`、`Containers`、`Web`、`Mobile` 和 `Back-end`。

   您可以为每个环境配置 Amazon CloudWatch 警报，以自动回滚有问题的配置更改。系统在部署配置期间监控警报。如果触发警报，系统将回滚配置。

3. 创建配置文件

   除其他外，配置文件包括一个 AWS AppConfig 允许在其存储位置查找配置数据的 URI 和配置文件类型。 AWS AppConfig 支持两种配置文件类型：功能标志和自由格式配置。功能标志配置文件将其数据存储在 AWS AppConfig 托管配置存储中，URI 很简单`hosted`。对于自由格式配置文件，您可以将数据存储在 AWS AppConfig 托管配置存储库或与集成的其他 Systems Manager 功能或 AWS 服务中 AWS AppConfig，如中所在中创建自由表单配置文件 AWS AppConfig述。

   配置文件还可以包括可选的验证器，以确保您的配置数据在语法和语义上都是正确的。 AWS AppConfig 开始部署时使用验证器执行检查。在对配置目标进行任何更改之前，如果检测到任何错误，部署将会停止。

   > **Note**
   >
   > 除非您有在亚马逊简单存储服务 (Amazon S3) Simple Service (Amazon S3) 中存储密钥 AWS Secrets Manager 或管理数据的特定需求，否则我们建议您将配置数据托管在 AWS AppConfig 托管配置存储中，因为它提供了最多的功能和增强功能。

主题

示例配置

使用AWS AppConfig、的 AWS Systems Manager功能创建、管理和快速部署应用程序配置。配置 是一组影响应用程序行为的设置。下面是一些示例。

功能标志配置

以下功能标志配置可按区域启用或禁用移动支付和默认付款方式。

JSON

```
{
  "allow_mobile_payments": {
    "enabled": false
  },
  "default_payments_per_region": {
    "enabled": true
  }
}
```

YAML

```
---
allow_mobile_payments:
  enabled: false
default_payments_per_region:
  enabled: true
```

操作配置

以下自由格式配置对应用程序处理请求的方式进行了限制。

JSON

```
{
  "throttle-limits": {
    "enabled": "true",
    "throttles": [
      {
        "simultaneous_connections": 12
      },
      {
        "tps_maximum": 5000
      }
    ],
    "limit-background-tasks": [
      true
    ]
  }
}
```

YAML

```
---
throttle-limits:
  enabled: 'true'
  throttles:
  - simultaneous_connections: 12
  - tps_maximum: 5000
  limit-background-tasks:
  - true
```

访问控制列表配置

以下访问控制列表自由格式配置指定了哪些用户或组可以访问应用程序。

JSON

```
{
  "allow-list": {
    "enabled": "true",
    "cohorts": [
      {
        "internal_employees": true
```

```
      },
      {
        "beta_group": false
      },
      {
        "recent_new_customers": false
      },
      {
        "user_name": "Jane_Doe"
      },
      {
        "user_name": "John_Doe"
      }
    ]
  }
}
```

YAML

```
---
allow-list:
  enabled: 'true'
  cohorts:
  - internal_employees: true
  - beta_group: false
  - recent_new_customers: false
  - user_name: Jane_Doe
  - user_name: Ashok_Kumar
```

关于配置文件 IAM 角色

您可以使用创建提供配置数据访问权限的 IAM 角色 AWS AppConfig。或者您可以自行创建 IAM 角色。如果您使用创建角色 AWS AppConfig，则系统将创建该角色并根据您选择的配置源类型指定以下权限策略之一。

配置源为 Secrets Manager 密钥

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

```
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
             ],
            "Resource": [
                "arn:aws:secretsmanager:AWS ##:account_ID:secret:secret_name-a1b2c3"
            ]
        }
    ]
}
```

配置源是 Parameter Store 参数

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter"
            ],
            "Resource": [
                "arn:aws:ssm:AWS ##:account_ID:parameter/parameter_name"
            ]
        }
    ]
    }
```

配置源是 SSM 文档

```
{

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetDocument"
            ],
            "Resource": [
                "arn:aws:ssm:AWS ##:account_ID:document/document_name"
            ]
        }
```

```
    ]
}
```

如果您使用创建角色 AWS AppConfig，则系统还会为该角色创建以下信任关系。

```
{

  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

在 AWS AppConfig中为应用程序创建命名空间

本节中的过程可帮助您创建名为 AWS AppConfig 应用程序的构件。应用程序只是一种组织构造，就像文件夹一样，用于标识应用程序的命名空间。此组织构造与某些可执行代码单元有关。例如，您可以创建一个名为的应用程序 MyMobileApp 来组织和管理用户安装的移动应用程序的配置数据。必须先创建这些工件，然后 AWS AppConfig 才能使用部署和检索功能标志或自由格式配置数据。

> **Note**
>
> 您可以使用 AWS CloudFormation 创建 AWS AppConfig 项目，包括应用程序、环境、配置配置文件、部署、部署策略和托管配置版本。有关更多信息，请参阅《AWS CloudFormation 用户指南》中的 AWS AppConfig 资源类型参考。

主题

创建 AWS AppConfig 应用程序（控制台）

使用以下步骤通过 AWS Systems Manager 控制台创建 AWS AppConfig 应用程序。

创建应用程序

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，选择应用程序，然后选择创建应用程序。
3. 对于 Name (名称)，请输入应用程序的名称。
4. 对于 Description (描述)，请输入有关应用程序的信息。
5. （可选）在“扩展”部分，从列表中选择一个分机。有关更多信息，请参阅 关于 AWS AppConfig 扩展。
6. （可选）在标签部分，输入密钥和可选值。您最多可以为一个资源指定 50 个标签。
7. 选择创建应用程序。

AWS AppConfig 创建应用程序，然后显示“环境”选项卡。继续执行在 AWS AppConfig中为应用程序创建环境。

创建 AWS AppConfig 应用程序（命令行）

以下过程介绍如何使用 AWS CLI（在 Linux 或 Windows 上）或 AWS Tools for PowerShell 如何创建 AWS AppConfig 应用程序。

分步创建应用程序

1. 打开 AWS CLI.
2. 运行以下命令以创建应用程序。

   Linux

```
aws appconfig create-application \
  --name A_name_for_the_application \
  --description A_description_of_the_application \
  --tags User_defined_key_value_pair_metadata_for_the_application
```

Windows

```
aws appconfig create-application ^
  --name A_name_for_the_application ^
  --description A_description_of_the_application ^
  --tags User_defined_key_value_pair_metadata_for_the_application
```

PowerShell

```
New-APPCApplication `
  -Name Name_for_the_application `
  -Description Description_of_the_application `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_for_the_application
```

系统将返回类似于以下内容的信息。

Linux

```
{
   "Id": "Application ID",
   "Name": "Application name",
   "Description": "Description of the application"
}
```

Windows

```
{
   "Id": "Application ID",
   "Name": "Application name",
   "Description": "Description of the application"
}
```

PowerShell

```
ContentLength    : Runtime of the command
Description      : Description of the application
HttpStatusCode   : HTTP Status of the runtime
Id               : Application ID
Name             : Application name
```

```
ResponseMetadata : Runtime Metadata
```

在 AWS AppConfig中为应用程序创建环境

您可以为每个 AWS AppConfig 应用程序定义一个或多个环境。环境是由 AppConfig 目标组成的逻辑部署组，例如Production环境中的应用程序、 AWS Lambda 函数或容器。Beta您也可以为应用程序子组件定义环境，例如应用程序的 Web、Mobile 和 Back-end。您可以为每个环境配置 Amazon CloudWatch 警报。系统在部署配置期间监控警报。如果触发警报，系统将回滚配置。

开始前的准备工作

如果您想启用回滚配置 AWS AppConfig 以响应 CloudWatch 警报，则必须为一个 AWS Identity and Access Management (IAM) 角色配置一个 AWS AppConfig 允许响应 CloudWatch 警报的权限。您可以在以下过程中选择该角色。有关更多信息，请参阅 （可选）根据警报配置回滚权限 CloudWatch 。

主题

- 创建 AWS AppConfig 环境（控制台）
- 创建 AWS AppConfig 环境（命令行）

创建 AWS AppConfig 环境（控制台）

使用以下过程通过 AWS Systems Manager 控制台创建 AWS AppConfig 环境。

创建环境

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，选择应用程序，然后选择应用程序的名称以打开详细信息页面。
3. 选择 “环境” 选项卡，然后选择 “创建环境”。
4. 对于 Name (名称)，请输入环境的名称。
5. 对于 Description (描述)，请输入有关环境的信息。
6. （可选）在监控器部分中，选择 IAM 角色字段，然后选择一个有权在触发警报时回滚配置的 IAM 角色。
7. 在CloudWatch 警报列表中，选择一个或多个要监控的警报。 AWS AppConfig 如果其中一个警报进入警报状态，则回滚您的配置部署。

8. （可选）在“关联分机”部分，从列表中选择一个分机。有关更多信息，请参阅 关于 AWS AppConfig 扩展。
9. （可选）在标签部分，输入密钥和可选值。您最多可以为一个资源指定 50 个标签。
10. 选择创建环境。

AWS AppConfig 创建环境，然后显示环境详细信息页面。继续执行在 AWS AppConfig中创建配置文件。

创建 AWS AppConfig 环境（命令行）

以下过程介绍如何使用 AWS CLI（在 Linux 或 Windows 上）或 AWS Tools for PowerShell 如何创建 AWS AppConfig 环境。

分步创建环境

1. 打开 AWS CLI.
2. 运行以下命令以创建环境。

   Linux

```
aws appconfig create-environment \
  --application-id The_application_ID \
  --name A_name_for_the_environment \
  --description A_description_of_the_environment \
  --monitors
 "AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn" \
  --tags User_defined_key_value_pair_metadata_of_the_environment
```

   Windows

```
aws appconfig create-environment ^
  --application-id The_application_ID ^
  --name A_name_for_the_environment ^
  --description A_description_of_the_environment ^
  --monitors
 "AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn" ^
  --tags User_defined_key_value_pair_metadata_of_the_environment
```

PowerShell

```
New-APPCEnvironment `
  -Name Name_for_the_environment `
  -ApplicationId The_application_ID
  -Description Description_of_the_environment `
  -Monitors
 @{"AlarmArn=ARN_of_the_Amazon_CloudWatch_alarm,AlarmArnRole=ARN_of_the_IAM
 role_for_AWS AppConfig_to_monitor_AlarmArn"} `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_environment
```

系统将返回类似于以下内容的信息。

Linux

```
{
   "ApplicationId": "The application ID",
   "Id": "The_environment ID",
   "Name": "Name of the environment",
   "State": "The state of the environment",
   "Description": "Description of the environment",

   "Monitors": [
      {
         "AlarmArn": "ARN of the Amazon CloudWatch alarm",
         "AlarmRoleArn": "ARN of the IAM role for AppConfig to monitor AlarmArn"
      }
   ]
}
```

Windows

```
{
   "ApplicationId": "The application ID",
   "Id": "The environment ID",
   "Name": "Name of the environment",
   "State": "The state of the environment"
   "Description": "Description of the environment",

   "Monitors": [
      {
```

```
            "AlarmArn": "ARN of the Amazon CloudWatch alarm",
            "AlarmRoleArn": "ARN of the IAM role for AppConfig to monitor AlarmArn"
        }
    ]
}
```

PowerShell

```
ApplicationId     : The application ID
ContentLength     : Runtime of the command
Description       : Description of the environment
HttpStatusCode    : HTTP Status of the runtime
Id                : The environment ID
Monitors          : {ARN of the Amazon CloudWatch alarm, ARN of the IAM role for
 AppConfig to monitor AlarmArn}
Name              : Name of the environment
Response Metadata : Runtime Metadata
State             : State of the environment
```

继续执行在 AWS AppConfig中创建配置文件。

在 AWS AppConfig中创建配置文件

除其他外，配置文件包括一个 URI（ AWS AppConfig 允许在存储位置查找配置数据）和配置类型。AWS AppConfig 支持两种类型的配置文件：功能标志和自由格式配置。功能标志配置将数据存储在 AWS AppConfig 托管配置存储中，URI 很简单`hosted`。自由格式配置可以将数据存储在 AWS AppConfig 托管配置存储区、各种 Systems Manager 功能或与集成的 AWS 服务中。 AWS AppConfig 有关更多信息，请参阅 在中创建自由表单配置文件 AWS AppConfig。

配置文件还可以包括可选的验证器，以确保您的配置数据在语法和语义上都是正确的。 AWS AppConfig 开始部署时使用验证器执行检查。在对配置目标进行任何更改之前，如果检测到任何错误，部署将会停止。

> **Note**
>
> 如果可能，我们建议将您的配置数据托管在 AWS AppConfig 托管配置存储中，因为它提供了最多的功能和增强功能。

主题

关于验证程序

创建配置文件时，最多可以指定两个验证程序。验证程序可确保您的配置数据在语法和语义上是正确的。如果您计划使用验证器，则必须在创建配置文件之前创建验证器。 AWS AppConfig 支持以下类型的验证器：

- AWS Lambda 函数：支持功能标志和自由格式配置。
- JSON 架构：支持自由表单配置。（ AWS AppConfig 根据 JSON 架构自动验证功能标志。）

主题

AWS Lambda 函数验证程序

Lambda 函数验证程序必须配置了以下事件架构。 AWS AppConfig 使用该架构调用 Lambda 函数。content 是 base64 编码的字符串，而 URI 是字符串。

```
{
    "applicationId": "The application ID of the configuration profile being
 validated",
    "configurationProfileId": "The ID of the configuration profile being validated",
    "configurationVersion": "The version of the configuration profile being validated",
    "content": "Base64EncodedByteString",
    "uri": "The configuration uri"
}
```

AWS AppConfig 验证响应中是否设置了 `X-Amz-Function-Error` Lambda 标头。如果函数抛出异常，Lambda 会设置该标头。有关 `X-Amz-Function-Error` 的详细信息，请参阅《AWS Lambda 开发人员指南》中的 AWS Lambda中的错误处理和自动重试。

这里是一个成功验证 Lambda 响应代码的简单示例。

```
import json

def handler(event, context):
     #Add your validation logic here
     print("We passed!")
```

这里是一个不成功验证 Lambda 响应代码的简单示例。

```
def handler(event, context):
     #Add your validation logic here
     raise Exception("Failure!")
```

下面是另一个示例，仅在配置参数是质数时才验证。

```
function isPrime(value) {
    if (value < 2) {
        return false;
    }

    for (i = 2; i < value; i++) {
        if (value % i === 0) {
            return false;
        }
    }

    return true;
}

exports.handler = async function(event, context) {
    console.log('EVENT: ' + JSON.stringify(event, null, 2));
    const input = parseInt(Buffer.from(event.content, 'base64').toString('ascii'));
    const prime = isPrime(input);
    console.log('RESULT: ' + input + (prime ? ' is' : ' is not') + ' prime');
    if (!prime) {
        throw input + "is not prime";
    }
}
```

AWS AppConfig 在调用`StartDeployment`和 `ValidateConfigurationActivity` API 操作时，会调用您的验证 Lambda。您必须提供 `appconfig.amazonaws.com` 权限才能调用 Lambda。有关

更多信息，请参阅向 AWS 服务授予函数访问权限。 AWS AppConfig 将验证 Lambda 运行时间限制为 15 秒，包括启动延迟。

JSON 架构验证程序

如果在 SSM 文档中创建配置，则必须为该配置指定或创建 JSON 架构。JSON 架构定义每个应用程序配置设置允许的属性。JSON 架构的作用类似于一组规则，用于确保新配置设置或更新的配置设置符合应用程序所需的最佳实践。下面是一个例子。

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "$id$",
  "description": "BasicFeatureToggle-1",
  "type": "object",
  "additionalProperties": false,
  "patternProperties": {
      "[^\\s]+$": {
          "type": "boolean"
      }
  },
  "minProperties": 1
}
```

当您从 SSM 文档创建配置时，系统会自动验证配置是否符合架构要求。如果不符合要求， AWS AppConfig 将返回验证错误。

Important

请注意有关 JSON 架构验证程序的以下重要信息：

- 在可以将配置添加到系统之前，SSM 文档中存储的配置数据必须对照关联的 JSON 架构进行验证。SSM 参数不需要验证方法，但我们建议您使用为新的或更新的 SSM 参数配置创建验证检查。 AWS Lambda
- SSM 文档中的配置使用 `ApplicationConfiguration` 文档类型。相应的 JSON 架构使用 `ApplicationConfigurationSchema` 文档类型。
- AWS AppConfig 支持 JSON 架构 4.X 版本的内联架构。如果您的应用程序配置需要不同的 JSON 架构版本，则必须创建 Lambda 验证程序。

在中创建功能标志配置文件 AWS AppConfig

您可以使用功能标志来启用或禁用应用程序中的功能，也可以使用标志属性配置应用程序功能的不同特性。 AWS AppConfig 以功能标志格式将功能标志配置存储在 AWS AppConfig 托管配置存储中，该格式包含有关您的标志和旗帜属性的数据和元数据。有关 AWS AppConfig 托管配置存储的更多信息，请参阅关于 AWS AppConfig 托管配置存储第节。

主题

- 创建功能标志配置文件（控制台）
- 创建功能标志和功能标志配置文件（命令行）
- AWS.AppConfig.FeatureFlags 的类型引用

开始前的准备工作

在以下步骤中，在可选的加密部分中，您可以选择 AWS Key Management Service (AWS KMS) 密钥。此客户托管密钥使您能够加密 AWS AppConfig 托管配置存储库中的新配置数据版本。有关此密钥的更多信息，请参阅中的AWS AppConfig 支持客户经理密钥AWS AppConfig 中的安全性。

以下过程还为您提供了将扩展与功能标志配置文件关联的选项。在创建或部署配置 AWS AppConfig 的工作流程中，扩展可以增强您在不同时刻注入逻辑或行为的能力。有关更多信息，请参阅 关于 AWS AppConfig 扩展。

最后，在功能标志属性部分，当您输入新功能标志的属性详细信息时，可以指定约束条件。约束可确保不会将任何意外的属性值部署到您的应用程序中。 AWS AppConfig 支持以下类型的标志属性及其相应的约束。

类型	约束	描述
String	正则表达式	字符串的正则表达式模式
	枚举	字符串的可接受值列表
数字	最低	属性的最小数值
	最高	属性的最大数值
布尔值	无	无
字符串数组	正则表达式	数组元素的正则表达式模式

类型	约束	描述
	枚举	数组元素的可接受值列表
数字数组	最低	数组元素的最小数值
	最高	数组元素的最大数值

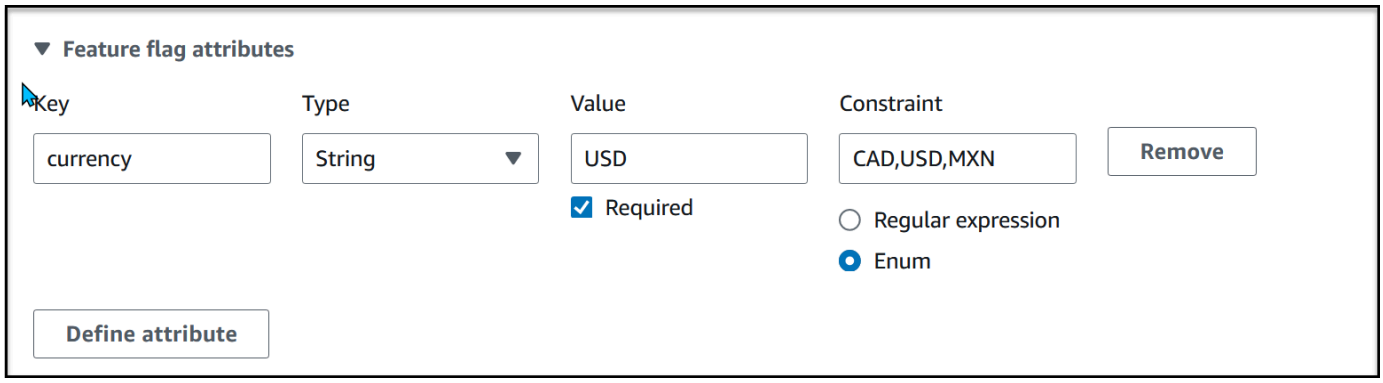
创建功能标志配置文件（控制台）

使用以下步骤使用 AWS AppConfig 控制台创建 AWS AppConfig 功能标志配置文件。

创建配置文件

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，选择应用程序，然后选择您在中创建的应用程序在 AWS AppConfig中为应用程序创建命名空间。
3. 选择配置文件和功能标志选项卡，然后选择创建配置。
4. 在配置选项部分中，选择功能标志。
5. 向下滚动。在配置配置文件部分中，为配置配置文件名称输入一个名称。
6. （可选）展开描述并输入描述。
7. （可选）展开其他选项并根据需要完成以下操作。
   a. 在加密列表中，从列表中选择一个 AWS Key Management Service (AWS KMS) 密钥。
   b. 在 “关联分机” 部分中，从列表中选择一个分机。
   c. 在 “标签” 部分中，选择 “添加新标签”，然后指定键和可选值。
8. 选择下一步。
9. 在 “功能标志定义” 部分中，为旗帜名称输入一个名称。
10. 对于 Fla g key，输入一个标志标识符以区分同一配置文件中的标志。同一配置文件中的标志不能具有相同的密钥。创建标志后，可以编辑标志名称，但不能编辑标志键。
11. （可选）展开描述并输入有关此标志的信息。
12. 选择 “这是短期标志”，也可以选择应禁用或删除该标志的日期。请注意，这 AWS AppConfig 不会禁用该标志。

13. 在 “旗帜属性” 部分中，选择定义属性。属性使您能够在标志中提供其他值。
14. 对于 Key，指定一个标志键并从 “类型” 列表中选择其类型。您可以选择根据指定的约束来验证属性值。下图显示了一个示例。



选择 “定义属性” 以添加其他属性。

> **Note**
>
> 请注意以下信息。
>
> - 对于属性名称，保留了“启用”一词。无法创建名为“已启用”的功能标志属性。没有其他保留字。
> - 仅当启用了功能标志时，该标志的属性才包含在 `GetLatestConfiguration` 响应中。
> - 给定标志的标志属性键必须是唯一的。
> - 选择必需值以指定属性值是否为必需值。

15. 在 “功能标志值” 部分，选择 “启用” 以启用该标志。如果适用，使用同样的开关在标志到达指定的弃用日期时将其禁用。
16. 选择下一步。
17. 在 “查看并保存” 页面上，验证标志的详细信息，然后单击 “保存” 并继续部署。

继续执行在 AWS AppConfig中部署功能标志和配置数据。

创建功能标志和功能标志配置文件（命令行）

以下过程介绍如何使用 AWS Command Line Interface （在 Linux 或 Windows 上）或适用于 Windows 的工具 PowerShell 来创建 AWS AppConfig 功能标志配置文件。如果您愿意，可以使用 AWS

CloudShell 运行下面列出的命令。有关更多信息，请参阅AWS CloudShell 《用户指南》中的什么是 AWS CloudShell？。

分步创建特征标志配置

1. 打开 AWS CLI.
2. 创建一个功能标志配置文件，将其类型指定为 `AWS.AppConfig.FeatureFlags`。配置文件必须使用 `hosted` 作为位置 URI。

   Linux

   ```
   aws appconfig create-configuration-profile \
     --application-id The_application_ID \
     --name A_name_for_the_configuration_profile \
     --location-uri hosted \
     --type AWS.AppConfig.FeatureFlags
   ```

   Windows

   ```
   aws appconfig create-configuration-profile ^
     --application-id The_application_ID ^
     --name A_name_for_the_configuration_profile ^
     --location-uri hosted ^
     --type AWS.AppConfig.FeatureFlags
   ```

   PowerShell

   ```
   New-APPCConfigurationProfile `
     -Name A_name_for_the_configuration_profile `
     -ApplicationId The_application_ID `
     -LocationUri hosted `
     -Type AWS.AppConfig.FeatureFlags
   ```

3. 创建功能标志配置数据。您的数据必须采用 JSON 格式，并符合 `AWS.AppConfig.FeatureFlags` JSON 架构。有关架构的更多信息，请参阅 AWS.AppConfig.FeatureFlags 的类型引用。
4. 使用 `CreateHostedConfigurationVersion` API 将功能标志配置数据保存到 AWS AppConfig。

Linux

```
aws appconfig create-hosted-configuration-version \
  --application-id The_application_ID \
  --configuration-profile-id The_configuration_profile_id \
  --content-type "application/json" \
  --content file://path/to/feature_flag_configuration_data \
  file_name_for_system_to_store_configuration_data
```

Windows

```
aws appconfig create-hosted-configuration-version ^
  --application-id The_application_ID ^
  --configuration-profile-id The_configuration_profile_id ^
  --content-type "application/json" ^
  --content file://path/to/feature_flag_configuration_data ^
  file_name_for_system_to_store_configuration_data
```

PowerShell

```
New-APPCHostedConfigurationVersion `
  -ApplicationId The_application_ID `
  -ConfigurationProfileId The_configuration_profile_id `
  -ContentType "application/json" `
  -Content file://path/to/feature_flag_configuration_data `
  file_name_for_system_to_store_configuration_data
```

下面是一个 Linux 示例命令。

```
aws appconfig create-hosted-configuration-version \
  --application-id 1a2b3cTestApp \
  --configuration-profile-id 4d5e6fTestConfigProfile \
  --content-type "application/json" \
  --content Base64Content
```

content 参数使用以下 base64 编码数据。

```
{
  "flags": {
```

```
    "flagkey": {
      "name": "WinterSpecialBanner"
    }
  },
  "values": {
    "flagkey": {
      "enabled": true
    }
  },
  "version": "1"
}
```

系统将返回类似于以下内容的信息。

Linux

```
{
   "ApplicationId"          : "1a2b3cTestApp",
   "ConfigurationProfileId" : "4d5e6fTestConfigProfile",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

Windows

```
{
   "ApplicationId"          : "1a2b3cTestApp",
   "ConfigurationProfileId" : "4d5e6fTestConfigProfile",
   "VersionNumber"          : "1",
   "ContentType"            : "application/json"
}
```

PowerShell

```
ApplicationId          : 1a2b3cTestApp
ConfigurationProfileId : 4d5e6fTestConfigProfile
VersionNumber          : 1
ContentType            : application/json
```

service_returned_content_file包含您的配置数据，其中包括一些 AWS AppConfig 生成的元数据。

> **Note**
>
> 创建托管配置版本时，请 AWS AppConfig 验证您的数据是否符合 AWS.AppConfig.FeatureFlags JSON 架构。 AWS AppConfig 此外，还会验证数据中的每个要素标志属性是否满足您为这些属性定义的约束条件。

AWS.AppConfig.FeatureFlags 的类型引用

使用 AWS.AppConfig.FeatureFlags JSON 架构作为参考来创建功能标志配置数据。

```
{
      "$schema": "http://json-schema.org/draft-07/schema#",
      "definitions": {
        "flagSetDefinition": {
          "type": "object",
          "properties": {
            "version": {
              "$ref": "#/definitions/flagSchemaVersions"
            },
            "flags": {
              "$ref": "#/definitions/flagDefinitions"
            },
            "values": {
              "$ref": "#/definitions/flagValues"
            }
          },
          "required": ["version", "flags"],
          "additionalProperties": false
        },
        "flagDefinitions": {
          "type": "object",
          "patternProperties": {
            "^[a-z][a-zA-Z\\d-]{0,63}$": {
              "$ref": "#/definitions/flagDefinition"
            }
          },
```

```
      "maxProperties": 100,
      "additionalProperties": false
    },
    "flagDefinition": {
      "type": "object",
      "properties": {
        "name": {
          "$ref": "#/definitions/customerDefinedName"
        },
        "description": {
          "$ref": "#/definitions/customerDefinedDescription"
        },
        "_createdAt": {
          "type": "string"
        },
        "_updatedAt": {
          "type": "string"
        },
        "_deprecation": {
          "type": "object",
          "properties": {
            "status": {
              "type": "string",
              "enum": ["planned"]
            }
          },
         "additionalProperties": false
        },
        "attributes": {
          "$ref": "#/definitions/attributeDefinitions"
        }
      },
      "additionalProperties": false
    },
    "attributeDefinitions": {
      "type": "object",
      "patternProperties": {
        "^[a-z][a-zA-Z\\d-_]{0,63}$": {
          "$ref": "#/definitions/attributeDefinition"
        }
      },
      "maxProperties": 25,
      "additionalProperties": false
    },
```

```
        "attributeDefinition": {
          "type": "object",
          "properties": {
            "description": {
              "$ref": "#/definitions/customerDefinedDescription"
            },
            "constraints": {
              "oneOf": [
                { "$ref": "#/definitions/numberConstraints" },
                { "$ref": "#/definitions/stringConstraints" },
                { "$ref": "#/definitions/arrayConstraints" },
                { "$ref": "#/definitions/boolConstraints" }
              ]
            }
          },
          "additionalProperties": false
        },
        "flagValues": {
          "type": "object",
          "patternProperties": {
            "^[a-z][a-zA-Z\\d-_]{0,63}$": {
              "$ref": "#/definitions/flagValue"
            }
          },
          "maxProperties": 100,
          "additionalProperties": false
        },
        "flagValue": {
          "type": "object",
          "properties": {
            "enabled": {
              "type": "boolean"
            },
            "_createdAt": {
              "type": "string"
            },
            "_updatedAt": {
              "type": "string"
            }
          },
          "patternProperties": {
            "^[a-z][a-zA-Z\\d-_]{0,63}$": {
              "$ref": "#/definitions/attributeValue",
              "maxProperties": 25
```

```
            }
          },
          "required": ["enabled"],
          "additionalProperties": false
        },
        "attributeValue": {
          "oneOf": [
            { "type": "string", "maxLength": 1024 },
            { "type": "number" },
            { "type": "boolean" },
            {
              "type": "array",
              "oneOf": [
                {
                  "items": {
                    "type": "string",
                    "maxLength": 1024
                  }
                },
                {
                  "items": {
                    "type": "number"
                  }
                }
              ]
            }
          ],
          "additionalProperties": false
        },
        "stringConstraints": {
          "type": "object",
          "properties": {
            "type": {
              "type": "string",
              "enum": ["string"]
      },
            "required": {
              "type": "boolean"
            },
            "pattern": {
              "type": "string",
              "maxLength": 1024
            },
            "enum": {
```

```
            "type": "array",
            "maxLength": 100,
            "items": {
              "oneOf": [
                {
                  "type": "string",
                  "maxLength": 1024
                },
                {
                  "type": "integer"
                }
              ]
            }
          }
        },
        "required": ["type"],
        "not": {
          "required": ["pattern", "enum"]
        },
        "additionalProperties": false
      },
      "numberConstraints": {
        "type": "object",
        "properties": {
          "type": {
            "type": "string",
            "enum": ["number"]
    },
          "required": {
            "type": "boolean"
          },
          "minimum": {
            "type": "integer"
          },
          "maximum": {
            "type": "integer"
          }
        },
        "required": ["type"],
        "additionalProperties": false
      },
      "arrayConstraints": {
        "type": "object",
        "properties": {
```

```
            "type": {
              "type": "string",
              "enum": ["array"]
      },
            "required": {
              "type": "boolean"
            },
            "elements": {
              "$ref": "#/definitions/elementConstraints"
      }
          },
          "required": ["type"],
          "additionalProperties": false
        },
        "boolConstraints": {
          "type": "object",
          "properties": {
            "type": {
              "type": "string",
              "enum": ["boolean"]
      },
            "required": {
              "type": "boolean"
            }
          },
          "required": ["type"],
          "additionalProperties": false
        },
        "elementConstraints": {
          "oneOf": [
            { "$ref": "#/definitions/numberConstraints" },
            { "$ref": "#/definitions/stringConstraints" }
          ]
        },
        "customerDefinedName": {
          "type": "string",
          "pattern": "^[^\\n]{1,64}$"
        },
        "customerDefinedDescription": {
          "type": "string",
          "maxLength": 1024
        },
        "flagSchemaVersions": {
          "type": "string",
```

```
          "enum": ["1"]
        }
      },
      "type": "object",
      "$ref": "#/definitions/flagSetDefinition",
      "additionalProperties": false
    }
```

> **Important**
>
> 若要检索功能标志配置数据，应用程序必须调用 GetLatestConfiguration API。无法通过调用 GetConfiguration 来检索功能标志配置数据，该数据已弃用。有关更多信息，请参阅 AWS AppConfig API 参考中的GetLatest配置。

当您的应用程序调用 GetLatestCon figuration 并收到新部署的配置时，定义您的功能标志和属性的信息将被删除。简化的 JSON 包含与您指定的每个标志键匹配的键映射。简化的 JSON 还包含 enabled 属性的 true 或 false 映射值。如果标志将 enabled 设置为 true，则该标志的任何属性也将存在。以下 JSON 架构描述了 JSON 输出的格式。

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "patternProperties": {
    "^[a-z][a-zA-Z\\d-_]{0,63}$": {
      "$ref": "#/definitions/attributeValuesMap"
    }
  },
  "maxProperties": 100,
  "additionalProperties": false,
  "definitions": {
    "attributeValuesMap": {
      "type": "object",
      "properties": {
        "enabled": {
          "type": "boolean"
        }
      },
      "required": ["enabled"],
      "patternProperties": {
        "^[a-z][a-zA-Z\\d-_]{0,63}$": {
          "$ref": "#/definitions/attributeValue"
```

```
        }
      },
      "maxProperties": 25,
      "additionalProperties": false
    },
    "attributeValue": {
      "oneOf": [
        { "type": "string","maxLength": 1024 },
        { "type": "number" },
        { "type": "boolean" },
        {
          "type": "array",
          "oneOf": [
            {
              "items": {
                "oneOf": [
                  {
                    "type": "string",
                    "maxLength": 1024
                  }
                ]
              }
            },
            {
              "items": {
                "oneOf": [
                  {
                    "type": "number"
                  }
                ]
              }
            }
          ]
        }
      ],
      "additionalProperties": false
    }
  }
}
```

在中创建自由表单配置文件 AWS AppConfig

除其他外，配置文件包括一个 AWS AppConfig 允许在其存储位置查找配置数据的 URI 和配置文件类型。 AWS AppConfig 支持两种配置文件类型：功能标志和自由格式配置。功能标志配置文件将其数据存储在 AWS AppConfig 托管配置存储中，URI 很简单`hosted`。对于自由格式的配置文件，您可以将数据存储在 AWS AppConfig 托管配置存储区或以下任何 AWS 服务和 Systems Manager 功能中：

位置	支持的文件类型
AWS AppConfig 托管配置存储	YAML、JSON 和文本（如果使用添加）。 AWS Management Console任何文件类型（如果使用 AWS AppConfig CreateHostedConfigurationVersionAPI 操作添加）。
Amazon Simple Storage Service (Amazon S3)	任何
AWS CodePipeline	管道（由服务定义）
AWS Secrets Manager	密钥（由服务定义）
AWS Systems Manager 参数存储	标准和安全字符串参数（由参数存储定义）
AWS Systems Manager 文档存储（SSM 文档）	YAML、JSON、文本

配置文件还可以包括可选的验证器，以确保您的配置数据在语法和语义上都是正确的。 AWS AppConfig 开始部署时使用验证器执行检查。在对配置目标进行任何更改之前，如果检测到任何错误，部署将会停止。

> **Note**
>
> 如果可能，我们建议将您的配置数据托管在 AWS AppConfig 托管配置存储中，因为它提供了最多的功能和增强功能。

对于存储在 AWS AppConfig 托管配置存储或 SSM 文档中的自由格式配置，您可以在创建配置文件时使用 Systems Manager 控制台创建自由格式配置。本主题稍后将介绍该过程。

对于存储在 Parameter Store、Secrets Manager 或 Amazon S3 中的自由格式配置，您必须先创建参数、密钥或对象，并将其存储在相关配置存储中。存储配置数据后，请使用本主题中的过程创建配置文件。

主题

关于配置存储配额和限制

支持的配置存储 AWS AppConfig 具有以下配额和限制。

	AWS AppConfig 托管配置存储	Amazon S3	Systems Manager Parameter Store	AWS Secrets Manager	Systems Manager 文档存储	AWS CodePipeline
配置大小限制	默认 2 MB，最大 4 MB	2 MB 由 S3 强制执行 AWS AppConfig，而不是 S3	4 KB（免费套餐）/8 KB（高级参数）	64 KB	64 KB	2 MB 由强制执行 AWS AppConfig，而不是 CodePipeline
资源存储限制	1 GB	无限制	10000 个参数（免费套餐）/100000 个参数（高级参数）	500,000	500 个文档	受每个应用程序配置文件数量限制（每个应用程序 100 个配置文件）

	AWS AppConfig 托管配置存储	Amazon S3	Systems Manager Parameter Store	AWS Secrets Manager	Systems Manager 文档存储	AWS CodePipeline
服务器端加密	是	SSE-S3、SSE-KMS	是	是	否	是
AWS CloudFormation 支持	是	不用于创建或更新数据	是	是	否	是
定价	免费	请参阅 Amazon S3 定价	请参阅 AWS Systems Manager 定价	请参阅 AWS Secrets Manager 定价	免费	请参阅 AWS CodePipeline 定价

关于 AWS AppConfig 托管配置存储

AWS AppConfig 包括内部或托管配置存储。配置必须为 2 MB 或更小。与其他配置存储选项相比，AWS AppConfig 托管配置存储具有以下优势。

- 您无需设置和配置其他服务，如 Amazon Simple Storage Service (Amazon S3) 或参数存储。
- 您无需配置 AWS Identity and Access Management (IAM) 权限即可使用配置存储。
- 您可以按 YAML、JSON 或文本文档格式存储配置。
- 使用存储不产生任何费用。
- 您可以创建配置并在创建配置文件时将其添加到存储。

关于存储在 Amazon S3 中的配置

您可以将配置存储在 Amazon Simple Storage Service (Amazon S3) 存储桶中。在创建配置文件时，将指定存储桶中单个 S3 对象的 URI。您还可以指定授予获取对象 AWS AppConfig 权限的 (IAM) 角色的 Amazon 资源名称 AWS Identity and Access Management (ARN)。在为 Amazon S3 对象创建配置文件之前，请注意以下限制。

限制	详细信息
大小	存储为 S3 对象的配置的最大大小可以为 1 MB。
Object encryption	配置文件可以 SSE-S3 和 SSE-KMS 加密对象为目标。
存储类	AWS AppConfig 支持以下 S3 存储类别：STANDARDINTELLIGENT_TIERING 、REDUCED_REDUNDANCY 、STANDARD_IA 、和ONEZONE_IA 。不支持以下类别：所有 S3 Glacier 类别（GLACIER 和 DEEP_ARCHIVE ）。
版本控制	AWS AppConfig 要求 S3 对象使用版本控制。

配置存储为 Amazon S3 对象的配置的权限

在为存储为 S3 对象的配置创建配置文件时，必须为授予获取对象 AWS AppConfig 权限的 IAM 角色指定 ARN。该角色必须包括以下权限。

对 S3 对象的访问权限

- s3：GetObject
- s3：GetObject版本

列出 S3 存储桶的权限

s3：ListAllMyBuckets

对用于存储对象的 S3 存储桶的访问权限

- s3：GetBucket位置
- s3：GetBucket版本控制
- s3：ListBucket
- s3：ListBucket版本

完成以下过程以创建允许 AWS AppConfig 获取存储在 S3 对象中的配置的角色。

创建用于访问 S3 对象的 IAM 策略

使用以下过程创建 IAM 策略，该策略 AWS AppConfig 允许获取存储在 S3 对象中的配置。

创建用于访问 S3 对象的 IAM 策略

1. 访问：https://console.aws.amazon.com/iam/，打开 IAM 控制台。
2. 在导航窗格中，选择 Policies (策略)，然后选择 Create policy (创建策略)。
3. 在创建策略页面上，选择 JSON 选项卡。
4. 使用有关 S3 存储桶和配置对象的信息更新以下示例策略。然后将策略粘贴到 JSON 选项卡上的文本字段中。用您自己的信息替换*####*。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/my-configurations/my-
configuration.json"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetBucketVersioning",
        "s3:ListBucketVersions",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "s3:ListAllMyBuckets",
      "Resource": "*"
```

```
    }
  ]
}
```

5. 选择查看策略。
6. 在 Review policy (查看策略) 页面上，在 Name (名称) 框中键入名称，然后键入描述。
7. 选择 创建策略。系统将让您返回到 角色 页面。

创建用于访问 S3 对象的 IAM 角色

使用以下过程创建一个 IAM 角色，该角色 AWS AppConfig 允许获取存储在 S3 对象中的配置。

创建一个用于访问 Amazon S3 的 IAM 角色

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择角色，然后选择创建角色。
3. 在 选择受信任实体的类型 部分中，选择 AWS 服务。
4. 在 Choose a use case (选择使用案例) 部分中，在 Common use cases (常见使用案例) 下，选择 EC2，然后选择 Next: Permissions (下一步: 权限)。
5. 在 Attach permissions policy (附加权限策略) 页面上的搜索框中，输入您在上一过程中创建的策略的名称。
6. 选择该策略，然后选择 Next: Tags (下一步: 标签)。
7. 在 添加标签(可选) 页面上，输入密钥和可选值，然后选择 下一步: 审核。
8. 在 Review (审核) 页面上，在 Role name (角色名称) 字段中键入名称，然后键入描述。
9. 选择 Create role (创建角色)。系统将让您返回到 角色 页面。
10. 在角色页面中，选择刚刚创建的角色以打开摘要页面。记下角色名称和角色 ARN。在本主题的后面部分中创建配置文件时，您将指定角色 ARN。

创建信任关系

使用以下过程将您刚刚创建的角色配置为信任 AWS AppConfig。

添加信任关系

1. 在刚刚创建的角色的摘要页面上，选择信任关系选项卡，然后选择编辑信任关系。
2. 删除 `"ec2.amazonaws.com"` 并添加 `"appconfig.amazonaws.com"`，如以下示例所示。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

3. 选择更新信任策略。

创建自由格式配置和配置文件

本节介绍如何创建自由格式配置和配置文件。开始之前，请注意以下信息。

- 以下过程要求您指定 IAM 服务角色，以便 AWS AppConfig 能够访问您选择的配置存储中的配置数据。如果您使用 AWS AppConfig 托管配置存储，则不需要此角色。如果您选择 S3、参数存储或 Systems Manager 文档存储，则必须选择现有 IAM 角色，或选择让系统自动为您创建角色的选项。有关该角色的更多信息，请参阅 关于配置文件 IAM 角色。
- 以下过程还为您提供了将扩展与功能标志配置文件关联的选项。在创建或部署配置 AWS AppConfig 的工作流程中，扩展可以增强您在不同时刻注入逻辑或行为的能力。有关更多信息，请参阅 关于 AWS AppConfig 扩展。
- 如果要为 S3 中存储的配置创建配置文件，则必须配置权限。有关使用 S3 作为配置存储的权限和其他要求的更多信息，请参阅 关于存储在 Amazon S3 中的配置。
- 如果您要使用验证程序，请查看使用验证程序的详细信息和要求。有关更多信息，请参阅 关于验证程序。

主题

- 创建 AWS AppConfig 自由格式配置文件（控制台）
- 创建 AWS AppConfig 自由格式配置文件（命令行）

创建 AWS AppConfig 自由格式配置文件（控制台）

使用以下步骤使用控制台创建 AWS AppConfig 自由格式配置文件和（可选）自由格式配置。 AWS Systems Manager

创建自由格式配置文件

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，选择应用程序，然后选择您在中创建的应用程序在 AWS AppConfig中为应用程序创建命名空间。
3. 选择配置文件和功能标志选项卡，然后选择创建配置。
4. 在配置选项部分，选择自由格式配置。
5. 在配置配置文件名称中，输入配置文件的名称。
6. （可选）展开描述并输入描述。
7. （可选）展开其他选项并根据需要完成以下操作。
   a. 在 “关联分机” 部分中，从列表中选择一个分机。
   b. 在 “标签” 部分中，选择 “添加新标签”，然后指定键和可选值。
8. 选择下一步。
9. 在 “指定配置数据” 页面的 “配置定义” 部分，选择一个选项。
10. 填写所选选项的字段，如下表所述。

已选择选项	详细信息
AWS AppConfig 托管配置	选择文本、JSON 或 YAML，然后在字段中输入您的配置。转到此过程中的步骤 12。
亚马逊 S3 对象	在 S3 对象源字段中输入对象 URI，然后转到此过程中的步骤 11。
AWS CodePipeline	选择 “下一步”，然后转到此过程中的步骤 12。
Secrets Manager 秘密	从列表中选择密钥，转到此过程中的步骤 11。

已选择选项	详细信息
AWS Systems Manager 参数	从列表中选择参数，然后转到此过程中的步骤 11。
AWS Systems Manager 文档	1. 从列表中选择一个文档或选择 “创建新文档”。 2. 如果选择 “创建新文档”，请在 “文档名称” 中输入一个名称。（可选）展开版本名称并输入文档版本的名称。 3. 对于应用程序配置架构，请从列表中选择 JSON 架构或选择创建架构。如果选择 创建架构，将打开创建架构 页面。输入架构详细信息，然后选择创建应用程序配置架构。 4. 在 Content (内容) 部分中，选择 YAML 或 JSON，然后在字段中输入配置数据。

11. 在服务角色部分中，选择新建服务角色以 AWS AppConfig 创建提供配置数据访问权限的 IAM 角色。 AWS AppConfig 根据您之前输入的名称自动填充角色名称字段。或者，选择现有服务角色。使用 Role ARN (角色 ARN) 列表选择角色。
12. 或者，在添加验证器页面上，选择 JSON 架构或。AWS Lambda如果选择 JSON Schema (JSON 架构)，请在字段中输入 JSON 架构。如果您选择 AWS Lambda，请从列表中选择函数 Amazon 资源名称 (ARN) 和版本。

> **Important**
>
> 在可以将配置添加到系统之前，SSM 文档中存储的配置数据必须对照关联的 JSON 架构进行验证。SSM 参数不需要验证方法，但我们建议您使用为新的或更新的 SSM 参数配置创建验证检查。 AWS Lambda

13. 选择下一步。
14. 在 “查看并保存” 页面上，选择 “保存” 并继续部署。

> **Important**
>
> 如果您为创建了配置文件 AWS CodePipeline，则必须在中创建指定 CodePipeline AWS AppConfig 为部署提供者的管道。您不需要执行 在 AWS AppConfig中部署功能标志和配置数据。但是，您必须配置客户端以接收应用程序配置更新，如 通过直接调用 API 获取配置 中所述。有关创建指定 AWS AppConfig 为部署提供者的管道的信息，请参阅《AWS CodePipeline 用户指南》中的教程：创建 AWS AppConfig 用作部署提供者的管道。

继续执行在 AWS AppConfig中部署功能标志和配置数据。

创建 AWS AppConfig 自由格式配置文件（命令行）

以下过程介绍如何使用 AWS CLI（在 Linux 或 Windows 上）或 AWS Tools for PowerShell 如何创建 AWS AppConfig 自由格式配置文件。如果你愿意，你可以 AWS CloudShell 使用运行下面列出的命令。有关更多信息，请参阅AWS CloudShell 《用户指南》中的什么是 AWS CloudShell？。

> **Note**
>
> 对于托管在托管的配置存储中 AWS AppConfig 托管的自由格式配置，您可以指定`hosted`位置 URI。

要创建配置文件，请使用 AWS CLI

1. 打开 AWS CLI.
2. 运行以下命令以创建自由格式配置文件。

   Linux

```
aws appconfig create-configuration-profile \
  --application-id The_application_ID \
  --name A_name_for_the_configuration_profile \
  --description A_description_of_the_configuration_profile \
  --location-uri A_URI_to_locate_the_configuration or hosted \
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified
 \
  --tags User_defined_key_value_pair_metadata_of_the_configuration_profile \
```

```
  --validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

Windows

```
aws appconfig create-configuration-profile ^
  --application-id The_application_ID ^
  --name A_name_for_the_configuration_profile ^
  --description A_description_of_the_configuration_profile ^
  --location-uri A_URI_to_locate_the_configuration or hosted  ^
  --retrieval-role-
arn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_the_specified
 ^
  --tags User_defined_key_value_pair_metadata_of_the_configuration_profile ^
  --validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

PowerShell

```
New-APPCConfigurationProfile `
  -Name A_name_for_the_configuration_profile `
  -ApplicationId The_application_ID `
  -Description Description_of_the_configuration_profile `
  -LocationUri A_URI_to_locate_the_configuration or hosted `
  -
RetrievalRoleArn The_ARN_of_the_IAM_role_with_permission_to_access_the_configuration_at_
 `
  -
Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_configuration_profile
 `
  -Validators "Content=JSON_Schema_content_or_the_ARN_of_an_AWS
 Lambda_function,Type=JSON_SCHEMA or LAMBDA"
```

Important

请注意以下重要信息。

- 如果您为创建了配置文件 AWS CodePipeline，则必须在中创建指定 CodePipeline AWS AppConfig 为部署提供者的管道。您不需要执行 在 AWS AppConfig中部署功能标志和配置数据。但是，您必须配置客户端以接收应用程序配置更新，如 通过直接调用 API 获取

配置 中所述。有关创建指定 AWS AppConfig 为部署提供者的管道的信息，请参阅《AWS CodePipeline 用户指南》中的教程：创建 AWS AppConfig 用作部署提供者的管道。

- 如果您在 AWS AppConfig 托管配置存储中创建了配置，则可以使用 CreateHostedConfigurationVersionAPI 操作创建配置的新版本。要查看此 API 操作的 AWS CLI 详细信息和示例命令，请参阅《命令参考》中的 create-hosted-configuration-version。AWS CLI

继续执行在 AWS AppConfig中部署功能标志和配置数据。

其他配置数据源

本主题包括有关与集成的其他 AWS 服务的信息 AWS AppConfig。

AWS AppConfig 与集成 AWS Secrets Manager

Secrets Manager 帮助您安全地加密、存储和检索数据库和其他服务的凭证。您可以在需要时调用 Secrets Manager 以检索您的凭证，而不是在应用程序中对凭证进行硬编码。Secrets Manager 使您能够轮换和管理对密钥的访问，从而帮助您保护对 IT 资源和数据的访问。

当您创建自由格式配置文件时，可以选择 Secrets Manager 作为配置数据的来源。在创建配置文件之前，您必须使用 Secrets Manager 注册并创建密钥。有关 Secrets Manager 的更多信息，请参阅 Secrets Manager 是什么 AWS Secrets Manager？ 在《AWS Secrets Manager 用户指南》中。有关创建使用 Secrets Manager 的配置文件的信息，请参阅 在中创建功能标志和自由格式配置数据 AWS AppConfig。

在 AWS AppConfig中部署功能标志和配置数据

在创建所需构件以处理功能标志和自由格式配置数据后，就可以创建新的部署。当您创建新的配置时，需要指定以下信息：

- 应用程序 ID
- 配置文件 ID
- 配置版本
- 要在其中部署配置数据的环境 ID
- 部署策略 ID，可定义更改生效的速度
- AWS Key Management Service (AWS KMS) 密钥 ID，用于使用客户管理的密钥对数据进行加密。

当您调用 StartDeploymentAPI 操作时， AWS AppConfig 会执行以下任务：

1. 使用配置文件中的位置 URI 从底层数据存储中读取配置数据。
2. 使用在创建配置文件时指定的验证程序，验证配置数据在语法和语义上是否正确。
3. 缓存数据副本，以便应用程序随时检索。此缓存副本被称为已部署数据。

AWS AppConfig 与 Amazon 集成 CloudWatch 以监控部署。如果部署触发了警报 CloudWatch，则 AWS AppConfig 会自动回滚部署以最大限度地减少对应用程序用户的影响。

主题

- 使用部署策略
- 部署配置
- AWS AppConfig 部署与集成 CodePipeline

使用部署策略

部署策略使您能够在几分钟或几小时内缓慢地将更改发布到生产环境。 AWS AppConfig 部署策略定义了配置部署的以下重要方面。

<table>
<tr><th>设置</th><th>描述</th></tr>
<tr><td>Deployment type（部署类型）</td><td>部署类型定义了配置的部署或部署方式。 AWS AppConfig 支持线性和指数部署类型。

• 线性：对于这种类型，按在部署中均匀分布的增长因子的增量来 AWS AppConfig 处理部署。下面是一个 10 小时部署的时间表示例，采用 20% 的线性增长：

<table>
<tr><th>运行时间</th><th>部署进度</th></tr>
<tr><td>0 小时</td><td>0%</td></tr>
<tr><td>2 小时</td><td>20%</td></tr>
<tr><td>4 小时</td><td>40%</td></tr>
<tr><td>6 小时</td><td>60%</td></tr>
<tr><td>8 小时</td><td>80%</td></tr>
<tr><td>10 小时</td><td>100%</td></tr>
</table>
• 指数：对于此类型， AWS AppConfig 使用以下公式以指数方式处理部署：<code>G*(2^N)</code>。在此公式中，<code>G</code> 是用户指定的步骤百分比，<code>N</code> 是在配置部署到所有目标之前的步骤数。例如，如果将增长系数指定为 2，则系统将按如下方式推出配置：

<pre>2*(2^0)
2*(2^1)
2*(2^2)</pre>以数字表示，部署的推出情况如下：2% 的目标、4% 的目标、8% 的目标，并持续到将配置部署到所有目标为止。</td></tr>
</table>

设置	描述
步骤百分比（增长系数）	该设置指定在部署的每个步骤中作为目标的调用方百分比。 **Note** 在开发工具包和 AWS AppConfig API 参考中，`step percentage` 称为 `growth factor`。
Deployment time (部署时间)	此设置指定 AWS AppConfig 部署到主机的时间。这不是超时值。这是一个按间隔处理部署的时段。
Bake time (烘焙时间)	此设置指定在将配置部署到 100% 的目标之后，在考虑部署已完成之前， AWS AppConfig 监控 Amazon CloudWatch 警报的时间长度。如果在此期间触发了警报， AWS AppConfig 将回滚部署。您必须配置权限 AWS AppConfig 才能根据 CloudWatch 警报进行回滚。有关更多信息，请参阅（可选）根据警报配置回滚权限 CloudWatch。

您可以选择随附的预定义策略， AWS AppConfig 也可以创建自己的策略。

主题

- 预定义的部署策略
- 创建部署策略

预定义的部署策略

AWS AppConfig 包括预定义的部署策略，可帮助您快速部署配置。您可以在部署配置时选择以下选项之一，而不是创建自己的策略。

部署策略	描述
AppConfig.Linear20 6 分钟 PercentEvery	AWS 推荐： 此策略每 6 分钟将配置部署到所有目标的 20%，以进行 30 分钟部署。系统会监控 Amazon CloudWatch 警报 30 分钟。如果此时未收到任何警报，则部署已完成。如果在这段时间内触发了警报，则会回 AWS AppConfig 滚部署。 我们建议将此策略用于生产部署，因为它符合 AWS 最佳实践，并且由于其持续时间和烘焙时间长，因此更加重视部署安全。
AppConfig.Canary 10% 20 分钟	AWS 推荐： 此策略在 20 分钟内使用 10% 的增长系数以指数方式处理部署。系统将监视 CloudWatch 警报 10 分钟。如果此时未收到任何警报，则部署已完成。如果在这段时间内触发了警报，则会回 AWS AppConfig 滚部署。 我们建议将此策略用于生产部署，因为它符合配置部署 AWS 的最佳实践。
AppConfig.AllAtOnce	快速： 此策略会立即将配置部署到所有目标。系统将监视 CloudWatch 警报 10 分钟。如果此时未收到任何警报，则部署已完成。如果在此期间触发了警报， AWS AppConfig 将回滚部署。
AppConfig.Linear50 30Seconds PercentEvery	测试/演示： 此策略每 30 秒将配置部署到所有目标的一半，以进行一分钟部署。系统会监视 Amazon CloudWatch 警报 1 分钟。如果此时未收到任何

部署策略	描述
	警报，则部署已完成。如果在这段时间内触发了警报，则会回 AWS AppConfig 滚部署。 我们建议仅将此策略用于测试或演示目的，因为它持续时间和烘焙时间短。

创建部署策略

如果您不想使用其中一种预定义的部署策略，则可以创建自己的部署策略。您最多可以创建 20 个部署策略。在部署配置时，您可以选择最适合应用程序和环境的部署策略。

创建 AWS AppConfig 部署策略（控制台）

使用以下过程通过 AWS Systems Manager 控制台创建 AWS AppConfig 部署策略。

创建部署策略

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，选择部署策略，然后选择创建部署策略。
3. 对于 Name (名称)，请输入部署策略的名称。
4. 对于 Description (描述)，请输入有关部署策略的信息。
5. 对于 Deployment type (部署类型)，选择类型。
6. 对于 Step percentage (步骤百分比)，请选择在部署的每个步骤中作为目标的调用方百分比。
7. 对于 Deployment time (部署时间)，请输入部署的总持续时间（以分钟或小时为单位）。
8. 在 Bake time 中，输入在继续部署的下一步或考虑完成部署之前，监控 Amazon CloudWatch 警报的总时间（以分钟或小时为单位）。
9. 在 标签 部分中，输入一个键和可选的值。您最多可以为一个资源指定 50 个标签。
10. 选择 Create deployment strategy (创建部署策略)。

> **Important**
>
> 如果您为创建了配置文件 AWS CodePipeline，则必须在中创建指定 CodePipeline AWS AppConfig 为部署提供者的管道。您不需要执行 部署配置。但是，您必须配置客户端以接收

应用程序配置更新，如 通过直接调用 API 获取配置 中所述。有关创建指定 AWS AppConfig 为部署提供者的管道的信息，请参阅《AWS CodePipeline 用户指南》中的教程：创建 AWS AppConfig 用作部署提供者的管道。

继续执行部署配置。

创建 AWS AppConfig 部署策略（命令行）

以下过程介绍如何使用 AWS CLI（在 Linux 或 Windows 上）或 AWS Tools for PowerShell 如何创建 AWS AppConfig 部署策略。

分步创建部署策略

1. 打开 AWS CLI.
2. 运行以下命令，创建部署策略。

   Linux

```
aws appconfig create-deployment-strategy \
  --name A_name_for_the_deployment_strategy \
  --description A_description_of_the_deployment_strategy \
  --deployment-duration-in-minutes Total_amount_of_time_for_a_deployment_to_last
 \
  --final-bake-time-in-minutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 \
  --growth-
factor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_interva
 \
  --growth-
type The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time
 \
  --replicate-
to To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document \
  --tags User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

   Windows

```
aws appconfig create-deployment-strategy ^
  --name A_name_for_the_deployment_strategy ^
```

```
  --description A_description_of_the_deployment_strategy ^
  --deployment-duration-in-minutes Total_amount_of_time_for_a_deployment_to_last
 ^
  --final-bake-time-in-minutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 ^
  --growth-
factor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_interva
 ^
  --growth-
type The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_time
 ^
  --name A_name_for_the_deployment_strategy ^
  --replicate-
to To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document ^
  --tags User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

PowerShell

```
New-APPCDeploymentStrategy `
  --Name A_name_for_the_deployment_strategy `
  --Description A_description_of_the_deployment_strategy `
  --DeploymentDurationInMinutes Total_amount_of_time_for_a_deployment_to_last `
  --FinalBakeTimeInMinutes Amount_of_time_AWS
 AppConfig_monitors_for_alarms_before_considering_the_deployment_to_be_complete
 `
  --
GrowthFactor The_percentage_of_targets_to_receive_a_deployed_configuration_during_each_i
 `
  --
GrowthType The_linear_or_exponential_algorithm_used_to_define_how_percentage_grows_over_
 `
  --
ReplicateTo To_save_the_deployment_strategy_to_a_Systems_Manager_(SSM)_document
 `
  --
Tag Hashtable_type_User_defined_key_value_pair_metadata_of_the_deployment_strategy
```

系统将返回类似于以下内容的信息。

Linux

```
{
   "Id": "Id of the deployment strategy",
   "Name": "Name of the deployment strategy",
   "Description": "Description of the deployment strategy",
   "DeploymentDurationInMinutes": "Total amount of time the deployment lasted",
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": "The percentage of targets that received a deployed
 configuration during each interval",
   "FinalBakeTimeInMinutes": "The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete",
   "ReplicateTo": "The Systems Manager (SSM) document where the deployment
 strategy is saved"
}
```

Windows

```
{
   "Id": "Id of the deployment strategy",
   "Name": "Name of the deployment strategy",
   "Description": "Description of the deployment strategy",
   "DeploymentDurationInMinutes": "Total amount of time the deployment lasted",
   "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
   "GrowthFactor": "The percentage of targets that received a deployed
 configuration during each interval",
   "FinalBakeTimeInMinutes": "The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete",
   "ReplicateTo": "The Systems Manager (SSM) document where the deployment
 strategy is saved"
}
```

PowerShell

```
ContentLength               : Runtime of the command
DeploymentDurationInMinutes : Total amount of time the deployment lasted
Description                 : Description of the deployment strategy
FinalBakeTimeInMinutes      : The amount of time AWS AppConfig monitored for
 alarms before considering the deployment to be complete
```

```
GrowthFactor                : The percentage of targets that received a deployed
 configuration during each interval
GrowthType                  : The linear or exponential algorithm used to define
 how percentage grew over time
HttpStatusCode              : HTTP Status of the runtime
Id                          : The deployment strategy ID
Name                        : Name of the deployment strategy
ReplicateTo                 : The Systems Manager (SSM) document where the
 deployment strategy is saved
ResponseMetadata            : Runtime Metadata
```

部署配置

在创建了处理功能标志和自由格式配置数据所需的工件后，您可以使用 AWS Management Console AWS CLI、或 SDK 创建新的部署。在中启动部署会 AWS AppConfig 调用 StartDeploymentAPI 操作。该调用包括 AWS AppConfig 应用程序、环境和配置文件的 ID 以及（可选）要部署的配置数据版本。该调用还包括要使用的部署策略的 ID，该策略确定如何部署配置数据。

如果您部署存储在中的密钥 AWS Secrets Manager、使用客户托管密钥加密的亚马逊简单存储服务 (Amazon S3) Service 对象，或者存储 AWS Systems Manager 在使用客户托管密钥加密的 Parameter Store 中的安全字符串参数，则必须为`KmsKeyIdentifier`该参数指定一个值。如果您的配置未加密或使用加密 AWS 托管式密钥，则无需为`KmsKeyIdentifier`参数指定值。

> **Note**
>
> 为 `KmsKeyIdentifier` 指定的值必须是客户管理的密钥。该密钥不必与您用来加密配置的密钥相同。
> 当您使用开始部署时`KmsKeyIdentifier`，附加到您的 AWS Identity and Access Management (IAM) 委托人的权限策略必须允许该`kms:GenerateDataKey`操作。

AWS AppConfig 监视向所有主机的分发情况并报告状态。如果分发失败，则回 AWS AppConfig 滚配置。

> **Note**
>
> 一次只能将一个配置部署到一个环境中。但是，您可以同时将一个配置部署到不同的环境。

部署配置（控制台）

使用以下过程通过 AWS Systems Manager 控制台部署 AWS AppConfig 配置。

使用控制台部署配置

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，选择应用程序，然后选择您在中创建的应用程序在 AWS AppConfig中为应用程序创建命名空间。
3. 在 “环境” 选项卡上，填写环境的单选按钮，然后选择 “查看详细信息”。
4. 选择开始部署。
5. 对于 Configuration (配置)，请从列表中选择一个配置。
6. 根据您的配置来源，使用版本列表选择要部署的版本。
7. 对于 Deployment strategy (部署策略)，请从列表中选择一个策略。
8. （可选）对于部署描述，输入描述。
9. 要查看其他加密选项，请从列表中选择一个 AWS Key Management Service 密钥。
10. （可选）在标签部分，选择添加新标签并输入密钥和可选值。您最多可以为一个资源指定 50 个标签。
11. 选择开始部署。

部署配置（命令行）

以下过程介绍如何使用 AWS CLI（在 Linux 或 Windows 上）或 AWS Tools for PowerShell 部署 AWS AppConfig 配置。

分步部署配置

1. 打开 AWS CLI.
2. 运行以下命令部署配置。

   Linux

```
aws appconfig start-deployment \
  --application-id The_application_ID \
  --environment-id The_environment_ID \
  --deployment-strategy-id The_deployment_strategy_ID \
```

```
  --configuration-profile-id The_configuration_profile_ID \
  --configuration-version The_configuration_version_to_deploy \
  --description A_description_of_the_deployment \
  --tags User_defined_key_value_pair_metadata_of_the_deployment
```

Windows

```
aws appconfig start-deployment ^
  --application-id The_application_ID ^
  --environment-id The_environment_ID ^
  --deployment-strategy-id The_deployment_strategy_ID ^
  --configuration-profile-id The_configuration_profile_ID ^
  --configuration-version The_configuration_version_to_deploy ^
  --description A_description_of_the_deployment ^
  --tags User_defined_key_value_pair_metadata_of_the_deployment
```

PowerShell

```
Start-APPCDeployment `
  -ApplicationId The_application_ID `
  -ConfigurationProfileId The_configuration_profile_ID `
  -ConfigurationVersion The_configuration_version_to_deploy `
  -DeploymentStrategyId The_deployment_strategy_ID `
  -Description A_description_of_the_deployment `
  -EnvironmentId The_environment_ID `
  -Tag Hashtable_type_user_defined_key_value_pair_metadata_of_the_deployment
```

系统将返回类似于以下内容的信息。

Linux

```
{
   "ApplicationId": "The ID of the application that was deployed",
   "EnvironmentId" : "The ID of the environment",
   "DeploymentStrategyId": "The ID of the deployment strategy that was
 deployed",
   "ConfigurationProfileId": "The ID of the configuration profile that was
 deployed",
   "DeploymentNumber": The sequence number of the deployment,
   "ConfigurationName": "The name of the configuration",
```

```
  "ConfigurationLocationUri": "Information about the source location of the
 configuration",
  "ConfigurationVersion": "The configuration version that was deployed",
  "Description": "The description of the deployment",
  "DeploymentDurationInMinutes": Total amount of time the deployment lasted,
  "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
  "GrowthFactor": The percentage of targets to receive a deployed configuration
 during each interval,
  "FinalBakeTimeInMinutes": Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete,
  "State": "The state of the deployment",

  "EventLog": [
     {
        "Description": "A description of the deployment event",
        "EventType": "The type of deployment event",
        "OccurredAt": The date and time the event occurred,
        "TriggeredBy": "The entity that triggered the deployment event"
     }
  ],

  "PercentageComplete": The percentage of targets for which the deployment is
 available,
  "StartedAt": The time the deployment started,
  "CompletedAt": The time the deployment completed
}
```

Windows

```
{
   "ApplicationId": "The ID of the application that was deployed",
   "EnvironmentId" : "The ID of the environment",
   "DeploymentStrategyId": "The ID of the deployment strategy that was
 deployed",
   "ConfigurationProfileId": "The ID of the configuration profile that was
 deployed",
   "DeploymentNumber": The sequence number of the deployment,
   "ConfigurationName": "The name of the configuration",
   "ConfigurationLocationUri": "Information about the source location of the
 configuration",
   "ConfigurationVersion": "The configuration version that was deployed",
   "Description": "The description of the deployment",
```

```
  "DeploymentDurationInMinutes": Total amount of time the deployment lasted,
  "GrowthType": "The linear or exponential algorithm used to define how
 percentage grew over time",
  "GrowthFactor": The percentage of targets to receive a deployed configuration
 during each interval,
  "FinalBakeTimeInMinutes": Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete,
  "State": "The state of the deployment",

  "EventLog": [
     {
        "Description": "A description of the deployment event",
        "EventType": "The type of deployment event",
        "OccurredAt": The date and time the event occurred,
        "TriggeredBy": "The entity that triggered the deployment event"
     }
  ],

  "PercentageComplete": The percentage of targets for which the deployment is
 available,
  "StartedAt": The time the deployment started,
  "CompletedAt": The time the deployment completed
}
```

PowerShell

```
ApplicationId               : The ID of the application that was deployed
CompletedAt                 : The time the deployment completed
ConfigurationLocationUri    : Information about the source location of the
 configuration
ConfigurationName           : The name of the configuration
ConfigurationProfileId      : The ID of the configuration profile that was
 deployed
ConfigurationVersion        : The configuration version that was deployed
ContentLength               : Runtime of the deployment
DeploymentDurationInMinutes : Total amount of time the deployment lasted
DeploymentNumber            : The sequence number of the deployment
DeploymentStrategyId        : The ID of the deployment strategy that was
 deployed
Description                 : The description of the deployment
EnvironmentId               : The ID of the environment that was deployed
```

```
EventLog                    : {Description : A description of the deployment
 event, EventType : The type of deployment event, OccurredAt : The date and time
 the event occurred,
        TriggeredBy : The entity that triggered the deployment event}
FinalBakeTimeInMinutes      : Time AWS AppConfig monitored for alarms before
 considering the deployment to be complete
GrowthFactor                : The percentage of targets to receive a deployed
 configuration during each interval
GrowthType                  : The linear or exponential algorithm used to define
 how percentage grew over time
HttpStatusCode              : HTTP Status of the runtime
PercentageComplete          : The percentage of targets for which the deployment
 is available
ResponseMetadata            : Runtime Metadata
StartedAt                   : The time the deployment started
State                       : The state of the deployment
```

AWS AppConfig 部署与集成 CodePipeline

AWS AppConfig 是 AWS CodePipeline (CodePipeline) 的集成部署操作。 CodePipeline 是一项完全托管的持续交付服务，可帮助您实现发布管道的自动化，从而实现快速可靠的应用程序和基础架构更新。CodePipeline 每次发生代码更改时，都会根据您定义的发布模型自动执行发布过程的构建、测试和部署阶段。有关更多信息，请参阅什么是 AWS CodePipeline？

AWS AppConfig 与的集成 CodePipeline 具有以下好处：

- 过去管理编排的客户现在 CodePipeline 可以轻量级地将配置更改部署到其应用程序，而不必部署整个代码库。
- 想要使用 AWS AppConfig 来管理配置部署，但由于 AWS AppConfig 不支持其当前代码或配置存储而受到限制的客户现在可以选择其他选项。 CodePipeline 支持 AWS CodeCommit GitHub、和 BitBucket （仅举几例）。

> **Note**
>
> AWS AppConfig 只有在可用的 AWS 区域情况下 CodePipeline 才支持与 CodePipeline 集成。

集成的工作方式

首先要进行设置和配置 CodePipeline。这包括将您的配置添加到 CodePipeline支持的代码存储中。接下来，通过执行以下任务来设置 AWS AppConfig 环境：

- 创建命名空间和配置文件
- 选择预定义的部署策略或创建自己的部署策略

完成这些任务后，您可以在中创建一个指定 CodePipeline AWS AppConfig 为部署提供者的管道。然后，您可以更改您的配置并将其上传到您的 CodePipeline 代码存储区。上传新配置会自动在中启动新的部署 CodePipeline。部署完成后，可以验证更改。有关创建指定 AWS AppConfig 为部署提供者的管道的信息，请参阅《AWS CodePipeline 用户指南》中的教程：创建 AWS AppConfig 用作部署提供者的管道。

正在检索 AWS AppConfig中的功能标志和配置数据

您的应用程序通过使用数据服务建立配置会话来检索功能标志和自由格式配置 AWS AppConfig 数据。如果您使用本节中描述的简化检索方法之一，则 A AWS AppConfig gent Lambda 扩展或 AWS AppConfig 代理将代表您管理一系列 API 调用和会话令牌。您可以将 AWS AppConfig Agent 配置为本地主机，并让代理轮 AWS AppConfig 询配置更新。代理调用会StartConfiguration话和GetLatest配置 API 操作并在本地缓存您的配置数据。要检索数据，您的应用程序需要对本地主机服务器进行 HTTP 调用。 AWS AppConfig 代理支持多种用例，如中所述简化的检索方法。

如果您愿意，可手动调用这些 API 操作来检索配置。API 过程的工作原理如下：

您的应用程序使用 `StartConfigurationSession` API 操作建立配置会话。然后，您的会话客户端会定期调用 `GetLatestConfiguration` 以检查和检索最新的可用数据。

调用时`StartConfigurationSession`，您的代码会发送会话跟踪的 AWS AppConfig 应用程序、环境和配置文件的标识符（ID 或名称）。

作为响应， AWS AppConfig 提供`InitialConfigurationToken`给会话的客户端，并在它第一次调`GetLatestConfiguration`用该会话时使用。

调用 `GetLatestConfiguration` 时，您的客户端代码会发送它所拥有的最新 `ConfigurationToken` 值,并接收响应：

- `NextPollConfigurationToken`：下次调用 `GetLatestConfiguration` 时要使用的 `ConfigurationToken` 值。
- 配置：用于会话的最新数据。如果客户端已有最新版本的配置，则此字段可能为空。

本节包含以下信息。

内容

关于 AWS AppConfig 数据平面服务

2021 年 11 月 18 日，AWS AppConfig 发布了一项新的数据平面服务。这项服务取代了之前使用 GetConfiguration API 操作检索配置数据的过程。数据平面服务使用两个新的 API 操作，即StartConfiguration会话和GetLatest配置。数据平面服务还使用新端点。

如果您在 2022 年 1 月 28 日 AWS AppConfig 之前开始使用，则该服务可能正在直接调用 GetConfiguration API 操作，或者可能正在使用提供的 AWS客户端（例如 AWS AppConfig 代理 Lambda 扩展）来调用此 API 操作。如果您直接调用 GetConfiguration API 操作，请执行步骤以使用 StartConfigurationSession 和 GetLatestConfiguration API 操作。如果您使用的是 AWS AppConfig 代理 Lambda 扩展，请参阅本主题后面标题为 “此更改如何影响代理 AWS AppConfig Lambda 扩展” 的部分。

与现已弃用的 API 操作相比，新的数据平面 GetConfiguration API 操作具有以下优点。

1. 无需管理 ClientID 参数。使用数据平面服务时，ClientID 由 StartConfigurationSession 创建的会话令牌在内部进行管理。
2. 您不再需要包含 ClientConfigurationVersion 来指示配置数据的缓存版本。使用数据平面服务时，ClientConfigurationVersion 由 StartConfigurationSession 创建的会话令牌在内部进行管理。
3. 新的数据平面 API 调用专用端点通过分离控制平面和数据平面调用来改进代码结构。
4. 新的数据平面服务提高了数据平面操作的未来可扩展性。通过利用管理配置数据检索的配置会话，AWS AppConfig 团队可以在将来创建更强大的增强功能。

从 **GetConfiguration** 迁移到 **GetLatestConfiguration**

要开始使用新的数据平面服务，您需要更新调用 GetConfiguration API 操作的代码。使用 StartConfigurationSession API 操作启动配置会话，然后调用 GetLatestConfiguration API 操作来检索配置数据。为提高性能，我们建议您在本地缓存配置数据。有关更多信息，请参阅 通过直接调用 API 获取配置。

此更改如何影响 AWS AppConfig Agent Lambda 扩展

此更改对 AWS AppConfig 代理 Lambda 扩展的工作方式没有直接影响。 AWS AppConfig Agent Lambda 扩展的旧版本代表您调GetConfiguration用了 API 操作。较新版本调用数据平面 API 操作。如果您使用的是 AWS AppConfig Lambda 扩展，我们建议您将扩展更新为最新的 Amazon 资源名称（ARN），并更新此新 API 调用的权限。有关更多信息，请参阅 使用 AWS AppConfig 代理 Lambda 扩展检索配置数据。

简化的检索方法

AWS AppConfig 提供了几种用于检索配置数据的简化方法。如果您在 AWS Lambda 函数中使用 AWS AppConfig 功能标志或自由格式配置数据，则可以使用 A AWS AppConfig gent Lambda 扩展来检索配置。如果您有在 Amazon EC2 实例上运行的应用程序，则可以使用 AWS AppConfig 代理来检索配置。 AWS AppConfig 代理还支持在亚马逊 Elastic Kubernetes Service（亚马逊 EKS）或亚马逊弹性容器服务（亚马逊 ECS）容器映像上运行的应用程序。

完成初始设置后，这些检索配置数据的方法比直接调用 AWS AppConfig API 更简单。它们会自动实施最佳实践，并且 AWS AppConfig 由于检索配置的 API 调用较少，因此可能会降低您的使用成本。

主题

使用 AWS AppConfig 代理 Lambda 扩展检索配置数据

AWS Lambda 扩展是一个配套流程，用于增强 Lambda 函数的功能。可以在调用函数之前启动扩展，与函数并行运行，并在处理函数调用后继续运行。实际上，Lambda 扩展类似于与 Lambda 调用并行的客户端。此并行客户端可以在其生命周期中的任何时刻与您的函数交互。

如果您在 Lambda 函数中使用 AWS AppConfig 功能标志或其他动态配置数据，那么我们建议您将代理 Lam AWS AppConfig bda 扩展作为一个层添加到您的 Lambda 函数中。这使得调用功能标志变得更加简单，而且扩展本身包含了在降低成本的 AWS AppConfig 同时简化使用的最佳实践。由于减少了对该 AWS AppConfig 服务的 API 调用，缩短了 Lambda 函数的处理时间，从而降低了成本。有关 Lambda 扩展的更多信息，请参阅《AWS Lambda 开发人员指南》中的Lambda 扩展。

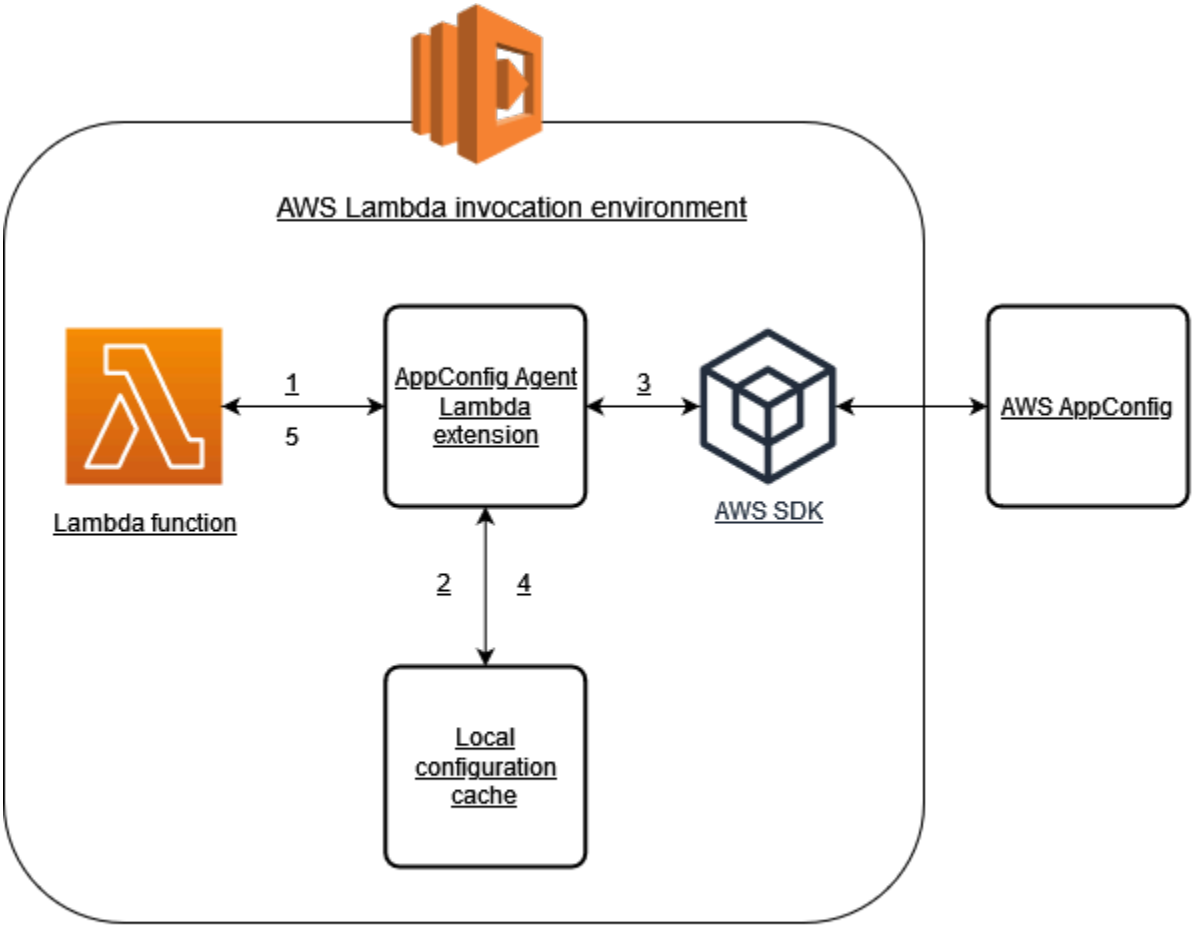
> **Note**
>
> AWS AppConfig 定价基于调用和接收配置的次数。如果您的 Lambda 执行多次冷启动并频繁检索新的配置数据，则成本会增加。

本主题包括有关 A AWS AppConfig gent Lambda 扩展的信息，以及如何配置该扩展以与您的 Lambda 函数配合使用的过程。

工作方式

[如果您使用 AWS AppConfig 管理没有 Lambda 扩展的 Lamb da 函数的配置，则必须通过与会话和配置 API 操作集成，将 Lambda 函数配置为接收配置更新。StartConfiguration GetLatest]

将 Lambda AWS AppConfig 代理扩展与您的 Lambda 函数集成可简化此过程。该扩展负责调用 AWS AppConfig 服务、管理检索到的数据的本地缓存、跟踪下一次服务调用所需的配置令牌以及定期在后台检查配置更新。下图显示了它的工作方式。



1. 您可以将 AWS AppConfig 代理 Lambda 扩展配置为 Lambda 函数的一个层。
2. 要访问其配置数据，您的函数在上运行的 HTTP 端点上调用该 AWS AppConfig 扩展程序`localhost:2772`。

3. 该扩展会维护配置数据的本地缓存。如果数据不在缓存中，则扩展程序会调用 AWS AppConfig 以获取配置数据。
4. 从服务接收配置后，扩展会将其存储在本地缓存中并将其传递给 Lambda 函数。
5. AWS AppConfig Agent Lambda 扩展定期在后台检查您的配置数据是否有更新。每次调用 Lambda 函数时，扩展都会检查自检索配置以来的运行时间。如果经过的时间大于配置的轮询间隔，则分机 AWS AppConfig 将调用以检查新部署的数据，如果有更改，则更新本地缓存，并重置经过的时间。

> **Note**
>
> - Lambda 会实例化与函数所需的并发级别相对应的单独实例。每个实例都是独立的，并维护自己的配置数据本地缓存。有关 Lambda 实例和并发的更多信息，请参阅《Lambda 开发人员指南》中的 管理 Lambda 函数并发 。
> - 从 AWS AppConfig部署更新的配置后，配置更改显示在 Lambda 函数中所需的时间取决于您用于部署的部署策略和为扩展配置的轮询间隔。

开始前的准备工作

在启用 AWS AppConfig 代理 Lambda 扩展之前，请执行以下操作：

- 在 Lambda 函数中组织配置，以便将其外部化为 AWS AppConfig。
- 创建 AWS AppConfig 构件和配置数据，包括功能标志或自由格式配置数据。有关更多信息，请参阅在中创建功能标志和自由格式配置数据 AWS AppConfig。
- 在 Lambda 函数执行角色使用的 AWS Identity and Access Management (IAM) 策略中添加`appconfig:StartConfigurationSession`和`appconfig:GetLatestConfiguration`。有关更多信息，请参阅AWS Lambda 开发人员指南 中的 AWS Lambda 执行角色。有关 AWS AppConfig 权限的更多信息，请参阅《服务授权参考》中的 AWS AppConfig操作、资源和条件键。

添加 AWS AppConfig 代理 Lambda 扩展

要使用 AWS AppConfig 代理 Lambda 扩展，您需要将扩展程序添加到您的 Lambda 中。这可以通过将 AWS AppConfig 代理 Lambda 扩展作为层添加到您的 Lambda 函数中，或者在 Lambda 函数上启用该扩展作为容器映像来实现。

> **Note**
>
> 该 AWS AppConfig 扩展程序与运行时无关，并且支持所有运行时。

使用层和 ARN 添加 AWS AppConfig 代理 Lambda 扩展

要使用 AWS AppConfig 代理 Lambda 扩展，您需要将该扩展作为一个层添加到 Lambda 函数中。有关如何向函数添加层的信息，请参阅《AWS Lambda 开发者指南》中的配置扩展。 AWS Lambda 控制台中扩展的名称是 AWS-AppConfig- Extension。另请注意，当您将扩展作为层添加到 Lambda 时，您必须指定 Amazon 资源名称（ARN）。从以下列表中选择一个 ARN，该列表与平台和您创建 Lambda 的 AWS 区域 位置相对应。

- x86-64 平台
- ARM64 平台

如果您想在将扩展添加到函数中之前对其进行测试，可以使用以下代码示例来验证其是否有效。

```
import urllib.request


def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name'
    config = urllib.request.urlopen(url).read()
    return config
```

要对其进行测试，请为 Python 创建一个新的 Lambda 函数，添加扩展，然后运行 Lambda 函数。运行 Lambda 函数后，Lambd AWS AppConfig a 函数会返回您为 http://localhost:2772 路径指定的配置。有关如何创建 Lambda 函数的信息，请参阅 AWS Lambda 《 Lambda 开发人员指南》中的 使用控制台创建 Lambda 函数。

要将 AWS AppConfig Agent Lambda 扩展添加为容器映像，请参阅。使用容器镜像添加 AWS AppConfig 代理 Lambda 扩展

配置 AWS AppConfig 代理 Lambda 扩展

您可以通过更改以下 AWS Lambda 环境变量来配置扩展。有关更多信息，请参阅《AWS Lambda 开发人员指南》中的使用 AWS Lambda 环境变量。

预取配置数据

环境变量 AWS_APPCONFIG_EXTENSION_PREFETCH_LIST 可以改善函数的启动时间。初始化 A AWS AppConfig gent Lambda 扩展时，它会检索 AWS AppConfig Lambda 开始初始化您的函数并调用您的处理程序之前的指定配置。在某些情况下，在函数请求配置数据之前，配置数据已在本地缓存中可用。

要使用预取功能，请将环境变量的值设置为与配置数据对应的路径。例如，如果您的配置对应于分别名为 “my_application”、“my_environmention” 和 “my_configuration_data” 的应用程序、环境和配置文件，则路径将为 `/applications/my_application/environments/my_environment/configurations/my_configuration_data`。您可以通过以逗号分隔的列表列出多个配置项目来指定这些配置项目（如果资源名称中包含逗号，请使用资源的 ID 值而不是其名称）。

从其他账户访问配置数据

A AWS AppConfig gent Lambda 扩展可以通过指定授予数据权限的 IAM 角色从其他账户检索配置数据。要设置此策略，请按照下列步骤操作：

1. 在 AWS AppConfig 用于管理配置数据的账户中，创建一个具有信任策略的角色，该策略允许运行 Lambda 函数的账户访问`appconfig:StartConfigurationSession`和`appconfig:GetLatestConfiguration`操作以及与配置资源对应的部分或完整 ARN。 AWS AppConfig
2. 在运行 Lambda 函数的账户中，使用在步骤 1 中创建的角色的 ARN 将 AWS_APPCONFIG_EXTENSION_ROLE_ARN 环境变量添加到 Lambda 函数中。
3. （可选）如果需要，可使用 AWS_APPCONFIG_EXTENSION_ROLE_EXTERNAL_ID 环境变量指定外部 ID。同样，可使用 AWS_APPCONFIG_EXTENSION_ROLE_SESSION_NAME 环境变量配置会话名称。

> **Note**
>
> 请注意以下信息。
>
> - A AWS AppConfig gent Lambda 扩展只能从一个账户检索数据。如果您指定 IAM 角色，则扩展将无法从运行 Lambda 函数的账户中检索配置数据。
> - AWS Lambda 使用亚马逊日志记录有关 AWS AppConfig Agent Lambda 扩展和 Lambda 函数的信息。 CloudWatch

环境变量	详细信息	默认值
AWS_APPCONFIG_EXTENSION_HTTP_PORT	此环境变量用于指定运行托管扩展的本地 HTTP 服务器的端口。	2772
AWS_APPCONFIG_EXTENSION_LOG_LEVEL	此环境变量指定将哪些 AWS AppConfig 特定于扩展的日志发送到函数的 Amazon CloudWatch Logs。不区分大小写的有效值包括 debug、info、warn、error 和 none。调试包括有关扩展的详细信息，包括计时信息。	info
AWS_APPCONFIG_EXTENSION_MAX_CONNECTIONS	此环境变量配置扩展用于从 AWS AppConfig检索配置的最大连接数。	3
AWS_APPCONFIG_EXTENSION_POLL_INTERVAL_SECONDS	此环境变量控制扩展程序轮询更新配置的频率（以秒 AWS AppConfig 为单位）。	45
AWS_APPCONFIG_EXTENSION_POLL_TIMEOUT_MILLIS	此环境变量控制扩展程序在刷新缓存中的数据 AWS AppConfig 时等待响应的最大时间（以毫秒为单位）。如果在指定的时间内 AWS AppConfig 没有响应，则扩展程序会跳过此轮询间隔并返回之前更新的缓存数据。	3000
AWS_APPCONFIG_EXTENSION_PREFETCH_LIST	此环境变量用于指定在函数初始化和处理程序运行之前，扩展开始检索的配置数据。它可以大大缩短函数的冷启动时间。	无

环境变量	详细信息	默认值
AWS_APPCONFIG_EXTENSION_PROXY_HEADERS	此环境变量用于指定 AWS_APPCONFIG_EXTENSION_PROXY_URL 环境变量中引用的代理所需的标头。其值是以逗号分隔的标头列表。每个标头使用以下形式： `"header: value"`	无
AWS_APPCONFIG_EXTENSION_PROXY_URL	此环境变量指定用于从 AWS AppConfig 扩展到的连接的代理 URL AWS 服务。 HTTPS并支持 HTTP URL。	无
AWS_APPCONFIG_EXTENSION_ROLE_ARN	此环境变量指定与该 AWS AppConfig 扩展程序为检索配置而应担任的角色相对应的 IAM 角色 ARN。	无
AWS_APPCONFIG_EXTENSION_ROLE_EXTERNAL_ID	此环境变量指定要与代入角色 ARN 结合使用的外部 ID。	无
AWS_APPCONFIG_EXTENSION_ROLE_SESSION_NAME	此环境变量指定要与代入的 IAM 角色的凭证关联的会话名称。	无
AWS_APPCONFIG_EXTENSION_SERVICE_REGION	此环境变量指定了扩展程序在调用 AWS AppConfig 服务时应使用的备用区域。如果未定义，则扩展将使用当前区域中的终端节点。	无

环境变量	详细信息	默认值
AWS_APPCONFIG_EXTENSION_MANIFEST	此环境变量将 AWS AppConfig Agent 配置为利用其他每个配置的功能，例如多帐户检索和将配置保存到磁盘。您可以输入以下值之一： • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" 有关使用这些功能的更多信息，请参阅 其他检索功能。	true
AWS_APPCONFIG_EXTENSION_WAIT_ON_MANIFEST	此环境变量将 AWS AppConfig Agent 配置为等到清单处理完毕后再完成启动。	true

从功能标志配置中检索一个或多个标志

对于功能标志配置（类型 AWS.AppConfig.FeatureFlags 的配置），Lambda 扩展使您能够检索配置中的单个标志或标志子集。如果您的 Lambda 只需要使用配置文件中的几个标志，则检索一个或两个标志非常有用。下面的示例使用了 Python。

> **Note**
>
> 只能在 A AWS AppConfig gent Lambda 扩展版本 2.0.45 及更高版本中调用配置中的单个功能标志或部分标志。

您可以从本地 HTTP 端点检索 AWS AppConfig 配置数据。要访问特定标志或标志列表，请使用 AWS AppConfig 配置文件的 ?flag=*flag_name* 查询参数。

访问单个标志及其属性

```
import urllib.request

def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name'
    config = urllib.request.urlopen(url).read()
    return config
```

访问多个标志及其属性

```
import urllib.request

def lambda_handler(event, context):
    url = f'http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two'
    config = urllib.request.urlopen(url).read()
    return config
```

查看 AWS AppConfig 代理 Lambda 扩展日志

您可以在日志中查看 Ag AWS AppConfig ent Lambda 扩展程序的 AWS Lambda 日志数据。日志条目开头为。`appconfig agent`以下为示例。

```
[appconfig agent] 2024/05/07 04:19:01 ERROR retrieve failure for
 'SourceEventConfig:SourceEventConfigEnvironment:SourceEventConfigProfile':
 StartConfigurationSession: api error AccessDenied: User:
 arn:aws:sts::0123456789:assumed-role/us-east-1-LambdaRole/extension1 is not authorized
 to perform: sts:AssumeRole on resource: arn:aws:iam::0123456789:role/test1 (retry in
 60s)
```

AWS AppConfig Agent Lambda 扩展的可用版本

本主题包含有关 AWS AppConfig 代理 Lambda 扩展版本的信息。 AWS AppConfig 代理 Lambda 扩展支持为 x86-64 和 ARM64(Graviton2)平台开发的 Lambda 函数。要正常运行，您的 Lambda 函数必须配置为使用其当前托管 AWS 区域 位置的特定 Amazon 资源名称 (ARN)。您可以在本节后 AWS 区域 面查看 ARN 详情。

> **Important**
>
> 请注意以下有关 AWS AppConfig 代理 Lambda 扩展的重要细节。

- `GetConfiguration` API 操作已于 2022 年 1 月 28 日弃用。接收配置数据的调用应改用 `StartConfigurationSession` 和 `GetLatestConfiguration` API。如果您使用的是在 2022 年 1 月 28 日之前创建的 A AWS AppConfig gent Lambda 扩展版本，则可能需要配置新 API 的权限。有关更多信息，请参阅 关于 AWS AppConfig 数据平面服务。
- AWS AppConfig 支持中列出的所有版本较旧的扩展版本。我们建议您定期更新到最新版本，以利用扩展程序增强功能。

主题

AWS AppConfig 代理 Lambda 扩展程序发行说明

下表描述了对 AWS AppConfig Lambda 扩展的最新版本所做的更改。

版本	推出日期	注意
2.0.358	12/01/2023	增加了对以下检索功能的支持： • 多账户检索：使用主账户或检索 AWS 账户 中的 AWS AppConfig 代理从多个供应商账户检索配置数据。 • 将配置副本写入磁盘：使用 AWS AppConfig 代理将配置数据写入磁盘。此功能使客户能够使用从磁盘读取配置数据以进行集成的应用程序 AWS AppConfig。

版本	推出日期	注意
2.0.181	08/14/2023	增加了对以色列（特拉维夫）il-central- AWS 区域 1 的支持。
2.0.165	02/21/2023	次要错误修复。不再通过 AWS Lambda 控制台将扩展程序的使用限制在特定的运行时版本上。增加了对以下 AWS 区域的支持： • 中东（阿联酋）- me-central-1 • 亚太（海得拉巴）- ap-south-2 • 亚太地区（墨尔本）区域（ap-southeast-4） • 欧洲（西班牙）- eu-south-2 • 欧洲（苏黎世）- eu-central-2
2.0.122	08/23/2022	添加了对隧道代理的支持，可以为该代理配置 `AWS_APPCONFIG_EXTENSION_PROXY_URL` 和 `AWS_APPCONFIG_EXTENSION_PROXY_HEADERS` 环境变量。添加了 .NET 6 作为运行时系统。有关环境变量的更多信息，请参阅 配置 AWS AppConfig 代理 Lambda 扩展。
2.0.58	05/03/2022	改进了对 Lambda 中 Graviton2 (ARM64) 处理器的支持。

版本	推出日期	注意
2.0.45	03/15/2022	添加了对调用单个功能标志的支持。以前，客户会调用分组到一个配置文件中的功能标志，并且必须在客户端一侧解析响应。在此版本中，客户可以在调用 HTTP 本地主机端点时使用 `flag=<flag-name>` 参数来获取单个标志的值。此外，还首次添加了对 Graviton2 (ARM64) 处理器的支持。

查找您的 Lambda 扩展程序版本号

使用以下过程查找您当前配置的 AWS AppConfig Agent Lambda 扩展的版本号。要正常运行，您的 Lambda 函数必须配置为使用其当前托管 AWS 区域 位置的特定 Amazon 资源名称 (ARN)。

1. 登录 AWS Management Console 并打开 AWS Lambda 控制台，网址为 https://console.aws.amazon.com/lambda/。
2. 选择要添加 `AWS-AppConfig-Extension` 层的 Lambda 函数。
3. 在 Layers（层）区域中，选择 Add a layer（添加层）。
4. 在 “选择图层” 部分，从图AWS 层列表中选择 AWS-AppConfig-Ex tension。
5. 使用版本列表选择版本号。
6. 选择添加。
7. 使用测试选项卡测试此函数。
8. 测试完成后，查看日志输出。在 “执行详情” 部分中找到 A AWS AppConfig gent Lambda 扩展版本。此版本必须与此版本所需的 URL 匹配。

x86-64 平台

如果您要将此扩展作为一个层添加到 Lambda，则必须指定 ARN。从下表中选择与您创建 Lambda 的 AWS 区域 位置相对应的 ARN。这些 ARN 适用于为 x86-64 平台开发的 Lambda 函数。

版本 2.0.358

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:128
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:93
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:141
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:161
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:93
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:106
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:47
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:125
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:93

区域	ARN
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:98
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:159
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:83
欧洲（西班牙）	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:44
中国（北京）	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:76
中国（宁夏）	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:76
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:83
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:98
亚太地区 (首尔)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:108

区域	ARN
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:101
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:106
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:106
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:79
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:20
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:107
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:47
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:128
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:83

区域	ARN
以色列（特拉维夫）	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:22
中东（阿联酋）	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:49
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:85
AWS GovCloud （美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:54
AWS GovCloud （美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:54

ARM64 平台

如果您要将此扩展作为一个层添加到 Lambda，则必须指定 ARN。从下表中选择与您创建 Lambda 的 AWS 区域 位置相对应的 ARN。这些 ARN 适用于为 ARM64 平台开发的 Lambda 函数。

版本 2.0.358

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:61
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:45

区域	ARN
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:18
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:63
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:13
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:49
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension-Arm64:5
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:63
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:45
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:17
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:18

区域	ARN
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:11
欧洲（西班牙）	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension-Arm64:5
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:11
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:51
亚太地区 (首尔)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:16
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:16
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:58
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:49
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:16

区域	ARN
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension-Arm64:5
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:49
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension-Arm64:5
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:16
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:11
中东（阿联酋）	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension-Arm64:5
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:13
以色列（特拉维夫）	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension-Arm64:5

较旧的扩展版本

本节列出了 ARN 和旧版本 AWS 区域 的 Lambd AWS AppConfig a 扩展。此列表不包含所有以前版本的 AWS AppConfig 代理 Lambda 扩展的信息，但会在发布新版本时进行更新。

较旧的扩展版本（x86-64 平台）

下表列出了 ARN 以及 AWS 区域 为 x86-64 平台开发的 A AWS AppConfig gent Lambda 扩展的旧版本。

由较新的扩展名取代的日期：2023 年 1 月 12 日

版本 2.0.181

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:113
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:81
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:124
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:146
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:81
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:93
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:32

区域	ARN
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:110
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:81
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:82
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:142
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:73
欧洲（西班牙）	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:29
中国（北京）	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:68
中国（宁夏）	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:68
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:73

区域	ARN
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:84
亚太地区 (首尔)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:93
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:86
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:91
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:93
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:64
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:5
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:94
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:32

区域	ARN
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:113
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:73
以色列（特拉维夫）	arn:aws:lambda:il-central-1:895787185223:layer:AWS-AppConfig-Extension:7
中东（阿联酋）	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:34
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:73
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:46
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:46

替换为较新扩展的日期：2023 年 8 月 14 日

版本 2.0.165

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:110
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:79
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:121
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:143
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:79
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:91
欧洲（苏黎世）	arn:aws:lambda:eu-central-2:758369105281:layer:AWS-AppConfig-Extension:29
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:108
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:79

区域	ARN
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:80
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:139
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:71
欧洲（西班牙）	arn:aws:lambda:eu-south-2:586093569114:layer:AWS-AppConfig-Extension:26
中国（北京）	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:66
中国（宁夏）	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:66
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:71
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:82
亚太地区 (首尔)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:91

区域	ARN
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:84
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:89
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:91
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:60
亚太地区（墨尔本）	arn:aws:lambda:ap-southeast-4:307021474294:layer:AWS-AppConfig-Extension:2
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:92
亚太地区（海得拉巴）	arn:aws:lambda:ap-south-2:489524808438:layer:AWS-AppConfig-Extension:29
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:110
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:71

区域	ARN
中东（阿联酋）	arn:aws:lambda:me-central-1:662846165436:layer:AWS-AppConfig-Extension:31
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:71
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:44
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:44

替换为较新扩展的日期：2023 年 2 月 21 日

版本 2.0.122

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:82
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:59
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:93

区域	ARN
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:114
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:59
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:70
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:82
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:59
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:60
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:111
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:54
中国（北京）	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:52

区域	ARN
中国（宁夏）	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:52
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:54
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:62
亚太地区 (首尔)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:70
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:59
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:64
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:70
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:37
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:71

区域	ARN
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:82
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:54
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:54
AWS GovCloud （美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:29
AWS GovCloud （美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:29

替换为较新扩展的日期：2022 年 8 月 23 日

版本 2.0.58

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:69
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:50

区域	ARN
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:78
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:101
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:50
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:59
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:69
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:50
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:51
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:98
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:47

区域	ARN
中国（北京）	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:46
中国（宁夏）	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:46
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:47
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:49
亚太地区 (首尔)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:59
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:46
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:51
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:59
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:24

区域	ARN
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:60
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:69
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:47
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:47
AWS GovCloud（美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:23
AWS GovCloud（美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:23

替换为较新扩展的日期：2022 年 4 月 21 日

版本 2.0.45

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:68

区域	ARN
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:49
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:77
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:100
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:49
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:58
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:68
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:49
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:50
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:97

区域	ARN
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:46
中国（北京）	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:45
中国（宁夏）	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:45
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:46
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:48
亚太地区 (首尔)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:58
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:45
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:50
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:58

区域	ARN
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:23
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:59
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:68
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:46
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:46
AWS GovCloud （美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:22
AWS GovCloud （美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:22

替换为较新扩展的日期：2022 年 3 月 15 日

版本 2.0.30

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:61
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension:47
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension:61
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension:89
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension:47
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension:54
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension:59
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension:47
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension:48

区域	ARN
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension:86
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension:44
中国（北京）	arn:aws-cn:lambda:cn-north-1:615057806174:layer:AWS-AppConfig-Extension:43
中国（宁夏）	arn:aws-cn:lambda:cn-northwest-1:615084187847:layer:AWS-AppConfig-Extension:43
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension:44
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension:45
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension:42
亚太地区（首尔）	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension:54
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension:45

区域	ARN
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension:54
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension:13
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension:55
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension:61
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension:44
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension:44
AWS GovCloud （美国东部）	arn:aws-us-gov:lambda:us-gov-east-1:946561847325:layer:AWS-AppConfig-Extension:20
AWS GovCloud （美国西部）	arn:aws-us-gov:lambda:us-gov-west-1:946746059096:layer:AWS-AppConfig-Extension:20

较旧的扩展版本（ARM64 平台）

下表列出了 ARN 以及为 ARM64 平台开发 AWS 区域 的 A AWS AppConfig gent Lambda 扩展的旧版本的 ARN。

由较新的扩展名取代的日期：2023 年 1 月 12 日

版本 2.0.181

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:46
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:33
美国西部（加利福尼亚北部）	arn:aws:lambda:us-west-1:958113053741:layer:AWS-AppConfig-Extension-Arm64:1
美国西部（俄勒冈）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:48
加拿大（中部）	arn:aws:lambda:ca-central-1:039592058896:layer:AWS-AppConfig-Extension-Arm64:1
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:36
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:48
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:33

区域	ARN
欧洲地区（巴黎）	arn:aws:lambda:eu-west-3:493207061005:layer:AWS-AppConfig-Extension-Arm64:1
Europe (Stockholm)	arn:aws:lambda:eu-north-1:646970417810:layer:AWS-AppConfig-Extension-Arm64:1
欧洲地区（米兰）	arn:aws:lambda:eu-south-1:203683718741:layer:AWS-AppConfig-Extension-Arm64:1
亚太地区（香港）	arn:aws:lambda:ap-east-1:630222743974:layer:AWS-AppConfig-Extension-Arm64:1
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:37
亚太地区 (首尔)	arn:aws:lambda:ap-northeast-2:826293736237:layer:AWS-AppConfig-Extension-Arm64:1
亚太地区（大阪）	arn:aws:lambda:ap-northeast-3:706869817123:layer:AWS-AppConfig-Extension-Arm64:1
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:43
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:36

区域	ARN
亚太地区（雅加达）	arn:aws:lambda:ap-southeast-3:418787028745:layer:AWS-AppConfig-Extension-Arm64:1
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:36
南美洲（圣保罗）	arn:aws:lambda:sa-east-1:000010852771:layer:AWS-AppConfig-Extension-Arm64:1
非洲（开普敦）	arn:aws:lambda:af-south-1:574348263942:layer:AWS-AppConfig-Extension-Arm64:1
中东（巴林）	arn:aws:lambda:me-south-1:559955524753:layer:AWS-AppConfig-Extension-Arm64:1

替换为较新扩展的日期：2023 年 3 月 30 日

版本 2.0.165

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:43
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:31

区域	ARN
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:45
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:34
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:46
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:31
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:35
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:41
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:34
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:34

替换为较新扩展的日期：2023 年 2 月 21 日

版本 2.0.122

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:15
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:11
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:16
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:13
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:20
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:11
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:15
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:16
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:13

区域	ARN
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:13

替换为较新扩展的日期：2022 年 8 月 23 日

版本 2.0.58

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:2
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:2
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:3
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:2
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:7
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:2

区域	ARN
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:2
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:3
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:2
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:2

替换为较新扩展的日期：2022 年 4 月 21 日

版本 2.0.45

区域	ARN
美国东部（弗吉尼亚州北部）	arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension-Arm64:1
美国东部（俄亥俄州）	arn:aws:lambda:us-east-2:728743619870:layer:AWS-AppConfig-Extension-Arm64:1
美国西部（俄勒冈州）	arn:aws:lambda:us-west-2:359756378197:layer:AWS-AppConfig-Extension-Arm64:2

区域	ARN
欧洲地区（法兰克福）	arn:aws:lambda:eu-central-1:066940009817:layer:AWS-AppConfig-Extension-Arm64:1
欧洲地区（爱尔兰）	arn:aws:lambda:eu-west-1:434848589818:layer:AWS-AppConfig-Extension-Arm64:6
欧洲地区（伦敦）	arn:aws:lambda:eu-west-2:282860088358:layer:AWS-AppConfig-Extension-Arm64:1
亚太地区（东京)	arn:aws:lambda:ap-northeast-1:980059726660:layer:AWS-AppConfig-Extension-Arm64:1
亚太地区（新加坡）	arn:aws:lambda:ap-southeast-1:421114256042:layer:AWS-AppConfig-Extension-Arm64:2
亚太地区（悉尼）	arn:aws:lambda:ap-southeast-2:080788657173:layer:AWS-AppConfig-Extension-Arm64:1
亚太地区（孟买）	arn:aws:lambda:ap-south-1:554480029851:layer:AWS-AppConfig-Extension-Arm64:1

使用容器镜像添加 AWS AppConfig 代理 Lambda 扩展

您可以将您的 A AWS AppConfig gent Lambda 扩展打包为容器映像，然后将其上传到托管在亚马逊弹性容器注册表 (Amazon ECR) Container Registry 上的容器注册表。

将 Lambda AWS AppConfig 代理扩展添加为 Lambda 容器镜像

1. 在 () 中输入 AWS 区域 和亚马逊资源名称 (ARN)， AWS Command Line Interface 如下所示。AWS CLI将区域和 ARN 值替换为您的地区和匹配的 ARN，以检索 Lambda 层的副本。 AWS AppConfig 为 x86-64 和 ARM64 平台提供 ARN。

```
aws lambda get-layer-version-by-arn \
  --region AWS ## \
  --arn extension ARN
```

以下为示例。

```
aws lambda get-layer-version-by-arn \
  --region us-east-1 \
  --arn arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-Extension:128
```

响应看起来与以下内容类似：

```
{
  "LayerVersionArn": "arn:aws:lambda:us-east-1:027255383542:layer:AWS-AppConfig-
Extension:128",
  "Description": "AWS AppConfig extension: Use dynamic configurations deployed via
 AWS AppConfig for your AWS Lambda functions",
  "CreatedDate": "2021-04-01T02:37:55.339+0000",
  "LayerArn": "arn:aws:lambda::layer:AWS-AppConfig-Extension",

  "Content": {
    "CodeSize": 5228073,
    "CodeSha256": "8otOgbLQbexpUm3rKlMhvcE6Q5TvwcLCKrc4Oe+vmMY=",
    "Location" : "S3-Bucket-Location-URL"
  },

  "Version": 30,
  "CompatibleRuntimes": [
    "python3.8",
    "python3.7",
    "nodejs12.x",
    "ruby2.7"
  ],
}
```

2. 在上面的响应中，为 `Location` 返回的值是包含 Lambda 扩展的 Amazon Simple Storage Service (Amazon S3) 存储桶的 URL。将 URL 粘贴到 Web 浏览器中，以便下载 Lambda 扩展 .zip 文件。

   > **Note**
   >
   > Amazon S3 存储桶 URL 仅在 10 分钟内可用。

   （可选）或者，您也可以使用以下 `curl` 命令下载 Lambda 扩展。

   ```
   curl -o extension.zip "S3-Bucket-Location-URL"
   ```

   （可选）或者，您可以结合步骤 1 和步骤 2 来检索 ARN 并一次下载 .zip 扩展名文件。

   ```
   aws lambda get-layer-version-by-arn \
     --arn extension ARN \
     | jq -r '.Content.Location' \
     | xargs curl -o extension.zip
   ```

3. 在 `Dockerfile` 中添加以下行，将扩展添加到容器映像。

   ```
   COPY extension.zip extension.zip
   RUN yum install -y unzip \
     && unzip extension.zip /opt \
     && rm -f extension.zip
   ```

4. 确保 Lambda 函数执行角色具有 appconfig: GetConfiguration 权限集。

示例

本节包括一个在基于容器映像的 Python Lambda 函数上启用 A AWS AppConfig gent Lambda 扩展的示例。

1. 创建一个类似于以下操作的 `Dockerfile`。

   ```
   FROM public.ecr.aws/lambda/python:3.8 AS builder
   COPY extension.zip extension.zip
   RUN yum install -y unzip \
     && unzip extension.zip -d /opt \
   ```

```
  && rm -f extension.zip

FROM public.ecr.aws/lambda/python:3.8
COPY --from=builder /opt /opt
COPY index.py ${LAMBDA_TASK_ROOT}
CMD [ "index.handler" ]
```

2. 将扩展层下载到与 Dockerfile 相同的目录。

```
aws lambda get-layer-version-by-arn \
  --arn extension ARN \
  | jq -r '.Content.Location' \
  | xargs curl -o extension.zip
```

3. 在与 Dockerfile 相同的目录中创建名为 index.py 的 Python 文件。

```
import urllib.request

def handler(event, context):
    return {
       # replace parameters here with your application, environment, and
 configuration names
       'configuration': get_configuration('myApp', 'myEnv', 'myConfig'),
    }

def get_configuration(app, env, config):
    url = f'http://localhost:2772/applications/{app}/environments/{env}/
configurations/{config}'
    return urllib.request.urlopen(url).read()
```

4. 然后，在该目录中执行以下步骤来构建 docker 镜像并将其上载到 Amazon ECR。

```
// set environment variables
export ACCOUNT_ID = <YOUR_ACCOUNT_ID>
export REGION = <AWS_REGION>

// create an ECR repository
aws ecr create-repository --repository-name test-repository

// build the docker image
docker build -t test-image .

// sign in to AWS
```

```
aws ecr get-login-password \
  | docker login \
  --username AWS \
  --password-stdin "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com"

// tag the image
docker tag test-image:latest "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/test-
repository:latest"

// push the image to ECR
docker push "$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/test-repository:latest"
```

5. 使用您刚刚创建的 Amazon ECR 镜像以创建 Lambda 函数。有关作为容器的 Lambda 函数的更多信息，请参阅创建定义为容器映像的 Lambda 函数。
6. 确保 Lambda 函数执行角色具有 appconfig: GetConfiguration 权限集。

与 OpenAPI 集成

您可以使用以下 OpenAPI 的 YAML 规范，通过 OpenAPI 生成器等工具创建 SDK。您可以更新此规范以包含“应用程序”、“环境”或“配置”的硬编码值。还可以添加其他路径（如果有多个配置类型）并包含配置架构，以便为 SDK 客户端生成特定于配置的类型化模型。有关 OpenAPI（也称为 Swagger）的详细信息，请参阅 OpenAPI 规范。

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: AppConfig Agent Lambda extension API
  description: An API model for the AppConfig Agent Lambda extension.
servers:
  - url: https://localhost:{port}/
    variables:
      port:
        default:
          '2772'
paths:
  /applications/{Application}/environments/{Environment}/configurations/
{Configuration}:
    get:
      operationId: getConfiguration
      tags:
        - configuration
```

```
    parameters:
      - in: path
        name: Application
        description: The application for the configuration to get. Specify either the
application name or the application ID.
        required: true
        schema:
          type: string
      - in: path
        name: Environment
        description: The environment for the configuration to get. Specify either the
environment name or the environment ID.
        required: true
        schema:
          type: string
      - in: path
        name: Configuration
        description: The configuration to get. Specify either the configuration name
or the configuration ID.
        required: true
        schema:
          type: string
    responses:
      200:
        headers:
          ConfigurationVersion:
            schema:
              type: string
        content:
          application/octet-stream:
            schema:
              type: string
              format: binary
        description: successful config retrieval
      400:
        description: BadRequestException
        content:
          application/text:
            schema:
              $ref: '#/components/schemas/Error'
      404:
        description: ResourceNotFoundException
        content:
          application/text:
```

```
              schema:
                $ref: '#/components/schemas/Error'
        500:
          description: InternalServerException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        502:
          description: BadGatewayException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'
        504:
          description: GatewayTimeoutException
          content:
            application/text:
              schema:
                $ref: '#/components/schemas/Error'

components:
  schemas:
    Error:
      type: string
      description: The response error
```

从 Amazon EC2 实例检索配置数据

您可以使用代理 AWS AppConfig 与在亚马逊弹性计算云 (Amazon EC2) Elastic Compute Cloud Linux 实例上运行的 AWS AppConfig 应用程序集成。代理通过以下方式增强应用程序处理和管理：

- 代理通过使用 AWS Identity and Access Management (IAM) 角色并管理配置数据的本地缓存来代表您进行调用 AWS AppConfig 。通过从本地缓存中提取配置数据，应用程序需要更少的代码更新来管理配置数据，在几毫秒内检索配置数据，并且不受可能中断对此类数据的调用的网络问题的影响。*
- 该代理为检索和解析 AWS AppConfig 功能标记提供了原生体验。
- 该代理开箱即用，提供了缓存策略、轮询间隔和本地配置数据可用性的最佳实践，同时跟踪后续服务调用所需的配置令牌。
- 在后台运行时，代理会定期轮询 AWS AppConfig 数据平面以获取配置数据更新。应用程序可以通过连接到端口 2772（可自定义的默认端口值）上的本地主机并调用 HTTP GET 来检索数据。

*AWS AppConfig 代理会在服务首次检索您的配置数据时缓存数据。因此，检索数据的第一次调用比后续调用慢。

主题

步骤 1：（必需）创建资源并配置权限

要 AWS AppConfig 与 Amazon EC2 实例上运行的应用程序集成，您 AWS AppConfig 必须创建项目和配置数据，包括功能标志或自由格式配置数据。有关更多信息，请参阅 在中创建功能标志和自由格式配置数据 AWS AppConfig。

要检索托管的配置数据 AWS AppConfig，必须将您的应用程序配置为可以访问 AWS AppConfig 数据平面。要向应用程序授予访问权限，请更新分配给 Amazon EC2 实例角色的 IAM 权限策略。具体而言，您必须将 `appconfig:StartConfigurationSession` 和 `appconfig:GetLatestConfiguration` 操作添加到策略中。示例如下：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "appconfig:StartConfigurationSession",
                "appconfig:GetLatestConfiguration"
            ],
            "Resource": "*"
        }
    ]
}
```

有关如何添加权限至策略的信息，请参阅《IAM 用户指南》中的添加和删除 IAM 身份权限。

步骤 2：（必需）在 Amazon EC2 实例上安装和启动 AWS AppConfig 代理

AWS AppConfig 代理托管在由管理的亚马逊简单存储服务 (Amazon S3) Service 存储桶中。 AWS使用以下过程在 Linux 实例上安装最新版本的 agent。如果应用程序分布在多个实例中，则必须在托管应用程序的每个实例上执行此过程。

> **Note**
>
> 请注意以下信息：
>
> - AWS AppConfig 代理适用于运行内核版本 4.15 或更高版本的 Linux 操作系统。不支持基于 Debian 的系统，例如 Ubuntu。
> - 该代理支持 x86_64 和 ARM64 体系结构。
> - 对于分布式应用程序，我们建议将 install 和 startup 命令添加到 Auto Scaling 组的 Amazon EC2 用户数据中。如果这样做，每个实例都会自动运行这些命令。有关更多信息，请参阅 Amazon EC2 用户指南中的启动时在您的 Linux 实例上运行命令。此外，请参阅《Amazon EC2 Auto Scaling 用户指南》中的教程：通过实例元数据检索目标生命周期状态。
> - 本主题中的过程介绍如何通过登录实例来运行命令以执行安装代理等操作。您可以从本地客户机运行命令，并使用 Run Command（这是一项功能）来定位一个或多个实例 AWS Systems Manager。有关更多信息，请参阅 AWS Systems Manager 用户指南中的 AWS Systems Manager Run Command。
> - AWS AppConfig 亚马逊 EC2 Linux 实例上的代理是一项`systemd`服务。

在实例上安装和启动 AWS AppConfig 代理

1. 登录您的 Linux 实例。
2. 打开终端，并使用 x86_64 体系结构的管理员权限运行以下命令：

   ```
   sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/linux/x86_64/latest/aws-appconfig-agent.rpm
   ```

   对于 ARM64 架构，请运行以下命令：

   ```
   sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/linux/arm64/latest/aws-appconfig-agent.rpm
   ```

如果要安装特定版本的 AWS AppConfig Agent，请将 URL `latest` 中的特定版本号替换为特定的版本号。以下是 x86_64 的一个示例：

```
sudo yum install https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/linux/x86_64/2.0.2/aws-appconfig-agent.rpm
```

3. 请运行以下命令以启动代理：

```
sudo systemctl start aws-appconfig-agent
```

4. 请运行以下命令以验证代理是否正在运行。

```
sudo systemctl status aws-appconfig-agent
```

如果成功，该命令将返回类似以下内容的信息：

```
aws-appconfig-agent.service - aws-appconfig-agent
  ...
  Active: active (running) since Mon 2023-07-26 00:00:00 UTC; 0s ago
  ...
```

> **Note**
>
> 要停止代理，请运行以下命令：
>
> ```
> sudo systemctl stop aws-appconfig-agent
> ```

步骤 3：（可选，但建议使用）将日志文件发送到 CloudWatch 日志

默认情况下， AWS AppConfig 代理会将日志发布到 STDERR。Systemd 将 Linux 实例上运行的所有服务的 STDOUT 和 STDERR 重定向到 systemd 日志。如果您只在一两个实例上运行 AWS AppConfig 代理，则可以查看和管理 systemd 日志中的日志数据。更好的解决方案（我们强烈推荐分布式应用程序使用的解决方案）是将日志文件写入磁盘，然后使用 Amazon CloudWatch 代理将日志数据上传到 AWS 云端。此外，您可以将 CloudWatch 代理配置为从您的实例中删除旧的日志文件，这样可以防止您的实例耗尽磁盘空间。

若要启用磁盘日志记录，必须设置 LOG_PATH 环境变量，如 步骤 4：（可选）使用环境变量为 Amazon EC2 配置 AWS AppConfig 代理 中所述。

要开始使用 CloudWatch 代理，请参阅亚马逊 CloudWatch 用户指南中的使用代理从 Amazon EC2 实例和本地服务器收集指标和日志。 CloudWatch 您可以使用 “快速设置”（Systems Manager 的一项功能）来快速安装 CloudWatch 代理。有关更多信息，请参阅 AWS Systems Manager 用户指南中的快速设置主机管理。

> **Warning**
>
> 如果您选择在不使用 CloudWatch 代理的情况下将日志文件写入磁盘，则必须删除旧的日志文件。 AWS AppConfig 代理每小时自动轮换一次日志文件。如果您不删除旧的日志文件，您的实例可能会耗尽磁盘空间。

在实例上安装 CloudWatch 代理后，创建 CloudWatch 代理配置文件。配置文件指示 CloudWatch 代理如何使用 AWS AppConfig 代理日志文件。有关创建 CloudWatch 代理配置文件的更多信息，请参阅创建 CloudWatch 代理配置文件。

将以下`logs`部分添加到实例的 CloudWatch 代理配置文件中，然后保存您的更改：

```
"logs": {
  "logs_collected": {
    "files": {
      "collect_list": [
        {
          "file_path": "/path_you_specified_for_logging",
          "log_group_name": "${YOUR_LOG_GROUP_NAME}/aws-appconfig-agent.log",
          "auto_removal": true
        },
        ...
      ]
    },
    ...
  },
  ...
}
```

如果的值为`true`，`auto_removal`则 CloudWatch 代理会自动删除轮换的 AWS AppConfig 代理日志文件。

步骤 4：（可选）使用环境变量为 Amazon EC2 配置 AWS AppConfig 代理

您可以使用环境变量为 Amazon EC2 配置 AWS AppConfig 代理。要为 systemd 服务设置环境变量，请创建一个插入式单元文件。以下示例说明如何创建嵌入式单元文件以将 AWS AppConfig Agent 日志级别设置为。DEBUG

如何为环境变量创建插入式单元文件的示例

1. 登录您的 Linux 实例。
2. 使用管理员权限打开终端，并运行下述命令。该命令创建一个配置目录：

```
sudo mkdir /etc/systemd/system/aws-appconfig-agent.service.d
```

3. 运行以下命令以创建插入单元文件。将 *file_name* 替换为文件的名称。扩展名必须是 .conf：

```
sudo touch /etc/systemd/system/aws-appconfig-agent.service.d/file_name.conf
```

4. 在插入式单元文件中输入信息。下面的示例添加一个定义环境变量的 Service 部分。该示例将 AWS AppConfig 代理日志级别设置为 DEBUG。

```
[Service]
Environment=LOG_LEVEL=DEBUG
```

5. 运行以下命令以重新加载系统配置。

```
sudo systemctl daemon-reload
```

6. 运行以下命令以重新启动 AWS AppConfig Agent：

```
sudo systemctl restart aws-appconfig-agent
```

您可以通过在嵌入式单元文件中指定以下环境变量来为 Amazon EC2 配置 AWS AppConfig 代理。

环境变量	详细信息	默认值
ACCESS_TOKEN	此环境变量定义从代理 HTTP 服务器请求配置数据时必须提供的令牌。必须在授权类型为 Bearer 的 HTTP 请求授权标	无

环境变量	详细信息	默认值
	头中设置令牌的值。下面是一个例子。 `GET /applications/my_app/... Host: localhost:2772 Authorization: Bearer <token value>`	
BACKUP_DIRECTORY	此环境变量使 AWS AppConfig 代理能够将其检索到的每个配置的备份保存到指定目录中。 **Important** 备份到磁盘的配置未加密。如果您的配置包含敏感数据，AWS AppConfig 建议您在文件系统权限方面采用最小权限原则。有关更多信息，请参阅 AWS AppConfig 中的安全性。	None（无）
HTTP_PORT	此环境变量指定运行代理的 HTTP 服务器的端口。	2772

环境变量	详细信息	默认值
LOG_LEVEL	此环境变量指定代理记录的详细程度。每个级别包括当前级别和所有更高级别。变量是区分大小写的。从最详细到最不详细，日志级别为：debug、info、warn、err和 none。Debug 包含有关代理的详细信息，包括计时信息。	info
LOG_PATH	写入日志的磁盘位置。如果未指定，则日志将写入 stderr。	无
MANIFEST	此环境变量将 AWS AppConfig Agent 配置为利用其他每个配置的功能，例如多帐户检索和将配置保存到磁盘。您可以输入以下值之一： • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" 有关使用这些功能的更多信息，请参阅 其他检索功能。	true
MAX_CONNECTIONS	此环境变量配置代理用于从 AWS AppConfig检索配置的最大连接数。	3

环境变量	详细信息	默认值
POLL_INTERVAL	此环境变量控制代理轮 AWS AppConfig 询更新配置数据的频率。您可以指定间隔的秒数。您还可以指定一个带有时间单位的数字：s 表示秒，m 表示分钟，h 表示小时。如果未指定单位，则代理默认为秒。例如，60 秒、60 秒和 1 分钟会产生相同的轮询间隔。	45 秒
PREFETCH_LIST	此环境变量指定代理启动后立即请求 AWS AppConfig 的配置数据。	无
PRELOAD_BACKUPS	如果设置为`true`， AWS AppConfig 代理会将中找到的配置备份加载BACKUP_DIRECTORY 到内存中，并立即检查服务中是否存在更新的版本。如果设置为`false`，则只有在无法从服务中检索配置数据时（例如，如果您的网络出现问题）， AWS AppConfig 代理才会加载配置备份中的内容。	true
PROXY_HEADERS	此环境变量指定 PROXY_URL 环境变量中引用的代理所需的标头。其值是以逗号分隔的标头列表。每个标头使用以下形式： `"header: value"`	无

环境变量	详细信息	默认值
PROXY_URL	此环境变量指定用于从代理到的连接的代理 URL AWS 服务，包括 AWS AppConfig。HTTPS并支持 HTTP URL。	无
REQUEST_TIMEOUT	此环境变量控制代理等待响应的 AWS AppConfig时间。如果服务没有响应，则请求将失败。 如果请求用于初始数据检索，则代理会向应用程序返回错误。 如果在对更新的数据进行后台检查期间发生超时，代理会记录错误，并在短暂延迟后重试。 您可以指定超时的毫秒数。您还可以指定一个带有时间单位的数字：ms 表示毫秒，s 表示秒。如果未指定单位，则代理默认为毫秒。例如，5000、5000ms 和 5s 会产生相同的请求超时值。	3000 毫秒
ROLE_ARN	此环境变量指定 IAM 角色的亚马逊资源名称 (ARN)。 AWS AppConfig 代理扮演此角色来检索配置数据。	无
ROLE_EXTERNAL_ID	此环境变量指定要与代入角色 ARN 一起使用的外部 ID。	无

环境变量	详细信息	默认值
ROLE_SESSION_NAME	此环境变量指定要与代入的 IAM 角色的凭证关联的会话名称。	无
SERVICE_REGION	此环境变量指定了 AWS AppConfig Agent AWS 区域 用来调用 AWS AppConfig 服务的替代方案。如果未定义，代理将尝试确定当前区域。如果不能，则代理无法启动。	无
WAIT_ON_MANIFEST	此环境变量将 AWS AppConfig Agent 配置为等到清单处理完毕后再完成启动。	true

步骤 5：（必需）检索配置数据

您可以使用 HTTP localhost 调用从 AWS AppConfig 代理中检索配置数据。以下示例将 `curl` 与 HTTP 客户端一起使用。您可以使用应用程序语言支持的任何可用 HTTP 客户端或可用库（包括 AWS SDK）来调用代理。

检索任何已部署配置的完整内容

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name"
```

从 **Feature Flag** 类型的 AWS AppConfig 配置中检索单个标志及其属性

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name"
```

从 **Feature Flag** 类型的 AWS AppConfig 配置访问多个标志及其属性

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two"
```

步骤 6（可选，但建议使用）：自动更新代理 AWS AppConfig

AWS AppConfig 代理会定期更新。为确保您在实例上运行最新版本的 AWS AppConfig Agent，我们建议您将以下命令添加到您的 Amazon EC2 用户数据中。您可以将命令添加到实例或 EC2 Auto Scaling 组上的用户数据中。每次实例启动或重启时，该脚本都会安装并启动最新版本的代理。

```
#!/bin/bash
# install the latest version of the agent
yum install -y https://s3.amazonaws.com/aws-appconfig-downloads/aws-appconfig-agent/
linux/x86_64/latest/aws-appconfig-agent.rpm
# optional: configure the agent
mkdir /etc/systemd/system/aws-appconfig-agent.service.d
echo "${MY_AGENT_CONFIG}" > /etc/systemd/system/aws-appconfig-agent.service.d/
overrides.conf
systemctl daemon-reload
# start the agent
systemctl start aws-appconfig-agent
```

从 Amazon ECS 和 Amazon EKS 检索配置数据

您可以使用代理 AWS AppConfig 与亚马逊弹性容器服务 (Amazon ECS) 和亚马逊 Elastic Kubernetes Service (Amazon EKS) 集成。 AWS AppConfig 该代理充当 sidecar 容器，与您的 Amazon ECS 和 Amazon EKS 容器应用程序一起运行。该代理通过以下方式增强容器化应用程序的处理和管理：

- 代理通过使用 AWS Identity and Access Management (IAM) 角色并管理配置数据的本地缓存来代表您进行调用 AWS AppConfig 。通过从本地缓存中提取配置数据，应用程序需要更少的代码更新来管理配置数据，在几毫秒内检索配置数据，并且不受可能中断对此类数据的调用的网络问题的影响。*
- 该代理为检索和解析 AWS AppConfig 功能标记提供了原生体验。
- 该代理开箱即用，提供了缓存策略、轮询间隔和本地配置数据可用性的最佳实践，同时跟踪后续服务调用所需的配置令牌。
- 在后台运行时，代理会定期轮询 AWS AppConfig 数据平面以获取配置数据更新。容器化应用程序可以通过连接到端口 2772（可自定义的默认端口值）上的本地主机并调用 HTTP GET 来检索数据。
- AWS AppConfig 代理无需重新启动或回收容器即可更新容器中的配置数据。

*AWS AppConfig 代理会在服务首次检索您的配置数据时缓存数据。因此，检索数据的第一次调用比后续调用慢。

主题

开始前的准备工作

要 AWS AppConfig 与容器应用程序集成，必须创建 AWS AppConfig 构件和配置数据，包括功能标志或自由格式配置数据。有关更多信息，请参阅 在中创建功能标志和自由格式配置数据 AWS AppConfig。

要检索托管的配置数据 AWS AppConfig，您的容器应用程序必须配置为可以访问 AWS AppConfig 数据平面。要为您的应用程序授予访问权限，请更新您的容器服务 IAM 角色使用的 IAM 权限策略。具体而言，您必须将 `appconfig:StartConfigurationSession` 和 `appconfig:GetLatestConfiguration` 操作添加到策略中。容器服务 IAM 角色包括以下内容：

- Amazon ECS 任务角色
- Amazon EKS 节点角色
- AWS Fargate (Fargate) 容器执行角色（如果您的 Amazon EKS 容器使用 Fargate 进行计算处理）

有关如何添加权限至策略的信息，请参阅《IAM 用户指南》中的添加和删除 IAM 身份权限。

启动 AWS AppConfig Agent for Amazon ECS 集成

AWS AppConfig Agent sidecar 容器将在您的 Amazon ECS 环境中自动可用。要使用 AWS AppConfig Agent sidecar 容器，必须将其启动。

启动 Amazon ECS（控制台）

1. 在 https://console.aws.amazon.com/ecs/v2 打开控制台。
2. 在导航窗格中，选择 Task definitions（任务定义）。

3. 选择应用程序的任务定义，然后选择最新版本。
4. 选择创建新的修订、创建新的修订。
5. 选择添加更多容器。
6. 在名称中，输入 AWS AppConfig 代理容器的唯一名称。
7. 对于 Image URI，输入：**`public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x`**
8. 对于 基本容器，选择 是。
9. 在 端口映射 部分中，选择 添加端口映射。
10. 对于容器端口，输入 **2772**。

> **Note**
>
> AWS AppConfig 默认情况下，代理在端口 2772 上运行。也可指定不同的端口。

11. 选择创建。Amazon ECS 将创建新的容器修订并显示详细信息。
12. 在导航窗格中，选择 集群，然后在列表中选择应用程序的集群。
13. 在 服务 选项卡上，为应用程序选择服务。
14. 选择更新。
15. 在 部署配置 下，对于 修订，选择最新的修订。
16. 选择更新。Amazon ECS 部署最新的任务定义。
17. 部署完成后，您可以在 “配置和任务” 选项卡上验证 AWS AppConfig 代理是否正在运行。在 任务 选项卡上，选择正在运行的任务。
18. 在 “容器” 部分，确认已列出 AWS AppConfig 代理容器。
19. 要验证 AWS AppConfig 代理是否已启动，请选择日志选项卡。在 A AWS AppConfig gent 容器中找到如下语句：`[appconfig agent] 1970/01/01 00:00:00 INFO serving on localhost:2772`

> **Note**
>
> 您可以通过输入或更改环境变量来调整 AWS AppConfig Agent 的默认行为。有关可用环境变量的更多信息，请参阅 使用环境变量为 Amazon ECS 和 Amazon EKS 配置 AWS AppConfig 代理。有关如何在 Amazon ECS 中更改环境变量的信息，请参阅 Amazon Elastic Container Service 开发人员指南 中的 将环境变量传递到容器。

启动 AWS AppConfig agent for Amazon EKS 集成

AWS AppConfig Agent sidecar 容器将在您的 Amazon EKS 环境中自动可用。要使用 AWS AppConfig Agent sidecar 容器，必须将其启动。以下过程介绍如何使用 Amazon EKS `kubectl` 命令行工具在容器应用程序的 `kubeconfig` 文件中进行更改。有关创建或编辑 `kubeconfig` 文件的更多信息，请参阅 Amazon EKS 用户指南 中的 为 Amazon EKS 集群创建或更新 kubeconfig 文件。

启动 AWS AppConfig 代理（kubectl 命令行工具）

1. 打开您的 `kubeconfig` 文件并验证您的 Amazon EKS 应用程序是否作为单容器部署运行。该文件的内容看起来类似于以下内容：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-application-label
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-application-label
  template:
    metadata:
      labels:
        app: my-application-label
    spec:
      containers:
      - name: my-app
        image: my-repo/my-image
        imagePullPolicy: IfNotPresent
```

2. 将 AWS AppConfig 代理容器定义详细信息添加到 YAML 部署文件中。

```
- name: appconfig-agent
        image: public.ecr.aws/aws-appconfig/aws-appconfig-agent:2.x
        ports:
        - name: http
          containerPort: 2772
          protocol: TCP
```

```
        env:
        - name: SERVICE_REGION
          value: region
        imagePullPolicy: IfNotPresent
```

> **Note**
>
> 请注意以下信息。
>
> - AWS AppConfig 默认情况下，代理在端口 2772 上运行。也可指定不同的端口。
> - 您可以通过输入环境变量来调整 AWS AppConfig Agent 的默认行为。有关更多信息，请参阅 使用环境变量为 Amazon ECS 和 Amazon EKS 配置 AWS AppConfig 代理。
> - 对于 *SERVICE_* REGION，请指定 AWS AppConfig 代理检索配置数据的 AWS 区域 代码（例如us-west-1）。

3. 在 kubectl 工具中运行以下命令，将更改应用于群集。

```
kubectl apply -f my-deployment.yml
```

4. 部署完成后，验证 AWS AppConfig 代理是否正在运行。使用以下命令查看应用程序 Pod 日志文件。

```
kubectl logs -n my-namespace -c appconfig-agent my-pod
```

在 A AWS AppConfig gent 容器中找到如下语句：[appconfig agent] 1970/01/01 00:00:00 INFO serving on localhost:2772

> **Note**
>
> 您可以通过输入或更改环境变量来调整 AWS AppConfig Agent 的默认行为。有关可用环境变量的更多信息，请参阅 使用环境变量为 Amazon ECS 和 Amazon EKS 配置 AWS AppConfig 代理。

使用环境变量为 Amazon ECS 和 Amazon EKS 配置 AWS AppConfig 代理

您可以通过更改 AWS AppConfig 代理容器的以下环境变量来配置代理。

环境变量	详细信息	默认值
ACCESS_TOKEN	此环境变量定义从代理 HTTP 服务器请求配置数据时必须提供的令牌。必须在授权类型为 Bearer 的 HTTP 请求授权标头中设置令牌的值。下面是一个例子。 `GET /applications/my_app/... Host: localhost:2772 Authorization: Bearer <token value>`	无
BACKUP_DIRECTORY	此环境变量使 AWS AppConfig 代理能够将其检索到的每个配置的备份保存到指定目录中。 **Important** 备份到磁盘的配置未加密。如果您的配置包含敏感数据，AWS AppConfig 建议您在文件系统权限方面采用最小权限原则。有关更多信息，请参阅 AWS AppConfig 中的安全性。	None（无）
HTTP_PORT	此环境变量指定运行代理的 HTTP 服务器的端口。	2772

环境变量	详细信息	默认值
LOG_LEVEL	此环境变量指定代理记录的详细程度。每个级别包括当前级别和所有更高级别。变量是区分大小写的。从最详细到最不详细，日志级别为：debug、info、warn、err和 none。 Debug 包含有关代理的详细信息，包括计时信息。	info
MANIFEST	此环境变量将 AWS AppConfig Agent 配置为利用其他每个配置的功能，例如多帐户检索和将配置保存到磁盘。您可以输入以下值之一： • "app:env:manifest-config" • "file:/fully/qualified/path/to/manifest.json" 有关使用这些功能的更多信息，请参阅 其他检索功能。	true
MAX_CONNECTIONS	此环境变量配置代理用于从 AWS AppConfig检索配置的最大连接数。	3

环境变量	详细信息	默认值
POLL_INTERVAL	此环境变量控制代理轮 AWS AppConfig 询更新配置数据的频率。您可以指定间隔的秒数。您还可以指定一个带有时间单位的数字：s 表示秒，m 表示分钟，h 表示小时。如果未指定单位，则代理默认为秒。例如，60 秒、60 秒和 1 分钟会产生相同的轮询间隔。	45 秒
PREFETCH_LIST	此环境变量指定代理启动后立即请求 AWS AppConfig 的配置数据。	无
PRELOAD_BACKUPS	如果设置为`true`， AWS AppConfig 代理会将中找到的配置备份加载BACKUP_DIRECTORY 到内存中，并立即检查服务中是否存在更新的版本。如果设置为`false`，则只有在无法从服务中检索配置数据时（例如，如果您的网络出现问题）， AWS AppConfig 代理才会加载配置备份中的内容。	true
PROXY_HEADERS	此环境变量指定 PROXY_URL 环境变量中引用的代理所需的标头。其值是以逗号分隔的标头列表。每个标头使用以下形式： `"header: value"`	无

环境变量	详细信息	默认值
PROXY_URL	此环境变量指定用于从代理到的连接的代理 URL AWS 服务，包括 AWS AppConfig。HTTPS并支持 HTTP URL。	无
REQUEST_TIMEOUT	此环境变量控制代理等待响应的 AWS AppConfig时间。如果服务没有响应，则请求将失败。 如果请求用于初始数据检索，则代理会向应用程序返回错误。 如果在对更新的数据进行后台检查期间发生超时，代理会记录错误，并在短暂延迟后重试。 您可以指定超时的毫秒数。您还可以指定一个带有时间单位的数字：ms 表示毫秒，s 表示秒。如果未指定单位，则代理默认为毫秒。例如，5000、5000ms 和 5s 会产生相同的请求超时值。	3000 毫秒
ROLE_ARN	此环境变量指定 IAM 角色的亚马逊资源名称 (ARN)。 AWS AppConfig 代理扮演此角色来检索配置数据。	无
ROLE_EXTERNAL_ID	此环境变量指定要与代入角色 ARN 一起使用的外部 ID。	无

环境变量	详细信息	默认值
ROLE_SESSION_NAME	此环境变量指定要与代入的 IAM 角色的凭证关联的会话名称。	无
SERVICE_REGION	此环境变量指定了 AWS AppConfig Agent AWS 区域 用来调用 AWS AppConfig 服务的替代方案。如果未定义，代理将尝试确定当前区域。如果不能，则代理无法启动。	无
WAIT_ON_MANIFEST	此环境变量将 AWS AppConfig Agent 配置为等到清单处理完毕后再完成启动。	true

检索配置数据

您可以使用 HTTP localhost 调用从 AWS AppConfig 代理中检索配置数据。以下示例将 `curl` 与 HTTP 客户端一起使用。您可以使用应用程序语言或可用库支持的任何可用 HTTP 客户端调用代理。

> **Note**
>
> 如果您的应用程序使用正斜杠（例如 “test-backend/test-service”），要检索配置数据，则需要使用 URL 编码。

检索任何已部署配置的完整内容

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name"
```

从 **`Feature Flag`** 类型的 AWS AppConfig 配置中检索单个标志及其属性

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?flag=flag_name"
```

从 **Feature Flag** 类型的 AWS AppConfig 配置访问多个标志及其属性

```
$ curl "http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name?
flag=flag_name_one&flag=flag_name_two"
```

其他检索功能

AWS AppConfig Agent 还提供以下附加功能来帮助您检索应用程序的配置。

- 多账号检索：使用主账户或检索 AWS 账户 中的 AWS AppConfig 代理从多个供应商账户检索配置数据。
- 将配置副本写入磁盘：使用 AWS AppConfig 代理将配置数据写入磁盘。此功能使客户能够使用从磁盘读取配置数据以进行集成的应用程序 AWS AppConfig。

关于代理清单

要启用这些 AWS AppConfig 代理功能，您需要创建清单。清单是您提供的一组配置数据，用于控制代理可以执行的操作。清单是用 JSON 编写的。它包含一组顶级密钥，这些密钥对应于您部署的不同配置 AWS AppConfig。

一份清单可以包含多个配置。此外，清单中的每个配置都可以标识用于指定配置的一个或多个代理功能。清单的内容使用以下格式：

```
{
    "application_name:environment_name:configuration_name": {
        "agent_feature_to_enable_1": {
            "feature-setting-key": "feature-setting-value"
        },
        "agent_feature_to_enable_2": {
            "feature-setting-key": "feature-setting-value"
        }
    }
}
```

以下是具有两种配置的清单的 JSON 示例。第一个配置 (*MyApp*) 不使用任何 AWS AppConfig 代理功能。第二种配置 (*my2ndApp*) 使用将配置副本写入磁盘和多账户检索功能：

```
{
```

```
        "MyApp:Test:MyAllowListConfiguration": {},

        "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
            "credentials": {
                "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
                "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
                "roleSessionName": "AwsAppConfigAgent",
                "credentialsDuration": "2h"
            },
            "writeTo": {
                "path": "/tmp/aws-appconfig/my-2nd-app/beta/my-enable-payments-feature-
flag-configuration.json"
            }
        }
    }
```

如何提供代理清单

您可以将清单作为文件存储在 AWS AppConfig 代理可以读取的位置。或者，您可以将清单存储为 AWS AppConfig 配置并将代理指向它。要提供代理清单，必须使用以下值之一设置MANIFEST环境变量：

清单位置	环境变量值	应用场景
文件	文件：/path/to/to/agent-manifest.j	如果您的清单不经常更改，请使用此方法。
AWS AppConfig 配置	################	使用此方法进行动态更新。您可以更新和部署 AWS AppConfig 作为配置存储在中的清单，方法与存储其他 AWS AppConfig 配置的方式相同。
环境变量	清单内容 (JSON)	如果您的清单不经常更改，请使用此方法。此方法在容器环境中很有用，在容器环境中，设置环境变量比公开文件更容易。

有关为 AWS AppConfig Agent 设置变量的更多信息，请参阅与您的用例相关的主题：

- 配置 AWS AppConfig 代理 Lambda 扩展
- 在 Amazon EC2 上使用 AWS AppConfig 代理
- 在 Amazon ECS 和 Amazon EKS 上使用 AWS AppConfig 代理

多账号检索

您可以通过在 AWS AppConfig 代理清单中输入凭据覆盖 AWS 账户 来配置 AWS AppConfig 代理以从多个中检索配置。证书覆盖包括 (IAM) 角色的 Amazon 资源名称 AWS Identity and Access Management (ARN)、角色 ID、会话名称以及代理可以担任该角色的时长。

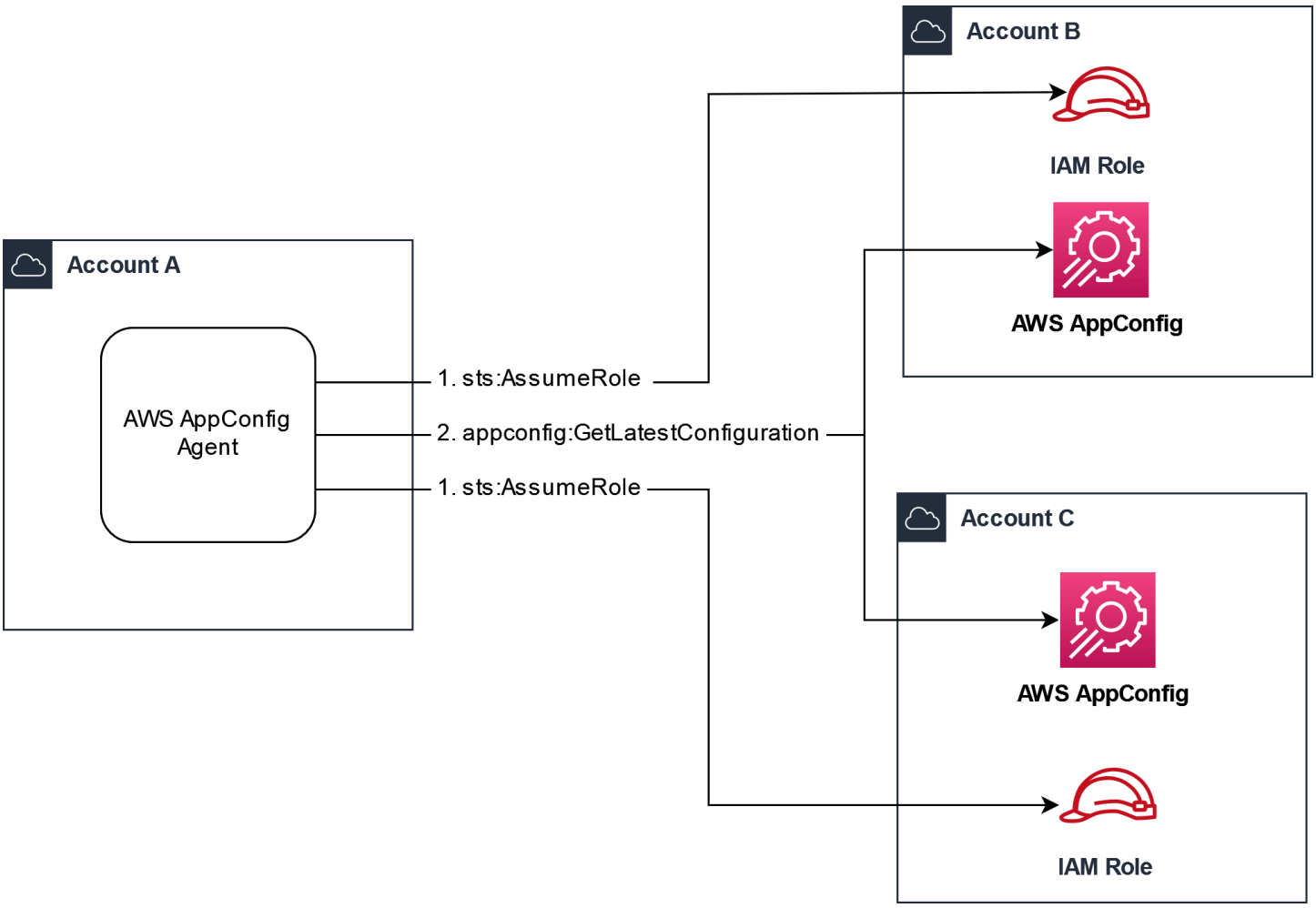
您可以在清单的 “凭证” 部分中输入这些详细信息。“凭证” 部分使用以下格式：

```
{
    "application_name:environment_name:configuration_name": {
        "credentials": {
            "roleArn": "arn:partition:iam::account_ID:role/roleName",
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
        }
    }
}
```

示例如下：

```
{
    "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
        "credentials": {
            "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
            "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
            "roleSessionName": "AWSAppConfigAgent",
            "credentialsDuration": "2h"
        }
    }
}
```

在检索配置之前，代理会从清单中读取配置的凭证详细信息，然后担任为该配置指定的 IAM 角色。您可以在单个清单中为不同的配置指定一组不同的凭据替换。下图显示 AWS AppConfig 代理在账户 A（检索账户）中运行时，如何扮演为账户 B 和账户 C（供应商账户）指定的不同角色，然后调用GetLatest配置 API 操作来检索在这些账户中 AWS AppConfig 运行的配置数据：



配置从供应商账户检索配置数据的权限

AWS AppConfig 在检索账户中运行的代理需要权限才能从供应商帐户检索配置数据。您可以通过在每个供应商账户中创建 AWS Identity and Access Management (IAM) 角色来授予代理权限。 AWS AppConfig 检索账户中的代理扮演此角色从供应商账户获取数据。完成本节中的步骤，创建 IAM 权限策略、IAM 角色并将代理覆盖添加到清单中。

开始前的准备工作

在 IAM 中创建权限策略和角色之前，请收集以下信息。

- 每个的 ID AWS 账户。检索账户是将调用其他账户获取配置数据的账户。供应商账户是将配置数据提供给检索账户的账户。
- 检索账户 AWS AppConfig 中使用的 IAM 角色的名称。以下是默认情况下使用的 AWS AppConfig角色列表：
  - 对于亚马逊弹性计算云 (Amazon EC2) AWS AppConfig ，使用实例角色。

- 对于 AWS Lambda， AWS AppConfig 使用 Lambda 执行角色。
- 对于亚马逊弹性容器服务 (Amazon ECS) 和亚马逊 Elastic Kubernetes Service (Amazon AWS AppConfig EKS)，使用容器角色。

如果您通过指定ROLE_ARN环境变量将 A AWS AppConfig gent 配置为使用不同的 IAM 角色，请记下该名称。

创建权限策略

使用以下过程通过 IAM 控制台创建权限策略。完成将为检索帐户提供配置数据的每个 AWS 账户 步骤中的步骤。

创建 IAM policy

1. 登录供应商账户。 AWS Management Console
2. 访问：https://console.aws.amazon.com/iam/，打开 IAM 控制台。
3. 在导航窗格中，选择 Policies (策略)，然后选择 Create policy (创建策略)。
4. 选择 JSON 选项。
5. 在策略编辑器中，将默认 JSON 替换为以下策略声明。使用供应商账户详细信息更新每个#######。

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "appconfig:StartConfigurationSession",
            "appconfig:GetLatestConfiguration"
        ],
        "Resource":
 "arn:partition:appconfig:region:vendor_account_ID:application/
vendor_application_ID/environment/vendor_environment_ID/
configuration/vendor_configuration_ID"
    }
   ]
}
```

示例如下：

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "appconfig:StartConfigurationSession",
            "appconfig:GetLatestConfiguration"
        ],
        "Resource": "arn:aws:appconfig:us-east-2:111122223333:application/abc123/
environment/def456/configuration/hij789"
    }
   ]
}
```

6. 选择下一步。
7. 在策略名称字段中，输入名称。
8. （可选）在 “添加标签” 中，添加一个或多个标签键值对来组织、跟踪或控制此策略的访问权限。
9. 选择 创建策略。系统将让您返回到 Policies 页面。
10. 在每个 AWS 账户 将为检索帐户提供配置数据的操作中重复此过程。

创建 IAM 角色

使用以下过程通过 IAM 控制台创建 IAM 角色。完成将为检索帐户提供配置数据的每个 AWS 账户 步骤中的步骤。

创建 IAM 角色

1. 登录供应商账户。 AWS Management Console
2. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
3. 在导航窗格中，选择角色，然后选择创建策略。
4. 对于 Trusted entity type（可信实体类型），选择 AWS 账户。
5. 在该AWS 账户部分中，选择其他 AWS 账户。
6. 在 “账户 ID” 字段中，输入检索账户 ID。
7. （可选）作为此代入角色的安全最佳实践，请选择 “需要外部 ID” 并输入字符串。
8. 选择下一步。
9. 在添加权限页面上，使用搜索字段查找您在上一个过程中创建的策略。选中其名称旁边的复选框。

10. 选择下一步。
11. 对于角色名称，输入一个名称。
12. （可选）对于描述，输入描述。
13. 对于 “步骤 1：选择可信实体”，选择 “编辑”。将默认 JSON 信任策略替换为以下策略。使用取回账户中的信息更新每个*#######*。

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

14. （可选）在 Tags（标签）中添加一个或多个标签密钥值对，以组织、跟踪或控制此角色的访问权限。
15. 选择 Create role (创建角色)。系统将让您返回到 角色 页面。
16. 搜索您刚刚创建的角色。选择该存储桶。在 ARN 部分中，复制 ARN。您将在下一个步骤中指定此信息。

向清单中添加凭证覆盖

在供应商账户中创建 IAM 角色后，更新取回账户中的清单。具体而言，添加证书块和用于从供应商账户检索配置数据的 IAM 角色 ARN。以下是 JSON 格式：

```
{
    "vendor_application_name:vendor_environment_name:vendor_configuration_name": {
        "credentials": {
            "roleArn":
 "arn:partition:iam::vendor_account_ID:role/name_of_role_created_in_vendor_account",
            "roleExternalId": "string",
            "roleSessionName": "string",
            "credentialsDuration": "time_in_hours"
        }
```

```
    }
}
```

示例如下：

```
{
    "My2ndApp:Beta:MyEnableMobilePaymentsFeatureFlagConfiguration": {
        "credentials": {
            "roleArn": "arn:us-west-1:iam::123456789012:role/MyTestRole",
            "roleExternalId": "00b148e2-4ea4-46a1-ab0f-c422b54d0aac",
            "roleSessionName": "AwsAppConfigAgent",
            "credentialsDuration": "2h"
        }
    }
}
```

验证多账户检索是否有效

您可以通过查看代理日志来验证该代理是否能够从多个账户检索配置数据。 AWS AppConfig 检索到 'YourApplicationName:YourEnvironmentName:YourConfigurationName' 的初始数据的INFO级别日志是成功检索的最佳指标。如果检索失败，您应该会看到一个指示失败原因的ERROR关卡日志。以下是成功从供应商账户检索的示例：

```
[appconfig agent] 2023/11/13 11:33:27 INFO AppConfig Agent 2.0.x
[appconfig agent] 2023/11/13 11:33:28 INFO serving on localhost:2772
[appconfig agent] 2023/11/13 11:33:28 INFO retrieved initial data for
 'MyTestApplication:MyTestEnvironment:MyDenyListConfiguration' in XX.Xms
```

将配置副本写入磁盘

您可以将 AWS AppConfig Agent 配置为以纯文本形式自动将配置副本存储到磁盘。此功能使客户能够使用从磁盘读取配置数据以进行集成的应用程序 AWS AppConfig。

此功能不是为用作配置备份功能而设计的。 AWS AppConfig 代理无法读取复制到磁盘的配置文件。如果要将配置备份到磁盘，请参阅在 Amazon EC2 中使用 AWS AppConfig 代理或在 Amazon ECS BACKUP_DIRECTORY 和 Amazon EKS 中使用 AWS AppConfig 代理的和PRELOAD_BACKUP环境变量。

> **Warning**
>
> 请注意有关此功能的以下重要信息：

- 保存到磁盘的配置以纯文本形式存储，便于用户阅读。请勿为包含敏感数据的配置启用此功能。
- 此功能写入本地磁盘。对文件系统权限使用最小权限原则。有关更多信息，请参阅 实施最低权限访问。

启用将配置复制到磁盘

1. 编辑清单。
2. 选择 AWS AppConfig 要写入磁盘的配置并添加`writeTo`元素。示例如下：

```
{
    "application_name:environment_name:configuration_name": {
        "writeTo": {
            "path": "path_to_configuration_file"
        }
    }
}
```

示例如下：

```
{
    "MyTestApp:MyTestEnvironment:MyNewConfiguration": {
        "writeTo": {
            "path": "/tmp/aws-appconfig/mobile-app/beta/enable-mobile-payments"
        }
    }
}
```

3. 保存您的更改。每次部署新的配置数据时，configuration.json 文件都会更新。

验证将配置副本写入磁盘是否正常工作

您可以通过查看 AWS AppConfig 代理日志来验证配置副本是否正在写入磁盘。带有 “INFO 将配置' *####*：*##*：*##* '写入*####” #INFO###### AWS AppConfig ##########*。

示例如下：

```
[appconfig agent] 2023/11/13 11:33:27 INFO AppConfig Agent 2.0.x
```

```
[appconfig agent] 2023/11/13 11:33:28 INFO serving on localhost:2772
[appconfig agent] 2023/11/13 11:33:28 INFO retrieved initial data for
 'MobileApp:Beta:EnableMobilePayments' in XX.Xms
[appconfig agent] 2023/11/13 17:05:49 INFO wrote configuration
 'MobileApp:Beta:EnableMobilePayments' to /tmp/configs/your-app/your-env/your-
config.json
```

AWS AppConfig 代理本地开发

AWS AppConfig 代理支持本地开发模式。如果启用本地开发模式，代理将从磁盘上的指定目录中读取配置数据。它不会从中检索配置数据 AWS AppConfig。您可以通过更新指定目录中的文件来模拟配置部署。对于以下用例，我们建议使用本地开发模式：

- 在使用部署不同的配置版本之前，先对其进行测试 AWS AppConfig。
- 在向代码存储库提交更改之前，请测试新功能的不同配置选项。
- 测试不同的配置方案，以验证它们是否能按预期运行。

> **Warning**
>
> 不要在生产环境中使用本地开发模式。此模式不支持重要的 AWS AppConfig 安全功能，例如部署验证和自动回滚。

使用以下步骤将 AWS AppConfig Agent 配置为本地开发模式。

将 AWS AppConfig Agent 配置为本地开发模式

1. 使用为您的计算环境描述的方法安装代理。 AWS AppConfig 代理使用以下内容 AWS 服务：
   - AWS Lambda
   - Amazon EC2
   - 亚马逊 ECS 和亚马逊 EKS
2. 如果代理正在运行，请将其停止。
3. `LOCAL_DEVELOPMENT_DIRECTORY`添加到环境变量列表中。在文件系统上指定一个为代理提供读取权限的目录。例如，`/tmp/local_configs`。
4. 在目录中创建文件。文件名必须使用以下格式：

```
application_name:environment_name:configuration_profile_name
```

示例如下：

```
Mobile:Development:EnableMobilePaymentsFeatureFlagConfiguration
```

> **Note**
>
> （可选）您可以根据您为文件提供的扩展名控制代理为配置数据返回的内容类型。例如，如果您使用.json 扩展名命名文件，则代理会在应用程序请求application/json时返回的内容类型为。如果省略扩展名，代理将使用application/octet-stream该内容类型。如果您需要精确的控制，可以提供格式的扩展名.*type*%*subtype*。代理将返回的内容类型为.type/subtype。

5. 运行以下命令以重新启动代理并请求配置数据。

```
curl http://localhost:2772/applications/application_name/
environments/environment_name/configurations/configuration_name
```

代理会按照为代理指定的轮询间隔检查本地文件是否有更改。如果未指定轮询间隔，代理将使用 45 秒的默认间隔。这种轮询间隔检查可确保代理在本地开发环境中的行为与配置为与 AWS AppConfig 服务交互时的行为相同。

> **Note**
>
> 要部署本地开发配置文件的新版本，请使用新数据更新该文件。

通过直接调用 API 获取配置

您的应用程序首先使用会话 API 操作建立配置会话来检索配置数据。StartConfiguration然后，您的会话客户端会定期调用 Con GetLatestfig uration 以检查和检索最新的可用数据。

在调用 StartConfigurationSession 时，代码发送以下信息：

- 会话跟踪的 AWS AppConfig 应用程序、环境和配置文件的标识符（ID 或名称）。

- （可选）会话的客户端在调用至 GetLatestConfiguration 之间必须等待的最短时间。

作为响应， AWS AppConfig 提供InitialConfigurationToken给会话的客户端，并在它第一次调GetLatestConfiguration用该会话时使用。

> **Important**
>
> 此令牌在第一次调用 GetLatestConfiguration 时只能使用一次。您必须在对 GetLatestConfiguration 的每次后续调用中在 GetLatestConfiguration 响应 (NextPollConfigurationToken) 中使用新令牌。为了支持长时间轮询用例，令牌的有效期最长为 24 小时。如果 GetLatestConfiguration 调用使用过期的令牌，系统将返回 BadRequestException。

调用 GetLatestConfiguration 时，您的客户端代码会发送它所拥有的最新 ConfigurationToken 值,并接收响应：

- NextPollConfigurationToken：下次调用 GetLatestConfiguration 时要使用的 ConfigurationToken 值。
- NextPollIntervalInSeconds：客户端在下次调用 GetLatestConfiguration 之前应等待的持续时间。
- 配置：用于会话的最新数据。如果客户端已有最新版本的配置，则此字段可能为空。

> **Important**
>
> 请注意以下重要信息。
>
> - 每个应用程序、环境、配置文件和客户端只能调用一次会StartConfiguration话 API，才能与服务建立会话。这通常在应用程序启动时或首次检索配置之前完成。
> - 如果您的配置是使用 KmsKeyIdentifier 部署的，则接收配置的请求必须包含调用 kms:Decrypt 的权限。有关更多信息，请参阅 AWS Key Management Service API 参考 中的 Decrypt。
> - 以前用于检索配置数据的 API 操作 GetConfiguration 已弃用。GetConfiguration API 接口不支持加密配置。

检索配置示例

以下 AWS CLI 示例演示如何使用数据StartConfigurationSession和 GetLatestConfiguration API 操作检索配置 AWS AppConfig 数据。第一个命令启动配置会话。此调用包括 AWS AppConfig 应用程序、环境和配置文件的 ID（或名称）。API 返回用于获取配置数据的 InitialConfigurationToken。

```
aws appconfigdata start-configuration-session \
    --application-identifier application_name_or_ID \
    --environment-identifier environment_name_or_ID \
    --configuration-profile-identifier configuration_profile_name_or_ID
```

系统使用以下格式的信息进行响应。

```
{
   "InitialConfigurationToken": initial configuration token
}
```

启动会话后，使用InitialConfiguration令牌调用GetLatest配置来获取您的配置数据。配置数据将保存到 mydata.json 文件中。

```
aws appconfigdata get-latest-configuration \
    --configuration-token initial configuration token mydata.json
```

对 GetLatestConfiguration 的第一次调用使用从 StartConfigurationSession 获取的 ConfigurationToken。返回信息如下。

```
{
    "NextPollConfigurationToken" : next configuration token,
    "ContentType" : content type of configuration,
    "NextPollIntervalInSeconds" : 60
}
```

对 GetLatestConfiguration 的后续调用必须提供上一个响应的 NextPollConfigurationToken。

```
aws appconfigdata get-latest-configuration \
    --configuration-token next configuration token mydata.json
```

Important

请注意有关 GetLatestConfiguration API 操作的以下重要详细信息：

- GetLatestConfiguration 响应包括显示配置数据的 Configuration 部分。仅当系统找到新的或更新的配置数据时，才会显示 Configuration 部分。如果系统找不到新的或更新的配置数据，则 Configuration 数据为空。
- 在来自 GetLatestConfiguration 的每个响应中，您都会收到一个新的 ConfigurationToken。
- 我们建议您根据预算、配置部署的预期频率以及配置目标数以调整 GetLatestConfiguration API 调用的轮询频率。

使用扩展程序扩展工作流

在创建或部署配置 AWS AppConfig 的工作流程中，扩展可以增强您在不同时刻注入逻辑或行为的能力。例如，您可以使用扩展程序来执行以下类型的任务（仅举几例）：

- 在部署配置文件时向 Amazon Simple Notification Service (Amazon SNS) 主题发送通知。
- 在部署开始之前，清理配置文件中敏感数据的内容。
- 每当对功能标志进行更改时，都会创建或更新 Atlassian Jira 问题。
- 在开始部署时，将服务或数据源中的内容合并到配置数据中。
- 每当部署配置时，将配置备份到 Amazon Simple Storage Service (Amazon S3) 存储桶。

您可以将这些类型的任务与 AWS AppConfig 应用程序、环境和配置文件相关联。

内容

关于 AWS AppConfig 扩展

本主题介绍 AWS AppConfig 扩展概念和术语。这些信息将在设置和使用 AWS AppConfig 扩展程序所需的每个步骤的背景下进行讨论。

主题

第 1 步：确定要对扩展程序执行哪些操作

您是否想在 AWS AppConfig 部署完成时收到向 Slack 发送消息的 webhook 的通知？ 您是否希望在部署配置之前将配置文件备份到 Amazon Simple Storage Service (Amazon S3) 存储桶？ 是否要在部署

配置之前清理配置数据中的敏感信息？ 您可以使用扩展程序来执行这些类型的任务以及更多任务。您可以创建自定义扩展，也可以使用随附 AWS AppConfig的 AWS 创作扩展。

> **Note**
>
> 对于大多数用例，要创建自定义扩展，必须创建一个 AWS Lambda 函数来执行扩展中定义的任何计算和处理。有关更多信息，请参阅 演练：创建自定义扩展 AWS AppConfig。

以下 AWS 编写的扩展可以帮助您将配置部署与其他服务快速集成。您可以在 AWS AppConfig 控制台中使用这些扩展，也可以直接从 AWS CLI AWS Tools for PowerShell、或 SDK 调用扩展 API 操作。

扩展程序	描述
Amazon CloudWatch 显然是 A/B 测试	此扩展允许您的应用程序在本地为用户会话分配变体，而不是通过调用EvaluateFeature操作。有关更多信息，请参阅 使用 Amazon CloudWatch Evently 扩展程序。
AWS AppConfig 部署事件到 EventBridge	部署配置时，此扩展将事件发送到 EventBridge 默认事件总线。
AWS AppConfig 向亚马逊简单通知服务 (Amazon SNS) 部署事件 Simple Notification SSERVICE	此扩展程序将消息发送到您在部署配置时指定的 Amazon SNS 主题。
AWS AppConfig 向亚马逊简单队列服务 (Amazon SQS) 部署事件 Simple Queue SQS	部署配置时，此扩展程序会将消息排入 Amazon SQS 队列的队列中。
集成扩展程序 - Atlassian Jira	每当你 AWS AppConfig 对功能标志进行更改时，此扩展都允许创建和更新问题。

步骤 2：确定扩展程序的运行时间

扩展程序定义了它在 AWS AppConfig 工作流程中执行的一个或多个操作。例如， AWS 创作的`AWS AppConfig deployment events to Amazon SNS`扩展程序包括向 Amazon SNS 主题发送通知的操作。每个操作都是在您与之交互时调用的， AWS AppConfig 或者 AWS AppConfig 是在代表您执行流程时调用的。这些被称为行动要点。 AWS AppConfig 扩展支持以下操作要点：

- PRE_CREATE_HOSTED_CONFIGURATION_VERSION
- PRE_START_DEPLOYMENT
- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_STEP
- ON_DEPLOYMENT_BAKING
- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

在PRE_*操作点上配置的扩展操作将在请求验证之后但在 AWS AppConfig 执行与操作点名称对应的活动之前应用。这些操作调用与请求同时处理。如果发出多个请求，则操作调用将按顺序运行。另请注意，PRE_* 操作点接收并可以更改配置的内容。 PRE_* 操作点还可以响应错误并防止操作发生。

扩展也可以使用ON_*操作点与 AWS AppConfig 工作流程并行运行。 ON_*操作点是异步调用的。ON_*操作点不接收配置的内容。如果扩展程序在 ON_* 操作点期间遇到错误，服务将忽略该错误并继续工作流。

步骤 3：创建扩展程序关联

要创建扩展或配置创 AWS 作的扩展，您需要定义在使用特定 AWS AppConfig 资源时调用扩展的操作点。例如，您可以选择运行 AWS AppConfig deployment events to Amazon SNS 扩展程序，并在为特定应用程序启动配置部署时接收有关 Amazon SNS 主题的通知。定义哪些操作点调用特定 AWS AppConfig 资源的扩展称为扩展关联。扩展关联是扩展与 AWS AppConfig 资源（例如应用程序或配置配置文件）之间的指定关系。

单个 AWS AppConfig 应用程序可以包含多个环境和配置文件。如果将扩展程序与应用程序或环境相关联，则会为与应用程序或环境资源相关的所有工作流程 AWS AppConfig 调用该扩展（如果适用）。

例如，假设你有一个名为的 AWS AppConfig 应用程序 MobileApps ，其中包含一个名为的配置文件 AccessList。假设该 MobileApps 应用程序包括测试版、集成和生产环境。您可以为创 AWS 作的 Amazon SNS 通知扩展信息创建扩展关联，并将该扩展程序与应用程序关联。 MobileApps 每当将应用程序的配置部署到三个环境中的任何一个时，都会调用 Amazon SNS 通知扩展程序。

> **Note**
>
> 您不必创建扩展即可使用创 AWS 作的扩展，但必须创建扩展关联。

步骤 4：部署配置并验证是否执行了扩展程序操作

创建关联后，在创建托管配置或部署配置时，会 AWS AppConfig 调用扩展并执行指定的操作。调用扩展时，如果系统在PRE-*操作时遇到错误，则 AWS AppConfig 返回有关该错误的信息。

使用 AWS 创作的扩展程序

AWS AppConfig 包括以下 AWS 创作的扩展。这些扩展可以帮助您将 AWS AppConfig 工作流程与其他服务集成。您可以直接从、 AWS Management Console 或 SDK 调用扩展 API 操作 AWS CLI AWS Tools for PowerShell，在或中使用这些扩展。

扩展程序	描述
Amazon CloudWatch 显然是 A/B 测试	此扩展允许您的应用程序在本地为用户会话分配变体，而不是通过调用EvaluateFeature操作。有关更多信息，请参阅 使用 Amazon CloudWatch Eviently 扩展程序。
AWS AppConfig 部署事件到 EventBridge	部署配置时，此扩展将事件发送到 EventBridge 默认事件总线。
AWS AppConfig 向亚马逊简单通知服务 (Amazon SNS) 部署事件 Simple Notification SSERVICE	此扩展程序将消息发送到您在部署配置时指定的 Amazon SNS 主题。
AWS AppConfig 向亚马逊简单队列服务 (Amazon SQS) 部署事件 Simple Queue SQS	部署配置时，此扩展程序会将消息排入 Amazon SQS 队列的队列中。
集成扩展程序 - Atlassian Jira	每当你 AWS AppConfig 对功能标志进行更改时，此扩展都允许创建和更新问题。

使用 Amazon CloudWatch Eviently 扩展程序

在推出新功能时，您可以使用 Amazon CloudWatch Evidently 向指定比例的用户提供新功能，从而安全地验证新功能。您可以监控新功能的性能，以帮助您决定何时向用户增加流量。这有助于您在完全启动该功能之前，降低风险并识别意外后果。您还可以进行 A/B 实验，以根据证据和数据制定功能设计决策。

CloudWatch Evicently 的 AWS AppConfig 扩展允许您的应用程序在本地为用户会话分配变体，而不是通过调用EvaluateFeature操作。本地会话可以降低 API 调用带来的延迟和可用性风险。有关如何配置和使用扩展程序的信息，请参阅 A mazon CloudWatch 用户指南中的使用 CloudWatch Evidently 执行启动和 A/B 实验。

与 AWS AppConfig deployment events to Amazon EventBridge 扩展程序协作

该AWS AppConfig deployment events to Amazon EventBridge扩展是 AWS 编写的扩展，可帮助您监控 AWS AppConfig 配置部署工作流程并对其采取行动。每当部署配置时，扩展都会向 EventBridge 默认事件总线发送事件通知。将扩展程序与某个 AWS AppConfig 应用程序、环境或配置文件关联后，在每次配置部署开始、结束和回滚之后，都会向事件总线 AWS AppConfig 发送事件通知。

如果您想更好地控制哪些操作点会发送 EventBridge 通知，可以创建自定义扩展插件，并在 URI 字段中输入 EventBridge 默认事件总线 Amazon 资源名称 (ARN)。有关创建扩展程序的信息，请参阅 演练：创建自定义扩展 AWS AppConfig。

> **Important**
>
> 此扩展仅支持 EventBridge 默认事件总线。

使用扩展程序

要使用该AWS AppConfig deployment events to Amazon EventBridge扩展，首先要通过创建扩展关联将扩展附加到您的一个 AWS AppConfig 资源。您可以使用 AWS AppConfig 控制台或CreateExtensionAssociationAPI 操作创建关联。创建关联时，您可以指定 AWS AppConfig 应用程序、环境或配置文件的 ARN。如果将扩展程序关联到应用程序或环境，则会为指定应用程序或环境中包含的任何配置文件发送事件通知。

创建关联后，部署指定 AWS AppConfig 资源的配置时，会 AWS AppConfig 调用该扩展程序并根据扩展中指定的操作点发送通知。

> **Note**
>
> 此扩展程序由以下操作点调用：
>
> - ON_DEPLOYMENT_START

- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

您无法自定义此扩展程序的操作点。若要调用不同的操作点，可以创建自己的扩展程序。有关更多信息，请参阅 演练：创建自定义扩展 AWS AppConfig。

使用以下过程通过 AWS Systems Manager 控制台或控制台创建 AWS AppConfig 扩展关联 AWS CLI。

创建扩展程序关联（控制台）

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，请选择 AWS AppConfig。
3. 在 扩展程序 选项卡上，选择 添加到资源。
4. 在扩展资源详细信息部分的资源类型中，选择一种 AWS AppConfig 资源类型。根据您选择的资源， AWS AppConfig 会提示您选择其他资源。
5. 选择创建与资源的关联。

以下是调用扩展程序 EventBridge 时发送到的示例事件。

```
{
   "version":"0",
   "id":"c53dbd72-c1a0-2302-9ed6-c076e9128277",
   "detail-type":"On Deployment Complete",
   "source":"aws.appconfig",
   "account":"111122223333",
   "time":"2022-07-09T01:44:15Z",
   "region":"us-east-1",
   "resources":[
      "arn:aws:appconfig:us-east-1:111122223333:extensionassociation/z763ff5"
   ],
   "detail":{
      "InvocationId":"5tfjcig",
       "Parameters":{

      },
```

```
      "Type":"OnDeploymentComplete",
      "Application":{
         "Id":"ba8toh7",
         "Name":"MyApp"
      },
      "Environment":{
         "Id":"pgil2o7",
         "Name":"MyEnv"
      },
      "ConfigurationProfile":{
         "Id":"ga3tqep",
         "Name":"MyConfigProfile"
      },
      "DeploymentNumber":1,
      "ConfigurationVersion":"1"
   }
}
```

与 AWS AppConfig deployment events to Amazon SNS 扩展程序协作

该AWS AppConfig deployment events to Amazon SNS扩展是 AWS 编写的扩展，可帮助您监控 AWS AppConfig 配置部署工作流程并对其采取行动。每当部署配置时，该扩展程序都会将消息发布到 Amazon SNS 主题。将扩展程序与您的一个 AWS AppConfig 应用程序、环境或配置文件关联后，在每次配置部署开始、结束和回滚之后，都会向该主题 AWS AppConfig 发布一条消息。

如果您希望更好地控制哪些操作点发送 Amazon SNS 通知，您可以创建自定义扩展程序，并在 URI 字段中输入 Amazon SNS 主题 Amazon 资源名称（ARN）。有关创建扩展程序的信息，请参阅 演练：创建自定义扩展 AWS AppConfig。

使用扩展程序

本节介绍如何使用 AWS AppConfig deployment events to Amazon SNS 扩展程序。

步骤 1：配置 AWS AppConfig 为向主题发布消息

向 Amazon SNS 主题添加访问控制策略，授予 AWS AppConfig (appconfig.amazonaws.com) 发布权限 (sns:Publish)。有关更多信息，请参阅 Amazon SNS 访问控制的示例案例。

步骤 2：创建扩展程序关联

通过创建扩展关联，将扩展附加到您的一个 AWS AppConfig 资源。您可以使用 AWS AppConfig 控制台或 CreateExtensionAssociationAPI 操作创建关联。创建关联时，您可以指定 AWS AppConfig 应用程序、环境或配置文件的 ARN。如果将扩展程序关联到应用程序或环境，则会针对指定应用程序或环境中包含的任何配置文件发送通知。在创建关联时，您必须为 `topicArn` 参数输入一个值，该参数包含要使用的 Amazon SNS 主题的 ARN。

创建关联后，部署指定 AWS AppConfig 资源的配置时，会 AWS AppConfig 调用该扩展程序并根据扩展中指定的操作点发送通知。

> **Note**
>
> 此扩展程序由以下操作点调用：
>
> - `ON_DEPLOYMENT_START`
> - `ON_DEPLOYMENT_COMPLETE`
> - `ON_DEPLOYMENT_ROLLED_BACK`
>
> 您无法自定义此扩展程序的操作点。若要调用不同的操作点，可以创建自己的扩展程序。有关更多信息，请参阅 演练：创建自定义扩展 AWS AppConfig。

使用以下过程通过 AWS Systems Manager 控制台或控制台创建 AWS AppConfig 扩展关联 AWS CLI。

创建扩展程序关联（控制台）

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，请选择 AWS AppConfig。
3. 在 扩展程序 选项卡上，选择 添加到资源。
4. 在扩展资源详细信息部分的资源类型中，选择一种 AWS AppConfig 资源类型。根据您选择的资源， AWS AppConfig 会提示您选择其他资源。
5. 选择创建与资源的关联。

以下是调用扩展程序时发送到 Amazon SNS 主题的消息示例。

```
{
```

```
    "Type": "Notification",
    "MessageId": "ae9d702f-9a66-51b3-8586-2b17932a9f28",
    "TopicArn": "arn:aws:sns:us-east-1:111122223333:MySNSTopic",
    "Message": {
        "InvocationId": "7itcaxp",
        "Parameters": {
            "topicArn": "arn:aws:sns:us-east-1:111122223333:MySNSTopic"
        },
        "Application": {
            "Id": "1a2b3c4d",
            "Name": MyApp
        },
        "Environment": {
            "Id": "1a2b3c4d",
            "Name": MyEnv
        },
        "ConfigurationProfile": {
            "Id": "1a2b3c4d",
            "Name": "MyConfigProfile"
        },
        "Description": null,
        "DeploymentNumber": "3",
        "ConfigurationVersion": "1",
        "Type": "OnDeploymentComplete"
    },
    "Timestamp": "2022-06-30T20:26:52.067Z",
    "SignatureVersion": "1",
    "Signature": "<...>",
    "SigningCertURL": "<...>",
    "UnsubscribeURL": "<...>",
    "MessageAttributes": {
        "MessageType": {
            "Type": "String",
            "Value": "OnDeploymentStart"
        }
    }
}
```

与 AWS AppConfig deployment events to Amazon SQS 扩展程序协作

该AWS AppConfig deployment events to Amazon SQS扩展是 AWS 编写的扩展，可帮助您监控 AWS AppConfig 配置部署工作流程并对其采取行动。每当部署配置时，该扩展程序都会将消息排入 Amazon Simple Queue Service (Amazon SQS) 队列。将扩展程序与您的一个 AWS AppConfig 应用程序、环境或配置文件关联后，在每次配置部署开始、 AWS AppConfig 结束和回滚之后，都会将一条消息排队到队列中。

如果您希望更好地控制哪些操作点发送 Amazon SQS 通知，您可以创建自定义扩展并在 URI 字段中输入 Amazon SQS 队列 Amazon 资源名称（ARN）。有关创建扩展程序的信息，请参阅 演练：创建自定义扩展 AWS AppConfig。

使用扩展程序

本节介绍如何使用 AWS AppConfig deployment events to Amazon SQS 扩展程序。

步骤 1：配置为 AWS AppConfig 将消息入队

将 Amazon SQS 策略添加到您的 Amazon SQS 队列，授予 AWS AppConfig (appconfig.amazonaws.com) 发送消息权限 (sqs:SendMessage)。有关更多信息，请参阅 Amazon SQS 策略的基本示例。

步骤 2：创建扩展程序关联

通过创建扩展关联，将扩展附加到您的一个 AWS AppConfig 资源。您可以使用 AWS AppConfig 控制台或 CreateExtensionAssociationAPI 操作创建关联。创建关联时，您可以指定 AWS AppConfig 应用程序、环境或配置文件的 ARN。如果将扩展程序关联到应用程序或环境，则会针对指定应用程序或环境中包含的任何配置文件发送通知。在创建关联时，您必须输入一个 Here 参数，该参数包含要使用的 Amazon SQS 队列的 ARN。

创建关联后，在创建或部署指定 AWS AppConfig 资源的配置时，会 AWS AppConfig 调用该扩展程序并根据扩展中指定的操作点发送通知。

> **Note**
>
> 此扩展程序由以下操作点调用：
>
> - ON_DEPLOYMENT_START
> - ON_DEPLOYMENT_COMPLETE

- ON_DEPLOYMENT_ROLLED_BACK

您无法自定义此扩展程序的操作点。若要调用不同的操作点，可以创建自己的扩展程序。有关更多信息，请参阅 演练：创建自定义扩展 AWS AppConfig。

使用以下过程通过 AWS Systems Manager 控制台或控制台创建 AWS AppConfig 扩展关联 AWS CLI。

创建扩展程序关联（控制台）

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，请选择 AWS AppConfig。
3. 在 扩展程序 选项卡上，选择 添加到资源。
4. 在扩展资源详细信息部分的资源类型中，选择一种 AWS AppConfig 资源类型。根据您选择的资源， AWS AppConfig 会提示您选择其他资源。
5. 选择创建与资源的关联。

以下是调用扩展程序时发送到 Amazon SQS 队列的消息示例。

```
{
   "InvocationId":"7itcaxp",
   "Parameters":{
      "queueArn":"arn:aws:sqs:us-east-1:111122223333:MySQSQueue"
   },
   "Application":{
      "Id":"1a2b3c4d",
      "Name":MyApp
   },
   "Environment":{
      "Id":"1a2b3c4d",
      "Name":MyEnv
   },
   "ConfigurationProfile":{
      "Id":"1a2b3c4d",
      "Name":"MyConfigProfile"
   },
   "Description":null,
```

```
    "DeploymentNumber":"3",
    "ConfigurationVersion":"1",
    "Type":"OnDeploymentComplete"
}
```

使用 Atlassian Jira 扩展程序来实现 AWS AppConfig

通过与 Atlassian Jira 集成，每当你对指定的功能标记进行更改时， AWS AppConfig 都可以在 Atlassian 控制台中创建和更新议题。 AWS 账户 AWS 区域每个 Jira 问题都包含标志名称、应用程序 ID、配置文件 ID 和标志值。在更新、保存和部署标志更改后，Jira 会使用此更改的详细信息更新现有问题。

> **Note**
>
> 每当您创建或更新功能标志时，Jira 都会更新问题。当您从父级标志中删除子级标志属性时，Jira 还会更新问题。当您删除父级标志时，Jira 不会记录信息。

要配置集成，必须执行以下操作：

- 为 AWS AppConfig Jira 集成配置权限
- 配置 AWS AppConfig Jira 集成应用程序

为 AWS AppConfig Jira 集成配置权限

配置与 Jira 的 AWS AppConfig 集成时，需要为用户指定凭据。具体而言，您需要在 Jira 应用程序的 AWS AppConfig 中输入用户的访问密钥 ID 和私有密钥。此用户授予 Jira 与 AWS AppConfig之通信的权限。 AWS AppConfig 曾经使用这些凭证在 AWS AppConfig 和 Jira 之间建立关联。不存储凭据。您可以通过卸载 for Jira 应用程序来 AWS AppConfig 删除关联。

用户帐户需要包含以下操作的权限策略：

- `appconfig:CreateExtensionAssociation`
- `appconfig:GetConfigurationProfile`
- `appconfig:ListApplications`
- `appconfig:ListConfigurationProfiles`

- appconfig:ListExtensionAssociations
- sts:GetCallerIdentity

完成以下任务以创建 IAM 权限策略以及用于 AWS AppConfig 和 Jira 集成的用户：

任务

- 任务 1：为 AWS AppConfig 和 Jira 集成创建 IAM 权限策略
- 任务 2：为 AWS AppConfig 和 Jira 集成创建用户

任务 1：为 AWS AppConfig 和 Jira 集成创建 IAM 权限策略

使用以下过程创建允许 Atlassian Jira 与之通信的 IAM 权限策略。 AWS AppConfig我们建议您创建一个新策略，并将此策略附加到新的 IAM 角色。向现有 IAM 策略和角色添加所需权限违反了最低权限原则，因此不建议这样做。

为 AWS AppConfig 和 Jira 集成创建 IAM 策略

1. 访问：https://console.aws.amazon.com/iam/，打开 IAM 控制台。
2. 在导航窗格中，选择 Policies (策略)，然后选择 Create policy (创建策略)。
3. 在创建策略页面上，选择 JSON 选项卡，然后将默认内容替换为以下策略。在以下策略中，将 *Region*、*account_ID*、*application_ID* 和 *configuration_profile_ID* 替换为 AWS AppConfig 功能标志环境中的信息。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                      "appconfig:CreateExtensionAssociation",
                      "appconfig:ListExtensionAssociations",
                      "appconfig:GetConfigurationProfile"
             ],
            "Resource": [

 "arn:aws:appconfig:Region:account_ID:application/application_ID",

 "arn:aws:appconfig:Region:account_ID:application/application_ID/
configurationprofile/configuration_profile_ID"
```

```
            ]
        },
       {
            "Effect": "Allow",
            "Action": [
                      "appconfig:ListApplications"

            ],
            "Resource": [
                      "arn:aws:appconfig:Region:account_ID:*"
            ]
        },
         {
            "Effect": "Allow",
            "Action": [
                      "appconfig:ListConfigurationProfiles"
            ],
            "Resource": [

 "arn:aws:appconfig:Region:account_ID:application/application_ID"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "sts:GetCallerIdentity",
            "Resource": "*"
        }
    ]
}
```

4. 选择 下一步：标签。
5. （可选）添加一个或多个标签键值对，以组织、跟踪或控制此策略的访问，然后选择 Next: Review (下一步：审核)。
6. 在 Review policy（查看策略）页面上的 Name（名称）框中输入一个名称，例如 **AppConfigJiraPolicy**，然后输入可选描述。
7. 选择 创建策略。

任务 2：为 AWS AppConfig 和 Jira 集成创建用户

使用以下步骤为 Atlassian J AWS AppConfig ira 集成创建用户。创建用户后，您可以复制访问密钥 ID 和私有密钥，您将在完成集成时指定这两个密钥。

为 AWS AppConfig 和 Jira 集成创建用户

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择 Users（用户），然后选择 Add users（添加用户）。
3. 在用户名字段中，输入名称，例如 **AppConfigJiraUser**。
4. 在 “选择 AWS 凭据类型” 中，选择访问密钥-编程访问权限。
5. 选择下一步：权限。
6. 在 设置权限 页面，选择 直接附加现有策略。对于您在 任务 1：为 AWS AppConfig 和 Jira 集成创建 IAM 权限策略 创建的策略，搜索并选中复选框，然后选择下一步：标签。
7. 在添加标签（可选）页面，添加一个或多个标签键值对，以组织、跟踪或控制此用户的访问权限。选择 下一步: 审核。
8. 在 审核 页面上，验证用户详细信息。
9. 选择 创建用户。系统显示用户的访问密钥 ID 和密钥。下载 .csv 文件或将这些凭据复制到单独的位置。您将在配置集成时指定这些凭据。

配置 AWS AppConfig Jira 集成应用程序

使用以下步骤在 for Jira 应用程序中 AWS AppConfig 配置所需的选项。完成此过程后，Jira 会 AWS 账户 为你中指定的 AWS 区域每个功能标志创建一个新问题。如果您在中更改功能标志 AWS AppConfig，Jira 会在现有问题中记录详细信息。

> **Note**
>
> 一个 AWS AppConfig 功能标志可以包含多个子级标志属性。Jira 为每个父级功能标志创建一个问题。如果更改子级标志属性，则可以在父级标志的 Jira 事务中查看该更改的详细信息。

配置集成

1. 登录到 Atlassian Marketplace。
2. 在搜索字段中，键入 **AWS AppConfig** 并按 Enter。
3. 在 Jira 实例上安装应用程序。
4. 在 Atlassian 控制台中，选择管理应用，然后为 Jira 选择AWS AppConfig 。
5. 选择 配置。
6. 在 配置详细信息 下，选择 Jira 项目，然后选择要与 AWS AppConfig 功能标志关联的项目。

7. 选择 AWS 区域 ，然后选择 AWS AppConfig 功能标志所在的区域。
8. 在 应用程序 ID 字段中，输入包含功能标志的 AWS AppConfig 应用程序的名称。
9. 在 配置文件 ID 字段中，输入功能标志的 AWS AppConfig 配置文件的名称。
10. 在 访问密钥 ID 和 密钥 字段中，输入您在 任务 2：为 AWS AppConfig 和 Jira 集成创建用户 中复制的凭据。（可选）还可以指定会话令牌。
11. 选择提交。
12. 在 Atlassian 控制台中，选择 “项目”，然后选择要集成的项目。 AWS AppConfig “问题” 页面显示指定 AWS 账户 和中每个功能标志的问题 AWS 区域。

删除 Jira 应用程序和数据中的 AWS AppConfig

如果您不想再使用带有 AWS AppConfig 功能标志的 Jira 集成，可以在 Atlassian AWS AppConfig 控制台中删除 Jira 应用程序。删除集成应用程序将执行以下操作：

- 删除您的 Jira 实例与之间的关联 AWS AppConfig
- 从中删除你的 Jira 实例详细信息 AWS AppConfig

删除 f AWS AppConfig or Jira 应用程序

1. 在 Atlassian 控制台中，选择管理应用程序。
2. 为 Jira 选择AWS AppConfig 。
3. 选择卸载。

演练：创建自定义扩展 AWS AppConfig

要创建自定义 AWS AppConfig 扩展模块，请完成以下任务。在后面的主题中将更详细地介绍每个任务。

> **Note**
>
> 你可以在以下网址查看自定义 AWS AppConfig 扩展程序的示例 GitHub：
>
> - 使用 Systems Manager 更改日历阻止使用`blocked day`暂停日历进行部署的示例扩展
> - 使用 git-secrets 防止机密泄漏到配置数据中的示例扩展
> - 使用 Amazon Comprehend 防止个人身份信息 (PII) 泄漏到配置数据中的示例扩展

1. 创建 AWS Lambda 函数

   对于大多数用例，要创建自定义扩展，必须创建一个 AWS Lambda 函数来执行扩展中定义的任何计算和处理。此规则的一个例外是，您创建了 AWS 创作的通知扩展程序的自定义版本来添加或删除操作点。有关此例外的更多详细信息，请参阅 创建自定义 AWS AppConfig 扩展。

2. 为自定义扩展程序配置权限

   若要为自定义扩展程序配置权限，可以执行下列操作之一：

   - 创建包含`InvokeFunction`权限的 AWS Identity and Access Management (IAM) 服务角色。
   - 使用 Lambda AddPermissionAPI 操作创建资源策略。

   此预排描述如何创建 IAM 服务角色。

3. 创建扩展程序

   您可以使用 AWS AppConfig 控制台或从 AWS CLI、 AWS Tools for PowerShell或 SDK 中调用 CreateExtensionAPI 操作来创建扩展。本演练使用控制台。

4. 创建扩展程序关联

   您可以使用 AWS AppConfig 控制台或从 AWS CLI、 AWS Tools for PowerShell或 SDK 调用 CreateExtensionAssociationAPI 操作来创建扩展关联。本演练使用控制台。

5. 执行调用扩展程序的操作

   创建关联后，当扩展定义的操作点出现在该资源上时，将 AWS AppConfig 调用该扩展。例如，如果关联包含 PRE_CREATE_HOSTED_CONFIGURATION_VERSION 操作的扩展程序，则每次创建新的托管配置版本时都会调用该扩展程序。

本节中的主题介绍创建自定义 AWS AppConfig 扩展程序所涉及的每个任务。每个任务都在一个使用案例的上下文中进行描述，在该用例中，客户希望创建一个扩展程序，该扩展程序会自动将配置备份到 Amazon Simple Storage Service (Amazon S3) 存储桶。每当创建 (PRE_CREATE_HOSTED_CONFIGURATION_VERSION) 或部署 (PRE_START_DEPLOYMENT) 托管配置时，扩展程序就会运行。

主题

- 为自定义扩展程序创建 Lambda 函数 AWS AppConfig
- 为自定义 AWS AppConfig 扩展程序配置权限
- 创建自定义 AWS AppConfig 扩展
- 为自定义扩展程序创建 AWS AppConfig 扩展关联

- 执行调用自定义扩展程序的操作 AWS AppConfig

为自定义扩展程序创建 Lambda 函数 AWS AppConfig

对于大多数用例，要创建自定义扩展，必须创建一个 AWS Lambda 函数来执行扩展中定义的任何计算和处理。本节包括自定义 AWS AppConfig 扩展的 Lambda 函数示例代码。本部分还包括有效负载请求和响应参考详细信息。有关创建 Lambda 函数的更多信息，请参阅 AWS Lambda 开发人员指南中的 Lambda 入门。

代码示例

以下 Lambda 函数的示例代码在调用时会自动将 AWS AppConfig 配置备份到 Amazon S3 存储桶。每当创建或部署新配置时，都会备份该配置。该示例使用扩展程序参数，因此不必在 Lambda 函数中对存储桶名称进行硬编码。通过使用扩展程序参数，用户可以将扩展程序附加到多个应用程序，并将配置备份到不同的存储桶。代码示例包含用于进一步解释该函数的注释。

扩展程序的 Lambda 函数示例 AWS AppConfig

```
from datetime import datetime
import base64
import json

import boto3


def lambda_handler(event, context):
    print(event)

    # Extensions that use the PRE_CREATE_HOSTED_CONFIGURATION_VERSION and
 PRE_START_DEPLOYMENT
    # action points receive the contents of AWS AppConfig configurations in Lambda
 event parameters.
    # Configuration contents are received as a base64-encoded string, which the lambda
 needs to decode
    # in order to get the configuration data as bytes. For other action points, the
 content
    # of the configuration isn't present, so the code below will fail.
    config_data_bytes = base64.b64decode(event["Content"])

    # You can specify parameters for extensions. The CreateExtension API action lets
 you define
```

```
    # which parameters an extension supports. You supply the values for those
 parameters when you
    # create an extension association by calling the CreateExtensionAssociation API
 action.
    # The following code uses a parameter called S3_BUCKET to obtain the value
 specified in the
    # extension association. You can specify this parameter when you create the
 extension
    # later in this walkthrough.
    extension_association_params = event.get('Parameters', {})
    bucket_name = extension_association_params['S3_BUCKET']
    write_backup_to_s3(bucket_name, config_data_bytes)

    # The PRE_CREATE_HOSTED_CONFIGURATION_VERSION and PRE_START_DEPLOYMENT action
 points can
    # modify the contents of a configuration. The following code makes a minor change
    # for the purposes of a demonstration.
    old_config_data_string = config_data_bytes.decode('utf-8')
    new_config_data_string = old_config_data_string.replace('hello', 'hello!')
    new_config_data_bytes = new_config_data_string.encode('utf-8')

    # The lambda initially received the configuration data as a base64-encoded string
    # and must return it in the same format.
    new_config_data_base64string =
 base64.b64encode(new_config_data_bytes).decode('ascii')

    return {
        'statusCode': 200,
        # If you want to modify the contents of the configuration, you must include the
 new contents in the
        # Lambda response. If you don't want to modify the contents, you can omit the
 'Content' field shown here.
        'Content': new_config_data_base64string
    }


def write_backup_to_s3(bucket_name, config_data_bytes):
    s3 = boto3.resource('s3')
    new_object = s3.Object(bucket_name,
 f"config_backup_{datetime.now().isoformat()}.txt")
    new_object.put(Body=config_data_bytes)
```

如果要在本演练中使用此示例，请使用名称 **MyS3ConfigurationBackUpExtension** 保存它，并复制该函数的 Amazon 资源名称（ARN）。在下一节中创建 AWS Identity and Access Management (IAM) 代入角色时，您将指定 ARN。您可以在创建扩展程序时指定 ARN 和名称。

有效负载参考

本节包括使用自定义 AWS AppConfig 扩展程序的有效载荷请求和响应参考详细信息。

请求结构

PreCreateHostedConfigurationVersion

```
{
    'InvocationId': 'vlns753', // id for specific invocation
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'ContentType': 'text/plain',
    'ContentVersion': '2',
    'Content': 'SGVsbG8gZWFydGgh', // Base64 encoded content
    'Application': {
        'Id': 'abcd123',
        'Name': 'ApplicationName'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'Description': '',
    'Type': 'PreCreateHostedConfigurationVersion',
    'PreviousContent': {
        'ContentType': 'text/plain',
        'ContentVersion': '1',
        'Content': 'SGVsbG8gd29ybGQh'
    }
}
```

PreStartDeployment

```
{
    'InvocationId': '765ahdm',
    'Parameters': {
```

```
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'ContentType': 'text/plain',
    'ContentVersion': '2',
    'Content': 'SGVsbG8gZWFydGgh',
    'Application': {
        'Id': 'abcd123',
        'Name': 'ApplicationName'
    },
    'Environment': {
        'Id': 'ibpnqlq',
        'Name': 'EnvironmentName'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'Type': 'PreStartDeployment'
}
```

异步事件

OnStartDeployment, OnDeploymentStep, OnDeployment

```
{
    'InvocationId': 'o2xbtm7',
    'Parameters': {
        'ParameterOne': 'ValueOne',
        'ParameterTwo': 'ValueTwo'
    },
    'Type': 'OnDeploymentStart',
    'Application': {
        'Id': 'abcd123'
    },
    'Environment': {
        'Id': 'efgh456'
    },
    'ConfigurationProfile': {
        'Id': 'ijkl789',
        'Name': 'ConfigurationName'
```

```
    },
    'DeploymentNumber': 2,
    'Description': 'Deployment description',
    'ConfigurationVersion': '2'
}
```

响应结构

以下示例显示了您的 Lambda 函数在响应自定义扩展程序的请求时返回的内容。 AWS AppConfig

同步事件 - 成功响应

如果要转换内容，请使用以下命令：

```
"Content": "SomeBase64EncodedByteArray"
```

如果不想转换内容，则不返回任何内容。

异步事件 - 成功响应

无返回内容。

所有错误事件

```
{
        "Error": "BadRequestError",
        "Message": "There was malformed stuff in here",
        "Details": [{
            "Type": "Malformed",
            "Name": "S3 pointer",
            "Reason": "S3 bucket did not exist"
        }]
    }
```

为自定义 AWS AppConfig 扩展程序配置权限

使用以下过程创建和配置 AWS Identity and Access Management (IAM) 服务角色（或代入角色）。AWS AppConfig 使用此角色来调用 Lambda 函数。

创建 IAM 服务角色并 AWS AppConfig 允许代入该角色

1. 通过 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。

2. 在导航窗格中，选择角色，然后选择创建角色。
3. 在选择可信实体类型下，选择自定义信任策略。
4. 将以下 JSON 策略粘贴到自定义信任策略字段中。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "appconfig.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

选择下一步。

5. 在附加权限页面上，选择创建策略。此时将在新选项卡中打开创建策略页面。
6. 选择 JSON 选项卡，然后将以下权限策略粘贴到编辑器中。`lambda:InvokeFunction` 操作用于 `PRE_*` 操作点。`lambda:InvokeAsync` 操作用于 `ON_*` 操作点。将*## Lambda ARN* 替换为 Lambda 的 Amazon 资源名称（ARN）。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "lambda:InvokeFunction",
        "lambda:InvokeAsync"
      ],
      "Resource": "Your Lambda ARN"
    }
  ]
}
```

7. 选择下一步：标签。
8. 在 添加标签 （可选） 页面上，添加一个或多个键值对，然后选择下一步： 审核。

9. 在 Review policy（检查策略）页面上输入一个名称和描述，然后选择 Create policy（创建策略）。
10. 在自定义信任策略的浏览器选项卡上，选择刷新图标，然后搜索您刚刚创建的权限策略。
11. 选择权限策略对应的复选框，然后选择下一步。
12. 在名称、审查和创建页面上，在角色名称框中输入名称，然后输入描述。
13. 选择 Create role (创建角色)。系统将让您返回到 角色 页面。在横幅中选择查看角色。
14. 复制 ARN。您可以在创建扩展程序时指定此 ARN。

创建自定义 AWS AppConfig 扩展

扩展程序定义了它在 AWS AppConfig 工作流程中执行的一个或多个操作。例如， AWS 创作的AWS AppConfig deployment events to Amazon SNS扩展程序包括向 Amazon SNS 主题发送通知的操作。每个操作都是在您与之交互时调用的， AWS AppConfig 或者 AWS AppConfig 是在代表您执行流程时调用的。这些被称为行动要点。 AWS AppConfig 扩展支持以下操作要点：

- PRE_CREATE_HOSTED_CONFIGURATION_VERSION
- PRE_START_DEPLOYMENT
- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_STEP
- ON_DEPLOYMENT_BAKING
- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

在PRE_*操作点上配置的扩展操作将在请求验证之后但在 AWS AppConfig 执行与操作点名称对应的活动之前应用。这些操作调用与请求同时处理。如果发出多个请求，则操作调用将按顺序运行。另请注意，PRE_* 操作点接收并可以更改配置的内容。 PRE_* 操作点还可以响应错误并防止操作发生。

扩展也可以使用ON_*操作点与 AWS AppConfig 工作流程并行运行。 ON_*操作点是异步调用的。 ON_*操作点不接收配置的内容。如果扩展程序在 ON_* 操作点期间遇到错误，服务将忽略该错误并继续工作流。

以下示例扩展程序定义了一个调用 PRE_CREATE_HOSTED_CONFIGURATION_VERSION 操作点的操作。在 Uri 字段中，该操作指定在本演练前面创建的 MyS3ConfigurationBackUpExtension

Lambda 函数的 Amazon 资源名称（ARN）。该操作还指定了本演练前面创建的 AWS Identity and Access Management (IAM) 代入角色 ARN。

AWS AppConfig 扩展示例

```
{
    "Name": "MySampleExtension",
    "Description": "A sample extension that backs up configurations to an S3 bucket.",
    "Actions": {
        "PRE_CREATE_HOSTED_CONFIGURATION_VERSION": [
            {
                "Name": "PreCreateHostedConfigVersionActionForS3Backup",
                "Uri": "arn:aws:lambda:aws-
region:111122223333:function:MyS3ConfigurationBackUpExtension",
                "RoleArn": "arn:aws:iam::111122223333:role/ExtensionsTestRole"
            }
        ]
    },
    "Parameters" : {
        "S3_BUCKET": {
            "Required": false
        }
    }
}
```

> **Note**
>
> 要查看创建扩展程序时的请求语法和字段描述，请参阅 AWS AppConfig API 参考中的CreateExtension主题。

创建扩展程序（控制台）

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，请选择 AWS AppConfig。
3. 在 扩展程序 选项卡上，选择 创建扩展程序。
4. 对于 扩展程序名称，输入唯一的名称。在本演练中，请输入 **MyS3ConfigurationBackUpExtension**。（可选）输入描述。
5. 在 操作 部分中，选择 添加新操作。

6. 对于 操作名称，输入唯一的名称。在本演练中，请输入 **PreCreateHostedConfigVersionActionForS3Backup**。此名称描述操作使用的操作点和扩展程序目的。
7. 在操作点列表中，选择 PRE_C REATE_HOSTED_CONFIGURATION_VERSION。
8. 对于 Uri，选择 Lambda 函数，然后在 Lambda 函数列表中选择该函数。如果您没有看到自己的函数，请确认您与创建函数的 AWS 区域 位置相同。
9. 对于 IAM 角色，选择您之前在本演练中创建的角色。
10. 在 扩展参数（可选） 部分中，选择添加新参数。
11. 对于参数名称，请输入名称。在本演练中，请输入 **S3_BUCKET**。
12. 重复步骤 5-11，为 PRE_START_DEPLOYMENT 操作点创建第二个操作。
13. 选择 创建扩展程序。

自定义 AWS 创作的附加通知信息

您无需创建 Lambda 或扩展即可使用 AWS 编写的通知扩展程序。您只需创建一个扩展程序关联，然后执行调用其中一个支持的操作点的操作即可。默认情况下， AWS 创作的附加通知信息支持以下操作点：

- ON_DEPLOYMENT_START
- ON_DEPLOYMENT_COMPLETE
- ON_DEPLOYMENT_ROLLED_BACK

如果创建 AWS AppConfig deployment events to Amazon SNS 扩展程序和 AWS AppConfig deployment events to Amazon SQS 扩展程序的自定义版本，则可以指定要接收其通知的操作点。

> **Note**
>
> AWS AppConfig deployment events to EventBridge 扩展不支持 PRE_* 操作点。如果要移除分配给创 AWS 作版本的一些默认操作点，则可以创建自定义版本。

如果您创建 AWS 创作的通知扩展程序的自定义版本，则无需创建 Lambda 函数。您只需在新扩展程序版本的 Uri 字段中指定 Amazon 资源名称（ARN）。

- 对于自定义附加 EventBridge 通知信息，请在字段中输入 EventBridge 默认事件的 ARN。`Uri`
- 对于自定义 Amazon SNS 通知扩展，请在 `Uri` 字段中输入 Amazon SNS 主题的 ARN。
- 对于自定义 Amazon SQS 通知扩展，请在 `Uri` 字段中输入 Amazon SQS 消息队列的 ARN。

为自定义扩展程序创建 AWS AppConfig 扩展关联

要创建扩展或配置创 AWS 作的扩展，您需要定义在使用特定 AWS AppConfig 资源时调用扩展的操作点。例如，您可以选择运行 `AWS AppConfig deployment events to Amazon SNS` 扩展程序，并在为特定应用程序启动配置部署时接收有关 Amazon SNS 主题的通知。定义哪些操作点调用特定 AWS AppConfig 资源的扩展称为扩展关联。扩展关联是扩展与 AWS AppConfig 资源（例如应用程序或配置配置文件）之间的指定关系。

单个 AWS AppConfig 应用程序可以包含多个环境和配置文件。如果将扩展程序与应用程序或环境相关联，则会为与应用程序或环境资源相关的所有工作流程 AWS AppConfig 调用该扩展（如果适用）。

例如，假设你有一个名为的 AWS AppConfig 应用程序 MobileApps ，其中包含一个名为的配置文件 AccessList。假设该 MobileApps 应用程序包括测试版、集成和生产环境。您可以为创 AWS 作的 Amazon SNS 通知扩展信息创建扩展关联，并将该扩展程序与应用程序关联。 MobileApps 每当将应用程序的配置部署到三个环境中的任何一个时，都会调用 Amazon SNS 通知扩展程序。

使用以下过程通过 AWS AppConfig 控制台创建 AWS AppConfig 扩展关联。

创建扩展程序关联（控制台）

1. 打开 AWS Systems Manager 控制台，网址为 https://console.aws.amazon.com/systems-manager/appconfig/。
2. 在导航窗格中，请选择 AWS AppConfig。
3. 在扩展程序选项卡上，为扩展程序选择一个选项按钮，然后选择添加到资源。在本演练中，请选择 MyS ConfigurationBackUpExtension 3。
4. 在扩展资源详细信息部分的资源类型中，选择一种 AWS AppConfig 资源类型。根据您选择的资源， AWS AppConfig 会提示您选择其他资源。在本演练中，请选择应用程序。
5. 在列表中选择应用程序。
6. 在参数部分中，验证键字段中是否列出了 S3_BUCKET。在值字段中，粘贴 Lambda 扩展程序的 ARN。例如：`arn:aws:lambda:aws-region:111122223333:function:MyS3ConfigurationBackUpExtension`。
7. 选择创建与资源的关联。

执行调用自定义扩展程序的操作 AWS AppConfig

创建关联后，可以通过创建一个新的配置文件来调用 `MyS3ConfigurationBackUpExtension` 扩展程序，该配置文件会对其 `SourceUri` 指定 `hosted`。在创建新配置的工作流程中， AWS AppConfig 会遇到`PRE_CREATE_HOSTED_CONFIGURATION_VERSION`操作点。遇到此操作点会调用 `MyS3ConfigurationBackUpExtension` 扩展，该扩展程序会自动将新创建的配置备份到此扩展程序关联中 `Parameter` 部分指定的 S3 存储桶。

AWS AppConfig 扩展与 Atlassian Jira 集成

AWS AppConfig 与 Atlassian Jira 集成。只要您 AWS AppConfig 对指定的功能标记进行更改，集成即可 AWS 账户 在 Atlassian 控制台中创建和更新问题。 AWS 区域每个 Jira 问题都包含标志名称、应用程序 ID、配置文件 ID 和标志值。在更新、保存和部署标志更改后，Jira 会使用此更改的详细信息更新现有问题。有关更多信息，请参阅 使用 Atlassian Jira 扩展程序来实现 AWS AppConfig。

AWS AppConfig 代码示例

本节包含以编程方式执行常见AWS AppConfig操作的代码示例。我们建议您将这些示例与Java、Python 和JavaScript软件开发工具包一起使用，以便在测试环境中执行操作。本节包含一个代码示例，用于在完成测试后清理测试环境。

主题

创建或更新存储在托管配置存储中的自由格式配置

以下每个示例都包含有关代码所执行操作的注释。本节中的示例调用以下 API：

- CreateApplication
- CreateConfigurationProfile
- CreateHostedConfigurationVersion

Java

```
public CreateHostedConfigurationVersionResponse createHostedConfigVersion() {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a hosted, freeform configuration profile
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
```

```
            .name("MyConfigProfile")
            .locationUri("hosted")
            .type("AWS.Freeform"));

        // Create a hosted configuration version
        CreateHostedConfigurationVersionResponse hcv =
 appconfig.createHostedConfigurationVersion(req -> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .contentType("text/plain; charset=utf-8")
            .content(SdkBytes.fromUtf8String("my config data")));

        return hcv;
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create a hosted, freeform configuration profile
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='hosted',
    Type='AWS.Freeform')

# create a hosted configuration version
hcv = appconfig.create_hosted_configuration_version(
    ApplicationId=application['Id'],
    ConfigurationProfileId=config_profile['Id'],
    Content=b'my config data',
    ContentType='text/plain')
```

JavaScript

```
import {
  AppConfigClient,
  CreateApplicationCommand,
```

```
  CreateConfigurationProfileCommand,
  CreateHostedConfigurationVersionCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create a hosted, freeform configuration profile
const profile = await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "hosted",
    Type: "AWS.Freeform",
  })
);

// create a hosted configuration version
await appconfig.send(
  new CreateHostedConfigurationVersionCommand({
    ApplicationId: application.Id,
    ConfigurationProfileId: profile.Id,
    ContentType: "text/plain",
    Content: "my config data",
  })
);
```

为存储在 Secrets Manager 中的密钥创建配置文件

以下每个示例都包含有关代码所执行操作的注释。本节中的示例调用以下 API：

- CreateApplication
- CreateConfigurationProfile

Java

```
private void createSecretsManagerConfigProfile() {
```

```
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a configuration profile for Secrets Manager Secret
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("secretsmanager://MySecret")
            .retrievalRoleArn("arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret")
            .type("AWS.Freeform"));
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create a configuration profile for Secrets Manager Secret
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='secretsmanager://MySecret',
    RetrievalRoleArn='arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret',
    Type='AWS.Freeform')
```

JavaScript

```
import {
  AppConfigClient,
  CreateConfigurationProfileCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();
```

```
// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create a configuration profile for Secrets Manager Secret
await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "secretsmanager://MySecret",
    RetrievalRoleArn: "arn:aws:iam::000000000000:role/
RoleTrustedByAppConfigThatCanRetrieveSecret",
    Type: "AWS.Freeform",
  })
);
```

部署配置文件

以下每个示例都包含有关代码所执行操作的注释。本节中的示例调用以下 API：

- CreateApplication
- CreateConfigurationProfile
- CreateHostedConfigurationVersion
- CreateEnvironment
- StartDeployment
- GetDeployment

Java

```
private void createDeployment() throws InterruptedException {
        AppConfigClient appconfig = AppConfigClient.create();

        // Create an application
        CreateApplicationResponse app = appconfig.createApplication(req ->
 req.name("MyDemoApp"));

        // Create a hosted, freeform configuration profile
```

```
        CreateConfigurationProfileResponse configProfile =
 appconfig.createConfigurationProfile(req -> req
            .applicationId(app.id())
            .name("MyConfigProfile")
            .locationUri("hosted")
            .type("AWS.Freeform"));

        // Create a hosted configuration version
        CreateHostedConfigurationVersionResponse hcv =
 appconfig.createHostedConfigurationVersion(req -> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .contentType("text/plain; charset=utf-8")
            .content(SdkBytes.fromUtf8String("my config data")));


        // Create an environment
        CreateEnvironmentResponse env = appconfig.createEnvironment(req -> req
            .applicationId(app.id())
            .name("Beta")
            // If you have CloudWatch alarms that monitor the health of your
 service, you can add them here and they
            // will trigger a rollback if they fire during an appconfig deployment
            //.monitors(Monitor.builder().alarmArn("arn:aws:cloudwatch:us-
east-1:520900602629:alarm:MyAlarm")
            //
  .alarmRoleArn("arn:aws:iam::520900602629:role/MyAppConfigAlarmRole").build())
        );

        // Start a deployment
        StartDeploymentResponse deploymentResponse = appconfig.startDeployment(req -
> req
            .applicationId(app.id())
            .configurationProfileId(configProfile.id())
            .environmentId(env.id())
            .configurationVersion(hcv.versionNumber().toString())
            .deploymentStrategyId("AppConfig.Linear50PercentEvery30Seconds")
        );

        // Wait for deployment to complete
        List<DeploymentState> nonFinalDeploymentStates = Arrays.asList(
            DeploymentState.DEPLOYING,
            DeploymentState.BAKING,
            DeploymentState.ROLLING_BACK,
```

```
            DeploymentState.VALIDATING);
        GetDeploymentRequest getDeploymentRequest =
 GetDeploymentRequest.builder().applicationId(app.id())

 .environmentId(env.id())

 .deploymentNumber(deploymentResponse.deploymentNumber()).build();
        GetDeploymentResponse deployment =
 appconfig.getDeployment(getDeploymentRequest);
        while (nonFinalDeploymentStates.contains(deployment.state())) {
            System.out.println("Waiting for deployment to complete: " + deployment);
            Thread.sleep(1000L);
            deployment = appconfig.getDeployment(getDeploymentRequest);
        }

        System.out.println("Deployment complete: " + deployment);
    }
```

Python

```
import boto3

appconfig = boto3.client('appconfig')

# create an application
application = appconfig.create_application(Name='MyDemoApp')

# create an environment
environment = appconfig.create_environment(
    ApplicationId=application['Id'],
    Name='MyEnvironment')

# create a configuration profile
config_profile = appconfig.create_configuration_profile(
    ApplicationId=application['Id'],
    Name='MyConfigProfile',
    LocationUri='hosted',
    Type='AWS.Freeform')

# create a hosted configuration version
hcv = appconfig.create_hosted_configuration_version(
    ApplicationId=application['Id'],
    ConfigurationProfileId=config_profile['Id'],
```

```
    Content=b'my config data',
    ContentType='text/plain')

# start a deployment
deployment = appconfig.start_deployment(
    ApplicationId=application['Id'],
    EnvironmentId=environment['Id'],
    ConfigurationProfileId=config_profile['Id'],
    ConfigurationVersion=str(hcv['VersionNumber']),
    DeploymentStrategyId='AppConfig.Linear20PercentEvery6Minutes')
```

JavaScript

```
import {
  AppConfigClient,
  CreateApplicationCommand,
  CreateEnvironmentCommand,
  CreateConfigurationProfileCommand,
  CreateHostedConfigurationVersionCommand,
  StartDeploymentCommand,
} from "@aws-sdk/client-appconfig";

const appconfig = new AppConfigClient();

// create an application
const application = await appconfig.send(
  new CreateApplicationCommand({ Name: "MyDemoApp" })
);

// create an environment
const environment = await appconfig.send(
  new CreateEnvironmentCommand({
    ApplicationId: application.Id,
    Name: "MyEnvironment",
  })
);

// create a configuration profile
const config_profile = await appconfig.send(
  new CreateConfigurationProfileCommand({
    ApplicationId: application.Id,
    Name: "MyConfigProfile",
    LocationUri: "hosted",
```

```
    Type: "AWS.Freeform",
  })
);

// create a hosted configuration version
const hcv = await appconfig.send(
  new CreateHostedConfigurationVersionCommand({
    ApplicationId: application.Id,
    ConfigurationProfileId: config_profile.Id,
    Content: "my config data",
    ContentType: "text/plain",
  })
);

// start a deployment
await appconfig.send(
  new StartDeploymentCommand({
    ApplicationId: application.Id,
    EnvironmentId: environment.Id,
    ConfigurationProfileId: config_profile.Id,
    ConfigurationVersion: hcv.VersionNumber.toString(),
    DeploymentStrategyId: "AppConfig.Linear20PercentEvery6Minutes",
  })
);
```

使用 AWS AppConfig Agent 读取自由格式配置文件

以下每个示例都包含有关代码所执行操作的注释。

Java

```
public void retrieveConfigFromAgent() throws Exception {
        /*
        In this sample, we will retrieve configuration data from the AWS AppConfig
 Agent.
        The agent is a sidecar process that handles retrieving configuration data
 from AppConfig
        for you in a way that implements best practices like configuration caching.

        For more information about the agent, see Simplified retrieval methods
        */
```

```
        // The agent runs a local HTTP server that serves configuration data
        // Make a GET request to the agent's local server to retrieve the
 configuration data
        URL url = new URL("http://localhost:2772/applications/MyDemoApp/
environments/Beta/configurations/MyConfigProfile");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        StringBuilder content;
        try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
            content = new StringBuilder();
            int ch;
            while ((ch = in.read()) != -1) {
                content.append((char) ch);
            }
        }
        con.disconnect();
        System.out.println("Configuration from agent via HTTP: " + content);
    }
```

Python

```
# in this sample, we will retrieve configuration data from the AWS AppConfig Agent.
# the agent is a sidecar process that handles retrieving configuration data from AWS
 AppConfig
# for you in a way that implements best practices like configuration caching.
#
# for more information about the agent, see
# Simplified retrieval methods
#

import requests

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'

# the agent runs a local HTTP server that serves configuration data
# make a GET request to the agent's local server to retrieve the configuration data
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{config_profile_name}")
config = response.content
```

JavaScript

```
// in this sample, we will retrieve configuration data from the AWS AppConfig Agent.
// the agent is a sidecar process that handles retrieving configuration data from
 AppConfig
// for you in a way that implements best practices like configuration caching.

// for more information about the agent, see
// Simplified retrieval methods

const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";

// the agent runs a local HTTP server that serves configuration data
// make a GET request to the agent's local server to retrieve the configuration data
const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${config_profile_name}`;
const response = await fetch(url);
const config = await response.text(); // (use `await response.json()` if your config
 is json)
```

使用AWS AppConfig代理读取特定的功能标志

以下每个示例都包含有关代码所执行操作的注释。

Java

```
public void retrieveSingleFlagFromAgent() throws Exception {
        /*
          You can retrieve a single flag's data from the agent by providing the
 "flag" query string parameter.
          Note: the configuration's type must be AWS.AppConfig.FeatureFlags
        */

        URL url = new URL("http://localhost:2772/applications/MyDemoApp/
environments/Beta/configurations/MyFlagsProfile?flag=myFlagKey");
        HttpURLConnection con = (HttpURLConnection) url.openConnection();
        con.setRequestMethod("GET");
        StringBuilder content;
        try (BufferedReader in = new BufferedReader(new
 InputStreamReader(con.getInputStream()))) {
```

```
            content = new StringBuilder();
            int ch;
            while ((ch = in.read()) != -1) {
                content.append((char) ch);
            }
        }
        con.disconnect();
        System.out.println("MyFlagName from agent: " + content);
    }
```

Python

```
import requests

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'
flag_key = 'MyFlag'

# retrieve a single flag's data by providing the "flag" query string parameter
# note: the configuration's type must be AWS.AppConfig.FeatureFlags
response = requests.get(f"http://localhost:2772/applications/{application_name}/
environments/{environment_name}/configurations/{config_profile_name}?
flag={flag_key}")
config = response.content
```

JavaScript

```
const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";
const flag_name = "MyFlag";

// retrieve a single flag's data by providing the "flag" query string parameter
// note: the configuration's type must be AWS.AppConfig.FeatureFlags
const url = `http://localhost:2772/applications/${application_name}/environments/
${environment_name}/configurations/${config_profile_name}?flag=${flag_name}`;
const response = await fetch(url);
const flag = await response.json(); // { "enabled": true/false }
```

使用 GetLatestConfig API 操作读取自由格式配置文件

以下每个示例都包含有关代码所执行操作的注释。本节中的示例调用以下 API：

- GetLatestConfiguration
- StartConfigurationSession

Java

```
public void retrieveConfigFromApi() {
        /*
        The example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
        For more information on these APIs, see AWS AppConfig Data         */
        AppConfigDataClient appConfigData = AppConfigDataClient.create();

        /*
        Start a new configuration session using the StartConfigurationSession API.
 This operation does not return configuration data.
        Rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
        IMPORTANT: This operation should only be performed once (per configuration),
 prior to the first GetLatestConfiguration
        call you preform. Each GetLatestConfiguration will return a new
 configuration token that you should then use in the
        next GetLatestConfiguration call.
        */
        StartConfigurationSessionResponse session =
            appConfigData.startConfigurationSession(req -> req
                .applicationIdentifier("MyDemoApp")
                .configurationProfileIdentifier("MyConfigProfile")
                .environmentIdentifier("Beta"));

        /*
        Retrieve configuration data using the GetLatestConfiguration API. The first
 time you call this API your configuration
        data will be returned. You should cache that data (and the configuration
 token) and update that cache asynchronously
        by regularly polling the GetLatestConfiguration API in a background thread.
 If you already have the latest configuration
        data, subsequent GetLatestConfiguration calls will return an empty response.
 If you then deploy updated configuration
```

```
        data the next time you call GetLatestConfiguration it will return that
 updated data.

        You can also avoid all the complexity around writing this code yourself by
 leveraging our agent instead.
        For more information about the agent, see Simplified retrieval methods
        */

        // The first getLatestConfiguration call uses the token from
 StartConfigurationSession
        String configurationToken = session.initialConfigurationToken();
        GetLatestConfigurationResponse configuration =

 appConfigData.getLatestConfiguration(GetLatestConfigurationRequest.builder().configurationT

        System.out.println("Configuration retrieved via API: " +
 configuration.configuration().asUtf8String());

        // You'll want to hold on to the token in the getLatestConfiguration
 response because you'll need to use it
        // the next time you call
        configurationToken = configuration.nextPollConfigurationToken();
        configuration =

 appConfigData.getLatestConfiguration(GetLatestConfigurationRequest.builder().configurationT

        // Try creating a new deployment at this point to see how the output below
 changes.
        if (configuration.configuration().asByteArray().length != 0) {
            System.out.println("Configuration contents have changed
 since the last GetLatestConfiguration call, new contents = " +
 configuration.configuration().asUtf8String());
        } else {
            System.out.println("GetLatestConfiguration returned an empty response
 because we already have the latest configuration");
        }
    }
```

Python

```
# the example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
#
```

```
# for more information on these APIs, see
# AWS AppConfig Data
#

import boto3

application_name = 'MyDemoApp'
environment_name = 'MyEnvironment'
config_profile_name = 'MyConfigProfile'

appconfigdata = boto3.client('appconfigdata')

# start a new configuration session.
# this operation does not return configuration data.
# rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
#
# note: this operation should only be performed once (per configuration).
#   all subsequent calls to AppConfigData should be via GetLatestConfiguration.
scs = appconfigdata.start_configuration_session(
    ApplicationIdentifier=application_name,
    EnvironmentIdentifier=environment_name,
    ConfigurationProfileIdentifier=config_profile_name)
initial_token = scs['InitialConfigurationToken']

# retrieve configuration data from the session.
# this operation returns your configuration data.
# each invocation of this operation returns a unique token that should be passed to
 the subsequent invocation.
#
# note: this operation does not always return configuration data after the first
 invocation.
#   data is only returned if the configuration has changed within AWS AppConfig
 (i.e. a deployment occurred).
#   therefore, you should cache the data returned by this call so that you can use
 it later.
glc = appconfigdata.get_latest_configuration(ConfigurationToken=initial_token)
config = glc['Configuration'].read()
```

JavaScript

```
// the example below uses two AppConfigData APIs: StartConfigurationSession and
 GetLatestConfiguration.
```

```
// for more information on these APIs, see
// AWS AppConfig Data

import {
  AppConfigDataClient,
  GetLatestConfigurationCommand,
  StartConfigurationSessionCommand,
} from "@aws-sdk/client-appconfigdata";

const appconfigdata = new AppConfigDataClient();

const application_name = "MyDemoApp";
const environment_name = "MyEnvironment";
const config_profile_name = "MyConfigProfile";

// start a new configuration session.
// this operation does not return configuration data.
// rather, it returns an initial configuration token that should be passed to
 GetLatestConfiguration.
//
// note: this operation should only be performed once (per configuration).
//   all subsequent calls to AppConfigData should be via GetLatestConfiguration.
const scs = await appconfigdata.send(
  new StartConfigurationSessionCommand({
    ApplicationIdentifier: application_name,
    EnvironmentIdentifier: environment_name,
    ConfigurationProfileIdentifier: config_profile_name,
  })
);
const { InitialConfigurationToken } = scs;

// retrieve configuration data from the session.
// this operation returns your configuration data.
// each invocation of this operation returns a unique token that should be passed to
 the subsequent invocation.
//
// note: this operation does not always return configuration data after the first
 invocation.
//   data is only returned if the configuration has changed within AWS AppConfig
 (i.e. a deployment occurred).
//   therefore, you should cache the data returned by this call so that you can use
 it later.
const glc = await appconfigdata.send(
```

```
  new GetLatestConfigurationCommand({
    ConfigurationToken: InitialConfigurationToken,
  })
);
const config = glc.Configuration.transformToString();
```

清理您的环境

如果您运行了本节中的一个或多个代码示例，我们建议您使用以下示例之一来查找和删除由这些代码示例创建的AWS AppConfig资源。本节中的示例调用以下 API：

- ListApplications
- DeleteApplication
- ListEnvironments
- DeleteEnvironments
- ListConfigurationProfiles
- DeleteConfigurationProfile
- ListHostedConfigurationVersions
- DeleteHostedConfigurationVersion

Java

```
/*
    This sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.

    WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
    configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
    an application that you may need in the future.
    */

    public void cleanUpDemoResources() {
        AppConfigClient appconfig = AppConfigClient.create();

        // The name of the application to delete
```

```
        // IMPORTANT: verify this name corresponds to the application you wish to
 delete
        String applicationToDelete = "MyDemoApp";


 appconfig.listApplicationsPaginator(ListApplicationsRequest.builder().build()).items().forE
 -> {
            if (app.name().equals(applicationToDelete)) {
                System.out.println("Deleting App: " + app);
                appconfig.listConfigurationProfilesPaginator(req ->
 req.applicationId(app.id())).items().forEach(cp -> {
                    System.out.println("Deleting Profile: " + cp);
                    appconfig
                        .listHostedConfigurationVersionsPaginator(req -> req
                            .applicationId(app.id())
                            .configurationProfileId(cp.id()))
                        .items()
                        .forEach(hcv -> {
                            System.out.println("Deleting HCV: " + hcv);
                            appconfig.deleteHostedConfigurationVersion(req -> req
                                .applicationId(app.id())
                                .configurationProfileId(cp.id())
                                .versionNumber(hcv.versionNumber()));
                        });
                    appconfig.deleteConfigurationProfile(req -> req
                        .applicationId(app.id())
                        .configurationProfileId(cp.id()));
                });

                appconfig.listEnvironmentsPaginator(req-
 >req.applicationId(app.id())).items().forEach(env -> {
                    System.out.println("Deleting Environment: " + env);
                    appconfig.deleteEnvironment(req-
 >req.applicationId(app.id()).environmentId(env.id()));
                });

                appconfig.deleteApplication(req -> req.applicationId(app.id()));
            }
        });
    }
```

Python

```
# this sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.
#
# WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
#   configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
#   an application that you may need in the future.
#

import boto3

# the name of the application to delete
# IMPORTANT: verify this name corresponds to the application you wish to delete
application_name = 'MyDemoApp'

# create and iterate over a list paginator such that we end up with a list of pages,
 which are themselves lists of applications
# e.g. [ [{'Name':'MyApp1',...},{'Name':'MyApp2',...}], [{'Name':'MyApp3',...}] ]
list_of_app_lists = [page['Items'] for page in
 appconfig.get_paginator('list_applications').paginate()]
# retrieve the target application from the list of lists
application = [app for apps in list_of_app_lists for app in apps if app['Name'] ==
 application_name][0]
print(f"deleting application {application['Name']} (id={application['Id']})")

# delete all configuration profiles
list_of_config_lists = [page['Items'] for page in
 appconfig.get_paginator('list_configuration_profiles').paginate(ApplicationId=application['
for config_profile in [config for configs in list_of_config_lists for config in
 configs]:
    print(f"\tdeleting configuration profile {config_profile['Name']}
 (Id={config_profile['Id']})")

    # delete all hosted configuration versions
    list_of_hcv_lists = [page['Items'] for page in
 appconfig.get_paginator('list_hosted_configuration_versions').paginate(ApplicationId=applic
 ConfigurationProfileId=config_profile['Id'])]
    for hcv in [hcv for hcvs in list_of_hcv_lists for hcv in hcvs]:

 appconfig.delete_hosted_configuration_version(ApplicationId=application['Id'],
 ConfigurationProfileId=config_profile['Id'], VersionNumber=hcv['VersionNumber'])
```

```
        print(f"\t\tdeleted hosted configuration version {hcv['VersionNumber']}")

    # delete the config profile itself
    appconfig.delete_configuration_profile(ApplicationId=application['Id'],
 ConfigurationProfileId=config_profile['Id'])
    print(f"\tdeleted configuration profile {config_profile['Name']}
 (Id={config_profile['Id']})")

# delete all environments
list_of_env_lists = [page['Items'] for page in
 appconfig.get_paginator('list_environments').paginate(ApplicationId=application['Id'])]
for environment in [env for envs in list_of_env_lists for env in envs]:
    appconfig.delete_environment(ApplicationId=application['Id'],
 EnvironmentId=environment['Id'])
    print(f"\tdeleted environment {environment['Name']} (Id={environment['Id']})")

# delete the application itself
appconfig.delete_application(ApplicationId=application['Id'])
print(f"deleted application {application['Name']} (id={application['Id']})")
```

JavaScript

```
// this sample provides cleanup code that deletes all the AWS AppConfig resources
 created in the samples above.

// WARNING: this code will permanently delete the given application and all of its
 sub-resources, including
//   configuration profiles, hosted configuration versions, and environments. DO NOT
 run this code against
//   an application that you may need in the future.

import {
  AppConfigClient,
  paginateListApplications,
  DeleteApplicationCommand,
  paginateListConfigurationProfiles,
  DeleteConfigurationProfileCommand,
  paginateListHostedConfigurationVersions,
  DeleteHostedConfigurationVersionCommand,
  paginateListEnvironments,
  DeleteEnvironmentCommand,
} from "@aws-sdk/client-appconfig";
```

```
const client = new AppConfigClient();

// the name of the application to delete
// IMPORTANT: verify this name corresponds to the application you wish to delete
const application_name = "MyDemoApp";

// iterate over all applications, deleting ones that have the name defined above
for await (const app_page of paginateListApplications({ client }, {})) {
  for (const application of app_page.Items) {

    // skip applications that dont have the name thats set
    if (application.Name !== application_name) continue;

    console.log( `deleting application ${application.Name} (id=${application.Id})`);

    // delete all configuration profiles
    for await (const config_page of paginateListConfigurationProfiles({ client },
 { ApplicationId: application.Id })) {
      for (const config_profile of config_page.Items) {
        console.log(`\tdeleting configuration profile ${config_profile.Name} (Id=
${config_profile.Id})`);

        // delete all hosted configuration versions
        for await (const hosted_page of
 paginateListHostedConfigurationVersions({ client },
          { ApplicationId: application.Id, ConfigurationProfileId:
 config_profile.Id }
        )) {
          for (const hosted_config_version of hosted_page.Items) {
            await client.send(
              new DeleteHostedConfigurationVersionCommand({
                ApplicationId: application.Id,
                ConfigurationProfileId: config_profile.Id,
                VersionNumber: hosted_config_version.VersionNumber,
              })
            );
            console.log(`\t\tdeleted hosted configuration version
 ${hosted_config_version.VersionNumber}`);
          }
        }

        // delete the config profile itself
        await client.send(
          new DeleteConfigurationProfileCommand({
```

```
            ApplicationId: application.Id,
            ConfigurationProfileId: config_profile.Id,
          })
        );
        console.log(`\tdeleted configuration profile ${config_profile.Name} (Id=
${config_profile.Id})`)
      }

      // delete all environments
      for await (const env_page of paginateListEnvironments({ client },
 { ApplicationId: application.Id })) {
        for (const environment of env_page.Items) {
          await client.send(
            new DeleteEnvironmentCommand({
              ApplicationId: application.Id,
              EnvironmentId: environment.Id,
            })
          );
          console.log(`\tdeleted environment ${environment.Name} (Id=
${environment.Id})`)
        }
      }
    }

    // delete the application itself
    await client.send(
      new DeleteApplicationCommand({ ApplicationId: application.Id })
    );
    console.log(`deleted application ${application.Name} (id=${application.Id})`)
  }
}
```

AWS AppConfig 中的安全性

AWS 十分重视云安全性。作为 AWS 客户，您将从专为满足大多数安全敏感型组织的要求而打造的数据中心和网络架构中受益。

安全性是 AWS 和您的共同责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云的安全性 – AWS 负责保护在 AWS Cloud 中运行 AWS 服务的基础设施。AWS 还向您提供可安全使用的服务。第三方审核员定期测试和验证我们的安全性的有效性，作为 AWS Compliance Programs 的一部分。要了解适用于 AWS Systems Manager 的合规性计划，请参阅合规性计划范围内的 AWS 服务。
- 云中的安全性——您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括数据的敏感性、公司的要求以及适用的法律法规。

AWS AppConfig 是 AWS Systems Manager 的功能。了解如何在使用 AWS AppConfig 时应用责任共担模式，请参阅 Security in AWS Systems Manager。本节介绍了如何配置 Systems Manager 以实现 AWS AppConfig 的安全性和合规性目标。

实施最低权限访问

作为安全最佳实践，应授予身份在特定条件下对特定资源执行特定操作所需的最低权限。 AWS AppConfig代理提供两种功能，使代理能够访问实例或容器的文件系统：备份和写入磁盘。如果启用这些功能，请确认只有AWS AppConfig代理才有权写入文件系统上的指定配置文件。还要验证只有从这些配置文件中读取所需的进程才能读取这些配置文件。实施最低权限访问对于减小安全风险以及可能由错误或恶意意图造成的影响至关重要。

有关实现最低权限访问的更多信息，请参阅《AWS Well-Architected Tool用户指南》中的 SEC03-BP02 授予最低权限访问权限。有关本节中提及的AWS AppConfig代理功能的更多信息，请参阅其他检索功能。

AWS AppConfig 中的静态数据加密

AWS AppConfig 默认提供加密，使用 AWS 拥有的密钥 保护客户静态数据。

AWS 拥有的密钥— AWS AppConfig 默认使用这些密钥自动加密由服务部署并托管在 AWS AppConfig 数据存储中的数据。您无法查看、管理或使用 AWS 拥有的密钥，或者审查其使用情况。但是您无

需执行任何工作或更改任何计划即可保护用于加密数据的密钥。有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的 AWS 拥有的密钥。

虽然您无法禁用此加密层或选择其他加密类型，但您可以指定在保存 AWS AppConfig 数据存储中托管的配置数据以及部署配置数据时使用的客户托管密钥。

客户托管密钥 — AWS AppConfig 支持使用您创建、拥有和管理的对称客户托管密钥，在现有 AWS 拥有的密钥 密钥的基础上添加第二层加密。由于您可以完全控制这一层加密，因此可以执行以下任务：

- 建立和维护密钥政策和授权
- 建立和维护 IAM 策略
- 启用和禁用密钥政策
- 轮换密钥加密材料
- 添加标签
- 创建密钥别名
- 计划删除密钥

有关更多信息，请参阅 AWS Key Management Service 开发人员指南中的客户托管密钥。

AWS AppConfig支持客户托管密钥

AWS AppConfig 为配置数据的客户托管密钥加密提供支持。对于保存到AWS AppConfig托管数据存储的配置版本，客户可以在相应的配置文件上设置 `KmsKeyIdentifier`。每次使用 `CreateHostedConfigurationVersion` API 操作创建新版本的配置数据时，AWS AppConfig 都会从 `KmsKeyIdentifier` 中生成一个 AWS KMS 数据密钥，以便在存储数据之前对其进行加密。之后在 `GetHostedConfigurationVersion` 或 `StartDeployment` API 操作期间访问数据时，AWS AppConfig 会使用有关生成的数据密钥的信息解密配置数据。

AWS AppConfig还支持对已部署的配置数据进行客户托管密钥加密。要加密配置数据，客户可以为其部署提供 `KmsKeyIdentifier`。AWS AppConfig 使用此 `KmsKeyIdentifier` 生成 AWS KMS 数据密钥，以加密 `StartDeployment` API 操作中的数据。

AWS AppConfig 加密访问

创建客户托管密钥时，请使用以下密钥政策来确保密钥可以使用。

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "Allow use of the key",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::account_ID:role/role_name"
        },
        "Action": [
            "kms:Decrypt",
            "kms:GenerateDataKey"
            ],
        "Resource": "*"
        }
 ]
```

要使用客户托管密钥加密托管配置数据，CreateHostedConfigurationVersion 的身份调用需要以下可分配给用户、群组或角色的策略声明：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:GenerateDataKey,
            "Resource": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ]
}
```

如果您使用的是 Secrets Manager 密钥或使用客户托管密钥加密的任何其他配置数据，则您的 retrievalRoleArn 将需要 kms:Decrypt 来解密和检索数据。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kms:Decrypt",
      "Resource": "arn:aws:kms:Region:account_ID:configuration source/object"
    }
  ]
```

```
}
```

调用 AWS AppConfig StartDeploymentAPI 操作时，身份调用StartDeployment需要以下 IAM 策略，该策略可以分配给用户、群组或角色：

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey*"
      ],
      "Resource": "arn:aws:kms:Region:account_ID:key_ID"
    }
  ]
}
```

在调用 AWS AppConfig GetLatestConfigurationAPI 操作时，身份调用GetLatestConfiguration需要以下可以分配给用户、组或角色的策略：

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kms:Decrypt,
            "Resource": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ]
}
```

加密上下文

加密上下文是一组可选的键值对，可以包含有关数据的其他上下文信息。

AWS KMS 将加密上下文用作其他已经过验证的数据来支持经过身份验证的加密。在请求中包含加密上下文以加密数据时，AWS KMS 将加密上下文绑定到加密的数据。要解密数据，您必须在请求中传入相同的加密上下文。

AWS AppConfig 加密上下文：AWS AppConfig 在所有 AWS KMS 加密操作中使用加密上下文来处理加密托管配置数据和部署。上下文包含一个与数据类型对应的键和一个用于标识特定数据项的值。

为 AWS 监控您的加密密钥

当您将AWS KMS客户托管密钥与一起使用时AWS AppConfig，您可以使用AWS CloudTrail或 Amazon CloudWatch Logs 来跟踪AWS AppConfig发送到的请求AWS KMS。

以下示例是一个 CloudTrail 事件，用于监控Decrypt为访问由AWS AppConfig您的客户托管密钥加密的数据而调用的AWS KMS操作：

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "appconfig.amazonaws.com"
    },
    "eventTime": "2023-01-03T02:22:28z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "Region",
    "sourceIPAddress": "172.12.34.56",
    "userAgent": "ExampleDesktop/1.0 (V1; OS)",
    "requestParameters": {
        "encryptionContext": {
            "aws:appconfig:deployment:arn":
 "arn:aws:appconfig:Region:account_ID:application/application_ID/
environment/environment_ID/deployment/deployment_ID"
        },
        "keyId": "arn:aws:kms:Region:account_ID:key/key_ID",
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "eventID": "ff000af-00eb-00ce-0e00-ea000fb0fba0SAMPLE",
    "readOnly": true,
    "resources": [
        {
            "accountId": "account_ID",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:Region:account_ID:key_ID"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
```

```
    "recipientAccountId": "account_ID",
    "sharedEventID": "dc129381-1d94-49bd-b522-f56a3482d088"
}
```

使用接口端点 (AWS PrivateLink) 访问 AWS AppConfig

您可以使用 AWS PrivateLink 在您的 VPC 和 AWS AppConfig 之间创建私有连接。您可以像在 VPC 中一样访问 AWS AppConfig，而无需使用互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例不需要公有 IP 地址即可访问 AWS AppConfig。

您可以通过创建由 AWS PrivateLink 提供支持的接口端点来建立此私有连接。我们将在您为接口端点启用的每个子网中创建一个端点网络接口。这些是请求者托管的网络接口，用作发往 AWS AppConfig 的流量的入口点。

有关更多信息，请参阅《AWS PrivateLink 指南》中的通过 AWS PrivateLink 访问 AWS 服务。

AWS AppConfig 的注意事项

在为 AWS AppConfig 设置接口端点之前，请首先查看《AWS PrivateLink 指南》中的注意事项。

AWS AppConfig支持通过接口端点对`appconfig`和`appconfigdata`服务进行调用。

为 AWS AppConfig 创建接口端点

您可以使用 Amazon VPC 控制台或 AWS Command Line Interface (AWS CLI) 为 AWS AppConfig 创建接口端点。有关更多信息，请参阅《AWS PrivateLink 指南》中的创建接口端点。

使用以下服务名称为 AWS AppConfig 创建接口端点：

```
com.amazonaws.region.appconfig
```

```
com.amazonaws.region.appconfigdata
```

如果为接口端点启用私有 DNS，则可使用其默认区域 DNS 名称向 AWS AppConfig 发出 API 请求。例如，`appconfig.us-east-1.amazonaws.com` 和 `appconfigdata.us-east-1.amazonaws.com`。

为接口端点创建端点策略

端点策略是一种 IAM 资源，您可以将其附加到接口端点。默认端点策略允许通过接口端点完全访问 AWS AppConfig API。要控制允许从 VPC 访问 AWS AppConfig 的权限，请将自定义端点策略附加到接口端点。

端点策略指定以下信息：

- 可执行操作的主体（AWS 账户、IAM 用户和 IAM 角色）。
- 可执行的操作。
- 可对其执行操作的资源。

有关更多信息，请参阅《AWS PrivateLink 指南》中的使用端点策略控制对服务的访问权限。

示例：AWS AppConfig 操作的 VPC 端点策略

以下是自定义端点策略的一个示例。将此策略附加到接口端点时，其会向所有资源上的所有主体授予对列出的 AWS AppConfig 操作的访问权限。

```
{
   "Statement": [
      {
         "Principal": "*",
         "Effect": "Allow",
         "Action": [
            "appconfig:CreateApplication",
            "appconfig:CreateEnvironment",
            "appconfig:CreateConfigurationProfile",
            "appconfig:StartDeployment",
            "appconfig:GetLatestConfiguration"
            "appconfig:StartConfigurationSession"
         ],
         "Resource":"*"
      }
   ]
}
```

Secrets Manager 密钥轮换

本节介绍了有关 AWS AppConfig 与 Secrets Manager 集成的重要安全信息。有关 Secrets Manager 的更多信息，请参阅 AWS Secrets Manager 用户指南中的什么是 AWS Secrets Manager？。

设置 AWS AppConfig 部署的 Secrets Manager 密钥的自动轮换

Rotation 是定期更新存储在 Secrets Manager 中的密钥的过程。当轮换密钥时，会同时更新密钥和数据库或服务中的凭据。您可以通过使用 AWS Lambda 函数在 Secrets Manager 中配置自动密钥轮换，从而更新密钥和数据库。有关更多信息，请参阅 AWS Secrets Manager用户指南中的轮换 AWS Secrets Manager 密钥。

要启用 AWS AppConfig 部署的 Secrets Manager 密钥的密钥轮换，请更新您的轮换 Lambda 函数并部署轮换后的密钥。

> **Note**
>
> 在您的密钥已轮换并完全更新到新版本后，部署您的 AWS AppConfig 配置文件。您可以通过 `VersionStage` 的状态从 `AWSPENDING` 变为 `AWSCURRENT` 来确定密钥是否已轮换。密钥轮换完成发生在 Secrets Manager 轮换模板 `finish_secret` 函数中。

下面是一个示例函数，该函数在轮换密钥后启动 AWS AppConfig 部署。

```
import time
import boto3
client = boto3.client('appconfig')

def finish_secret(service_client, arn, new_version):
    """Finish the rotation by marking the pending secret as current
    This method finishes the secret rotation by staging the secret staged AWSPENDING
 with the AWSCURRENT stage.
    Args:
        service_client (client): The secrets manager service client
        arn (string): The secret ARN or other identifier
        new_version (string): The new version to be associated with the secret
    """
    # First describe the secret to get the current version
    metadata = service_client.describe_secret(SecretId=arn)
    current_version = None
    for version in metadata["VersionIdsToStages"]:
```

```
        if "AWSCURRENT" in metadata["VersionIdsToStages"][version]:
            if version == new_version:
                # The correct version is already marked as current, return
                logger.info("finishSecret: Version %s already marked as AWSCURRENT for
%s" % (version, arn))
                return
            current_version = version
            break

    # Finalize by staging the secret version current
    service_client.update_secret_version_stage(SecretId=arn, VersionStage="AWSCURRENT",
MoveToVersionId=new_version, RemoveFromVersionId=current_version)

    # Deploy rotated secret
    response = client.start_deployment(
            ApplicationId='TestApp',
            EnvironmentId='TestEnvironment',
            DeploymentStrategyId='TestStrategy',
            ConfigurationProfileId='ConfigurationProfileId',
            ConfigurationVersion=new_version,
            KmsKeyIdentifier=key,
            Description='Deploy secret rotated at ' + str(time.time())
        )

    logger.info("finishSecret: Successfully set AWSCURRENT stage to version %s for
secret %s." % (new_version, arn))
```

监控 AWS AppConfig

监控是保持 AWS AppConfig 和您的其他 AWS 解决方案的可靠性、可用性和性能的重要方面。AWS 提供了以下一些监控工具来监控 AWS AppConfig、在出现错误时进行报告并适时自动采取措施。

- AWS CloudTrail 捕获由您的 AWS 账户或代表该账户发出的 API 调用和相关事件，并将日志文件传输到您指定的 Amazon S3 桶。您可以标识哪些用户和账户调用了 AWS、发出调用的源 IP 地址以及调用的发生时间。有关更多信息，请参阅AWS CloudTrail《用户指南》。
- Amazon CloudWatch Logs 允许您监控、存储和访问来自 Amazon EC2 实例和其他来源的日志文件。 CloudTrail CloudWatch 日志可以监视日志文件中的信息，并在达到特定阈值时通知您。您还可以在高持久性存储中检索您的日志数据。有关更多信息，请参阅 Amazon CloudWatch 日志用户指南。

主题

使用 AWS CloudTrail 记录 AWS AppConfig API 调用

AWS AppConfig与AWS CloudTrail一项服务集成，该服务提供用户、角色或AWS服务在中执行的操作的记录AWS AppConfig。 CloudTrail 将所有 API 调用捕获AWS AppConfig为事件。捕获的调用包含来自 AWS AppConfig 控制台和代码的 AWS AppConfig API 操作调用。如果您创建了跟踪，则可以允许将 CloudTrail事件持续传输到 Amazon S3 存储桶，包括的事件AWS AppConfig。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向哪个请求发出AWS AppConfig、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅AWS CloudTrail用户指南。

AWS AppConfig信息在 CloudTrail

CloudTrail 在您创建账户AWS 账户时已在您的账户上启用。当活动发生在中时AWS AppConfig，该活动会与其他AWS服务 CloudTrail 事件一起记录在事件历史记录中。您可以在 AWS 账户 中查看、搜索和下载最新事件。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

对于 AWS 账户中的事件的持续记录（包括 AWS AppConfig 的事件），请创建跟踪记录。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。默认情况下，在使用控制台创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪记录在 AWS 分区中记录所有区域中的事件，并将日志文件传送至您指定的 Amazon S3 桶。此外，您可以配置其他AWS服务，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- Overview for creating a trail
- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个地区的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

所有AWS AppConfig操作均由《API 参考》记录 CloudTrail 并记录在《AWS AppConfigAPI 参考》中。例如，调用`GetApplication`和`ListApplications`操作会在 CloudTrail 日志文件中生成条目。`CreateApplication`

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 AWS Identity and Access Management（IAM）用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其它 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

AWS AppConfig中的数据事件 CloudTrail

数据事件提供有关在资源上或在资源中执行的资源操作的信息（例如，通过调用检索最新部署的配置 GetLatestConfiguration）。这些也称为数据层面操作。数据事件通常是高容量活动。默认情况下，CloudTrail 不记录数据事件。 CloudTrail 事件历史记录不记录数据事件。

记录数据事件将收取额外费用。有关 CloudTrail 定价的更多信息，请参阅AWS CloudTrail定价。

您可以使用 CloudTrail 控制台或 CloudTrail API 操作记录AWS AppConfig资源类型的数据事件。AWS CLI本节中的表格显示了可用的资源类型AWS AppConfig。

- 要使用 CloudTrail 控制台记录数据事件，请创建跟踪或事件数据存储以记录数据事件，或者更新现有的跟踪或事件数据存储以记录数据事件。

  1. 选择数据事件以记录数据事件。

2. 从数据事件类型列表中选择AWS AppConfig。
3. 选择要使用的日志选择器模板。您可以记录资源类型的所有数据事件、记录所有readOnly事件、记录所有writeOnly事件，或者创建自定义日志选择器模板来筛选readOnlyeventName、和resources.ARN字段。
4. 在选择器名称中，输入AppConfigDataEvents。有关为数据事件跟踪启用 A CloudWatch mazon Logs 的信息，请参阅记录AWS AppConfig数据平面调用的指标。

- 要使用记录数据事件AWS CLI，请将--advanced-event-selectors参数配置为将eventCategory字段设置为等于Data并将该resources.type字段设置为资源类型值（参见表）。您可以添加条件来筛选readOnlyeventName、和resources.ARN字段的值。
  - 要配置记录数据事件的跟踪，请运行put-event-selectors命令。有关更多信息，请参阅使用记录跟踪的数据事件AWS CLI。
  - 要将事件数据存储配置为记录数据事件，请运行create-event-data-store命令创建新的事件数据存储以记录数据事件，或者运行update-event-data-store命令来更新现有的事件数据存储。有关更多信息，请参阅使用记录事件数据存储的数据事件AWS CLI。

下表列出了 AWS AppConfig 资源类型。数据事件类型（控制台）列显示要从控制 CloudTrail 台上的数据事件类型列表中选择的值。res ources.type 值列显示该resources.type值，您将在使用或 API 配置高级事件选择器时指定该值。AWS CLI CloudTrail “记录到的数据 API CloudTrail” 列显示了CloudTrail针对该资源类型记录的 API 调用。

数据事件类型（控制台）	resources.type 值	数据 API 已记录到 CloudTrail *
AWS AppConfig	AWS::AppConfig::Configuration	• GetLatestConfiguration • StartConfigurationSession

*您可以将高级事件选择器配置为在eventName、和resources.ARN字段上进行筛选readOnly，以仅记录那些对您很重要的事件。有关这些字段的更多信息，请参阅AdvancedFieldSelector。

AWS AppConfig中的管理事件 CloudTrail

管理事件提供对您 AWS 账户内的资源所执行管理操作的相关信息。这些也称为控制层面操作。默认情况下， CloudTrail 记录管理事件。

AWS AppConfig将所有AWS AppConfig控制平面操作记录为管理事件。有关AWS AppConfig记录到的AWS AppConfig控制平面操作的列表 CloudTrail，请参阅 AWS AppConfigAPI 参考。

了解 AWS AppConfig 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间、请求参数等的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下示例显示了演示该StartConfigurationSession操作的 CloudTrail 日志条目。


```
{
      "eventVersion": "1.09",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "sessionIssuer": {},
          "attributes": {
            "creationDate": "2024-01-11T14:37:02Z",
            "mfaAuthenticated": "false"
          }
        }
      },
      "eventTime": "2024-01-11T14:45:15Z",
      "eventSource": "appconfig.amazonaws.com",
      "eventName": "StartConfigurationSession",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "203.0.113.0",
      "userAgent": "Boto3/1.34.11 md/Botocore#1.34.11 ua/2.0 os/macos#22.6.0
 md/arch#x86_64 lang/python#3.11.4 md/pyimpl#CPython cfg/retry-mode#legacy
 Botocore/1.34.11",
      "requestParameters": {
        "applicationIdentifier": "rrfexample",
        "environmentIdentifier": "mexampleqe0",
        "configurationProfileIdentifier": "3eexampleu1"
      },
      "responseElements": null,
      "requestID": "a1b2c3d4-5678-90ab-cdef-aaaaaEXAMPLE",
      "eventID": "a1b2c3d4-5678-90ab-cdef-bbbbbEXAMPLE",
      "readOnly": false,
      "resources": [
```

```
        {
          "accountId": "123456789012",
          "type": "AWS::AppConfig::Configuration",
          "ARN": "arn:aws:appconfig:us-east-1:123456789012:application/rrfexample/
environment/mexampleqe0/configuration/3eexampleu1"
        }
      ],
      "eventType": "AwsApiCall",
      "managementEvent": false,
      "recipientAccountId": "123456789012",
      "eventCategory": "Data",
      "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "appconfigdata.us-east-1.amazonaws.com"
      }
    }
```

记录AWS AppConfig数据平面调用的指标

如果您配置AWS CloudTrail为记录AWS AppConfig数据事件，则可以启用 Amazon Logs 来 CloudWatch 记录AWS AppConfig数据平面调用的指标。然后，您可以通过创建一个或多个指标筛选器来搜索和筛选 CloudWatch 日志中的日志数据。指标筛选器定义了发送到日志时要 CloudWatch 在日志数据中查找的术语和模式。 CloudWatch 日志使用指标筛选器将日志数据转换为数字 CloudWatch 指标。您可以绘制指标图表或使用警报对其进行配置。

开始之前

在中启用AWS AppConfig数据事件的记录AWS CloudTrail。以下过程介绍如何为中的现有AWS AppConfig跟踪启用指标记录 CloudTrail。有关如何为AWS AppConfig数据计划调用启用 CloudTrail 日志记录功能的信息，请参阅AWS AppConfig中的数据事件 CloudTrail。

使用以下过程启用 Log CloudWatch s 来记录AWS AppConfig数据平面调用的指标。

启用 Logs 来 CloudWatch 记录AWS AppConfig数据平面调用的指标

1. 打开 CloudTrail 控制台，网址为 https://console.aws.amazon.com/cloudtrail/。
2. 在仪表板上，选择您的AWS AppConfig路线。
3. 在CloudWatch 标签部分中，选择编辑。
4. 选择 Enabled (已启用)。

5. 对于日志组名称，请保留默认名称或输入名称。记录下名称。稍后您将在 “日志” 控制台中选择 CloudWatch 日志组。
6. 对于角色名称，输入一个名称。
7. 选择保存更改。

使用以下过程在 Lo CloudWatch gs AWS AppConfig 中为其创建指标和指标筛选器。该过程介绍如何为operation和的调用（可选）创建指标筛选器Amazon Resource Name (ARN)。operation

在 Lo CloudWatch gs AWS AppConfig 中为创建指标和指标筛选器

1. 打开 CloudWatch 控制台，网址为https://console.aws.amazon.com/cloudwatch/。
2. 在导航窗格中，选择 Logs（日志），然后选择 Log groups（日志组）。
3. 选中AWS AppConfig日志组旁边的复选框。
4. 选择 Actions（操作），然后选择 Create metric filter（创建指标筛选条件）。
5. 在筛选器名称中，输入一个名称。
6. 在 “筛选模式” 中，输入以下内容：

```
{ $.eventSource = "appconfig.amazonaws.com" }
```

7. （可选）在测试模式部分，从选择要测试的日志数据列表中选择您的日志组。如果 CloudTrail 尚未记录任何通话，则可以跳过此步骤。
8. 请选择 Next（下一步）。
9. 在指标命名空间中，输入**AWS AppConfig**。
10. 对于 Metric name（指标名称），请输入 **Calls**。
11. 对于 Metric value（指标值），输入 **1**。
12. 跳过 “默认值” 和 “单位”。
13. 在 “维度名称” 中，输入**operation**。
14. 在 “维度值” 中输入**$.eventName**。

    （可选）您可以输入第二个维度，其中包括发出呼叫的 Amazon 资源名称 (ARN)。要添加第二个维度，请在维度名称中输入**resource**。在 “维度值” 中输入**$.resources[0].ARN**。

    请选择 Next（下一步）。
15. 查看筛选器的详细信息并创建指标筛选器。

（可选）您可以重复此过程，为特定的错误代码（例如）创建新的指标筛选器AccessDenied。如果这样做，请输入以下详细信息：

1. 在筛选器名称中，输入一个名称。
2. 在 “筛选模式” 中，输入以下内容：

   ```
   { $.errorCode = "codename" }
   ```

   例如

   ```
   { $.errorCode = "AccessDenied" }
   ```

3. 在指标命名空间中，输入**AWS AppConfig**。
4. 对于 Metric name（指标名称），请输入 **Errors**。
5. 对于 Metric value（指标值），输入 **1**。
6. 在默认值中，输入零 (0)。
7. 跳过单位、尺寸和警报。

CloudTrail 记录 API 调用后，您可以在中查看指标 CloudWatch。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的在控制台中查看您的指标和日志。有关如何找到您创建的指标的信息，请参阅搜索可用指标。

> **Note**
>
> 如果您按照此处所述设置了没有维度的错误指标，则可以在没有维度的指标页面上查看这些指标。

为 CloudWatch指标创建警报

创建指标后，可以在中创建指标警报 CloudWatch。例如，您可以为在上一个过程中创建的AWS AppConfig通话量指标创建警报。具体而言，您可以为超出阈值AWS AppConfigStartConfigurationSession的 API 操作调用创建警报。有关如何为指标创建警报的信息，请参阅 Amazon CloudWatch 用户指南中的基于静态阈值创建 CloudWatch 警报。有关AWS AppConfig数据平面调用的默认限制的信息，请参阅中的数据平面默认限制Amazon Web Services 一般参考。

AWS AppConfig 用户指南文档历史记录

下表描述了自上次发布以来对文档所做的重要更改 AWS AppConfig。

当前 API 版本：2019-10-09

变更	说明	日期
AWS AppConfig 自定义扩展示例	演练：创建自定义 AWS AppConfig 扩展模块主题现在包含指向以下示例扩展的GitHub链接： • 使用 Systems Manager 更改日历阻止使用blocked day暂停日历进行部署的示例扩展 • 使用 git-secrets 防止机密泄漏到配置数据中的示例扩展 • 使用 Amazon Comprehend 防止个人身份信息 (PII) 泄漏到配置数据中的示例扩展	2024年2月28日
新主题：使用记录 AWS AppConfig API 调用 AWS CloudTrail	AWS AppConfig 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在中执行的操作的记录 AWS AppConfig。 CloudTrail 将所有 API 调用捕获 AWS AppConfig 为事件。这个新主题提供 AWS AppConfig特定内容，而不是链接到《AWS Systems Manager 用户指南》中的相应内容。有关更多信息，请参阅使用记录 AWS	2024 年 1 月 18 日

	AppConfig API 调用 AWS CloudTrail。	
AWS AppConfig 现在支持 AWS PrivateLink	您可以使用 AWS PrivateLink 在您的 VPC 和之间创建私有连接 AWS AppConfig。您可以像在 VPC 中 AWS AppConfig 一样进行访问，无需使用互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。VPC 中的实例不需要公有 IP 地址即可访问 AWS AppConfig。有关更多信息，请参阅AWS AppConfig 使用接口终端节点进行访问 (AWS PrivateLink)。	2023 年 12 月 6 日

其他 AWS AppConfig 代理检索功能和新的本地开发模式	AWS AppConfig Agent 还提供以下附加功能来帮助您检索应用程序的配置。 其他检索功能 • 多账户检索：使用主账户或检索 AWS 账户 中的 AWS AppConfig 代理从多个供应商账户检索配置数据。 • 将配置副本写入磁盘：使用 AWS AppConfig 代理将配置数据写入磁盘。此功能使客户能够使用从磁盘读取配置数据以进行集成的应用程序 AWS AppConfig。 **Note** 将配置写入磁盘并不是作为配置备份功能设计的。 AWS AppConfig 代理无法读取复制到磁盘的配置文件。如果要将配置备份到磁盘，请参阅在 Amazon EC2 中使用 AWS AppConfig 代理或在 Amazon ECS 和 Amazon EKS 中使用 AWS AppConfig 代理的和PRELOAD_BACKUP 环境变量。	2023 年 12 月 1 日

	本地开发模式 AWS AppConfig 代理支持本地开发模式。如果启用本地开发模式，代理将从磁盘上的指定目录中读取配置数据。它不会从中检索配置数据 AWS AppConfig。您可以通过更新指定目录中的文件来模拟配置部署。对于以下用例，我们建议使用本地开发模式： • 在使用部署不同的配置版本之前，先对其进行测试 AWS AppConfig。 • 在向代码存储库提交更改之前，测试新功能的不同配置选项。 • 测试不同的配置方案，以验证它们是否按预期运行。	
新的代码示例主题	在本指南中添加了新的代码示例主题。本主题包括 Java、Python 中的示例，以及如何 JavaScript 以编程方式执行六种常见 AWS AppConfig 操作的示例。	2023 年 11 月 17 日
修订了目录，以更好地反映 AWS AppConfig 工作流程	本用户指南中的内容现已分组在 “创建、部署、检索和扩展工作流程” 标题下。这种组织方式更好地反映了使用 AWS AppConfig 的工作流程，旨在有助于便于查找内容。	2023 年 11 月 7 日

添加了有效负载参考	为自定义 AWS AppConfig 扩展创建 Lambda 函数主题目前已包含请求和响应有效负载参考。	2023 年 11 月 7 日
新的 AWS 预定义部署策略	AWS AppConfig 现在提供并推荐了`AppConfig.Linear20PercentEvery6Minutes` 预定义的部署策略。有关更多信息，请参阅 预定义的部署策略。	2023 年 8 月 11 日
AWS AppConfig 与亚马逊 EC2 集成	您可以使用代理 AWS AppConfig 与在亚马逊弹性计算云 (Amazon EC2) Elastic Compute Cloud Linux 实例上运行的 AWS AppConfig 应用程序集成。该代理支持亚马逊 EC2 的 x86_64 和 ARM64 架构。有关更多信息，请参阅 AWS AppConfig 与 Amazon EC2 集成。	2023 年 7 月 20 日
AWS CloudFormation 支持新 AWS AppConfig 资源和功能标志示例	AWS CloudFormation 现在支持AWS::AppConfig::Extension和AWS::AppConfig::ExtensionAssociation资源，可帮助您开始使用 AWS AppConfig 扩展。 AWS::AppConfig::ConfigurationProfile和AWS::AppConfig::HostedConfigurationVersion资源现在包括在 AWS AppConfig 托管配置存储中创建功能标志配置文件的示例。	2023 年 4 月 12 日

AWS AppConfig 与集成 AWS Secrets Manager	AWS AppConfig 与。集成 AWS Secrets Manager。Secrets Manager 帮助您安全地加密、存储和检索数据库和其他服务的凭证。您可以在需要时调用 Secrets Manager 以检索您的凭证，而不是在应用程序中对凭证进行硬编码。Secrets Manager 使您能够轮换和管理对密钥的访问，从而帮助您保护对 IT 资源和数据的访问。 当您创建自由格式配置文件时，可以选择 Secrets Manager 作为配置数据的来源。在创建配置文件之前，您必须使用 Secrets Manager 注册并创建密钥。有关 Secrets Manager 的更多信息，请参阅 Secrets Manager 是什么 AWS Secrets Manager？ 在《AWS Secrets Manager 用户指南》中。有关创建配置文件的信息，请参阅创建自由格式配置文件。	2023 年 2 月 2 日

AWS AppConfig 与 Amazon ECS 和 Amazon EKS 集成	您可以使用该代理 AWS AppConfig 与亚马逊弹性容器服务 (Amazon ECS) 和亚马逊 Elastic Kubernetes Service (Amazon EKS) 集成。 AWS AppConfig 该代理充当 sidecar 容器，与您的 Amazon ECS 和 Amazon EKS 容器应用程序一起运行。该代理通过以下方式增强容器化应用程序的处理和管理： • 代理通过使用 AWS Identity and Access Management (IAM) 角色并管理配置数据的本地缓存来代表您进行调用 AWS AppConfig 。通过从本地缓存中提取配置数据，您的应用程序需要更少的代码更新来管理配置数据，在几毫秒内即可检索配置数据，且不受可能中断对此类数据的调用的网络问题的影响。 • 该代理为检索和解析 AWS AppConfig 功能标记提供了原生体验。 • 该代理开箱即用，提供了缓存策略、轮询间隔和本地配置数据可用性的最佳实践，同时跟踪后续服务调用所需的配置令牌。 • 在后台运行时，代理会定期轮询 AWS AppConfig 数据平面以获取配置数据更新。容器化应用程序可以通过连接到端口 2772（可自定义的	2022 年 12 月 2 日

	默认端口值）上的本地主机并调用 HTTP GET 来检索数据。 • AWS AppConfig 代理无需重新启动或回收容器即可更新容器中的配置数据。 有关更多信息，请参阅AWS AppConfig 与 Amazon ECS 和 Amazon EKS 集成。	
新扩展： CloudWatch Evient AWS AppConfig ly 的扩展	在推出新功能时，您可以使用 Amazon CloudWatch Evidently 向指定比例的用户提供新功能，从而安全地验证新功能。您可以监控新功能的性能，以帮助您决定何时向用户增加流量。这有助于您在完全启动该功能之前，降低风险并识别意外后果。您还可以进行 A/B 实验，以根据证据和数据制定功能设计决策。 CloudWatch Evicently 的 AWS AppConfig 扩展允许您的应用程序在本地为用户会话分配变体，而不是通过调用EvaluateFeature操作。本地会话可以降低 API 调用带来的延迟和可用性风险。有关如何配置和使用扩展程序的信息，请参阅 Amazon CloudWatch 用户指南中的使用 CloudWatchEvidently 执行启动和 A/B 实验。	2022 年 9 月 13 日

弃用 GetConfiguration API 操作	2021 年 11 月 18 日，AWS AppConfig 发布了一项新的数据平面服务。这项服务取代了之前使用 GetConfiguration API 操作检索配置数据的过程。数据平面服务使用两个新的 API 操作，StartConfigurationSession和GetLatestConfiguration。数据平面服务还使用新端点。 有关更多信息，请参阅关于 AWS AppConfig 数据平面服务。	2022 年 9 月 13 日
AWS AppConfig Agent Lambda 扩展的新版本	Agent AWS AppConfig Lambda 扩展的 2.0.122 版本现已推出。新扩展程序使用不同的 Amazon 资源名称(ARN)。有关更多信息，请参阅 AWS AppConfig 代理 Lambda 扩展程序发行说明。	2022 年 8 月 23 日
推出扩 AWS AppConfig 展	在创建或部署配置 AWS AppConfig 的工作流程中，扩展可以增强您在不同时刻注入逻辑或行为的能力。你可以使用 AWS创作的扩展，也可以创建自己的扩展。有关更多信息，请参阅使用 AWS AppConfig 扩展。	2022 年 7 月 12 日

AWS AppConfig Agent Lambda 扩展的新版本	Agent AWS AppConfig Lambda 扩展的 2.0.58 版本现已推出。新扩展程序使用不同的 Amazon 资源名称(ARN)。有关更多信息，请参阅 AWS AppConfig Lambda 扩展的可用版本。	2022 年 5 月 3 日
AWS AppConfig 与 Atlassian Jira 集成	与 Atlassian Jira 集成 AWS AppConfig ，每当你对指定的功能标记进行更改时，都可以在 Atlassian 控制台中创建和更新议题。 AWS 账户 AWS 区域每个 Jira 问题都包含标志名称、应用程序 ID、配置文件 ID 和标志值。在更新、保存和部署标志更改后，Jira 会使用此更改的详细信息更新现有问题。有关更多信息，请参阅 AWS AppConfig 与 Atlassian Jira 集成。	2022 年 4 月 7 日

ARM64 (Graviton2) 处理器的功能标志和 Lambda 扩展支持正式上市	借助 AWS AppConfig 功能标志，您可以开发新功能并将其部署到生产环境中，同时向用户隐藏该功能。首先，将标志添加 AWS AppConfig 为配置数据。该功能准备好发布后，无需部署任何代码即可更新标志配置数据。此功能可提高开发运营环境的安全性，因为您无需部署新代码即可发布该功能。有关更多信息，请参阅创建功能标志配置文件。 AWS AppConfig 中功能标志的正式发布包括以下增强功能： • 控制台包括一个将标志指定为短期标志的选项。您可以对短期标志的标志列表进行筛选和排序。 • 对于在 AWS Lambda中使用功能标志的客户，新的 Lambda 扩展程序允许您使用 HTTP 端点调用各个功能标志。有关更多信息，请参阅从功能标志配置中检索一个或多个标志。 此更新还支持为 ARM64 (Graviton2)处理器开发的 AWS Lambda 扩展程序。有关更多信息，请参阅 AWS AppConfig Lambda 扩展的可用版本。	2022 年 3 月 15 日

GetConfiguration API 操作已被弃用	`GetConfiguration` API 操作已被弃用。接收配置数据的调用应改用 `StartConfigurationSession` 和 `GetLatestConfiguration` API。有关这些 API 及其使用方法的更多信息，请参阅检索配置。	2022 年 1 月 28 日
Lambda AWS AppConfig 扩展的新区域 ARN	AWS AppConfig Lambda 扩展已在新的亚太地区（大阪）区域推出。在该区域中创建 Lambda 时需要提供 Amazon 资源名称（ARN）。有关亚太地区（大阪）地区 ARN 的更多信息，请参阅添加 Lambda 扩展 AWS AppConfig。	2021 年 3 月 4 日
AWS AppConfig Lambda 扩展	如果您使用 AWS AppConfig 管理 Lambda 函数的配置，那么我们建议您添加 Lambda 扩展模块 AWS AppConfig 。此扩展包括可在降低成本的 AWS AppConfig 同时简化使用的最佳实践。降低成本的原因是减少了对 AWS AppConfig 服务的 API 调用，而且 Lambda 函数处理时间缩短也降低了成本。有关更多信息，请参阅AWS AppConfig 与 Lambda 扩展程序集成。	2020 年 10 月 8 日
新章节	添加了一个新章节，其中提供了设置 AWS AppConfig的说明。有关更多信息，请参阅设置 AWS AppConfig。	2020 年 9 月 30 日

添加了命令行程序	本用户指南中的过程现在包括 AWS Command Line Interface (AWS CLI) 的命令行步骤和适用于 Windows 的工具。PowerShell有关更多信息，请参阅使用 AWS AppConfig。	2020 年 9 月 30 日
发布 AWS AppConfig 用户指南	使用 AWS AppConfig、的 AWS Systems Manager功能创建、管理和快速部署应用程序配置。 AWS AppConfig 支持对任何规模的应用程序进行受控部署，并包括内置的验证检查和监控。您可以 AWS AppConfig 与 EC2 实例上托管的应用程序、容器 AWS Lambda、移动应用程序或 IoT 设备一起使用。	2020 年 7 月 31 日

AWS 术语表

有关最新的 AWS 术语，请参阅《AWS 词汇表参考》中的 AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。