开发人员指南

AWS App Runner

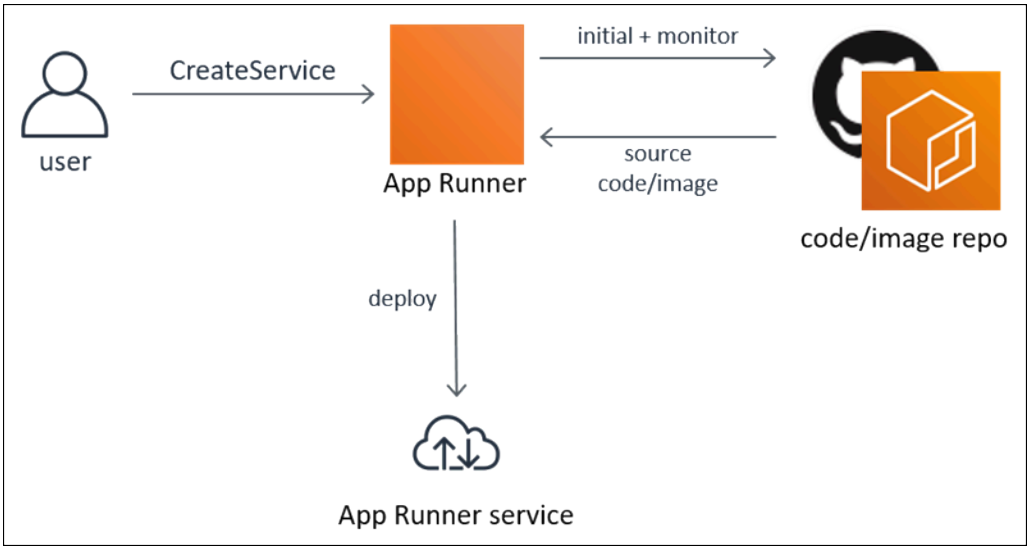
AWS App Runner: 开发人员指南

Table of Contents

什么是 AWS App Runner？

AWS App Runner是一个AWS服务，它提供了一种快速、简单且经济高效的方式来从源代码或容器映像直接部署到AWS云。您不需要学习新技术、决定要使用的计算服务或了解如何配置和配置AWS资源的费用。

应用程序 Runner 直接连接到您的代码或图像存储库。它提供了一个自动集成和交付管道，具有完全托管的操作、高性能、可扩展性和安全性。



应用程序亚军为谁服务？

如果您是开发人员，您可以使用 App Runner 简化部署新版本的代码或映像存储库的过程。

适用于操作团队，App Runner 会在每次将提交推送到代码存储库或将新容器映像版本推送到映像存储库时启用自动部署。

访问应用程序运行程序

您可以使用以下任一接口定义和配置 App Runner 服务部署：

- 应用程序 Runner 控制台— 提供用于管理应用程序运行程序服务的 Web 界面。
- 应用程序运行程序 API— 提供用于执行应用程序运行程序操作的 REST 风格 API。有关更多信息，请参阅 AWS App Runner API 参考。
- AWS命令行界面 (AWS CLI) — 提供大量通过AWS服务（包括 Amazon VPC ），同时被 Windows、macOS 和 Linux 支持。有关更多信息，请参阅AWS Command Line Interface。

- AWS开发工具包— 提供特定于语言的 API，并关注许多连接详细信息，例如计算签名、处理请求重试和错误处理。有关更多信息，请参阅 AWS 软件开发工具包。

应用程序运行者的定价

App Runner 提供了一种经济高效的方式来运行您的应用程序。您只需为您的 App Runner 服务使用的资源付费。当请求流量较慢时，您的服务可以缩减到较少的计算实例。您可以控制可扩展性设置：预配置实例的最低和最高数量，以及实例处理的最高负载。

有关 App Runner 自动扩展的更多信息，请参阅the section called “Auto Scaling”。

有关定价信息，请参阅 AWS App Runner 定价。

接下来做什么

了解如何通过以下主题开始使用 App Runner：

- 设置— 完成使用应用程序运行程序的必备步骤。
- 入门— 将第一个应用程序部署到应用程序运行程序。

设置应用程序运行者

如果你是一个新的AWS客户，请先完成此页上列出的安装先决条件，然后再开始使用AWS App Runner。

对于这些设置过程，您可以使用AWS Identity and Access Management(IAM) 服务。有关 IAM 的完整信息，请参阅以下参考资料：

- AWS Identity and Access Management (IAM)
- IAM User Guide

创建 AWS 账户

当您注册AWS，您将获得一个帐号，可以访问AWS优惠，包括AWS App Runner。

如果您已有AWS 账户，请跳到下一个必备任务。

如果您没有AWS账户，请完成以下步骤创建一个账户。

如何注册 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，您将接到一通电话，要求您使用电话键盘输入一个验证码。

创建 IAM 用户

要访问AWS服务时，需要提供凭证。这些凭据决定身份验证（你是谁）和authorization（您必须在哪些权限上执行操作AWS资源的费用）。

首次创建 Amazon Web Services (AWS) 账户，您将以单一登录身份开始。此身份具有对所有AWS服务和资源的费用。此身份称为AWS账户根用户。在登录时，请输入您用于创建账户的电子邮件地址和密码。

> **Important**
>
> 强烈建议您不使用根用户执行日常任务，即使是管理任务。相反，请遵循仅使用根用户创建您的第一个 IAM 用户的最佳实践。然后请妥善保存根用户凭证，仅用它们执行少数账户和服务管理任务。要查看需要您以根用户身份登录的任务，请参阅需要根用户凭证的任务。

有关根用户和 IAM 用户凭证的更多信息，请参阅AWS 账户根用户证书和 IAM 用户证书中的AWS一般参考。

自行创建管理员用户并将该用户添加到管理员组（控制台）

1. 登录到IAM 控制台作为帐户所有者，方法是选择根用户并输入您的AWS账户电子邮件地址。在下一页上，输入您的密码。

   > **Note**
   >
   > 强烈建议您遵守使用**Administrator**IAM 用户遵守并安全锁定根用户凭证。只在执行少数账户和服务管理任务时才作为根用户登录。

2. 在导航窗格中，选择用户，然后选择添加用户。
3. 对于 User name (用户名)，输入 **Administrator**。
4. 选中 旁边的复选框AWS Management Console访问。然后选择自定义密码，并在文本框中输入新密码。
5. （可选）默认情况下，AWS要求新用户在首次登录时创建新密码。您可以清除 User must create a new password at next sign-in (用户必须在下次登录时创建新密码) 旁边的复选框以允许新用户在登录后重置其密码。
6. 选择 Next:。Permissions (下一步：权限)。
7. 在设置权限下，选择将用户添加到组。
8. 选择创建组。
9. 在 Create group (创建组) 对话框中，对于 Group name (组名称)，输入 **Administrators**。
10. 选择筛选策略，然后选择。AWS托管-作业功能来过滤表格内容。
11. 在策略列表中，选中 AdministratorAccess 的复选框。然后选择 Create group (创建组)。

> **Note**
>
> 您必须先激活 IAM 用户和角色对账单的访问权限，然后才能使用AdministratorAccess权限以访问AWS Billing and Cost Management控制台。为此，请按照“向账单控制台委派访问权限”教程第 1 步中的说明进行操作。

12. 返回到组列表中，选中您的新组所对应的复选框。如有必要，选择 Refresh 以在列表中查看该组。
13. 选择 Next:。标签。
14. （可选）通过以键值对的形式附加标签来向用户添加元数据。有关在 IAM 中使用标签的更多信息，请参阅标记 IAM 实体中的IAM 用户指南。
15. 选择 Next:。审核以查看要添加到新用户的组成员资格的列表。如果您已准备好继续，请选择 Create user。

您可使用此相同的流程创建更多的组和用户，并允许您的用户访问 AWS 账户资源。要了解有关使用限制用户对特定AWS资源，请参阅访问管理和示例策略。

> **Important**
>
> 保护AWS 账户。切勿向组织以外的任何人发送或共享您的凭据。合法代表 Amazon 的任何人永远都不会要求您提供凭证。

创建 IAM 用户后，请使用其凭证登录AWS Management Console。有关更多信息，请参阅 。IAM 用户如何登录AWS 账户中的IAM 用户指南。

为您的 IAM 用户创建访问密钥

访问密钥包含访问密钥 ID 和秘密访问密钥，用于对向AWS。如果您没有访问密钥，您可以在AWS Management Console。作为最佳实践，请勿使用AWS账户根用户访问密钥，以执行不必要的任务。相反，创建新的管理员 IAM 用户，为您自己提供访问密钥。

您只能在创建密钥时查看或下载私有访问密钥。以后您无法恢复这些属性。但是，您可以随时创建新的访问密钥。您还必须拥有执行所需 IAM 操作的权限。有关更多信息，请参阅 。访问 IAM 资源所需的权限中的IAM 用户指南。

为 IAM 用户创建访问密钥

1. 登录 AWS Management Console，单击 https://console.aws.amazon.com/iam/ 打开 IAM 控制台。
2. 在导航窗格中，选择 Users。
3. 选择要创建其访问密钥的用户的名称，然后选择安全凭证选项卡。
4. 在访问密钥部分中，选择。创建访问密钥。
5. 要查看新的访问 key pair，请选择。Show (显式)。关闭此对话框后，您将无法再次访问该秘密访问密钥。您的凭证与如下显示类似：

   - 访问密钥 ID：秋季花园 7 个例子
   - 专属访问密钥：wJalrrXUtnFEMI/K7MDENG/bPxRPxRecrRecrRecrRecrRecrRecrRecurtxRecrRec
6. 要下载密钥对，请选择下载 .csv 文件。将密钥存储在安全位置。关闭此对话框后，您将无法再次访问该秘密访问密钥。

   请对密钥保密以保护您的AWS账户，切勿发送电子邮件。请勿对组织外部共享密钥，即使有来自 AWS 或 Amazon.com 的询问。合法代表 Amazon 的任何人永远都不会要求您提供秘密密钥。
7. 在您下载`.csv`文件中，选择Close。在创建访问密钥时，默认情况下，密钥对处于活动状态，并且您可以立即使用此密钥对。

相关主题

- 什么是 IAM？中的IAM 用户指南
- AWS安全凭证在AWS一般参考

接下来做什么

您已完成必备步骤。要将第一个应用程序部署到 App Runner，请参阅入门。

开始使用应用程序

AWS App Runner是一个AWS服务，它提供了一种快速、简单且经济高效的方法，将现有容器映像或源代码直接转换为AWS Cloud。

本教程介绍了如何使用AWS App Runner将应用程序部署到应用程序运行程序服务。它演示了配置源代码和部署、服务构建和服务运行时。它还显示了如何部署代码版本、进行配置更改以及查看日志。最后，本教程介绍了如何清理您在遵循教程的过程中创建的资源。

主题

Prerequisites

开始使用本教程之前，确保执行以下操作：

1. 完成中的设置步骤设置。
2. 创建GitHub账户（如果您还没有）。如果您是 GitHub 的新用户，请参阅GitHub 入门中的GitHub 文档。
3. 在 GitHub 账户中创建存储库。本教程使用存储库名称`python-hello`。使用以下示例中指定的名称和内容在存储库的根目录中创建文件。

的文件**`python-hello`**示例存储库

Example requirements.txt

```
Click==7.0
```

```
Flask==1.0.2
itsdangerous==1.1.0
Jinja2==2.10
MarkupSafe==1.1.1
Werkzeug==0.15.5
```

Example server.py

```
from flask import Flask
import os

PORT = 8080
name = os.environ['NAME']
if name == None or len(name) == 0:
  name = "world"
MESSAGE = "Hello, " + name + "!"
print("Message: '" + MESSAGE + "'")

app = Flask(__name__)


@app.route("/")
def root():
  print("Handling web request. Returning message.")
  result = MESSAGE.encode("utf-8")
  return result


if __name__ == "__main__":
  app.run(debug=True, host="0.0.0.0", port=PORT)
```

第 1 步：创建应用程序 Runner 服务

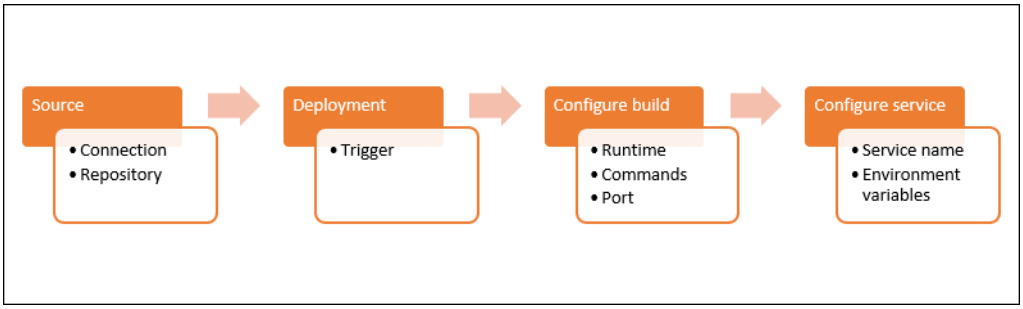
在此步骤中，您将根据您在 GitHub 上创建的示例源代码存储库创建一个 App Runner 服务，该存储库是the section called “Prerequisites”。该示例包含一个简单的 Python 网站。以下是创建服务所采取的主要步骤：

1. 配置您的源代码。
2. 配置源部署。
3. 配置应用程序构建。

4. 配置您的服务。

5. 查看并确认。

下图概述了创建 App Runner 服务的步骤：

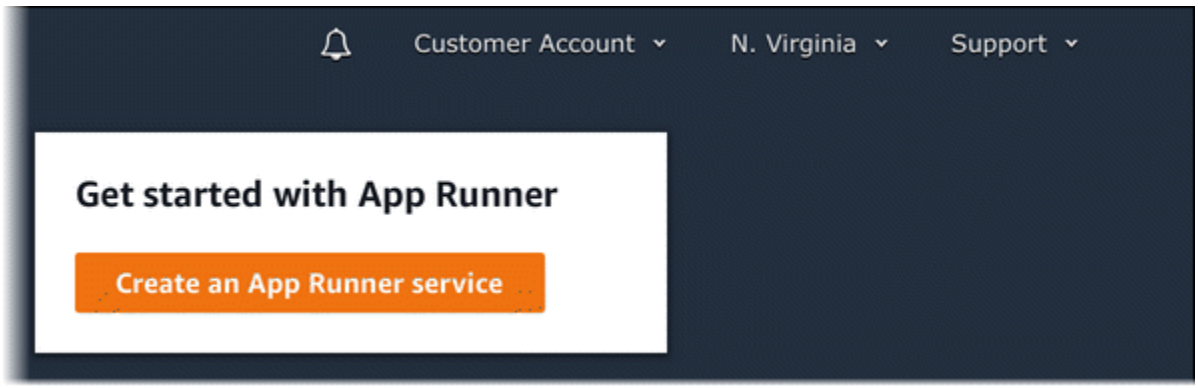


基于源代码存储库创建 App Runner 服务

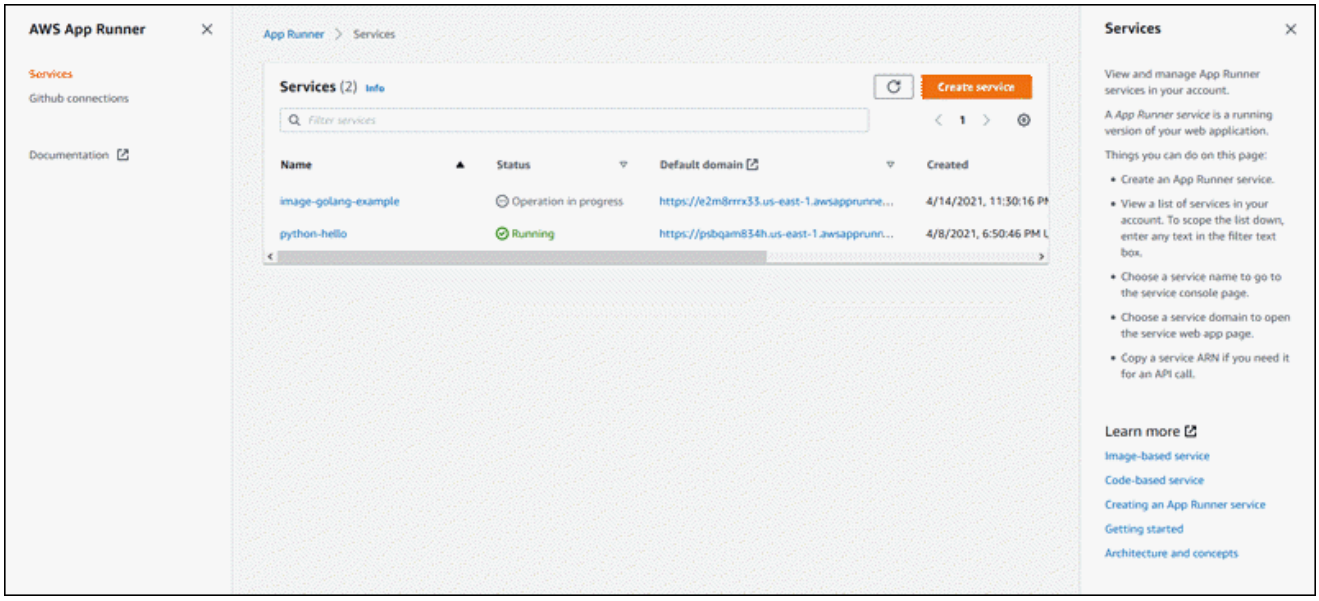
1. 配置您的源代码。

   a. 打开应用程序 Runner 控制台，并在区域列表中选择您的AWS 区域。

   b. 如果AWS 账户还没有任何 App Runner 服务，则会显示控制台主页。选择创建应用程序 Runner 服务。



   如果AWS 账户具有现有服务，服务页面，其中包含您的服务列表。选择 Create service。



c. 在存储库的源和部署页面上的源部分，适用于存储库类型中，选择源代码存储库。

d. UNDERConnect 到 GitHub选择添加新，然后，如果系统提示，请提供您的 GitHub 凭证。

> **Note**
>
> 以下步骤以安装AWSGitHub 连接器连接到您的 GitHub 帐户是一次性步骤。您可以重复使用连接来创建基于此帐户中的存储库的多个 App Runner 服务。当您拥有现有连接时，请选择该连接并跳到存储库选择。

e. 在安装AWSGitHub 的连接器对话框中，如果出现提示，请选择您的 GitHub 帐户名称。

f. 如果系统提示您授权AWS用于 GitHub 的连接器，请选择授权 AWS 连接。

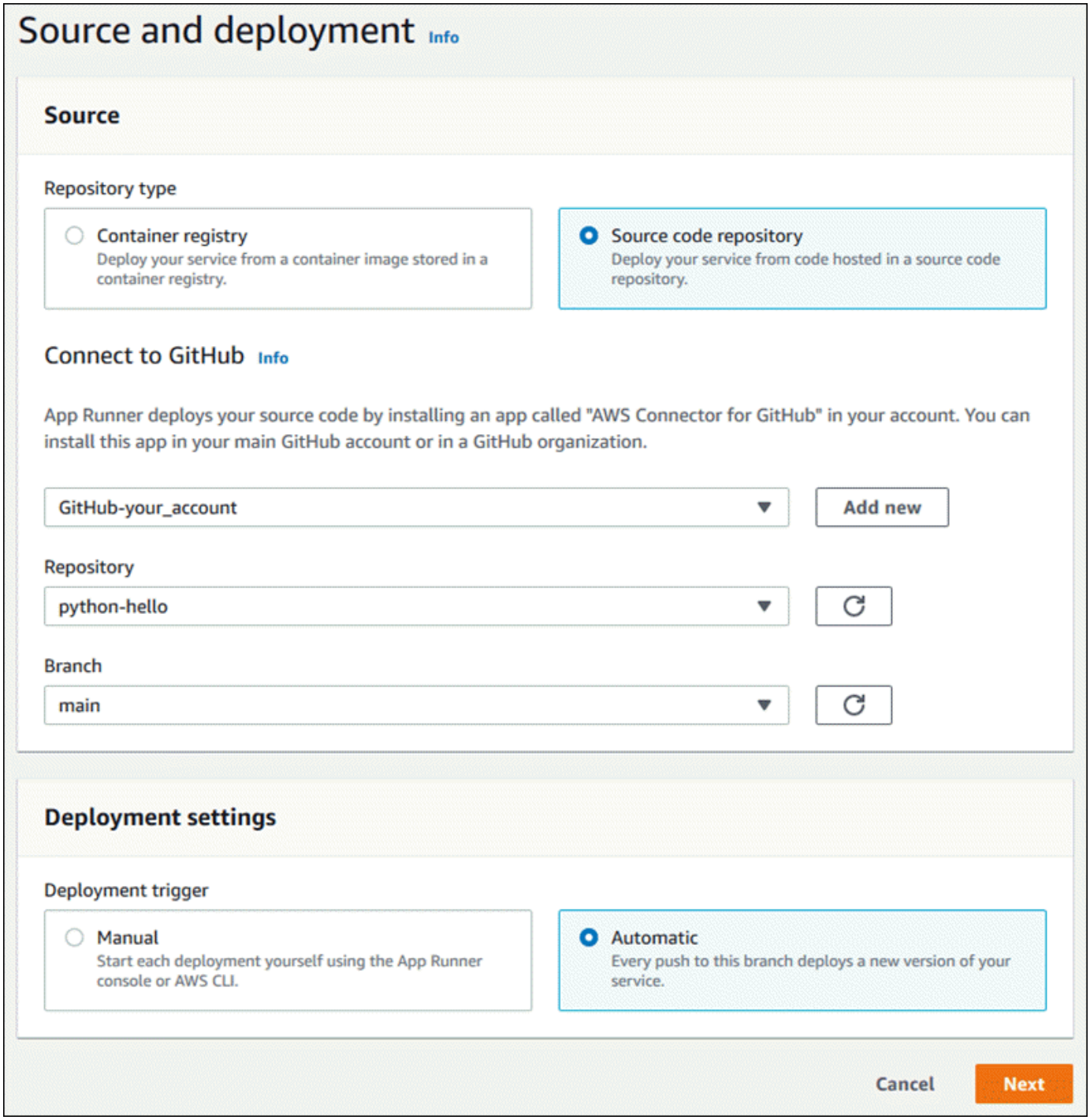
g. 选择安装。

您的帐户名称将显示为所选GitHub 帐户/组织。现在，您可以在您的账户中选择存储库。

h. 适用于Repository中，选择您创建的示例存储库`python-hello`。适用于分支中，选择存储库的默认分支名称（例如主色）。

2. 配置您的部署：在部署设置部分，选择。自动，然后选择。下一步。

> **Note**
>
> 通过自动部署，对存储库的每次新提交都会自动部署服务的新版本。



3. 配置应用程序构建。

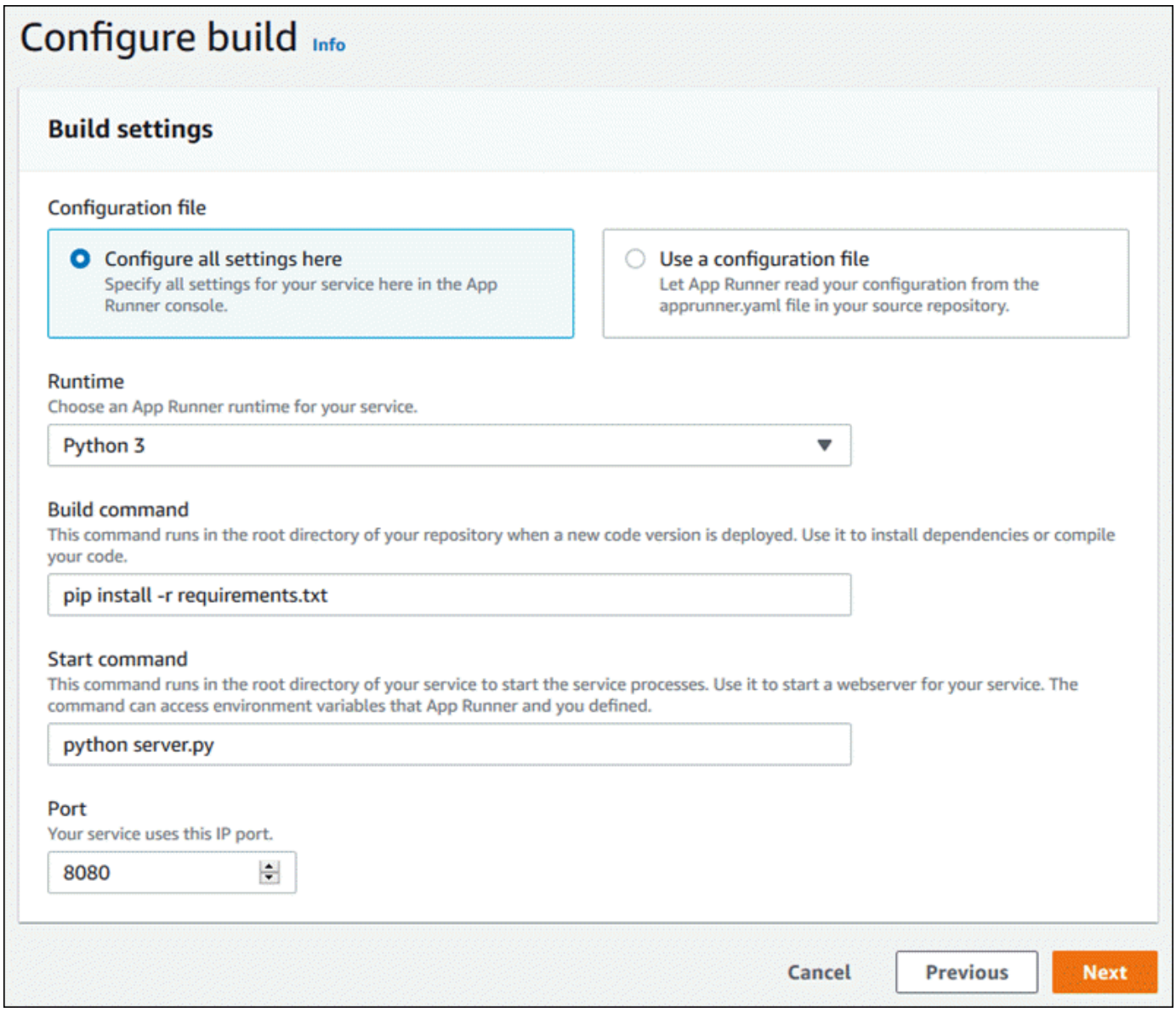
   a. 在存储库的配置生成页面, 适用于配置文件中，选择在此处配置所有设置。

   b. 提供以下构建设置：

- 运行时— 选择。Python 3。
- 生成命令— 输入**`pip install -r requirements.txt`**。
- 启动命令— 输入**`python server.py`**。
- 端口— 输入**`8080`**。

c. 选择 Next。

> **Note**
>
> Python 3 运行时使用基本 Python 3 图像和您的示例 Python 代码构建一个 Docker 映像。然后，它启动运行此映像的容器实例的服务。



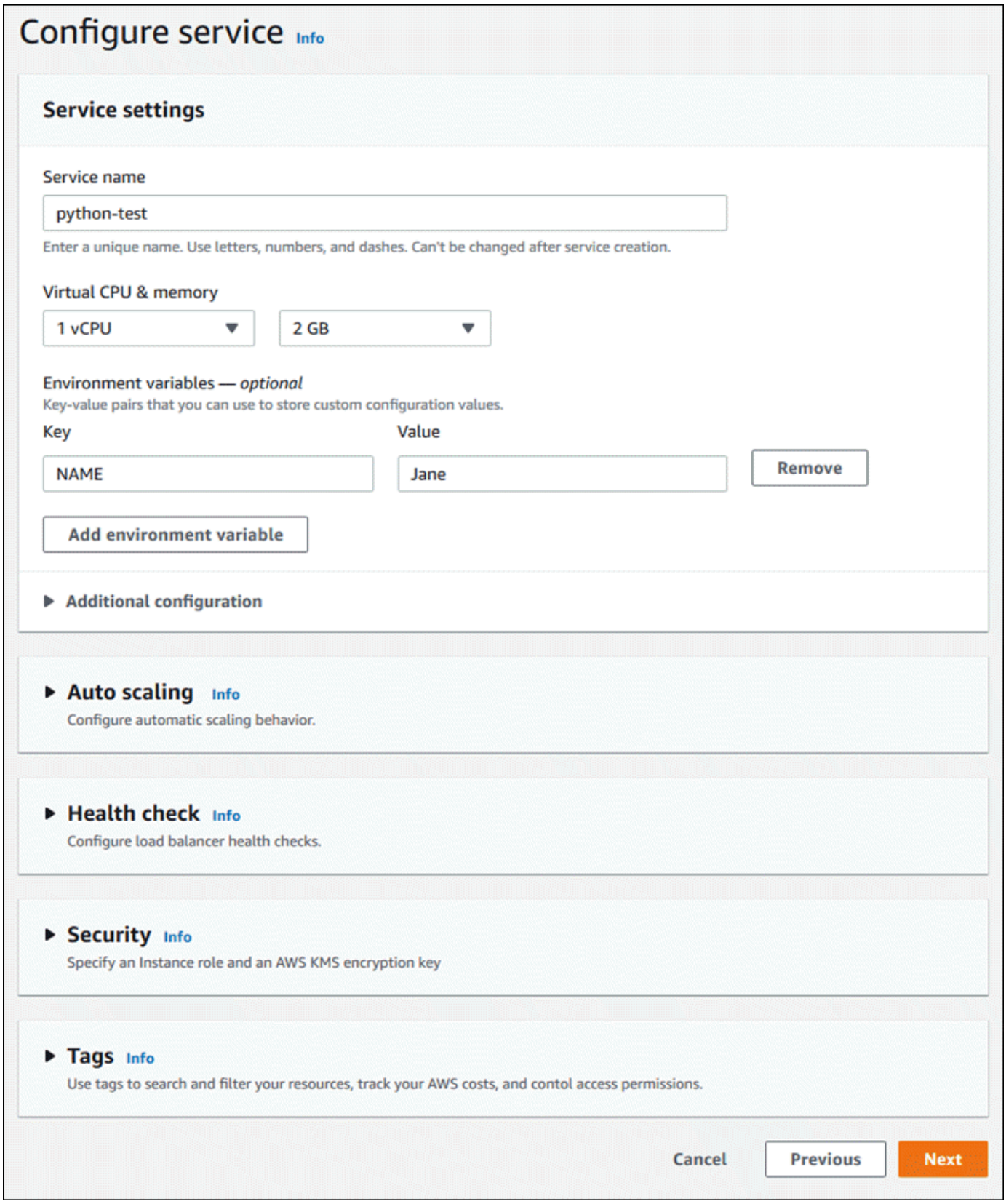
4. 配置您的服务。

   a. 在存储库的配置服务页面上的服务设置部分中，输入服务名称。

   b. UNDER环境变量中，添加单个环境变量。适用于密钥输入。**NAME**，以及值中，输入任何名称（例如，您的名字）。

> **Note**
>
> 示例应用程序读取您在此环境变量中设置的名称，并在其网页上显示该名称。

c. 选择 Next。


Configure service Info
Service settings
Service name
python-test
Enter a unique name. Use letters, numbers, and dashes. Can't be changed after service creation.
Virtual CPU & memory
1 vCPU
2 GB
Environment variables — optional
Key-value pairs that you can use to store custom configuration values.
Key
Value
NAME
Jane
Remove
Add environment variable
Additional configuration
Auto scaling Info
Configure automatic scaling behavior.
Health check Info
Configure load balancer health checks.
Security Info
Specify an Instance role and an AWS KMS encryption key
Tags Info
Use tags to search and filter your resources, track your AWS costs, and contol access permissions.
Cancel
Previous
Next

5. 在存储库的审核和创建页面上，验证您输入的所有详细信息，然后选择创建和部署。

   如果服务创建成功，控制台将显示服务仪表板，并显示服务概览的新服务。

6. 验证您的服务是否正在运行。

   a. 在服务仪表板页面上，等待服务状态是正在运行。

   b. 选择默认域值-它是指向您服务网站的 URL。

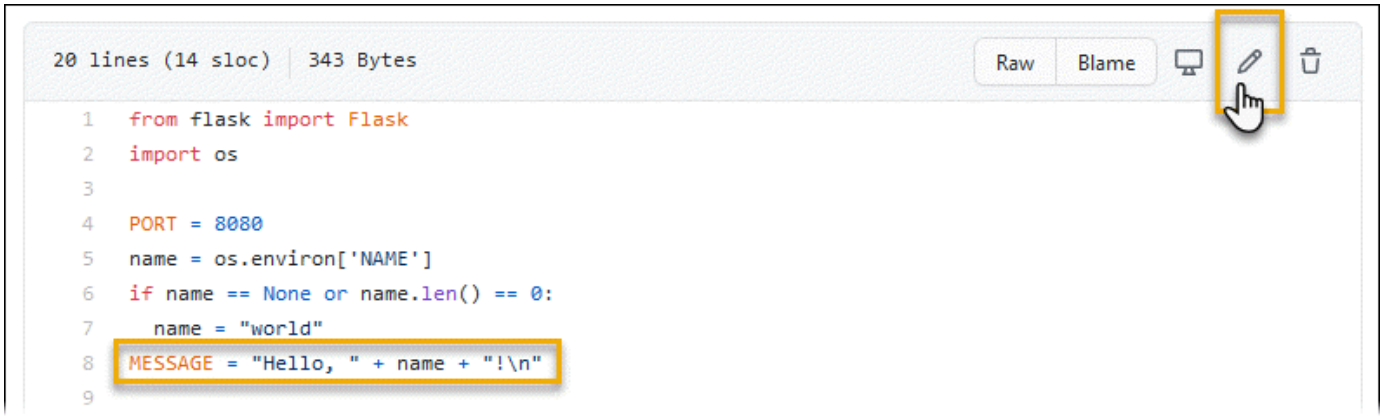
      将显示一个网页：Hello####！

第 2 步：更改您的服务代码

在此步骤中，您将更改源代码存储库。应用程序运行程序 CI/CD 功能会自动构建并将更改部署到您的服务中。

更改您的服务代码

1. 导航到您的示例 GitHub 存储库。
2. 选择文件名称`server.py`导航到该文件。
3. 选择编辑此文件（铅笔图标）。
4. 在分配给变量的表达式`MESSAGE`，更改文本`Hello`到`Good morning`。



5. 选择提交更改。

   新的提交开始部署。在服务仪表板页面上，状态对 的更改正在进行中的操作。

6. 等待部署结束。在服务仪表板页面上，状态应该更改回正在运行。

7. 验证部署是否成功：刷新显示服务网页的浏览器选项卡。

   此页面现在显示已修改的消息：早上好**####**！

第 3 步：进行配置更改

在这个步骤中，您将更改**NAME**环境变量值，以演示服务配置更改。

要查看服务的日志

1. 打开应用程序 Runner 控制台，并在区域列表中选择您的AWS 区域。

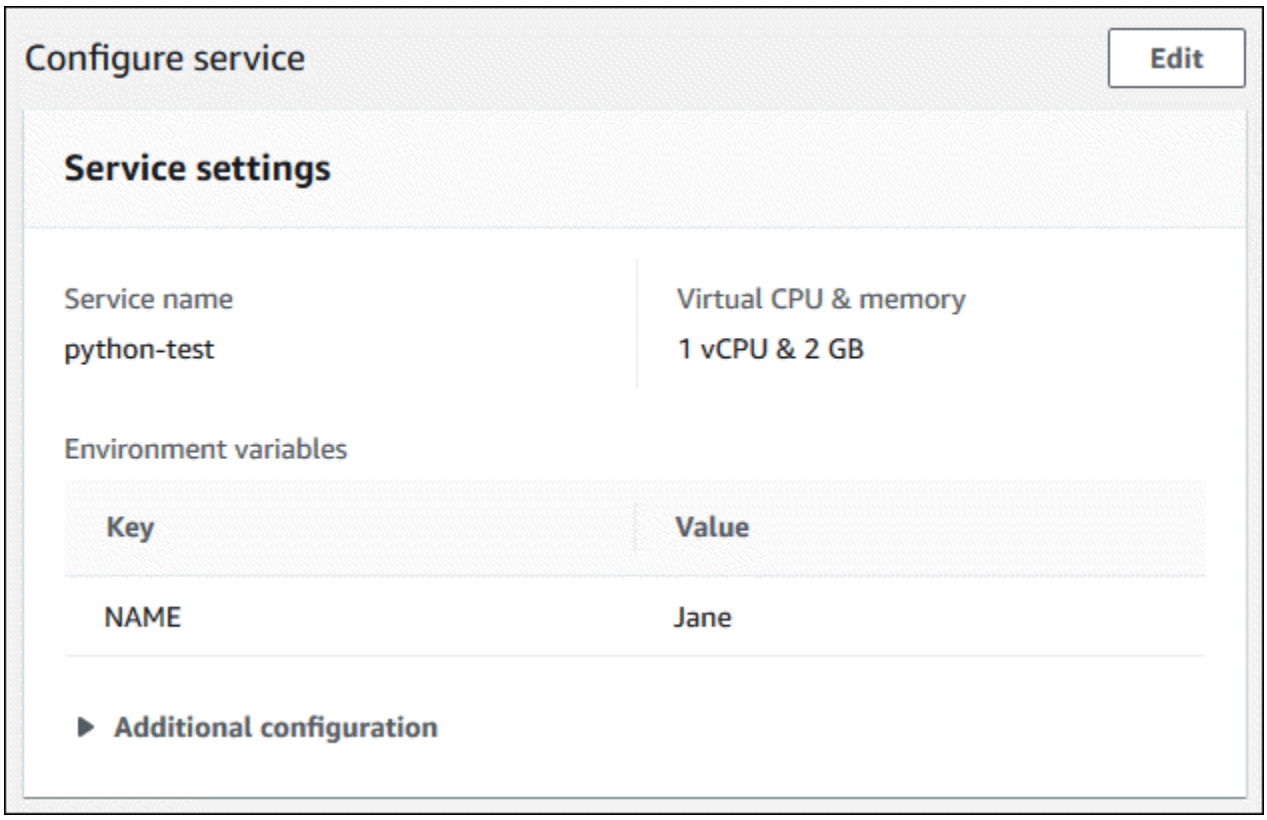
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板服务概览。

3. 在服务控制面板页面上，选择配置选项卡。

   控制台分为几个部分显示您的服务配置设置。

4. 在配置服务部分，选择。编辑。



5. 对于带有键的环境变量**NAME**中，将值更改为不同名称。
6. 选择 Apply changes。

   应用程序运行程序将开始更新过程。在服务仪表板页面上，状态对 的更改正在进行中的操作。
7. 请等待更新结束。在服务仪表板页面上，状态应该更改回正在运行。
8. 验证更新是否成功：刷新显示服务网页的浏览器选项卡。

   该页面现在显示修改后的名称：早上好**###**！

第 4 步：查看服务的日志

在此步骤中，您可以使用 App Runner 控制台查看 App Runner 服务的日志。应用程序运行程序将日志流式传输到 Amazon CloudWatch Logs（CloudWatch 日志），并将其显示在您的服务的仪表板上。有关应用程序运行程序日志的信息，请参阅the section called “日志（CloudWatch Logs）”。

要查看服务的日志

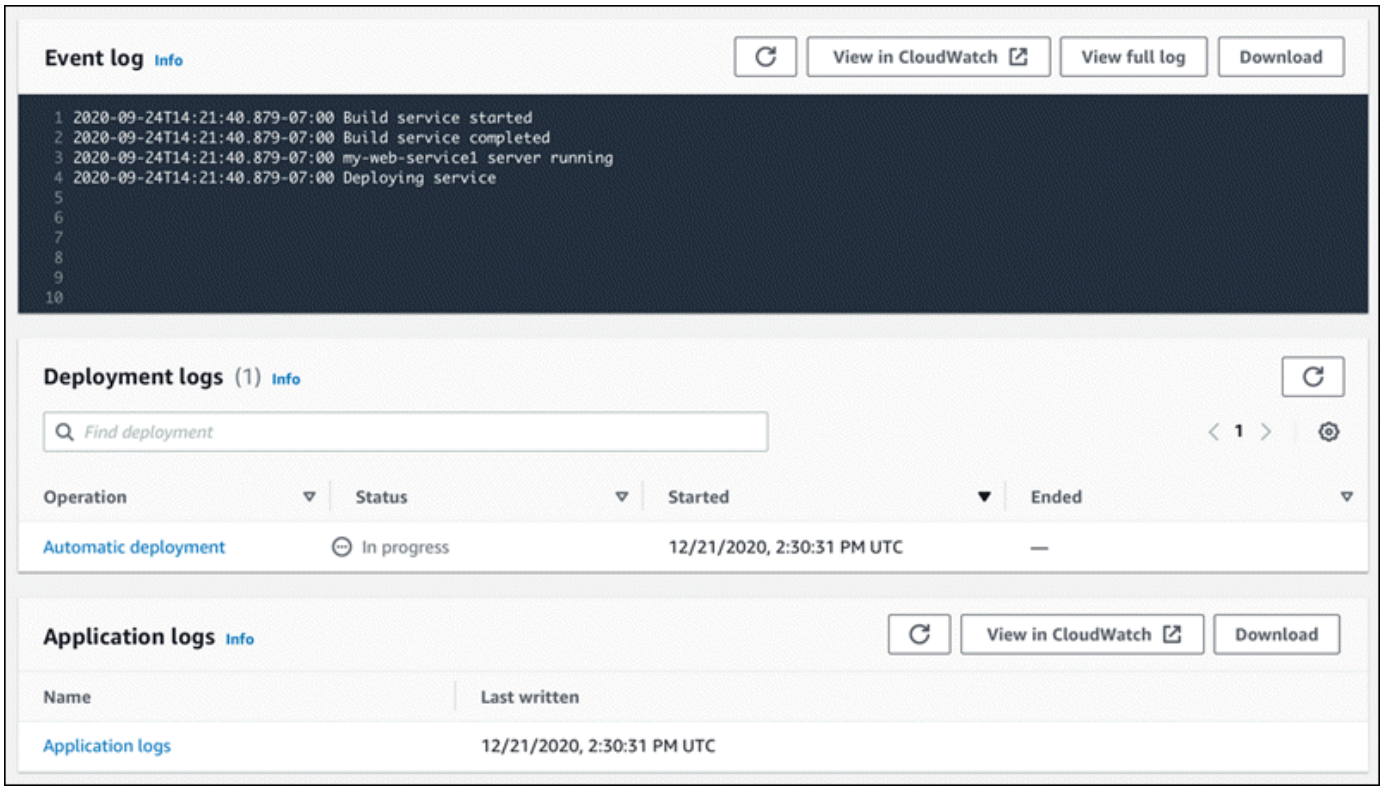
1. 打开应用程序 Runner 控制台，并在区域列表中选择您的AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

控制台将显示服务仪表板服务概览。

3. 在服务控制面板页面上，选择日志选项卡。

   控制台在几个部分显示几种类型的日志：

   - 事件日志— 应用程序运行程序服务生命周期中的活动。控制台将显示最新事件。
   - 部署日志— 将源存储库部署到您的应用程序运行程序服务。控制台为每个部署显示单独的日志流。
   - 应用程序日志— 部署到您的 App Runner 服务的 Web 应用程序的输出。控制台将所有正在运行的实例的输出结合到单个日志流中。



4. 要查找特定部署，请通过输入搜索词来缩小部署日志列表的范围。您可以搜索表中显示的任何值。
5. 要查看日志的内容，请选择查看完整日志（事件日志）或日志流名称（部署和应用程序日志）。
6. 选择下载以下载日志。对于部署日志流，请先选择一个日志流。
7. 选择CloudWatch CloudWatch打开 CloudWatch 控制台，并使用其全部功能来浏览您的应用程序运行程序服务日志。对于部署日志流，请先选择一个日志流。

> **Note**
>
> 如果您希望查看特定实例的应用程序日志而不是组合的应用程序日志，CloudWatch 控制台特别有用。

第 5 步：清除

现在，您已经学会了如何创建 App Runner 服务、查看日志以及进行一些更改。在此步骤中，您将删除该服务以删除不再需要的资源。

删除您的服务

1. 在服务控制面板页面上，选择操作，然后选择。删除服务。
2. 在确认对话框中，输入请求的文本，然后选择Delete。

   结果：控制台将导航到服务页. 您刚刚删除的服务将显示DELETING。很短的时间后，它从列表中消失。

同时考虑删除在本教程中创建的 GitHub 连接。有关更多信息，请参阅the section called “连接”。

接下来做什么

现在，您已经部署了第一个 App Runner 服务，请参阅以下主题了解更多信息：

- 体系结构和概念— 体系结构、主要概念和AWS与应用程序运行者相关的资源。
- 基于映像的服务和基于代码的服务— App Runner 可以部署的两种类型的应用程序源。
- 面向应用程序运行者的开发— 在开发或迁移应用程序代码以部署到 App Runner 时，您应该了解的事情。
- 应用程序 Runner— 使用 App Runner 控制台管理和监控您的服务。
- 管理您的服务— 管理应用程序运行程序服务的生命周期。
- 日志记录和监控— 通过查看指标、阅读日志和跟踪服务操作调用来监控您的 App Runner 服务。
- 应用程序运行程序配置文件— 一种基于配置的方法，用于指定应用程序 Runner 服务的构建和运行时行为的选项。

- 应用程序开发工具包— 使用 App Runner 应用程序编程接口 (API) 来创建、读取、更新和删除 App Runner 资源。
- 安全性— 不同的方式AWS，并且在使用 App Runner 和其他服务时确保云安全。

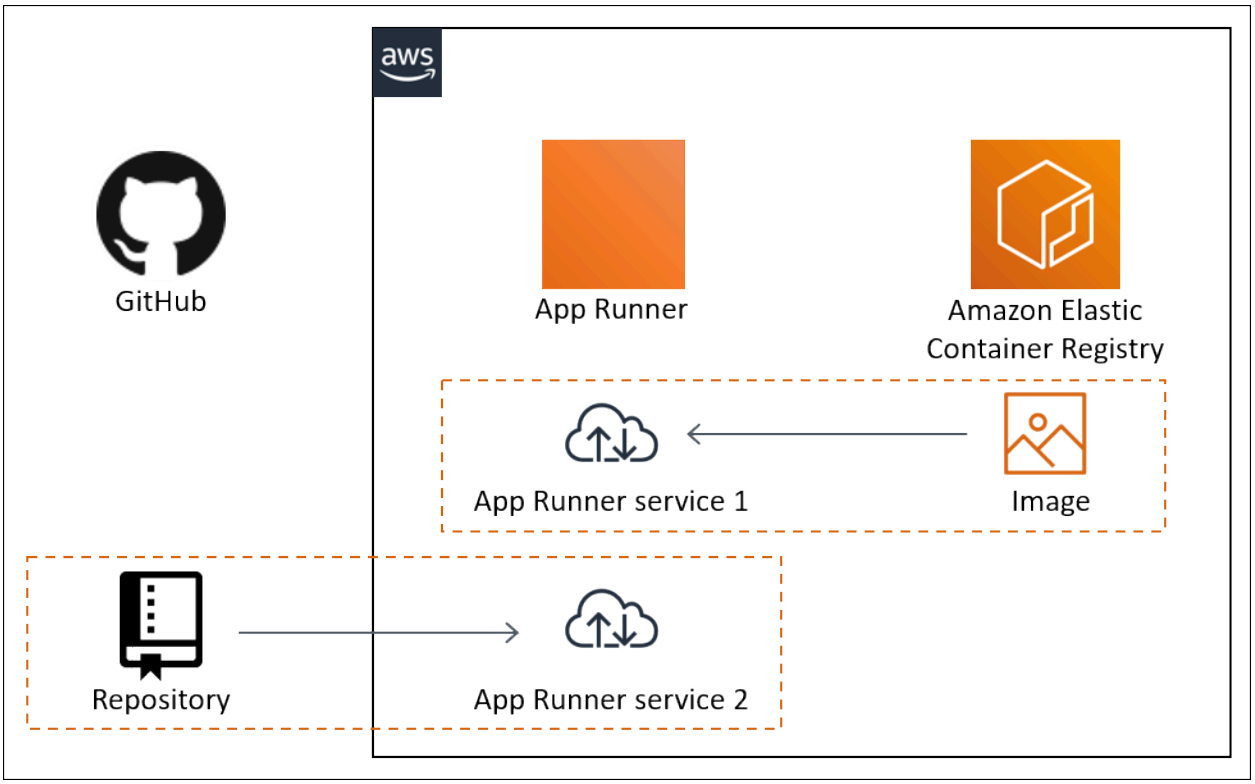
应用程序运行者体系结构和概念

AWS App Runner从存储库中获取源代码或源图像，并在AWS Cloud。通常情况下，您只需调用一个应用程序运行程序操作，CreateService，以创建您的服务。

通过源映像存储库，您可以提供一个随时可用的容器映像，App Runner 可以部署该映像来运行 Web 服务。使用源代码存储库，您可以提供有关构建和运行 Web 服务的代码和说明，这些服务针对由 App Runner 管理的多个运行时环境之一而设计。

此时，App Runner 可以从GitHub存储库，或者从Amazon Elastic Container Registry (Amazon ECR)在您的AWS 账户。

下图概述了 App Runner 服务体系结构。在图中，有两个示例服务：一个部署来自 GitHub 的源代码，另一个部署来自 Amazon ECR 的源映像。



App Runner 概念

以下是与在 App Runner 中运行的 Web 服务相关的关键概念：

- App Runner 服务— 一个AWS资源，App Runner 根据源代码存储库或容器映像部署和管理您的应用程序。应用程序运行程序服务是应用程序的正在运行的版本。有关创建服务的更多信息，请参阅the section called “Creation”。
- 源类型— 您为部署应用程序 Runner 服务而提供的源资料库的类型：源代码或者源镜像。
- 存储库提供商— 包含应用程序源的存储库服务（例如GitHub或者Amazon ECR)。
- App Runner 连接— 一个AWS资源，该资源允许 App Runner 访问存储库提供商帐户（例如 GitHub 帐户或组织）。有关连接的更多信息，请参阅the section called “连接”。
- 运行时— 用于部署源代码存储库的基础映像。App Runner 提供了各种管理的运行时适用于不同的编程环境。有关更多信息，请参阅基于代码的服务。
- 部署— 将源存储库的版本（代码或图像）应用到 App Runner 服务的操作。服务的第一次部署是作为服务创建的一部分进行的。以后的部署可以通过以下两种方式之一进行：
  - 自动部署— CI/CD 功能。您可以将 App Runner 服务配置为自动构建（针对源代码）并在存储库中显示的应用程序的每个版本进行部署。这可以是源代码存储库中的新提交，也可以是源映像存储库中的新映像版本。
  - 手动部署— 您显式启动的应用程序 Runner 服务的部署。
- CUSTOM 域— 您与应用程序运行程序服务关联的域。Web 应用程序的用户可以使用此域访问您的 Web 服务，而不是默认的 App Runner 子域。有关更多信息，请参阅the section called “自定义域名”。
- 维护— App Runner 偶尔在运行您的 App Runner 服务的基础架构上执行的活动。维护正在进行时，服务状态会临时更改为OPERATION_IN_PROGRESS(正在进行操作在控制台中）几分钟。在此期间阻止对服务执行的操作（例如，部署、配置更新、暂停/恢复或删除）。几分钟后，当服务状态返回到RUNNING。

> **Note**
>
> 如果您的操作失败，这并不意味着您的 App Runner 服务已关闭。您的应用程序处于活动状态并持续处理请求。您的服务不太可能遇到任何停机时间。

特别是，如果 App Runner 检测到托管服务的底层硬件中存在问题，则会迁移您的服务。为了防止任何服务停机，App Runner 会将您的服务部署到一组新的实例，并将流量转移到这些实例（蓝绿色部署）。您可能偶尔会看到费用略有暂时增加。

App Runner 资源

使用 App Runner 时，您可以在AWS 账户。这些资源用于访问您的代码和管理您的服务。

下表提供了这些资源的概览：

资源名称	描述
Service	表示应用程序的正在运行的版本。本指南的大部分其余部分介绍了服务类型、管理、配置和监控。 ARN：`arn:aws:apprunner:` *`region`*`:`*`account-id`* `:service/` *`service-name`* `[/`*`service-id`* `]`
Connection	为您的 App Runner 服务提供访问由第三方提供商存储的私有存储库的权限。作为单独的资源存在，用于跨多个服务共享。有关连接的更多信息，请参阅the section called “连接”。 ARN：`arn:aws:apprunner:` *`region`*`:`*`account-id`* `:connection/` *`connection-name`* `[/`*`connection-id`* `]`
AutoScalingConfiguration	为您的 App Runner 服务提供控制应用程序自动缩放的设置。作为单独的资源存在，用于跨多个服务共享。有关自动扩展的更多信息，请参阅the section called “Auto Scaling”。 ARN：`arn:aws:apprunner:` *`region`*`:`*`account-id`* `:autoscalingconfiguration/` *`config-name`* `[/`*`config-revision`* `[/`*`config-id`* `]]`

App Runner 资源配额

AWS对您的帐户规定了一些配额（也称为限制）AWS资源使用量AWS 区域。下表列出了与 App Runner 资源相关的配额。配额也将在AWS App Runner终端节点和配额中的AWS一般参考。

资源配额	描述	默认值	可调整?
Services	您可以在账户中为每个创建的最大服务数AWS 区域。	10	✓ 是

资源配额	描述	默认值	可调整?
Connections	您可以在账户中创建的每个连接的最大数量AWS 区域。您可以跨多个服务共享单个连接。	10	✓ 是
Auto scaling configurations-名称	您可以在自动扩展配置中创建的唯一名称的最大数量，这些名称在您的帐户中为每个AWS 区域。您可以在多个服务中使用自动扩展配置。	10	✓ 是
Auto scaling configurations-每个名称的修订	您可以在账户中为每个创建的 Auto Scaling 配置修订的最大数量AWS 区域对应每个唯一名称。您可以在多个服务中使用自动扩展配置修订版。	10	X 否

大多数配额是可调的，您可以请求增加配额。有关更多信息，请参阅 。请求增加配额在 Service Quotas 用户指南中。

基于源图像的应用程序运行程序服务

您可以使用AWS App Runner基于两种根本不同类型的服务源创建和管理服务：源代码和源镜像。无论源类型如何，App Runner 都会负责启动、运行、扩展和负载平衡服务。您可以使用 App Runner 的 CI/CD 功能来跟踪源图像或代码的更改。当 App Runner 发现更改时，它会自动生成（针对源代码）并将新版本部署到您的 App Runner 服务。

本章讨论基于源映像的服务。有关基于源代码的服务的信息，请参阅基于代码的服务。

A源镜像是存储在图像存储库中的公共或私有容器映像。您将 App Runner 指向图像，并启动基于此映像运行容器的服务。无需构建阶段。相反，您提供了一个可随时部署的映像。

镜像存储库提供商

App Runner 支持以下镜像存储库提供程序：

- Amazon Elastic Container Registry (Amazon ECR)— 将私有图像存储在您的AWS 账户。
- Amazon Elastic Container Registry Pblic（Amazon ECR）— 存储可公开读取的图像。

从 Amazon ECR 部署

Amazon ECR将图像存储在存储库中。有私有和公共存储库。要从私有存储库将映像部署到 App Runner 服务，App Runner 需要获得许可才能从 Amazon ECR 读取映像。要将该权限授予应用程序运行者，您需要为应用程序运行者提供访问角色。这是一个AWS Identity and Access Management(IAM)角色，该角色具有必要 Amazon ECR 操作权限。当您使用 App Runner 控制台创建服务时，您可以选择帐户中的现有角色。或者，您可以使用 IAM 控制台创建新的自定义角色，或者选择使用 App Runner 控制台根据托管策略为您创建角色。

当您使用应用程序运行程序 API 或AWS CLI，您将完成一个分为两个步骤的流程。首先，使用 IAM 控制台创建访问角色。您可以使用 App Runner 提供的托管策略，也可以输入您自己的自定义权限。然后，您可以在服务创建过程中使用CreateServiceAPI 操作。

有关创建 App Runner 服务的信息，请参阅the section called “Creation”。

从亚马逊 ECR 公共部署

Amazon ECR 公共存储可公开读取的图像。以下是亚马逊 ECR 和亚马逊 ECR 公共之间的主要区别，您应在应用程序运行程序服务上下文中注意这些区别：

- 亚马逊 ECR 公共图片可公开读取。在基于 Amazon ECR 公共映像创建服务时，您无需提供访问角色。
- 应用程序运行程序不支持亚马逊 ECR 公共映像的自动部署。

基于源代码的应用程序运行程序服务

可以使用AWS App Runner基于两种根本不同类型的服务源创建和管理服务：源代码和源镜像。无论源类型如何，App Runner 都会负责启动、运行、扩展和负载平衡您的服务。您可以使用 App Runner 的 CI/CD 功能来跟踪源图像或代码的更改。当 App Runner 发现更改时，它会自动生成（针对源代码）并将新版本部署到您的 App Runner 服务。

本章讨论基于源代码的服务。有关基于源映像的服务的信息，请参阅基于映像的服务。

源代码是 App Runner 为您构建和部署的应用程序代码。您可以将 App Runner 指向源代码存储库，然后选择合适的runtime。App Runner 构建基于运行时的基本映像和应用程序代码的映像。然后，它启动一个基于此映像运行容器的服务。

应用程序运行程序提供方便的语言特定托管运行时。这些运行时中的每个运行时都会根据源代码构建容器图像，并将语言运行时依赖项添加到映像中。您不需要提供容器配置和构建指令，例如 Docker 文件。

本章的副主题讨论了各种运行时应用程序运行程序支持的通用 Docker 文件运行时和托管运行时适用于不同的编程环境。

主题

- 源代码存储库提供程序
- 应用程序运行器管理的运行时
- 使用 Python 托管运行时
- 使用 Node.js 托管运行时

源代码存储库提供程序

App Runner 通过从源代码存储库读取源代码来部署源代码。应用程序 Runner 支持一个源代码存储库提供程序：GitHub。

从 GitHub 部署

将源代码部署到应用程序运行程序服务GitHub存储库中，应用程序运行程序会建立到 GitHub 的连接。如果您的存储库是私有的（也就是说，它不能在 GitHub 上公开访问），则必须向 App Runner 提供连接详细信息。当您使用应用程序运行程序控制台创建服务，可以将连接详细信息作为服务创建过程的一部分提供。

当您使用应用程序运行程序 API 或AWS CLI，则连接是一个单独的资源。首先，使用CreateConnectionAPI 操作。然后，您在服务创建过程中提供连接的 ARN，使用CreateServiceAPI 操作。

有关 App Runner 服务创建更多信息，请参阅。the section called “Creation”。有关应用程序运行程序的更多信息，请参阅。the section called “连接”。

应用程序运行器管理的运行时

App Runner 为各种编程环境提供托管运行时。每个托管运行时都可以根据特定的编程语言或运行时环境轻松构建和运行容器。当您使用托管运行时，App Runner 将以托管运行时映像启动。此图像基于Amazon Linux Docker 镜像并包含一个语言运行时包以及一些工具和流行的依赖包。App Runner 使用此托管运行时映像作为基础映像，并添加应用程序代码以构建 Docker 映像。然后，它将部署此映像以在容器中运行 Web 服务。

您可以指定应用程序运行程序服务的运行时创建服务使用应用程序运行程序控制台或CreateServiceAPI。您还可以将运行时指定为源代码的一部分。使用`runtime`关键字应用程序运行程序配置文件，您将其包含在代码存储库中。托管运行时的命名约定是*`<language-name> <major-version>`*。

App Runner 会在每次部署或服务更新时将服务的运行时更新为最新版本。如果您的应用程序需要特定版本的托管运行时，您可以使用`runtime-version`关键字应用程序运行程序配置文件。将次要版本指定为*`<major>`*。*`<minor>`*锁定主要版本和次要版本（App Runner 仅更新修补程序版本）。将特定修补程序级别指定为*`<major>`*。*`<minor>`*。*`<patch>`*来锁定您的服务在特定的运行时版本上（App Runner 永远不会更新运行时）。

使用 Python 托管运行时

AWS App Runner提供了一个 Python 托管运行时。运行时实现了轻松构建和运行具有基于 Python 的 Web 应用程序的容器。当您使用 Python 运行时，应用程序运行程序从托管的 Python 运行时映像开始。此图像基于Amazon Linux 码头映像，并包含 Python 运行时包以及一些工具和流行的依赖包。App Runner 使用此托管运行时映像作为基础映像，并添加应用程序代码以构建 Docker 映像。然后，它将部署此映像以在容器中运行 Web 服务。

您可以指定应用程序运行程序服务的运行时创建服务使用应用程序运行程序控制台或CreateServiceAPI。您还可以将运行时指定为源代码的一部分。使用`runtime`关键字应用程序运行程序配置文件，您将其包含在代码存储库中。托管运行时的命名约定是*`<language-name> <major-version>`*。

有关有效的 Python 运行时名称，请参阅the section called “版本信息”。

App Runner 会在每次部署或服务更新时将服务的运行时更新为最新版本。如果您的应用程序需要特定版本的托管运行时，您可以使用`runtime-version`关键字应用程序运行程序配置文件。将次要版本指定为`<major>`。`<minor>`锁定主要版本和次要版本（App Runner 仅更新修补程序版本）。将特定修补程序级别指定为`<major>`。`<minor>`。`<patch>`来锁定您的服务在特定的运行时版本上（App Runner 永远不会更新运行时）。

主题

- Python 运行时配置
- Python 运行时示例
- Python 运行时版本信息

Python 运行时配置

选择托管运行时，还必须至少配置生成和运行命令。您可以配置它们，而creating或者正在更新您的应用程序运行者服务。可通过几种方法实现操作：

- 使用应用程序运行程序控制台— 指定配置生成部分或配置选项卡。
- 使用应用程序运行程序 API— 调用CreateService或者UpdateService。指定命令使用`BuildCommand`和`StartCommand`的成员代码配置值数据类型。
- 使用配置文件— 指定最多三个构建阶段的一个或多个构建命令，以及用于启动应用程序的单个运行命令。还有其他可选配置设置。

提供配置文件是可选的。使用控制台或 API 创建 App Runner 服务时，您可以指定 App Runner 是在创建过程中直接获取配置设置，还是从配置文件获取配置设置。

Python 运行时示例

以下示例显示了用于构建和运行 Python 服务的应用程序运行器配置文件。最后一个示例是可以部署到 Python 运行时服务的完整 Python 应用程序的源代码。

Python 配置文件

此示例显示了可与 Python 托管运行时一起使用的最小配置文件。有关 App Runner 使用最小配置文件做出的假设，请参阅the section called “配置文件”。

Example 呼吸器 .yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install pipenv
      - pipenv install
run:
  command: python app.py
```

Python 配置文件

此示例说明了在 Python 托管运行时使用所有配置密钥。

Example 呼吸器 .yaml

```
version: 1.0
runtime: python3
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/DOC-EXAMPLE-BUCKET/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip install pipenv
      - pipenv install
    post-build:
      - python manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.7.7
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
```

```
      value: "example"
```

端到端 Python 应用程序源

此示例显示了可以部署到 Python 运行时服务的完整 Python 应用程序的源代码。

Example requirements.txt

```
Click==7.0
Flask==1.0.2
itsdangerous==1.1.0
Jinja2==2.10
MarkupSafe==1.1.1
Werkzeug==0.15.5
```

Example server.py

```
from flask import Flask
import os

PORT = 8080
name = os.environ['NAME']
if name == None or len(name) == 0:
  name = "world"
MESSAGE = "Hello, " + name + "!"
print("Message: '" + MESSAGE + "'")

app = Flask(__name__)


@app.route("/")
def root():
  print("Handling web request. Returning message.")
  result = MESSAGE.encode("utf-8")
  return result


if __name__ == "__main__":
  app.run(debug=True, host="0.0.0.0", port=PORT)
```

Example 呼吸器 .yaml

```
version: 1.0
```

```
runtime: python3
build:
  commands:
    build:
      - pip install -r requirements.txt
run:
  command: python server.py
```

Python 运行时版本信息

本主题列出了 App Runner 支持的 Python 运行时版本的完整详细信息。

Python 3

Detail	描述
运行时名称	Python3
次要版本	3.7、3.8
包含的包裹	蟒蛇, 点子, 设置工具, 车轮, 虚拟

使用 Node.js 托管运行时

AWS App Runner提供了一个 Node.js 托管的运行时。运行时可轻松构建和运行具有 Node.jS 基于 Web 应用程序的容器。当您使用 Node.js 运行时，应用程序运行程序将以托管 Node.js 运行时映像启动。此图像基于Amazon Linux Docker 镜像并包含 Node.js 运行时软件包和一些工具。App Runner 使用此托管运行时映像作为基础映像，并添加应用程序代码以构建 Docker 映像。然后，它将部署此映像以在容器中运行 Web 服务。

您可以指定应用程序运行程序服务的运行时创建服务使用应用程序运行程序控制台或CreateServiceAPI。您还可以将运行时指定为源代码的一部分。使用`runtime`的关键字应用程序运行程序配置文件，您将其包含在代码存储库中。托管运行时的命名约定是*`<language-name>`* *`<major-version>`*。

有关有效的 Node.js 运行时名称，请参阅the section called “版本信息”。

App Runner 会在每次部署或服务更新时将服务的运行时更新为最新版本。如果您的应用程序需要特定版本的托管运行时，您可以使用`runtime-version`关键字，在应用程序运行程序配置文件。将次要版本指定为*`<major>`*。*`<minor>`*锁定主要版本和次要版本（App Runner 仅更新修补程序版本）。将特

定修补程序级别指定为*<major>*。*<minor>*。*<patch>*来锁定您的服务在特定的运行时版本上（App Runner 永远不会更新运行时）。

主题

- Node.js 运行时配置
- Node.js 运行时示例
- Node.js 运行时版本信息

Node.js 运行时配置

选择托管运行时，还必须至少配置生成和运行命令。您可以配置它们，而creating或者正在更新您的应用程序运行者服务。可通过几种方式执行此操作：

- 使用应用程序运行程序控制台— 指定配置生成部分或配置选项卡。
- 使用应用程序运行程序 API— 调用CreateService或者UpdateService。指定命令使用`BuildCommand`和`StartCommand`的成员代码配置值数据类型。
- 使用配置文件— 指定最多三个构建阶段的一个或多个构建命令，以及用于启动应用程序的单个运行命令。还有其他可选配置设置。

提供配置文件是可选的。使用控制台或 API 创建 App Runner 服务时，您可以指定 App Runner 是在创建过程中直接获取配置设置，还是从配置文件获取配置设置。

使用 Node.js 运行时，您还可以使用名为`package.json`放在源存储库的根目录中。使用此文件，您可以配置 Node.js 引擎版本、依赖项包和各种命令（命令行应用程序）。npm 或 yarn 等软件包管理器将此文件解释为其命令的输入。

例如：

- npm install会安装由`dependencies`和`devDependencies`中的节点`package.json`。
- npm start或者npm run start运行由`scripts/start`中的节点`package.json`。

下面是一个 `package.json` 示例文件。

package.json

```
{
  "name": "node-js-getting-started",
```

```
  "version": "0.3.0",
  "description": "A sample Node.js app using Express 4",
  "engines": {
    "node": "12.18.4"
  },
  "scripts": {
    "start": "node index.js",
    "test": "node test.js"
  },
  "dependencies": {
    "cool-ascii-faces": "^1.3.4",
    "ejs": "^2.5.6",
    "express": "^4.15.2"
  },
  "devDependencies": {
    "got": "^11.3.0",
    "tape": "^4.7.0"
  }
}
```

有关 的更多信息`package.json`，请参阅package.json 指南Node.js 网站上。

Tips

- 如果您的`package.json`文件定义了start命令，您可以将其用作run命令，如以下示例所示。

  Example

  package.json

  ```
  {
    "scripts": {
      "start": "node index.js"
    }
  }
  ```

  呼吸器 .yaml

  ```
  run:
    command: npm start
  ```

- 当您运行npm install在您的开发环境中，npm 会创建文件package-lock.json。此文件包含刚刚安装的软件包版本 npm 的快照。此后，当 npm 安装依赖关系时，它使用这些确切的版本。同样，纱线创建yarn.json。将这些文件提交到源代码存储库中，以确保您的应用程序与您开发和测试的依赖关系版本一起安装。
- 您还可以使用应用程序运行程序配置文件来配置 Node.js 版本和启动命令。当您执行此操作时，这些定义将覆盖package.json。之间的冲突node版本在package.json和runtime-version值会导致应用程序运行程序构建阶段失败。

Node.js 运行时示例

以下示例显示了用于构建和运行 Node.js 服务的应用程序运行器配置文件。

Node.js 配置文件最小

此示例显示了可与 Node.js 托管运行时一起使用的最小配置文件。有关 App Runner 使用最小配置文件做出的假设，请参阅the section called “配置文件”。

Example 呼吸器 .yaml

```
version: 1.0
runtime: nodejs12
build:
  commands:
    build:
      - npm install --production
run:
  command: node app.js
```

Node.js 扩展配置文件

此示例说明在 Node.js 托管运行时使用所有配置密钥。

Example 呼吸器 .yaml

```
version: 1.0
runtime: nodejs12
build:
  commands:
```

```
    pre-build:
      - npm install --only=dev
      - node test.js
    build:
      - npm install --production
    post-build:
      - node node_modules/ejs/postinstall.js
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 12.18.4
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Node.js 应用程序与咕噜

此示例说明如何配置 Grunt 开发的 Node.js 应用程序。咕噜是一个命令行 JavaScript 任务运行程序。它运行重复性任务并管理流程自动化，以减少人为错误。Grunt 和 Grunt 插件是使用 npm 安装和管理的。您可以通过包含`Gruntfile.js`文件放在源存储库的根目录中。

Example package.json

```
{
  "scripts": {
    "build": "grunt uglify",
    "start": "node app.js"
  },
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0"
  },
  "dependencies": {
    "express": "^4.15.2"
  },
}
```

Example Gruntfile.js

```
module.exports = function(grunt) {

  // Project configuration.
  grunt.initConfig({
    pkg: grunt.file.readJSON('package.json'),
    uglify: {
      options: {
        banner: '/*! <%= pkg.name %> <%= grunt.template.today("yyyy-mm-dd") %> */\n'
      },
      build: {
        src: 'src/<%= pkg.name %>.js',
        dest: 'build/<%= pkg.name %>.min.js'
      }
    }
  });

  // Load the plugin that provides the "uglify" task.
  grunt.loadNpmTasks('grunt-contrib-uglify');

  // Default task(s).
  grunt.registerTask('default', ['uglify']);

};
```

Example 呼吸器 .yaml

```
version: 1.0
runtime: nodejs12
build:
  commands:
    pre-build:
      - npm install grunt grunt-cli
      - npm install --only=dev
      - npm run build
    build:
      - npm install --production
run:
  runtime-version: 12.18.4
  command: node app.js
  network:
    port: 8000
```

```
    env: APP_PORT
```

Node.js 运行时版本信息

本主题列出了应用程序运行程序支持的 Node.js 运行时版本的完整详细信息。

Node.js 12

Detail	描述
运行时名称	节点
次要版本	latest
包含的包裹	节点（包括 npm），纱线

开发应用程序运行者的应用程序代码

本章讨论在开发或迁移应用程序代码以部署到AWS App Runner。

Runtime 信息

无论您提供容器映像还是 App Runner 为您构建容器映像，App Runner 都会在容器实例中运行您的应用程序代码。以下是容器实例运行时环境的几个关键方面。

- 框架支持— App Runner 支持任何实现 Web 应用程序的图像。它与您选择的编程语言以及您使用的 Web 应用程序服务器或框架（如果您使用的话）无关。为了您的方便起见，我们提供了特定于语言的托管运行时，以简化应用程序构建过程和抽象映像创建。
- 网络请求— 默认情况下，您的容器实例必须在端口 8080 上侦听 HTTP 请求。有关配置服务的更多信息，请参阅。the section called “配置”。您无需实施对 HTTPS 安全流量的处理。应用程序运行程序需要传入 HTTPS 流量，并在将请求传递到容器实例之前终止 HTTPS。
- 无状态应用— App Runner 不保证在处理单个传入 Web 请求的持续时间之后的状态持久性。
- 存储— 应用程序 Runner 将容器实例中的文件系统实现为短暂存储。文件是瞬态的。例如，当您暂停和恢复 App Runner 服务时，这些策略不会持久。更一般地说，作为应用程序无状态性质的一部分，文件不能保证在处理单个请求之外持续存在。但是，存储的文件在其生命周期内会占用 App Runner 服务的部分存储分配。

  > **Note**
  >
  > 尽管临时存储文件可能不会在请求之间保留，但它们有时确实坚持。在某些情况下，这会非常有用。例如，在处理请求时，您可以缓存应用程序下载的文件（如果将来的请求可能需要）。这可能会加快未来的请求处理速度，但不能保证速度提高。您的代码不应该假定在先前请求中下载的文件仍然存在。
  >
  > 要使用高吞吐量、低延迟的内存中数据存储保证缓存，请使用Amazon ElastiCache。

- 环境变量— 默认情况下，应用程序运行程序使PORT环境变量。您可以使用端口信息配置变量值，并添加自定义环境变量和值。有关配置服务的更多信息，请参阅。the section called “配置”。
- 实例角色— 如果您的应用程序代码调用AWS服务，使用服务 API 或AWSSDK，使用创建实例角色 AWS Identity and Access Management(IAM). 然后，在创建应用程序 Runner 服务时将其附加到应用程序 Runner 服务。包含全部AWS您的代码在您的实例角色中需要的服务操作权限。有关更多信息，请参阅the section called “实例角色”。

代码开发指南

在为 App Runner Web 应用程序开发代码时，请考虑这些准则。

- 设计无状态代码— 将部署到 App Runner 服务的 Web 应用程序设计为无状态。您的代码应该假定在处理单个传入 Web 请求的持续时间之后，没有任何状态持续存在。
- 删除临时文件— 创建文件时，这些文件将存储在文件系统中，并占用服务的存储分配的一部分。为避免存储不足错误，请不要长时间保留临时文件。在做出文件缓存决策时，平衡存储大小与请求处理速度。

使用应用程序运行程序控制台

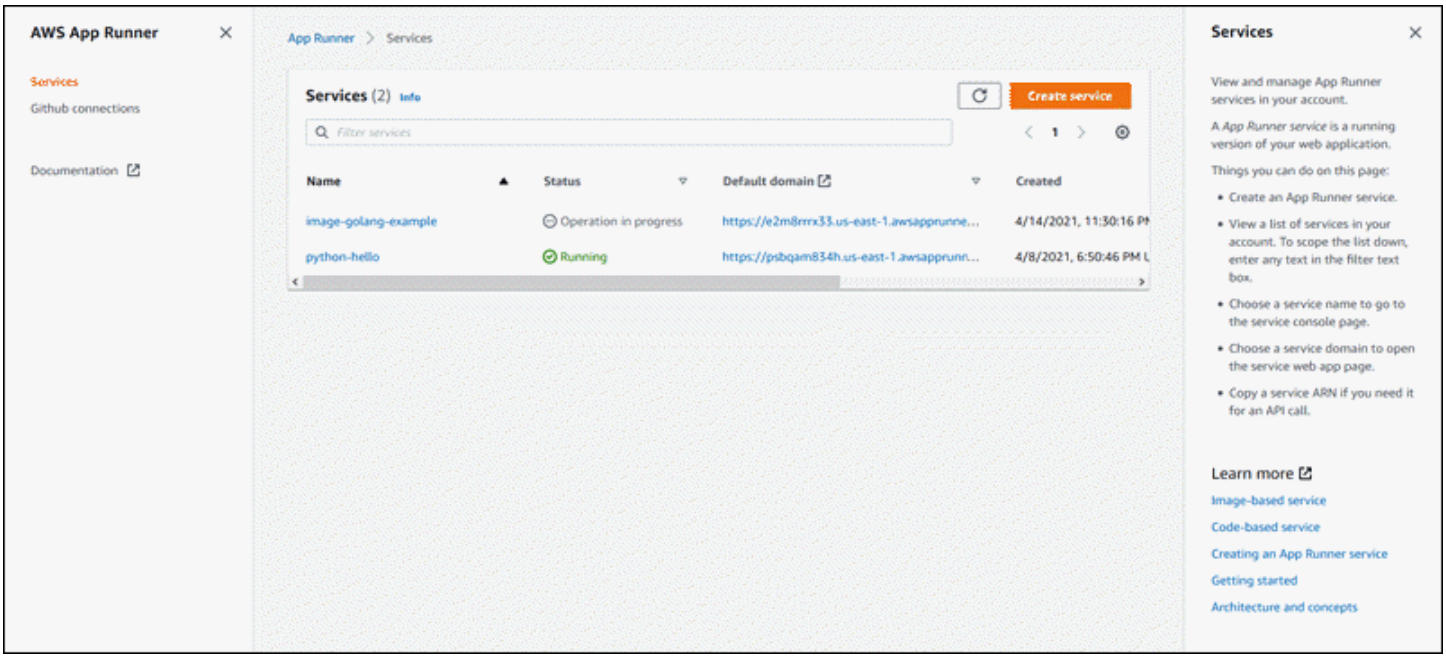
使用AWS App Runner控制台来创建、管理和监控您的 App Runner 服务和相关资源（如连接）。您可以查看现有服务、创建新服务和配置服务。您可以查看 App Runner 服务的状态以及查看日志、监视活动和跟踪指标。您还可以导航到服务的网站或源存储库。

以下各节介绍控制台的布局和功能，并指向相关信息。

总体控制台布局

应用程序运行者控制台有三个区域。从左到右：

- 导航窗格— 可折叠或展开的侧窗格。使用它可以选择要使用的顶级控制台页面。
- 内容窗格— 控制台页面的主要部分。使用它来查看信息并执行您的任务。
- 帮助窗格— 用于查看详细信息的侧面窗格。展开它以获取有关您所在页面的帮助。或者选择任何Info链接以获取上下文帮助。



“服务” 页面

这些区域有：服务页面会列出您帐户中的应用程序运行者服务。您可以使用筛选器文本框来确定列表的范围。

要访问Services页

1. 打开应用程序 Runner，并在区域列表中，选择您的AWS 区域。
2. 在导航窗格中，选择服务。

您可以在这里做的事情：

- 创建应用程序 Runner 服务。有关更多信息，请参阅the section called “Creation”。
- 选择一个服务名称以转到服务仪表板控制台页面。
- 选择服务域以打开服务 Web 应用程序页面。

服务仪表板页

您可以查看有关 App Runner 服务的信息，并从服务 dashbaord 页面对其进行管理。在页面顶部，您可以看到服务名称。

要访问服务仪表板，请导航到服务页面（请参阅上一节），然后选择您的 App Runner 服务。

这些区域有：服务概述部分提供了有关 App Runner 服务和您的应用程序的基本详细信息。您可以在这里做的事情：

- 查看服务详细信息，如状态、运行状况和 ARN。
- 导航到默认域— App Runner 为服务中运行的 Web 应用程序提供的域。这是`awsapprunner.com`域所拥有的应用程序运行者。
- 导航到部署到服务的源资料库。
- 启动源资料库部署到您的服务。
- 暂停、恢复和删除您的服务。

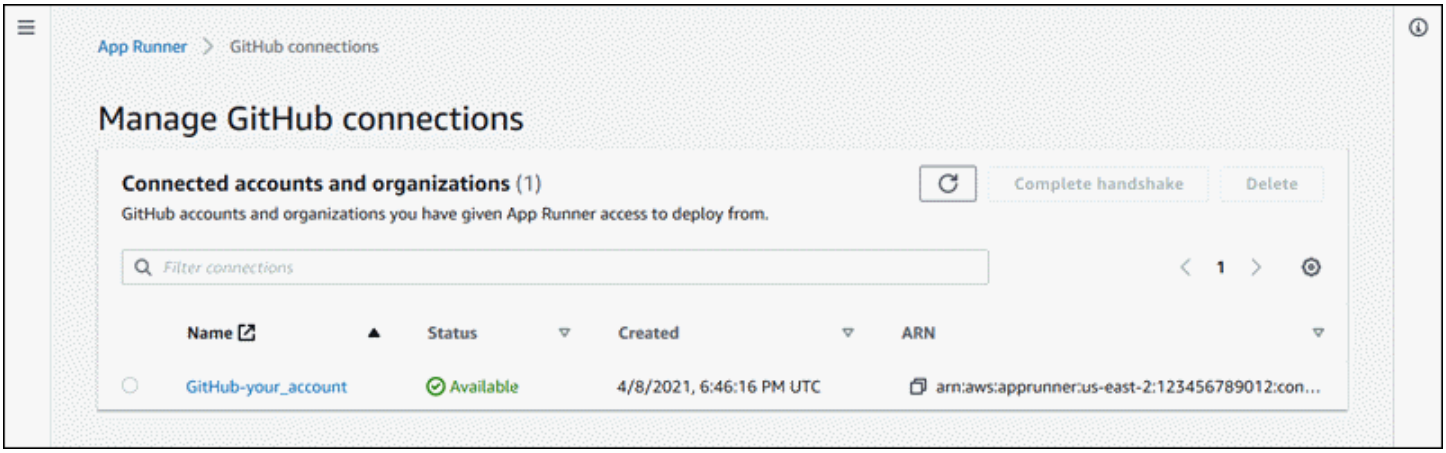
服务概述下面的选项卡适用于服务管理和监控。

GitHub 连接页

这些区域有：GitHub 连接页面列出了您帐户中与 GitHub 的应用程序运行程序连接。您可以使用筛选器文本框来确定列表的范围。有关连接的更多信息，请参阅the section called “连接”。

要访问GitHub 连接页

1. 打开应用程序 Runner，并在区域列表中，选择您的AWS 区域。
2. 在导航窗格中，选择GitHub 连接。



您可以在这里做的事情：

- 查看您的账户中 GitHub 连接的列表。要缩小列表范围，请在筛选器文本框中输入任何文本。
- 选择连接名称以转到相关 GitHub 帐户或组织。
- 选择一个连接以完成刚刚建立的连接的握手（作为创建服务的一部分），或删除连接。

管理您的应用运行程序服务生命周期

本章介绍如何在AWS App Runner服务。在本章中，您将了解如何创建、配置和删除服务，如何将新的应用程序版本部署到您的服务，以及如何管理连接。您还将了解如何通过暂停和恢复服务来控制 Web 服务的可用性。

主题

创建应用程序运行程序服务

AWS App Runner自动执行从容器映像或源代码存储库转变为可自动扩展的正在运行的 Web 服务的过程。您可以将 App Runner 指向源图像或代码，只指定少量所需的设置。App Runner 根据需要构建您的应用程序，配置计算资源，并部署您的应用程序以便在这些资源上运行。

当您创建服务时，应用程序 Runner 将创建服务资源的费用。在部分情况下，您可能需要提供连接资源的费用。如果您使用 App Runner 控制台，控制台将隐式创建连接资源。有关应用程序 Runner 资源类型的详细信息，请参阅the section called “App Runner 资源”。这些资源类型的配额与您的帐户在每个 AWS 区域。有关更多信息，请参阅the section called “App Runner 资源配额”。

根据源类型和提供程序，创建服务的过程有细微的差异。本主题介绍了创建这些不同源类型的完全独立的过程，以便您可以按照最符合您具体情况的任何一种来源类型进行操作。有关代码示例的基本启动过程，请参阅入门。

Prerequisites

在创建应用程序 Runner 服务之前，请确保完成以下操作：

- 完成中的设置步骤设置。

- 准备好您的应用程序源代码。您可以使用GitHub中的容器图像或Amazon EElastic Container Registry (Amazon ECR)创建应用程序运行程序服务。

创建服务

本节介绍了两种 App Runner 服务类型的创建过程：基于源代码和基于容器映像。

从 GitHub 代码库创建服务

下面几节说明了当您的源代码是GitHub。当您使用 GitHub 时，应用程序运行者必须连接到 GitHub 组织或帐户。因此，您需要帮助建立此连接。有关应用程序 Runner 连接的更多信息，请参阅。the section called “连接”。

在创建服务时，App Runner 将构建包含应用程序代码和依赖关系的 Docker 映像。然后，它启动运行此映像的容器实例的服务。

主题

- 使用应用程序运行程序控制台从代码创建服务
- 使用应用程序运行程序 API 或AWS CLI

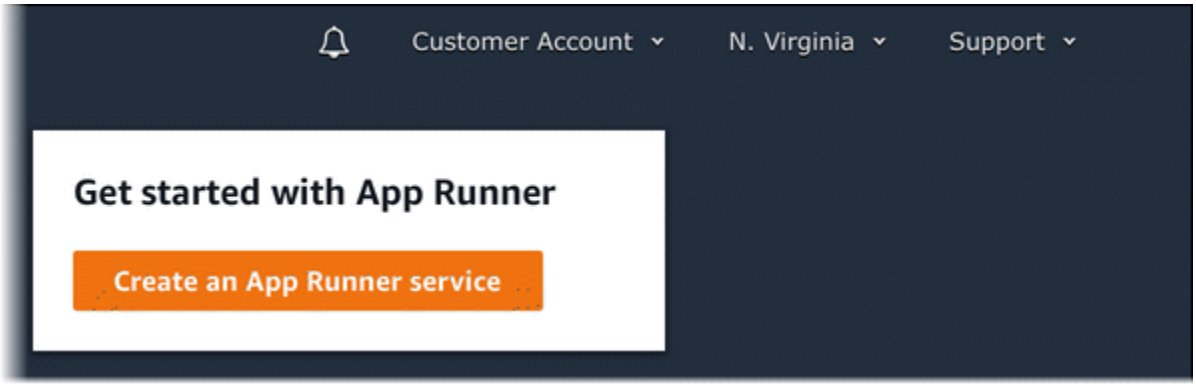
使用应用程序运行程序控制台从代码创建服务

使用控制台创建应用程序 Runner 服务

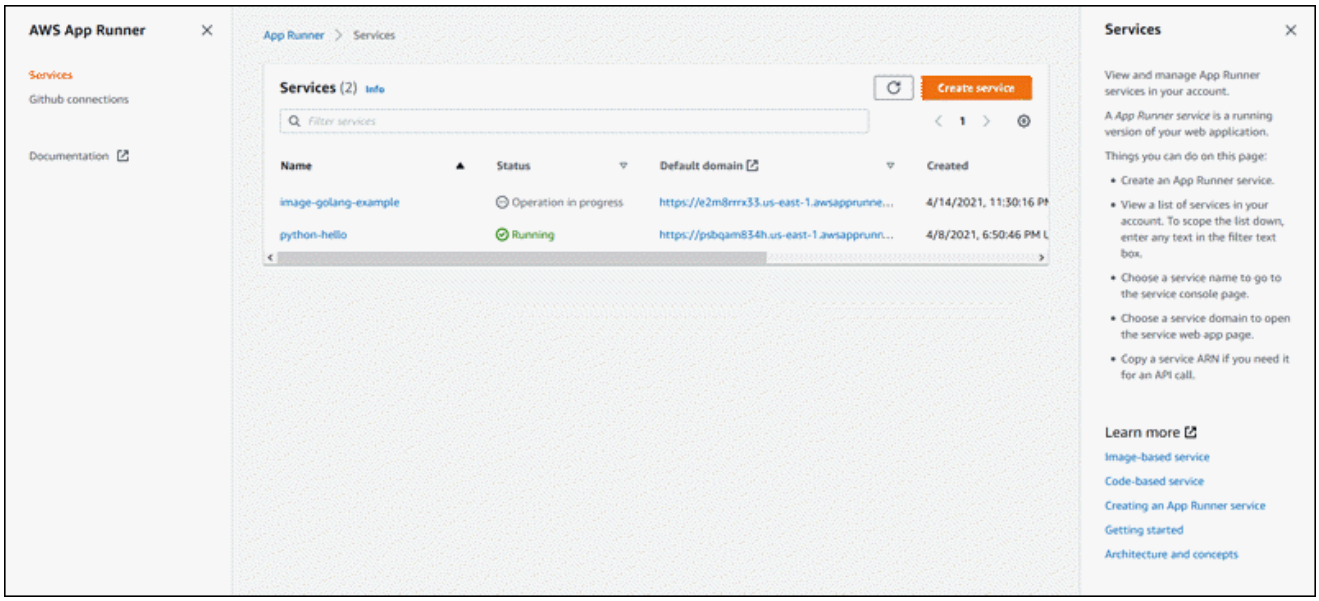
1. 配置您的源代码。

   a. 打开应用程序运行器控制台，并在区域列表中，选择您的AWS 区域。

   b. 如果AWS 账户还没有任何 App Runner 服务，则会显示控制台主页。选择创建应用程序 Runner 服务。



如果AWS 账户具有现有服务，服务页面，其中包含您的服务列表。选择 Create service。



c. 在存储库的源和部署” 页面中的源部分，用于存储库类型中，选择源代码存储库。

d. 适用于Connect to GitHub，选择您之前使用过的 GitHub 帐户或组织，或者选择Add New。然后，完成提供 GitHub 凭据并选择要连接到的 GitHub 帐户或组织的过程。

e. 适用于Repository下，选择包含应用程序代码的存储库。

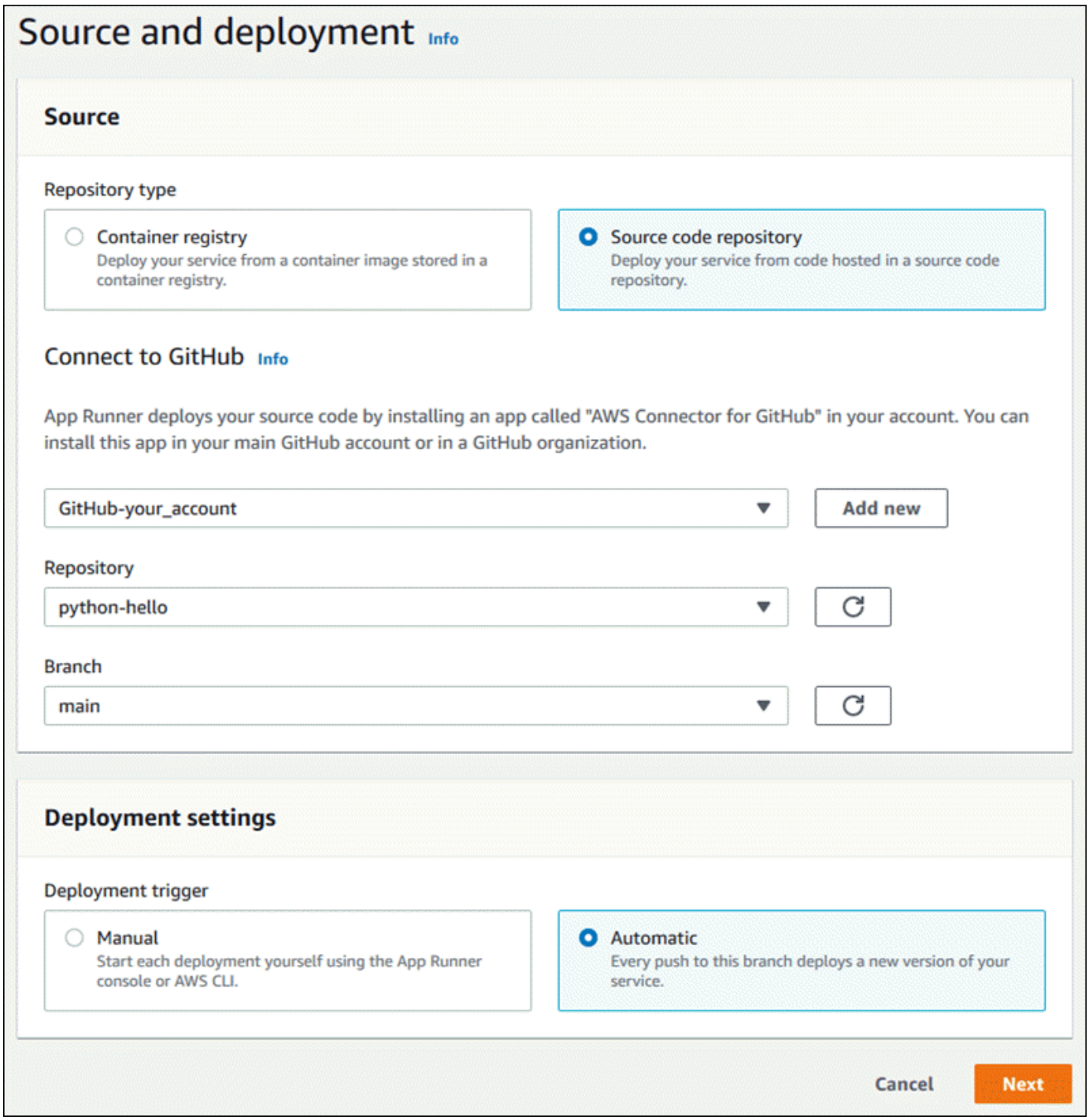
f. 适用于分支中，选择要部署的分支。

2. 配置您的部署。

a. 在部署设置部分，选择。手动或者自动。

有关部署方法的更多信息，请参阅。the section called “部署方法”。

b. 选择 Next。



3. 配置应用程序生成。

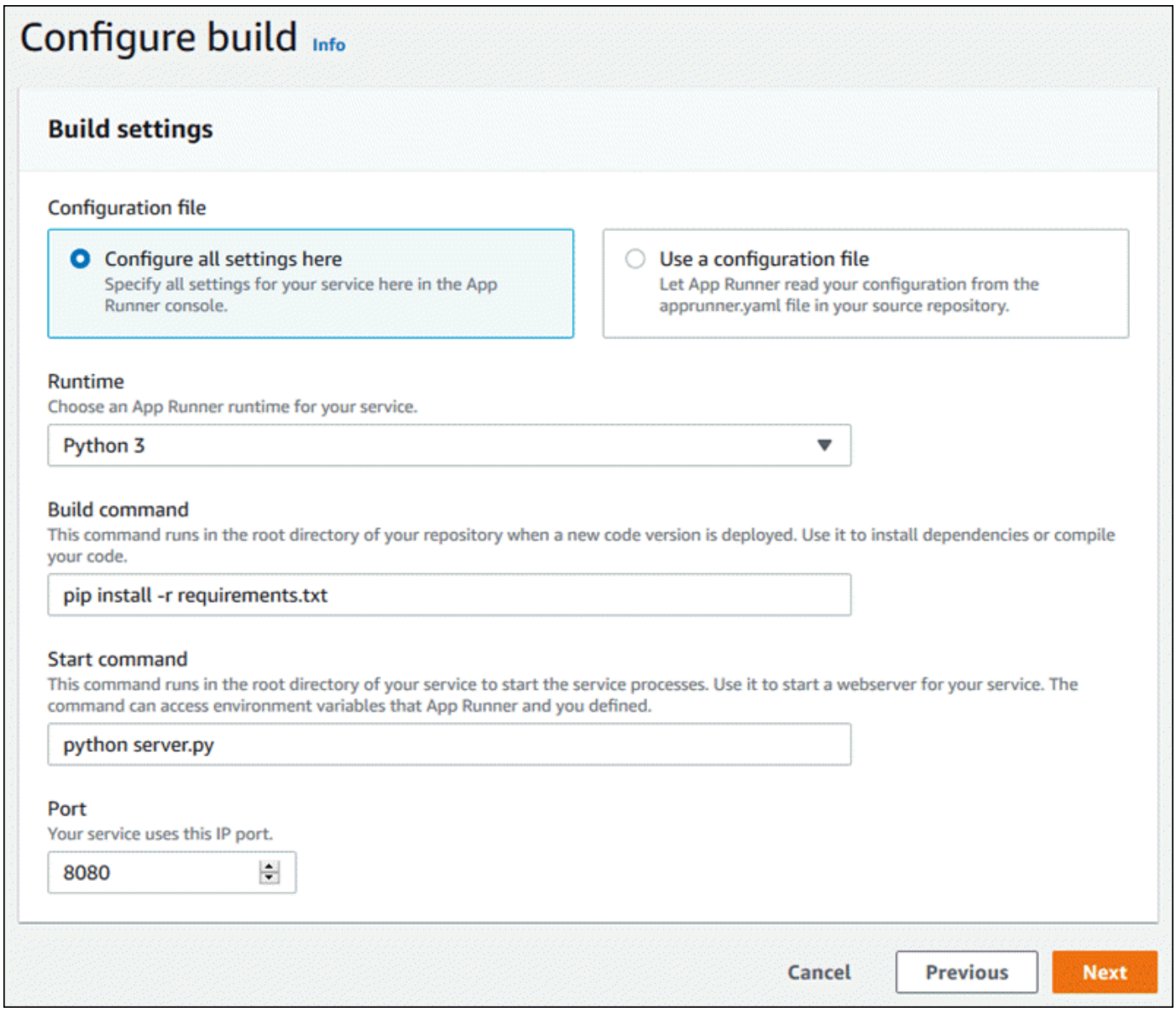
   a. 在存储库的配置生成页面, 用于配置文件中，选择在此处配置所有设置如果您的存储库不包含 App Runner 配置文件，或者使用配置文件如果确实如此。

> **Note**
>
> App Runner 配置文件是将构建配置作为应用程序源的一部分进行维护的一种方法。当您提供一个值时，App Runner 会从文件中读取一些值，并且不允许您在控制台中设置它们。

b. 提供以下构建设置：

- 运行时— 为您的应用程序选择特定的托管运行时。
- 生成命令— 输入从应用程序的源代码构建应用程序的命令。这可能是一个特定于语言的工具或代码随附的脚本。
- 启动命令— 输入启动 Web 服务的命令。
- 端口— 输入 Web 服务侦听的 IP 端口。

c. 选择 Next。



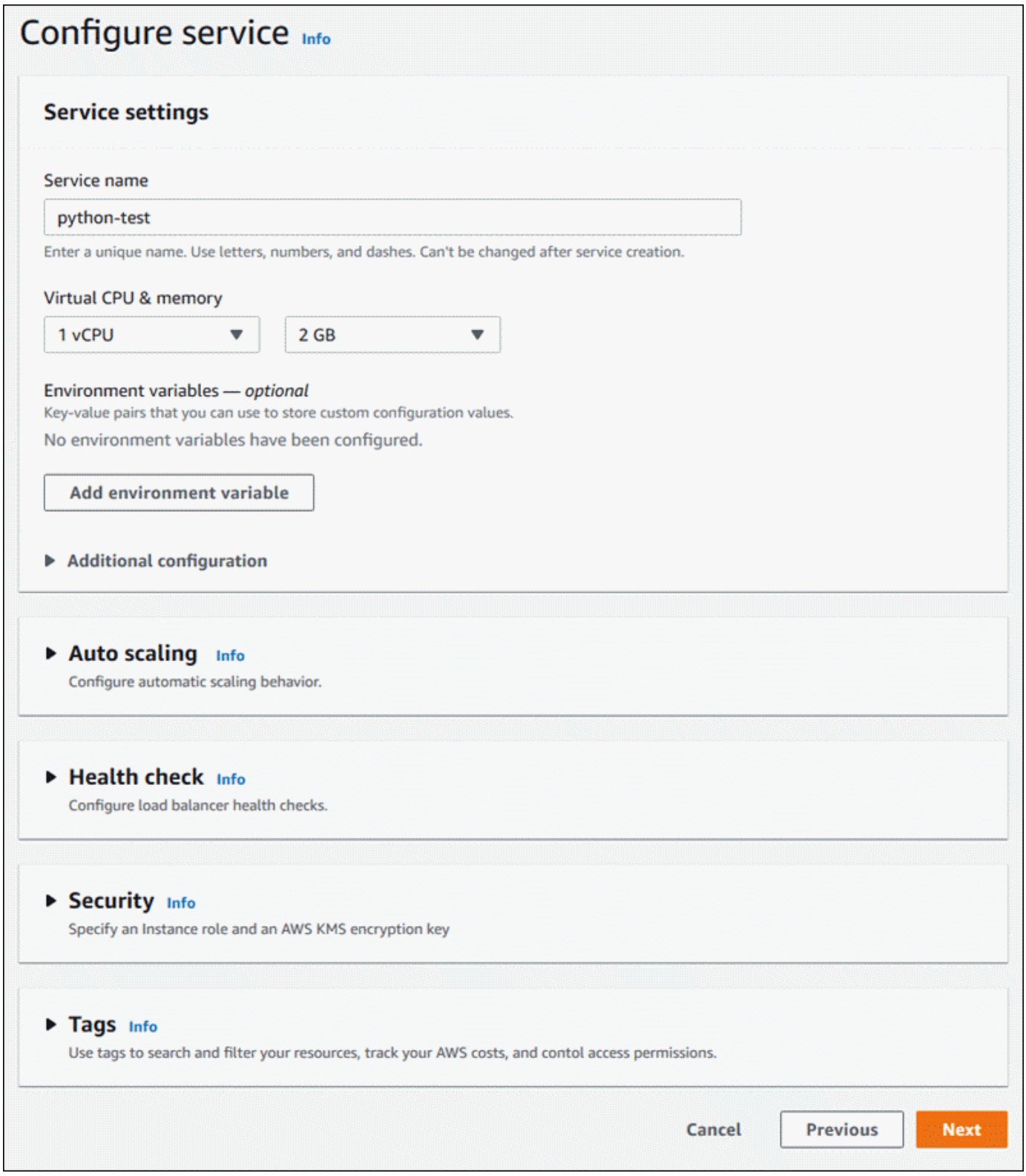
4. 配置您的服务。

   a. 在存储库的配置服务” 页面中的服务设置部分中，输入服务名称。

   > **Note**
   >
   > 所有其他服务设置都是可选的，或者具有控制台提供的默认值。

   b. （可选）更改或添加其他设置以满足应用程序要求。

   c. 选择 Next。



5. 在存储库的审核和创建页面上，验证您输入的所有详细信息，然后选择创建和部署。

结果：如果服务创建成功，控制台应显示服务仪表板，并显示服务概述的费用。

6. 验证您的服务是否正在运行。

   a. 在服务仪表板页面上，等待服务状态是正在运行。

   b. 选择默认域值-它是指向您服务网站的 URL。

   c. 使用您的网站并验证其是否正常运行。

使用应用程序运行程序 API 或AWS CLI

使用应用程序运行程序 API 或AWS CLI，调用`CreateService`API 操作。有关更多信息以及示例，请参阅。CreateService。如果这是您第一次使用特定 GitHub 组织或帐户创建服务，请首先调用CreateConnection。这建立了应用程序运行者和 GitHub 组织或帐户之间的连接。有关应用程序 Runner 连接的更多信息，请参阅。the section called “连接”。

如果调用返回一个成功响应，则启动服务创建服务对象显示`"Status": "CREATING"`。

有关示例调用，请参阅。创建源代码存储库服务中的AWS App RunnerAPI 参考

使用 Amazon ECR 映像创建服务

下面几节说明了当您的源代码是存储在Amazon ECR。Amazon ECR 是AWS服务。因此，要基于 Amazon ECR 映像创建服务，您需要为 App Runner 提供包含必要 Amazon ECR 操作权限的访问角色。

> **Note**
>
> 如果您的图片存储在 Amazon ECR PUblic（图片可公开发布）中，则不需要访问角色。

在服务创建过程中，App Runner 会启动运行您提供的映像的容器实例的服务。在这种情况下没有构建阶段。

主题

- 使用应用程序运行程序控制台从映像创建服务
- 使用应用程序运行程序 API 或AWS CLI

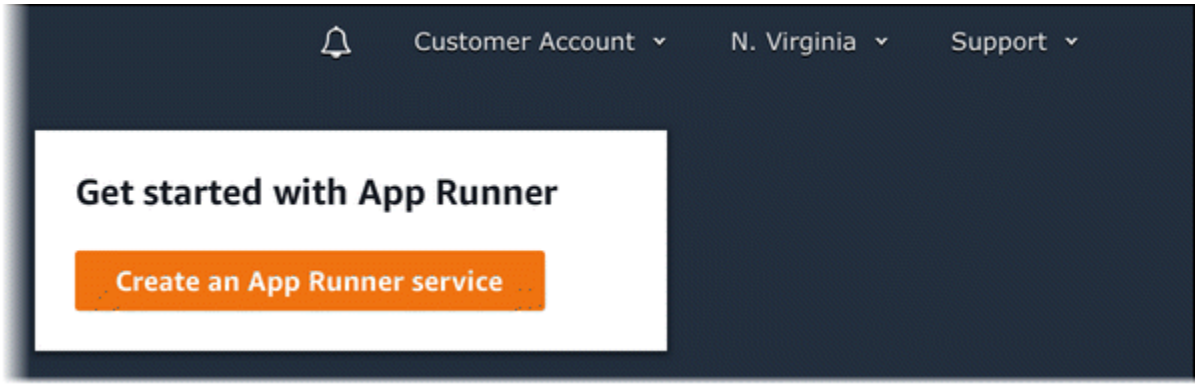
使用应用程序运行程序控制台从映像创建服务

使用控制台创建应用程序 Runner 服务

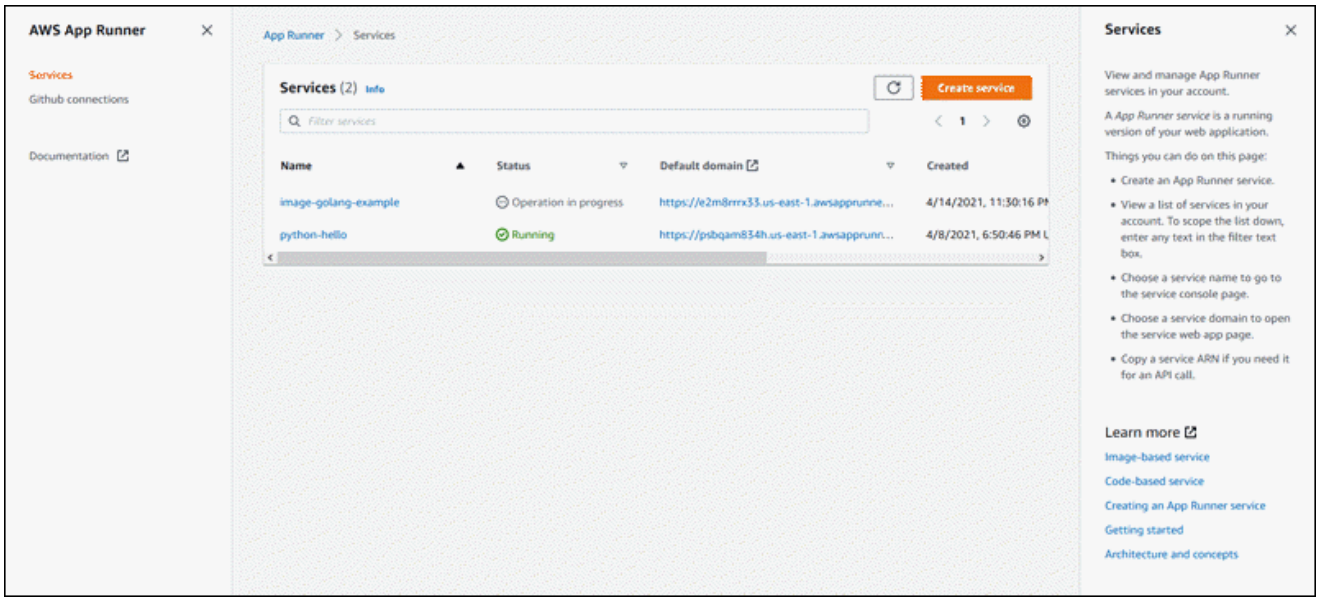
1. 配置您的源代码。

   a. 打开应用程序运行器控制台，并在区域列表中，选择您的AWS 区域。

   b. 如果AWS 账户还没有任何 App Runner 服务，则会显示控制台主页。选择创建应用程序 Runner 服务。



   如果AWS 账户具有现有服务，服务页面，其中包含您的服务列表。选择 Create service。



   c. 在存储库的源和部署” 页面中的源部分，用于存储库类型中，选择容器注册表。

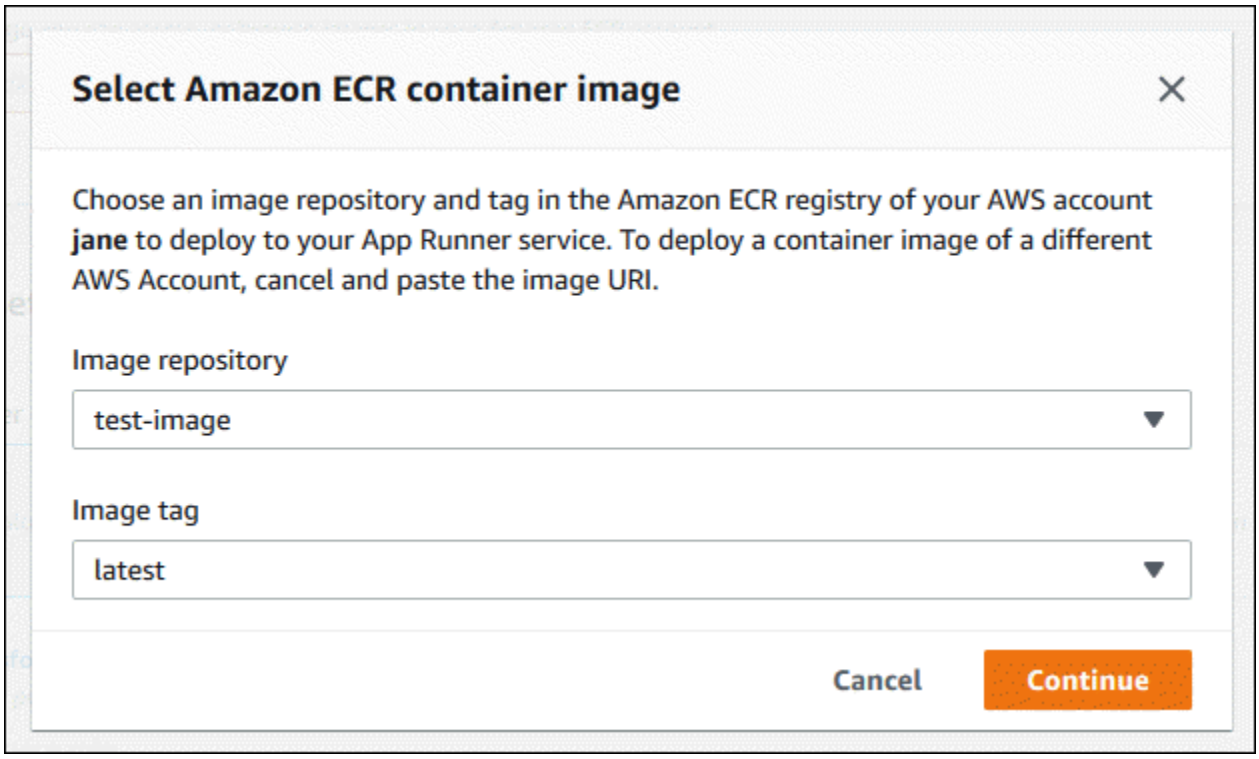
   d. 适用于提供程序下，选择存储图像的提供程序：

   - Amazon ECR— 存储在亚马逊 ECR 中的私有映像AWS 账户。
   - Amazon ECR 公开— 存储在亚马逊 ECR 公共中的可公开读取的图像。

e. 适用于容器映像 URI中，选择浏览。

f. 在选择亚马逊 ECR 容器图片对话框中，对于映像存储库中，选择包含映像的存储库。

g. 适用于映像标签下，选择要部署的特定映像标签，例如latest，然后选择。Continue。



2. 配置您的部署。

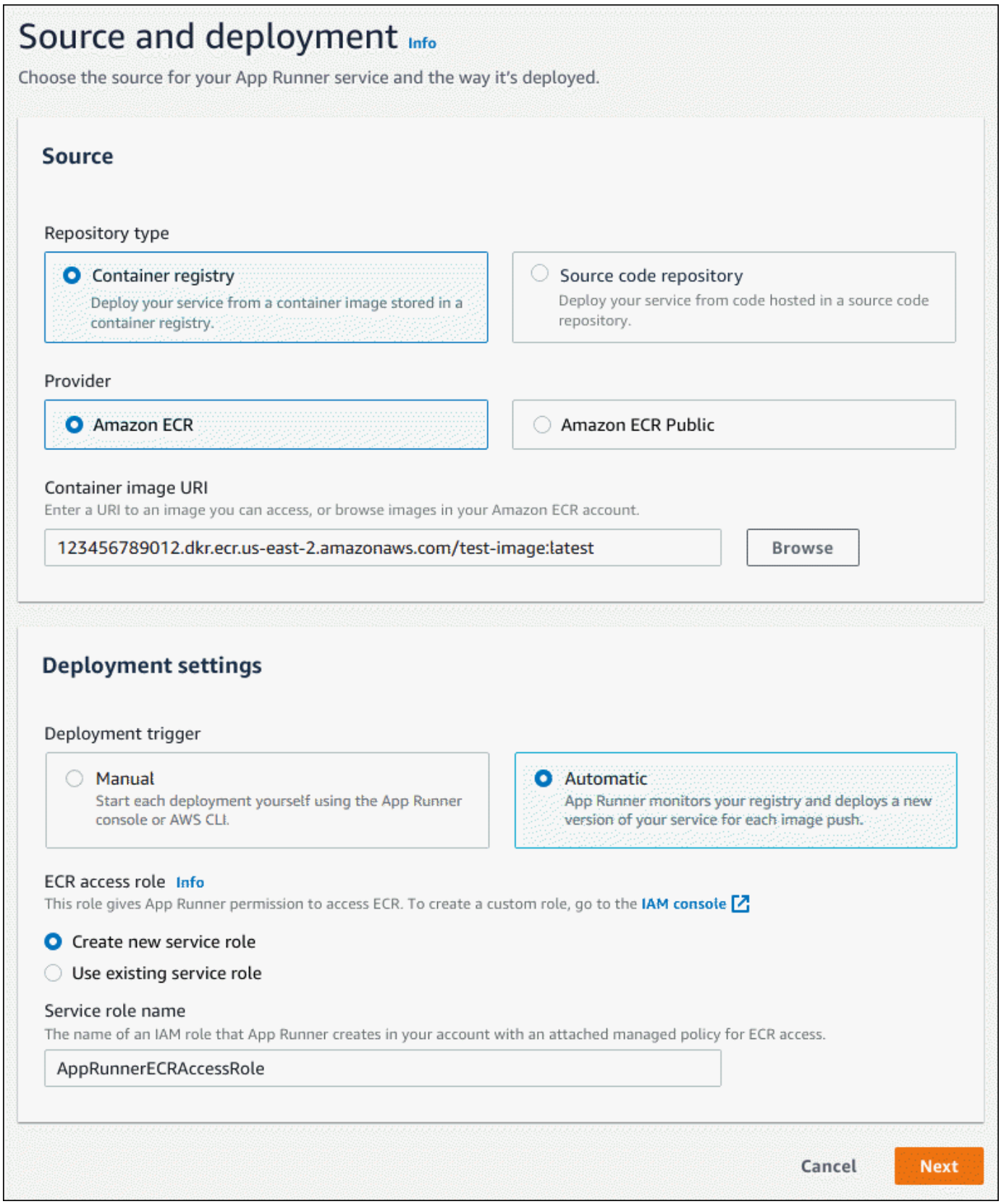
a. 在部署设置部分，选择。手动或者自动。

有关部署方法的更多信息，请参阅。the section called “部署方法”。

> **Note**
>
> 应用程序运行程序不支持亚马逊 ECR 公共映像的自动部署。

b. [Amazon ECR提供商] 对于ECR 访问角色下，选择您的帐户中的现有服务角色或选择创建新角色。如果您使用手动部署，您还可以选择在部署时使用 IAM 用户角色。

c. 选择 Next。


Source and deployment Info
Choose the source for your App Runner service and the way it's deployed.
Source
Repository type
Container registry
Deploy your service from a container image stored in a container registry.
Source code repository
Deploy your service from code hosted in a source code repository.
Provider
Amazon ECR
Amazon ECR Public
Container image URI
Enter a URI to an image you can access, or browse images in your Amazon ECR account.
123456789012.dkr.ecr.us-east-2.amazonaws.com/test-image:latest
Browse
Deployment settings
Deployment trigger
Manual
Start each deployment yourself using the App Runner console or AWS CLI.
Automatic
App Runner monitors your registry and deploys a new version of your service for each image push.
ECR access role Info
This role gives App Runner permission to access ECR. To create a custom role, go to the IAM console
Create new service role
Use existing service role
Service role name
The name of an IAM role that App Runner creates in your account with an attached managed policy for ECR access.
AppRunnerECRAccessRole
Cancel
Next

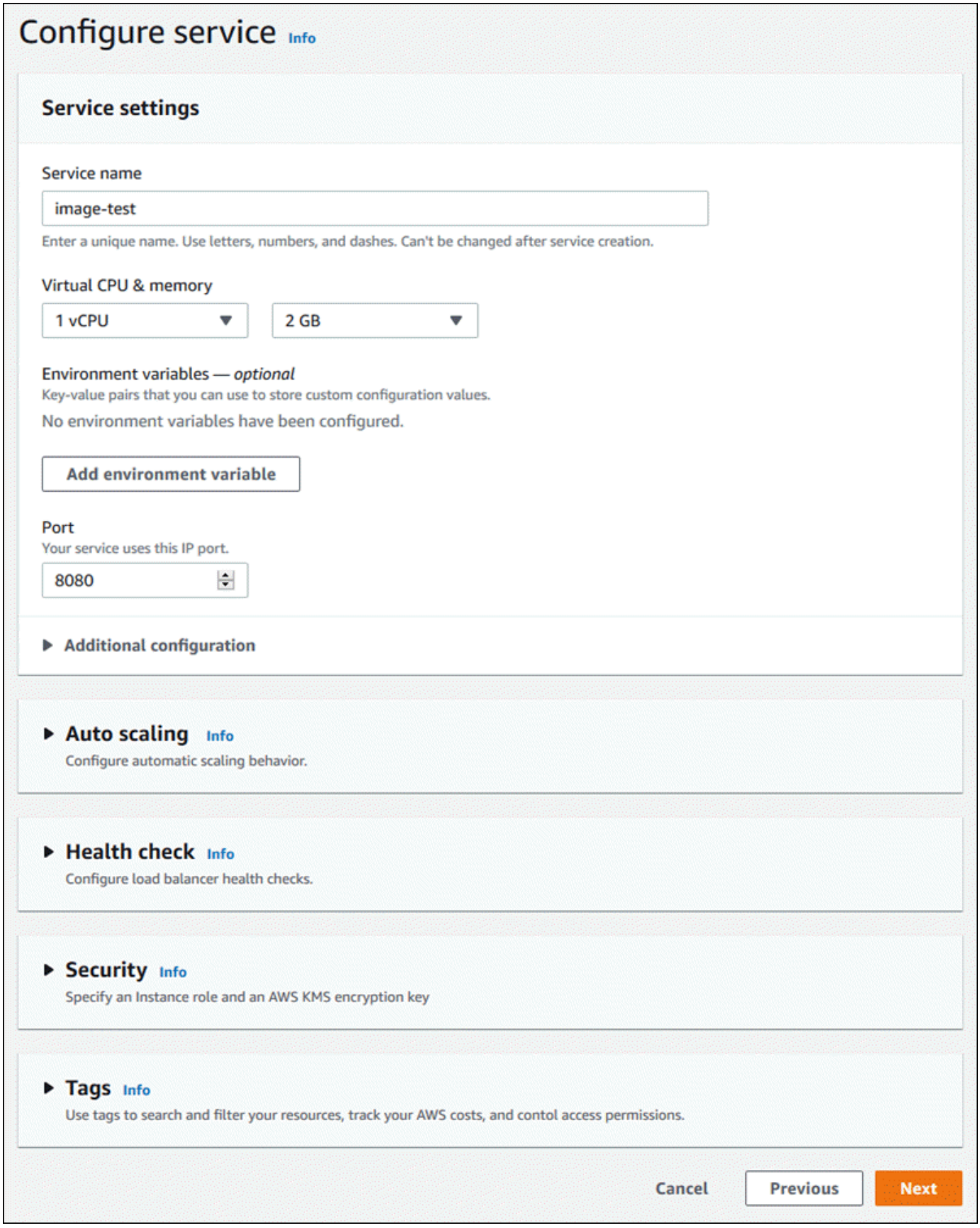
3. 配置您的服务。

   a. 在存储库的配置服务” 页面中的服务设置部分中，输入服务名称和服务网站侦听的 IP 端口。

   > **Note**
   >
   > 所有其他服务设置都是可选的，或者具有控制台提供的默认值。

   b. （可选）更改或添加其他设置以满足您的应用程序的需求。

   c. 选择 Next。


Configure service Info
Service settings
Service name
image-test
Enter a unique name. Use letters, numbers, and dashes. Can't be changed after service creation.
Virtual CPU & memory
1 vCPU
2 GB
Environment variables — optional
Key-value pairs that you can use to store custom configuration values.
No environment variables have been configured.
Add environment variable
Port
Your service uses this IP port.
8080
Additional configuration
Auto scaling Info
Configure automatic scaling behavior.
Health check Info
Configure load balancer health checks.
Security Info
Specify an Instance role and an AWS KMS encryption key
Tags Info
Use tags to search and filter your resources, track your AWS costs, and contol access permissions.
Cancel
Previous
Next

4. 在存储库的审核和创建页面上，验证您输入的所有详细信息，然后选择创建和部署。

   结果：如果服务创建成功，控制台应显示服务仪表板，并显示服务概述的费用。

5. 验证您的服务是否正在运行。

   a. 在服务仪表板页面上，等待服务状态是正在运行。

   b. 选择默认域值-它是指向您服务网站的 URL。

   c. 使用您的网站并验证其是否正常运行。

使用应用程序运行程序 API 或AWS CLI

使用应用程序运行程序 API 或AWS CLI，调用CreateServiceAPI 操作。

如果调用返回一个成功响应，则启动服务创建服务对象显示`"Status": "CREATING"`。

有关示例调用，请参阅。创建源映像存储库服务中的AWS App RunnerAPI 参考

创建服务失败时

如果尝试创建 App Runner 服务失败，该服务将显示`CREATE_FAILED`(创建失败）。

创建服务的尝试可能会失败，原因是应用程序代码、构建过程或配置中存在问题，因为您达到了资源配额，或者由于底层AWS服务需要使用的服务。要对故障进行故障排除，建议执行以下操作。首先，阅读服务事件和日志，找出故障导致的原因。接下来，对您的代码或配置进行任何必要的更改。最后，如果达到服务配额，请删除一个或多个服务。然后，完成所有这些步骤后，尝试再次创建该服务。

> **Important**
>
> 失败的服务不可用。除了初始创建尝试之外，您不会为此产生任何额外的费用。但是，App Runner 不会自动删除失败的服务，它仍会计入您的服务配额。完成故障分析后，请确保删除失败的服务。

将新应用程序版本部署到应用程序 Runner

当您 时创建服务在AWS App Runner中，您可以配置应用程序源-容器映像或源资料库。App Runner 配置资源以运行您的服务，并将您的应用程序部署到这些资源。

本主题介绍了在新版本可用时将应用程序源重新部署到 App Runner 服务的方法。这可以是映像存储库中的新映像版本，也可以是代码存储库中的新提交。App Runner 提供了两种部署到服务的方法：自动和手动方式。

部署方法

App Runner 为您提供了以下方法来控制应用程序部署的启动方式。

自动部署

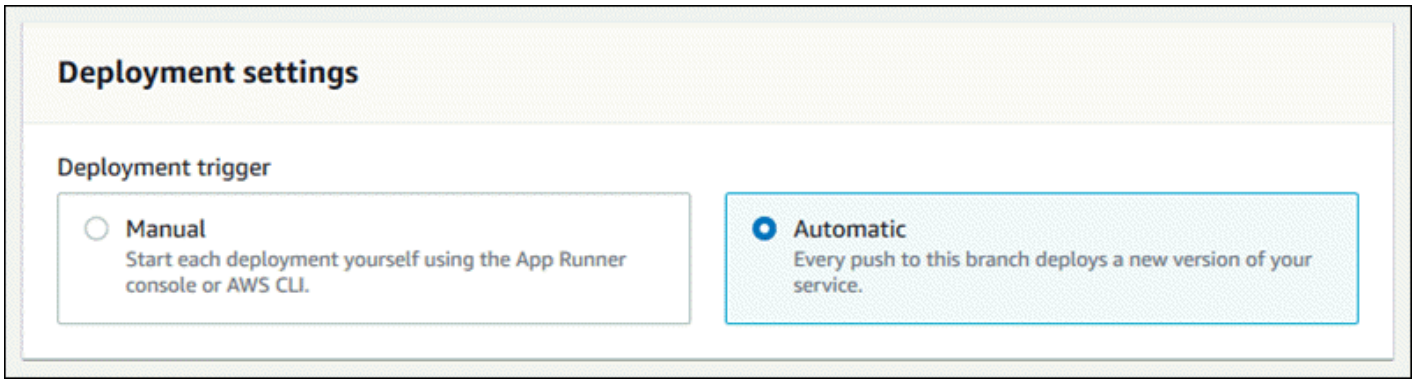
如果您希望服务的持续集成和部署 (CICD) 行为，请使用自动部署。应用程序运行程序监控您的图像或代码存储库。无论何时将新映像版本推送到映像存储库，或者新提交到代码存储库时，App Runner 都会自动将其部署到您的服务中，而无需进一步操作。

手动部署

如果要显式启动服务的每个部署，请使用手动部署。如果为服务配置的存储库具有要部署的新版本，则可以启动部署。有关更多信息，请参阅the section called “手动部署”。

您可以通过以下方式配置服务的部署方法：

- 控制台— 对于要创建的新服务或现有服务，请在部署设置的 部分源和部署配置页面上，选择手动或者自动。



- API 或AWS CLI— 在调用CreateService或者UpdateService操作中，设置`AutoDeploymentsEnabled`成员SourceConfiguration参数设置为`False`用于手动部署或`True`进行自动部署。

手动部署

使用手动部署，您需要明确地启动服务的每个部署。当您准备好部署应用程序映像或代码的新版本时，您可以参考以下部分以了解如何使用控制台和 API 执行部署。

使用应用程序运行程序控制台部署应用程序版本

使用应用程序 Runner 控制台进行部署

1. 打开应用程序 Runner 控制台，并在区域列表中，选择您的AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板服务概览。
3. 选择 Deploy。

   结果：开始部署新版本。在服务仪表板页面上，状态对 的更改正在进行操作。
4. 请等待部署结束。在服务仪表板页面上，状态应该更改回正在运行。
5. 要验证部署是否成功，请在服务仪表板页面上选择默认域值-它是指向您服务网站的 URL。检查 Web 应用程序或与其交互，并验证您的版本更改。

使用应用程序运行程序 API 或AWS CLI

要使用应用程序运行程序 API 或AWS CLI，调用StartDeploymentAPI 操作。唯一要传递的参数是您的服务 ARN。在创建服务时，您已经配置了应用程序源位置，App Runner 可以找到新版本。如果调用返回成功响应，则部署将启动。

配置应用程序 Runner 服务

当您 时创建AWS App Runner服务中，您可以设置各种配置值。创建服务后，可以更改其中某些配置设置。其他设置只能在创建服务时应用，此后不能更改。本主题讨论使用应用程序运行程序 API、应用程序运行程序控制台和应用程序运行程序配置文件进行服务的配置。

使用应用程序运行程序 API 或AWS CLI

API 定义了服务创建后可以更改哪些设置。下面的列表讨论了相关操作、类型和限制。

- UpdateService操作 — 可在创建后调用以更新某些配置设置。

- 可以更新— 您可以更新SourceConfiguration、InstanceConfiguration, 和HealthCheckConfiguration参数。然而，在SourceConfiguration，您不能将源代码类型从代码切换到图像或其他方式。您必须提供与创建服务时提供的相同的存储库参数。它要么CodeRepository或者ImageRepository。

  您还可以更新AutoScalingConfigurationArn，即与服务关联的自动扩展配置资源的 ARN。

- 无法更新— 您不能更改ServiceName和EncryptionConfiguration参数，这些参数可在CreateServiceaction. 创建之后无法更改。这些区域有：UpdateService操作不包含这些参数。

- API 与文件— 您可以设置ConfigurationSource参数代码配置类型（用于源代码存储库作为SourceConfiguration) 到Repository。在这种情况下，应用程序运行程序会忽略CodeConfigurationValues，并从配置文件在您的存储库中。如果您设置ConfigurationSource到API，App Runner 从 API 调用中获取所有配置设置，并忽略配置文件，即使存在配置文件也是如此。

- TagResource操作 — 可在创建服务后调用以向服务添加标签或更新现有标签的值。
- UntagResource操作 — 可以在创建服务以从服务中删除标签后调用。

使用应用程序运行程序控制台配置您的服务

控制台使用应用程序运行程序 API 应用配置更新。API 强加的更新规则（如上一节中所定义）决定了您可以使用控制台配置的内容。某些在服务创建过程中可用的设置不可用于稍后修改。此外，如果您决定使用配置文件，其他设置将隐藏在控制台中，并且 App Runner 从文件中读取它们。

配置服务

1. 打开应用程序 Runner 控制台，并在区域列表中，选择您的AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板服务概述。
3. 在服务控制面板页面上，选择配置选项卡。

   结果：控制台分为以下几个部分显示服务的当前配置设置：源和部署、配置生成, 和配置服务。
4. 要更新任何类别中的设置，请选择编辑。
5. 在配置编辑页面上，进行所需更改，然后选择保存更改。。

使用应用程序运行器配置文件配置您的服务

创建或更新 App Runner 服务时，可以指示 App Runner 从作为源存储库一部分提供的配置文件中读取某些配置设置。通过执行此操作，您可以在源代码管理下管理与源代码相关的设置以及代码本身。配置文件还提供了某些无法使用控制台或 API 设置的高级设置。有关更多信息，请参阅应用程序运行程序配置文件。

管理应用程序运行程序连接

当您 时创建服务在AWS App Runner中，您可以配置应用程序源-与提供程序一起存储的容器映像或源资料库。如果存储在第三方提供商的存储库是私有的（不可公开读取），则 App Runner 必须与提供商建立经过身份验证和授权的连接。然后，App Runner 可以读取您的存储库并将其部署到您的服务。当您创建访问存储在AWS 账户或公共代码位置。

应用程序运行程序将连接信息保存在名为连接。当您创建需要第三方连接信息的服务时，App Runner 需要连接资源。以下是关于连接的一些重要信息：

- 提供程序— 应用程序运行程序当前需要连接资源GitHub。
- 共享— 您可以使用连接资源创建多个使用同一存储库提供程序帐户的 App Runner 服务。
- 资源管理— 在应用程序运行程序中，您可以创建和删除连接。但是，您无法修改现有连接。
- 资源配额— 连接资源有一个设置的配额，该配额与AWS 账户在每个AWS 区域。如果达到此配额，则可能需要先删除连接，然后才能连接到新的提供商帐户。您可以使用应用程序 Runner 来删除连接控制台或者API。有关更多信息，请参阅the section called “App Runner 资源配额”。

使用应用程序运行程序控制台管理连接

当您使用应用程序运行程序控制台创建服务中，您将提供连接详细信息。您不必显式创建连接资源。在控制台中，您可以选择连接到之前连接的 GitHub 帐户，或连接到新帐户。如有必要，App Runner 会为您创建一个连接资源。对于新连接，某些提供程序（例如 GitHub）要求您在使用连接之前完成身份验证握手。控制台将指导您完成此过程。

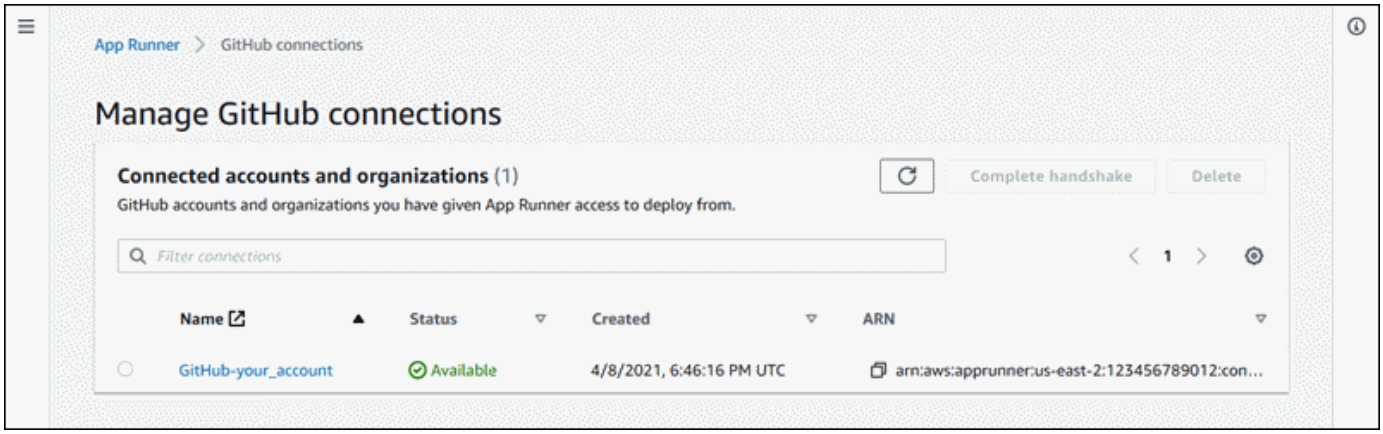
控制台还有一个用于管理现有连接的页面。如果在创建服务时没有执行此操作，则可以完成连接的身份验证握手。您还可以删除不再使用的连接。以下步骤演示如何管理 GitHub 连接。

管理您帐户中的 GitHub 连接

1. 打开应用程序 Runner 控制台，并在区域列表中，选择您的AWS 区域。

2. 在导航窗格中，选择GitHub 连接。

   控制台随后将显示您账户中 GitHub 连接的列表。



3. 现在，您可以对列表中的任何连接执行以下某种操作：

   - 打开 GitHub 帐户或组织— 选择连接的名称。
   - 完成身份验证握手— 选择连接，然后选择完成握手。控制台将引导您完成身份验证握手过程。
   - 删除连接— 选择连接，然后选择Delete。按照删除提示符上的说明进行操作。

使用应用程序运行程序 API 或AWS CLI

您可以通过以下 App Runner API 操作来管理您的连接。

- CreateConnection— 创建到存储库提供程序帐户的连接。创建连接后，您必须使用 App Runner 控制台手动完成身份验证握手。上一节介绍了此过程。
- 列出连接— 返回与AWS 账户。
- DeleteConnection— 删除连接。如果您达到AWS 账户。

管理应用程序运行程序自动扩展

AWS App Runner自动向上或向下扩展您的 App Runner 应用程序的计算资源（实例）。自动扩展可在传入流量高时提供充分的请求处理，并在流量减慢时降低成本。您可以配置几个参数来调整服务的自动缩放行为。

应用程序运行程序在名为自动缩放配置。您可以在创建或更新服务时提供自动扩展配置资源。当您创建新的 App Runner 服务时，App Runner 控制台会为您创建一个。提供自动扩展配置是可选的。如果您未提供密集策略，则 App Runner 会提供默认自动扩展配置，其中包含建议值。

自动扩展配置具有名称和一个数字修订。一个配置的多个修订版具有相同的名称和不同的修订版号。对于不同的自动扩展方案，如高可用性或低成本，您可以使用不同的配置名称。对于每个名称，您可以添加多个修订版来微调特定方案的设置。

以下是有关自动扩展配置的一些重要信息：

- 设置— 您可以配置以下内容：
  - 最大并发数— 实例处理的最大并发请求数。当并发请求数超过此配额时，App Runner 会扩展服务。
  - 最大大小— 您的服务扩展到的最大实例数。至多这些实例正在为您的服务提供流量。
  - 最小尺寸— App Runner 为您的服务配置的最小实例数。该服务始终至少具有此数量的预配置实例。他们中的一些积极服务交通。其余部分（预配置和非活动实例）可作为经济高效的计算容量预留，随时可以快速激活。您需要为所有预配置实例的内存使用量付费。您仅为活动子集的 CPU 使用率付费。

    App Runner 在部署期间临时将预配置实例的数量增加一倍，以保持旧代码和新代码的相同容量。
- 修订— 您使用名称创建的第一个配置获取修订号 1。具有相同名称的后续配置将获得连续的修订版号（从 2 开始）。您可以将您的 App Runner 服务与特定的自动扩展配置修订版或配置的最新版本相关联。
- 共享— 您可以跨多个 App Runner 服务共享单个自动扩展配置资源。如果它们具有类似的扩展要求，则这很有用。特别是，您可以通过指定配置名称但不指定修订来配置多个服务以使用最新版本的配置。通过执行此操作，您以此方式配置的任何服务在更新服务时都会收到 Auto Scaling 配置更新。有关配置更改的详细信息，请参阅the section called “配置”。
- 资源管理— 您可以使用 App Runner 创建和删除自动扩展配置。您无法直接更新配置。相反，您可以创建对现有配置名称的新修订版，以有效地更新配置。

  > **Note**
  >
  > 目前，您只能在 App Runner 控制台中使用单个修订版创建配置。要创建更多修订版并删除配置，请使用应用程序运行程序API。
- 资源配额— 您可以在每个AWS 区域。如果达到这些配额，则必须删除配置名称或至少一些修订版，然后才能创建更多配置名称。使用应用程序运行程序API删除它们。有关更多信息，请参阅the section called “App Runner 资源配额”。

使用应用程序运行程序控制台管理自动扩展

当您 时创建服务，您可以使用默认的自动扩展配置或自定义配置。要使用自定义配置，请选择现有配置或提供新名称和设置。如果是新配置，App Runner 会为您创建一个新的自动扩展配置资源，然后将其与您的新服务相关联。

使用应用程序运行程序 API 或AWS CLI

您可以使用下列应用程序 Runner API 操作来管理您的自动扩展配置。

- 创建自动缩放配置— 创建新的自动扩展配置或对现有配置的修订。
- 直接扩展配置— 返回与您的AWS 账户，其中包含摘要信息。
- 描述缩放配置— 返回自动扩展配置的完整描述。
- 删除自动缩放配置— 删除自动扩展配置。您可以删除特定修订版或最新的活动版本。如果您达到自动扩展配置配额，则可能需要删除不必要的自动扩展配置AWS 账户。

管理应用程序运行程序服务的自定义域名

在创建AWS App Runner服务时，应用程序运行程序会为其分配域名。这是awsapprunner.com属于应用程序运行者的域。它可用于访问服务中运行的 Web 应用程序。

如果您拥有一个域名，则可将它与您的 App Runner 服务相关联。App Runner 验证您的新域名后，除了 App Runner 域之外，还可以使用它来访问您的应用程序。您最多可以关联五个自定义域。

> **Note**
>
> 您可以选择包含www域的子域名。但是，目前仅在 API 中支持。App Runner 控制台不支持它。

当您将自定义域与服务关联时，App Runner 会为您提供一组证书验证记录。将它们添加到您的域名系统 (DNS)，以便应用程序运行程序可以验证您是否拥有或控制该域。此外，将别名记录或别名记录添加到您的 DNS 中，以定位应用程序运行程序域。您需要为自定义域添加一条记录，另一条记录用于www子域，如果您选择了此选项。然后等待自定义域状态变为处于活动状态在应用程序运行程序控制台中。这通常需要几分钟（但可能需要 24-48 小时）。此时，您的自定义域将被验证，并且 App Runner 开始将流量从此域路由到您的 Web 应用程序。

您可以通过以下方式指定要与 App Runner 服务关联的域：

- 根域— 例如，example.com。您可以选择将www.example.com也作为相同操作的一部分。
- 子域— 例如，login.example.com或者admin.login.example.com。您可以选择将www子域名。
- 一个通配符— 例如，*.example.com。您不能使用www选项。您只能将通配符指定为根域的直接子域，并且只能自行指定（这些不是有效的规范：login*.example.com、*.login.example.com)。此通配符规范关联所有直接子域，并且不关联根域本身（根域必须在单独的操作中关联）。

更具体的域关联会覆盖不太具体的域关联。例如，login.example.com覆盖*.example.com。使用更具体关联的证书和别名记录。

以下示例显示如何使用多个自定义域关联：

1. Associtsexample.com与您的服务的主页一起使用。启用www还要关联www.example.com。
2. Associtslogin.example.com与您的服务的登录页面。
3. Associts*.example.com与自定义 “未找到” 页面。

您可以取消自定义域与 App Runner 服务的关联（取消链接）。当您取消链接域时，App Runner 会停止将流量从此域路由到您的 Web 应用程序。您必须从 DNS 中删除此域的记录。

App Runner 在内部创建跟踪域有效性的证书。它们存储在AWS Certificate Manager(ACM). 在域与您的服务取消关联或服务被删除后七天内，App Runner 不会删除这些证书。

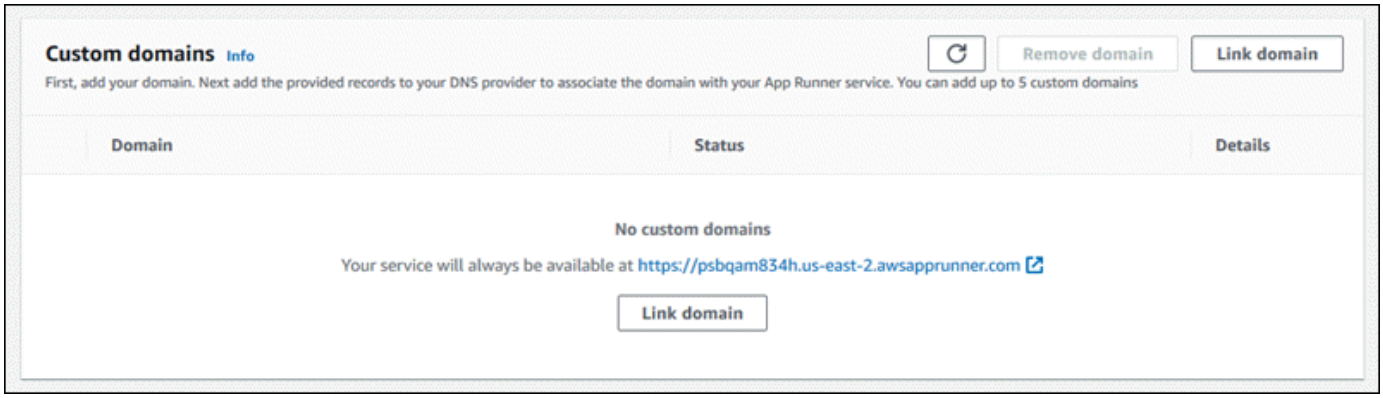
使用应用程序运行程序控制台管理自定义域

使用 App Runner 控制台关联（链接）自定义域

1. 打开应用程序运行者控制台，并在区域列表中，选择AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板，其中包含服务概述。
3. 在服务控制面板页面上，选择CUSTOM 域选项卡。

   控制台显示与您的服务关联的自定义域，或者无自定义域。



4. 在存储库的CUSTOM 域选项卡上，选择链接域。
5. 在链接自定义域对话框中，输入域名，然后选择查看 DNS 配置。
6. 按照配置 DNS页面启动域验证过程。
7. 当域状态更改为处于活动状态，请通过浏览到该域来验证该域是否适用于路由流量。

使用 App Runner 控制台取消关联（取消链接）自定义域的步骤

1. 在存储库的CUSTOM 域选项卡上，选择要取消关联的域的磁贴，然后选择断开关联域。
2. 在断开关联域对话框中，通过选择断开关联域。

使用应用程序运行程序 API 或AWS CLI

使用应用程序运行程序 API 或AWS CLI，调用关联自定义域API 操作。当调用成功时，它会返回一个CustomDomain对象，该对象描述与您的服务关联的自定义域。该对象应该显示CREATING，并包含证书验证记录对象。这些记录可以添加到 DNS 中。

使用应用程序运行程序 API 或AWS CLI，调用取消关联自定义域API 操作。当调用成功时，它会返回一个CustomDomain对象，该对象描述正在与您的服务断开关联的自定义域。该对象应该显示DELETING。

暂停和恢复应用程序运行程序服务

如果您需要暂时禁用 Web 应用程序并停止代码运行，则可以暂停AWS App Runner服务。应用程序运行程序将服务的计算容量降至零。

当您准备好再次运行应用程序时，您可以恢复您的 App Runner 服务。App Runner 配置新的计算容量，将您的应用程序部署到其中，并运行应用程序。您的应用程序源不会重新部署，也不需要构建。相反，应用程序运行程序将继续使用您当前部署的版本。您的应用程序将保留其应用程序运行程序域。

Important

- 暂停服务时，应用程序将失去其状态。例如，您的代码使用的任何临时存储都会丢失。对于您的代码，暂停和恢复服务等同于部署到新服务。
- 如果由于代码中的缺陷（例如发现的错误或安全问题）而暂停服务，则无法在恢复服务之前部署新版本。

  因此，我们建议您保持服务运行并回滚到最后一个稳定的应用程序版本。
- 恢复服务时，App Runner 会部署暂停服务之前使用的最后一个应用程序版本。如果在暂停服务后添加了任何新的源版本，即使选择了自动部署，App Runner 也不会自动部署它们。例如，假设映像存储库中有新的映像版本或代码存储库中的新提交。这些版本不会自动部署。

  要部署较新版本，请在恢复 App Runner 服务后执行手动部署或将其他版本添加到源存储库。

暂停和删除

Pause您的应用程序运行程序服务暂时禁用它。仅终止计算资源，并且存储的数据（例如，包含应用程序版本的容器映像）保持不变。恢复服务速度很快 — 您的应用程序已准备好部署到新的计算资源。App Runner 域保持不变。

Delete您的应用程序运行程序服务永久的删除它。您存储的数据将被删除。如果您需要重新创建服务，App Runner 需要再次获取源代码，并且如果它是代码存储库，也需要构建它。您的 Web 应用程序将获得一个新的应用程序运行程序域。

服务暂停时

当您暂停服务并且它位于Paused状态时，它会对操作请求（包括 API 调用或控制台操作）作出不同的响应。暂停服务时，您仍然可以执行不会以影响服务运行时的方式修改服务的定义或配置的 App Runner 操作。换句话说，如果操作更改了正在运行的服务的行为、缩放或其他特征，则无法对暂停的服务执行该操作。

以下列表提供了有关可以和不能对暂停的服务执行的 API 操作的信息。同样允许或拒绝等效的控制台操作。

您的操作can对暂停的服务执行

- *List**和*Describe**操作— 只读取信息的操作。
- *DeleteService*— 您可以随时删除服务。
- *TagResource*、*UntagResource*— 标签与服务关联，但不属于服务定义的一部分，不影响其运行时行为。

您的操作cannot对暂停的服务执行

- *StartDeployment*操作（或手动部署使用控制台）
- *UpdateService*（或使用控制台进行配置更改，标记更改除外）
- *CreateCustomDomainAssociations*, *DeleteCustomDomainAssociations*
- *CreateConnection*, *DeleteConnection*

使用应用程序运行程序控制台暂停和恢复服务

使用应用程序运行程序控制台暂停服务

1. 打开App Runner 控制台，并在区域列表中，选择AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板，其中包含服务概览。
3. 选择操作，然后选择Pause。

   在服务仪表板页面上，状态对 的更改操作正在进行中，然后更改为Paused。您的服务现已暂停。

使用应用程序运行程序控制台恢复服务

1. 选择操作，然后选择Resume。

   在服务仪表板页面上，状态对 的更改操作正在进行中。
2. 等待服务恢复。在服务仪表板页面上，状态更改回正在运行。
3. 要验证恢复服务是否成功，请在服务仪表板页面上选择应用程序 Runner值。它是您的服务网站的URL。验证 Web 应用程序是否正常运行。

使用应用程序运行程序 API 或AWS CLI

要使用应用程序运行程序 API 或AWS CLI，调用保姆服务API 操作。如果调用返回一个成功的响应服务对象显示`"Status": "OPERATION_IN_PROGRESS"`，应用程序运行程序开始暂停您的服务。

要使用应用程序运行程序 API 或AWS CLI，调用恢复服务API 操作。如果调用返回一个成功的响应服务对象显示`"Status": "OPERATION_IN_PROGRESS"`，应用程序运行程序开始恢复您的服务。

删除应用程序运行程序服务

当您想终止在AWS App Runner服务，则可删除服务。删除服务会停止正在运行的 Web 服务，删除基础资源，并删除关联的数据。

您可能会出于以下一个或多个原因而需要删除 App Runner 服务：

- 您不再需要 Web 应用程序了— 例如，它已停用，或者它是您使用完的开发版本。
- 您已达到 App Runner 服务配额— 您需要在相同的AWS 区域，并且您已达到与您的账户关联的配额。有关更多信息，请参阅the section called “App Runner 资源配额”。
- 安全或隐私注意事项— 您希望 App Runner 删除它为您的服务存储的数据。

暂停与删除

Pause您的应用程序运行程序服务到暂时禁用它。仅终止计算资源，并且存储的数据（例如，包含应用程序版本的容器映像）保持不变。恢复服务速度很快 — 您的应用程序已准备好部署到新的计算资源。您的 App Runner 域保持不变。

Delete您的应用程序运行程序服务到永久的将其删除。您存储的数据将被删除。如果您需要重新创建服务，App Runner 需要再次获取源代码，并且如果它是代码存储库，也需要构建它。您的 Web 应用程序将获得一个新的应用程序运行程序域。

应用程序运行者删除哪些内容？

删除服务时，App Runner 会删除一些关联的项目，而不会删除其他项目。下面的列表提供了详细信息。

应用程序运行者删除的项目：

- Container 映像— 您部署的映像或 App Runner 从源代码构建的映像的副本。存储在 Amazon Elastic Container Registry (Amazon ECR) 中，使用内部AWS 账户，由应用程序运行者拥有。

- 服务配置— 与您的应用程序运行程序服务关联的配置设置。它们存储在 Amazon DynamoDB 中，使用内部AWS 账户，由应用程序运行者拥有。

应用程序运行程序不会删除的项目：

- Connection— 您可能具有与您的服务关联的连接。应用程序 Runner 连接是一个单独的资源，可能在多个 App Runner 服务之间共享。如果您不再需要连接，则可将其明确删除。有关更多信息，请参阅the section called “连接”。
- 自定义域证书— 如果将自定义域链接到 App Runner 服务，App Runner 会在内部创建用于跟踪域有效性的证书。相关内容均存储在AWS Certificate Manager(ACM). 在域与您的服务取消链接或服务被删除后七天内，App Runner 不会删除该证书。有关更多信息，请参阅the section called “自定义域名”。

使用应用程序运行程序控制台删除您的服务

使用应用程序运行程序控制台删除您的服务

1. 打开App Runner 控制台，并在区域列表中，选择AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板，其中包含服务概述。
3. 选择 Actions (操作)，然后选择 Delete (删除)。

   控制台会将您带到服务页. 删除的服务将显示操作正在进行中状态，然后该服务从列表中消失。您的服务现在已被删除。

使用应用程序运行程序 API 或AWS CLI

要使用应用程序运行程序 API 或AWS CLI，调用DeleteServiceAPI 操作。如果调用返回一个成功的响应服务对象显示`"Status": "OPERATION_IN_PROGRESS"`，应用程序运行程序开始删除您的服务。

的应用程序运行程序服务的日志记录和监控

AWS App Runner与几个AWS服务，为您的 App Runner 服务提供广泛的日志记录和监控工具套件。本章中的主题介绍了这些功能。

主题

跟踪应用程序运行者服务活动

AWS App Runner使用操作列表来跟踪 App Runner 服务中的活动。操作表示对 API 操作的异步调用，例如创建服务、更新配置和部署服务。以下部分说明了如何跟踪 App Runner 控制台中的活动以及如何使用 API。

跟踪控制台中的应用程序运行程序服务活动

App Runner 控制台显示您的 App Runner 服务活动，并提供更多探索操作的方法。

查看您的服务活动

1. 打开应用程序 Runner 控制台，并在区域列表中，选择AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板，其中包含服务概览。
3. 在服务控制面板页面上，选择活动选项卡（如果尚未选择）。

   控制台显示了操作列表。
4. 要查找特定操作，请通过输入搜索词来缩小列表范围。您可以搜索表中显示的任何值。
5. 选择列出的任何操作以查看或下载相关日志。

使用应用程序运行程序 API 或AWS CLI

这些区域有：ListOperations操作（给定应用 Runner 服务的 Amazon 资源名称 (ARN)）将返回在此服务上发生的操作列表。每个列表项都包含一个操作 ID 和一些跟踪详细信息。

查看流式传输到 CloudWatch 日志的应用程序运行程序日志

您可以使用 Amazon CloudWatch Logs 监控、存储和访问日志文件AWS服务生成。有关更多信息，请参阅 。Amazon CloudWatch Logs 用户指南。

AWS App Runner收集应用程序部署和活动服务的输出，并将其流式传输到 CloudWatch Logs 中。以下部分列出了 App Runner 日志流，并向您展示如何在 App Runner 控制台中查看它们。

应用程序 Runner 日志组和流

CloudWatch Logs 将日志数据保存在日志流中，它进一步组织在日志组中。A日志流是来自特定源的一系列日志事件。日志组 是一组具有相同保留期、监控和访问控制设置的日志流。

应用程序运行程序定义了两个 CloudWatch Logs 组，每个日志组都有多个日志流，用于AWS 账户。

服务日志

服务日志组包含由 App Runner 生成的日志记录输出，因为它管理您的 App Runner 服务并对其执行操作。

日志组名称	示例
`/aws/apprunner/ service-name /service-id /service`	`/aws/apprunner/python-test/ ac7ec8b51ff34746bcb6654e0bcb23da/service`

在服务日志组中，App Runner 会创建一个事件日志流，以捕获 App Runner 服务生命周期中的活动。例如，这可能是启动您的应用程序或暂停它。

此外，App Runner 为与您的服务相关的每个长时间运行的异步操作创建一个日志流。日志流名称反映了操作类型和特定操作 ID。

A部署是一种操作类型。部署日志包含在您创建服务或部署应用程序的新版本时 App Runner 执行的构建和部署步骤的日志记录输出。部署日志流名称以deployment/，并以执行部署的操作的 ID 结束。此操作要么是CreateService调用初始应用程序部署或StartDeployment要求每次进一步部署.

在部署日志中，每条日志消息都以前缀开头：

- [AppRunner]— 应用程序运行程序在部署期间生成的输出。
- [Build]— 您自己的构建脚本的输出。

日志流名称	示例
events	N/A（固定名称）
operation-type /*operation-id*	deployment/c2c8eeedea164f45 9cf78f12a8953390

应用程序日志

应用程序日志组包含正在运行的应用程序代码的输出。

日志组名称	示例
/aws/apprunner/ *service-n ame* /*service-id* /application	/aws/apprunner/python-test/ ac7ec8b51ff34746bcb6654e0bcb23da/ application

在应用程序日志组中，App Runner 为运行应用程序的每个实例（扩展单元）创建日志流。

日志流名称	示例
instance/ *instance-id*	instance/1a80bc9134a84699b7 b3432ebeebb591

在控制台中查看应用程序 Runner 日志

App Runner 控制台显示服务所有日志的摘要，并允许您查看、浏览和下载这些日志。

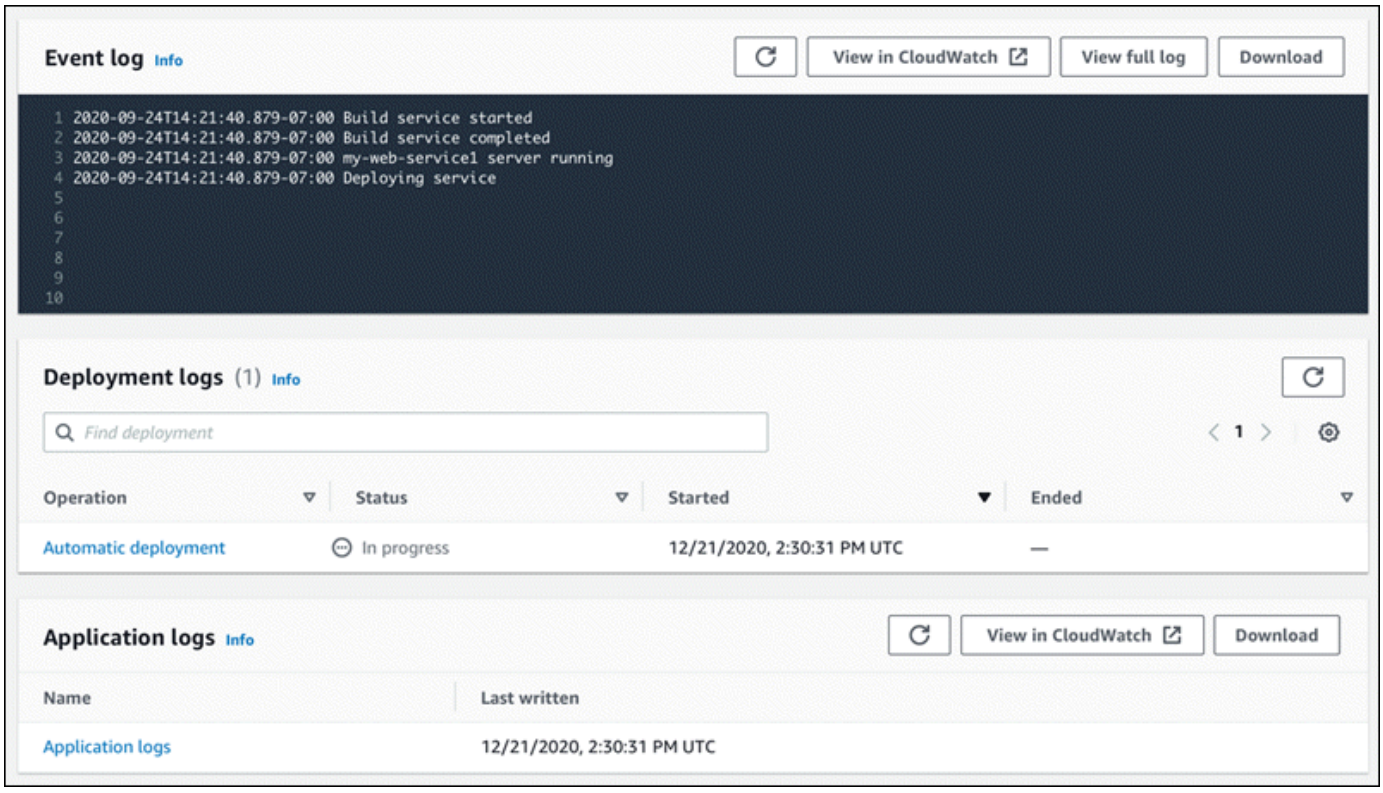
要查看服务的日志

1. 打开应用程序 Runner 控制台，并在区域列表中，选择您的AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板服务概述。
3. 在服务控制面板页面上，选择日志选项卡。

   控制台在几个部分显示几种类型的日志：

   - 事件日志— 应用程序运行程序服务生命周期中的活动。控制台将显示最新的事件。
   - 部署日志— 将源存储库部署到您的应用程序运行程序服务。控制台为每个部署显示单独的日志流。
   - 应用程序日志— 部署到您的 App Runner 服务的 Web 应用程序的输出。控制台将所有正在运行的实例的输出结合到单个日志流中。



4. 要查找特定部署，请通过输入搜索词来缩小部署日志列表的范围。您可以搜索表格中显示的任何值。
5. 要查看日志的内容，请选择查看完整日志（事件日志）或日志流名称（部署和应用程序日志）。
6. 选择下载以下载日志。对于部署日志流，请先选择一个日志流。
7. 选择在 CloudWatch Logs 中查看打开 CloudWatch 控制台，并使用其全部功能来浏览您的应用程序运行程序服务日志。对于部署日志流，请先选择一个日志流。

> **Note**
>
> 如果您希望查看特定实例的应用程序日志而不是组合的应用程序日志，CloudWatch 控制台特别有用。

查看向 CloudWatch 报告的应用程序运行程序服务指标

Amazon CloudWatch 可监控您的 Amazon Web Services (AWS) 资源以及您在上运行的应用程序AWS实时。您可以使用 CloudWatch 收集和跟踪指标，这些指标是您可衡量的相关资源和应用程序的变量。您还可以使用它来创建警报，监控指标。当达到特定阈值时，CloudWatch 会发送通知或自动更改受监控的资源。有关更多信息，请参阅 。Amazon CloudWatch 用户指南。

AWS App Runner收集各种指标，使您能够更好地了解 App Runner 服务的使用情况、性能和可用性。有些指标跟踪运行 Web 服务的单个实例，而其他指标则在整体服务级别。以下部分列出了 App Runner 指标，并向您展示如何在 App Runner 控制台中查看这些指标。

应用程序 Runner 指标

应用程序运行程序会收集以下与您的服务相关的指标，并 CloudWatch 它们发布到`AWS/AppRunner`命名空间。

实例级指标分别为每个实例（扩展单位）收集。

什么是测量的？	指标	描述
CPU utilization	`CPUUtilization`	一分钟内的平均 CPU 使用率。
Memory utilization	`MemoryUtilization`	一分钟内的平均内存使用情况。

服务等级指标收集整个服务。

什么是测量的？	指标	描述
HTTP request count	`Requests`	服务接收的 HTTP 请求的数量。

HTTP status counts	`2xxStatus Responses`
	
	`4xxStatus Responses`
	
	`5xxStatus Responses`
		返回每个响应状态的 HTTP 请求的数量，按类别（2XX、4XX、5XX）分组。

HTTP request latency	`RequestLatency`	您的 Web 服务处理 HTTP 请求所花费的时间。
Instance counts	`ActiveInstances`	正处理针对您的服务的 HTTP 请求的实例的数量。

在控制台中查看应用程序 Runner 指标

App Runner 控制台以图形方式显示 App Runner 为您的服务收集的指标，并提供更多探索这些指标的方法。

> **Note**
>
> 此时，控制台仅显示服务指标。要查看实例指标，请使用 CloudWatch 视控制台。

要查看服务的日志

1. 打开应用程序 Runner 控制台，并在区域列表中，选择您的AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的应用程序运行程序服务。

   控制台将显示服务仪表板服务概述。
3. 在服务控制面板页面上，选择指标选项卡。

控制台显示一组指标图表。

4. 选择持续时间（例如12 小时）将指标图形范围到该持续时间的最近时段。
5. 选择添加到控制面板或使用任何图形上的菜单，将相关指标添加到 CloudWatch 控制台中的仪表板以进行进一步调查。

处理 EventBridge 中的应用程序运行者事件

使用 Amazon EventBridge，您可以设置事件驱动型规则，以监控来自AWS App Runner服务的特定模式。当规则的模式匹配时，EventBridge 会在目标中启动一个操作，例如AWS Lambda、Amazon ECS、AWS Batch和 Amazon SNS。例如，每当服务部署失败时，您可以通过向 Amazon SNS 主题发出信号来设置发出电子邮件通知的规则。或者，您可以将 Lambda 函数设置为每当服务更新失败时通知 Slack 通道。有关 EventBridge 的更多信息，请参阅Amazon EventBridge 用户指南。

应用程序运行程序将以下事件类型发送到 EventBridge

- 服务状态更改— 应用程序运行程序服务状态的更改。例如，服务状态更改为`DELETE_FAILED`。
- 服务操作状态更改— 对 App Runner 服务进行长时间异步操作的状态更改。例如，服务开始创建，服务更新成功完成，或者服务部署完成时出现错误。

创建 EventBridge 规则以处理应用程序运行程序事件

EventBridgeevent是一个定义了一些标准 EventBridge 字段的对象，例如源AWS服务和详细信息（事件）类型，以及一组具有事件详细信息的特定于事件的字段。若要创建 EventBridge 梁规则，您可以使用 EventBridge 控制台定义事件模式（应该跟踪哪些事件），并指定目标操作（在比赛中应该做什么）。事件模式类似于它匹配的事件。指定要匹配的字段子集，并为每个字段指定一个可能值的列表。本主题提供了 App Runner 事件和事件模式的示例。

有关创建 EventBridge 规则的更多信息，请参阅为创建规则AWS服务中的Amazon EventBridge 用户指南。

> **Note**
>
> 一些服务支持预定义模式，将设置为 EventBridge。这简化了事件模式的创建方式。您可以在窗体上选择字段值，并且 EventBridge 为您生成模式。目前，App Runner 不支持预定义的模式。您必须输入模式作为 JSON 对象。您可以使用本主题中的示例作为起始点。

应用程序运行者事件示例

以下是应用程序运行者发送给 EventBridge 的事件的一些示例。

- 服务状态更改事件。具体来说，从OPERATION_IN_PROGRESS添加到RUNNING状态。

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "Service Status Change",
  "source": "aws.apprunner",
  "account": "111122223333",
  "time": "2021-04-29T11:54:23Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:apprunner:us-east-2:123456789012:service/my-
app/8fe1e10304f84fd2b0df550fe98a71fa"
  ],
  "detail": {
    "PreviousStatus": "OPERATION_IN_PROGRESS",
    "CurrentStatus": "RUNNING",
    "ServiceName": "my-app",
    "ServiceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "Message": "Service status is set to RUNNING.",
    "Severity": "INFO"
  }
}
```

- 操作状态更改事件。具体来说，UpdateService操作已成功完成。

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "Service Operation Status Change",
  "source": "aws.apprunner",
  "account": "111122223333",
  "time": "2021-04-29T18:43:48Z",
  "region": "us-east-2",
  "resources": [
    "arn:aws:apprunner:us-east-2:123456789012:service/my-
app/8fe1e10304f84fd2b0df550fe98a71fa"
  ],
  "detail": {
```

```
    "Status": "UpdateServiceCompletedSuccessfully",
    "ServiceName": "my-app",
    "ServiceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "Message": "Service update completed successfully. New application and
 configuration is deployed.",
    "Severity": "INFO"
  }
}
```

应用程序运行者事件模式示例

以下示例演示了事件模式，您可以在 EventBridge 规则中使用这些模式来匹配一个或多个应用程序运行程序事件。事件模式类似于事件。仅包含要匹配的字段，并为每个字段提供一个列表而不是标量。

- 匹配特定帐户的服务的所有服务状态更改事件，其中服务不再位于RUNNING状态。

```
{
  "detail-type": [ "Service Status Change" ],
  "source": [ "aws.apprunner" ],
  "account": [ "111122223333" ],
  "detail": {
    "PreviousStatus": [ "RUNNING" ]
  }
}
```

- 匹配操作失败的特定帐户服务的所有操作状态更改事件。

```
{
  "detail-type": [ "Service Operation Status Change" ],
  "source": [ "aws.apprunner" ],
  "account": [ "111122223333" ],
  "detail": {
    "Status": [
      "CreateServiceFailed",
      "DeleteServiceFailed",
      "UpdateServiceFailed",
      "DeploymentFailed",
      "PauseServiceFailed",
      "ResumeServiceFailed"
    ]
  }
```

```
}
```

应用程序运行者事件参考

服务状态更改

服务状态更改事件具有detail-type设置为Service Status Change。它具有以下详细信息字段和值：

```
"PreviousStatus": "any valid service status",
"CurrentStatus": "any valid service status",
"ServiceName": "your service name",
"ServiceId": "your service ID",
"Message": "Service status is set to CurrentStatus.",
"Severity": "varies"
```

操作状态更改

操作状态更改事件具有detail-type设置为Service Operation Status Change。它具有以下详细信息字段和值：

```
"Status": "see following table",
"ServiceName": "your service name",
"ServiceId": "your service ID",
"Message": "see following table",
"Severity": "varies"
```

下表列出了所有可能的状态码和相关消息。

状态	Message
CreateServiceStarted	服务创建已启动。
CreateServiceCompletedSuccessfully	服务创建已成功完成。
CreateServiceFailed	服务创建失败。有关详细信息，请参阅服务日志。
DeleteServiceStarted	服务删除已启动。

状态	Message
DeleteServiceCompletedSuccessfully	服务删除已成功完成。
DeleteServiceFailed	服务删除失败。
UpdateServiceStarted	
UpdateServiceCompletedSuccessfully	服务更新已成功完成。部署了新的应用程序和配置。
	服务更新已成功完成。部署了新的配置。
UpdateServiceFailed	服务更新失败。有关详细信息，请参阅服务日志。
DeploymentStarted	已开始部署。
DeploymentCompletedSuccessfully	部署已成功完成。
DeploymentFailed	部署失败。有关详细信息，请参阅服务日志。
PauseServiceStarted	服务暂停。
PauseServiceCompletedSuccessfully	服务暂停已成功完成。
PauseServiceFailed	服务暂停失败。
ResumeServiceStarted	服务恢复。
ResumeServiceCompletedSuccessfully	服务恢复已成功完成。
ResumeServiceFailed	服务恢复失败。

使用记录应用程序 Runner API 调用AWS CloudTrail

应用程序运行程序与AWS CloudTrail，该服务提供了用户、角色或AWS服务。CloudTrail 将应用程序 Runner 的所有 API 调用作为事件捕获。捕获的调用包括来自 App Runner 控制台的调用和对应用

Runner API 操作的代码调用。如果您创建跟踪，则可以使 CloudTrail 事件持续传送到 Amazon S3 存储桶（包括应用程序 Runner 的事件）。如果您不配置跟踪，则仍可在 CloudTrail 控制台的 Event history (事件历史记录) 中查看最新事件。使用 CloudTrail 收集的信息，您可以确定向 App Runner 发出了什么请求、发出请求的 IP 地址、请求的发出时间以及其他详细信息。

要了解 CloudTrail 的更多信息，请参阅AWS CloudTrail用户指南。

CloudTrail 中的应用程序运行者信息

CloudTrail 已在您的AWS 账户创建账户时，应用于此账户。当 App Runner 中发生活动时，该活动将记录在 CloudTrail 事件中，并与其他AWS中的服务事件事件历史记录。您可以在AWS 账户。有关更多信息，请参阅使用 CloudTrail 事件历史记录查看事件。

要持续记录在您的AWS 账户（包括 App Runner 的事件），应创建跟踪。通过 trail (跟踪)，CloudTrail 可将日志文件传送至 Amazon S3 存储桶。默认情况下，在控制台中创建跟踪时，此跟踪应用于所有 AWS 区域。此跟踪记录来自AWS分区，并将日志文件传送到您指定的 Amazon S3 存储桶。此外，您可以配置其他AWS服务，进一步分析在 CloudTrail 日志中收集的事件数据并采取操作。有关更多信息，请参阅下列内容：

- 创建跟踪概述
- CloudTrail 支持的服务和集成
- 为 CloudTrail 配置 Amazon SNS 通知
- 接收多个区域中的 CloudTrail 日志文件和从多个账户中接收 CloudTrail 日志文件

所有应用程序运行器操作都由 CloudTrail 记录，并记录在AWS App RunnerAPI 参考。例如，对 `CreateService`、`DeleteConnection` 和 `StartDeployment` 操作的调用会在 CloudTrail 日志文件中生成条目。

每个事件或日志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身份用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅 CloudTrail userIdentity 元素。

了解应用程序 Runner 日志文件条目

跟踪是一种配置，可用于将事件作为日志文件传送到您指定的 Amazon S3 存储桶。CloudTrail 日志文件包含一个或多个日志条目。一个事件表示来自任何源的一个请求，包括有关所请求的操作、操作的日期和时间以及请求参数的信息。CloudTrail 日志文件不是公用 API 调用的有序堆栈跟踪，因此它们不会按任何特定顺序显示。

下面的示例显示了一个 CloudTrail 日志条目，该条目说明了 `CreateService` 操作。

> **Note**
>
> 出于安全原因，某些属性值在日志中被编辑，并替换为文本`HIDDEN_DUE_TO_SECURITY_REASONS`。这样可以防止意外泄露秘密信息。但是，您仍然可以看到这些属性已在请求中传递或在响应中返回。

的 CloudTrail 日志条目示例**`CreateService`**应用程序运行者操作

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::123456789012:user/aws-user",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "aws-user",
  },
  "eventTime": "2020-10-02T23:25:33Z",
  "eventSource": "apprunner.amazonaws.com",
  "eventName": "CreateService",
  "awsRegion": "us-east-2",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/77.0.3865.75 Safari/537.36",
  "requestParameters": {
    "serviceName": "python-test",
    "sourceConfiguration": {
      "codeRepository": {
        "repositoryUrl": "https://github.com/github-user/python-hello",
        "sourceCodeVersion": {
          "type": "BRANCH",
```

```
          "value": "main"
        },
        "codeConfiguration": {
          "configurationSource": "API",
          "codeConfigurationValues": {
            "runtime": "python3",
            "buildCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
            "startCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
            "port": "8080",
            "runtimeEnvironmentVariables": "HIDDEN_DUE_TO_SECURITY_REASONS"
          }
        }
      },
      "autoDeploymentsEnabled": true,
      "authenticationConfiguration": {
        "connectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/your-
connection/e7656250f67242d7819feade6800f59e"
      }
    },
    "healthCheckConfiguration": {
      "protocol": "HTTP"
    },
    "instanceConfiguration": {
      "cpu": "256",
      "memory": "1024"
    }
  },
  "responseElements": {
    "service": {
      "serviceName": "python-test",
      "serviceId": "dfa2b7cc7bcb4b6fa6c1f0f4efff988a",
      "serviceArn": "arn:aws:apprunner:us-east-2:123456789012:service/python-test/
dfa2b7cc7bcb4b6fa6c1f0f4efff988a",
      "serviceUrl": "generated domain",
      "createdAt": "2020-10-02T23:25:32.650Z",
      "updatedAt": "2020-10-02T23:25:32.650Z",
      "status": "OPERATION_IN_PROGRESS",
      "sourceConfiguration": {
        "codeRepository": {
          "repositoryUrl": "https://github.com/github-user/python-hello",
          "sourceCodeVersion": {
            "type": "Branch",
            "value": "main"
          },
```

```
          "codeConfiguration": {
            "codeConfigurationValues": {
            "configurationSource": "API",
              "runtime": "python3",
              "buildCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
              "startCommand": "HIDDEN_DUE_TO_SECURITY_REASONS",
              "port": "8080",
              "runtimeEnvironmentVariables": "HIDDEN_DUE_TO_SECURITY_REASONS"
            }
          }
        },
        "autoDeploymentsEnabled": true,
        "authenticationConfiguration": {
          "connectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/your-
connection/e7656250f67242d7819feade6800f59e"
        }
      },
      "healthCheckConfiguration": {
        "protocol": "HTTP",
        "path": "/",
        "interval": 5,
        "timeout": 2,
        "healthyThreshold": 3,
        "unhealthyThreshold": 5
      },
      "instanceConfiguration": {
        "cpu": "256",
        "memory": "1024"
      },
      "autoScalingConfigurationSummary": {
        "autoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
        "autoScalingConfigurationName": "DefaultConfiguration",
        "autoScalingConfigurationRevision": 1
      }
    }
  },
  "requestID": "1a60af60-ecf5-4280-aa8f-64538319ba0a",
  "eventID": "e1a3f623-4d24-4390-a70b-bf08a0e24669",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "123456789012"
```

```
}
```

使用配置文件设置应用程序运行程序服务选项

> **Note**
>
> 配置文件仅适用于基于源代码的服务。您不能将配置文件与基于映像的服务。

在创建时AWS App Runner服务，使用源代码存储库AWS App Runner需要有关构建和启动服务的信息。您可以在每次使用 App Runner 控制台或 API 创建服务时提供此信息。此外，您可以通过使用配置文件。您在文件中指定的选项将成为源存储库的一部分，对这些选项的任何更改都会跟踪跟踪源代码更改的方式相似。您可以使用 App Runner 配置文件指定比 API 支持的更多选项。如果您只需要 API 支持的基本选项，则无需提供配置文件。

应用程序运行程序配置文件是名为`apprunner.yaml`在应用程序存储库的根目录中。它为您的服务提供构建和运行时选项。此文件中的值指示 App Runner 如何构建和启动服务，并提供运行时上下文（如网络设置和环境变量）。

App Runner 配置文件不包含操作设置，例如 CPU 和内存。

有关 App Runner 配置文件的示例，请参阅the section called “示例”。有关完整的参考指南，请参阅the section called “参考”。

主题

- 应用程序运行程序配置文件示例
- 应用程序运行程序配置文件参考

应用程序运行程序配置文件示例

> **Note**
>
> 配置文件仅适用于基于源代码的服务。不能将配置文件与基于映像的服务。

以下示例展示AWS App Runner配置文件。有些是最小的，只包含必需的设置。其他部分已完成，包括所有配置文件部分。有关 App Runner 配置文件的概述，请参阅应用程序运行程序配置文件。

配置文件

最小配置文件

使用最小的配置文件，App Runner 会做出以下假设：

- 构建或运行期间不需要自定义环境变量。
- 使用最新的运行时版本。
- 使用默认端口号和端口环境变量。

Example 呼吸器 .yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install pipenv
      - pipenv install
run:
  command: python app.py
```

完整的配置文件

此示例说明使用托管运行时的所有配置密钥。

Example 呼吸器 .yaml

```
version: 1.0
runtime: python3
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/DOC-EXAMPLE-BUCKET/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip install pipenv
      - pipenv install
    post-build:
      - python manage.py test
  env:
```

```
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.7.7
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

有关特定托管运行时配置文件的示例，请参阅基于代码的服务。

应用程序运行程序配置文件参考

> **Note**
>
> 配置文件仅适用于基于源代码的服务。不能将配置文件与基于映像的服务。

本主题是一个全面的参考指南，介绍了AWS App Runner配置文件。有关 App Runner 配置文件的概述，请参阅应用程序运行程序配置文件。

应用程序运行程序配置文件是 YAML 文件。命名它`apprunner.yaml`，并将其放入您的应用程序存储库的根目录中。

结构概述

应用程序运行程序配置文件是 YAML 文件。命名它`apprunner.yaml`，并将其放入您的应用程序存储库的根目录中。

应用程序运行器配置文件包含以下主要部分：

- 顶边— 包含顶级键
- 构建部分— 配置构建阶段
- 运行部分— 配置运行时阶段

顶边

文件顶部的键提供有关文件和服务运行时的一般信息。可用键如下：

- `version`–必填项。应用程序运行程序配置文件版本。理想情况下，请使用最新版本的。

  语法

  ```
  version: version
  ```

  Example

  ```
  version: 1.0
  ```

- `runtime`–必填项。应用程序使用的运行时的名称。以下运行时目前可用：

  `python3`, `nodejs12`

  > **Note**
  > 托管运行时的命名约定是*<language-name> <major-version>*。

  语法

  ```
  runtime: runtime-name
  ```

  Example

  ```
  runtime: python3
  ```

构建部分

构建部分配置 App Runner 服务部署的构建阶段。您可以指定构建命令和环境变量。生成命令是必需的。

该部分以`build:`键，并具有以下子项：

- `commands`–必填项。指定 App Runner 在各个构建阶段运行的命令。包含以下子项：

- `pre-build`–可选。App Runner 在构建之前将运行的命令。例如，安装npm依赖关系或测试库。
- `build`–必填项。App Runner 为构建应用程序而运行的命令。例如，使用pipenv。
- `post-build`–可选。App Runner 在构建后将运行的命令。例如，使用 Maven 将构建项目打包成 JAR 或 WAR 文件，还可以运行测试。

语法

```
build:
  commands:
    pre-build:
      - command
      - …
    build:
      - command
      - …
    post-build:
      - command
      - …
```

Example

```
build:
  commands:
    pre-build:
      - yum install openssl
    build:
      - pip install -r requirements.txt
    post-build:
      - python manage.py test
```

- env–可选。指定构建阶段的自定义环境变量。定义为名称-值标量映射。您可以在构建命令中按名称引用这些变量。

语法

```
build:
  env:
    - name: name1
      value: value1
    - name: name2
      value: value2
```

```
    - …
```

Example

```
build:
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
```

运行部分

运行部分配置 App Runner 应用程序部署的容器运行阶段。您可以指定运行时版本、启动命令、网络端口和环境变量。

该部分以`run:`键，并具有以下子项：

- `runtime-version`–可选。指定要为 App Runner 服务锁定的运行时版本。

  默认情况下，只有主要版本被锁定。App Runner 使用每个部署或服务更新时可用于运行时的最新次要版本和修补程序版本。如果您指定了主要版本和次要版本，则两者都会被锁定，并且 App Runner 仅更新修补程序版本。如果指定主要版本、次版本和修补程序版本，则服务将锁定在特定的运行时版本上，而 App Runner 永远不会更新它。

  语法

  ```
  run:
    runtime-version: major[.minor[.patch]]
  ```

  Example

  ```
  runtime: python3
  run:
    runtime-version: 3.7
  ```

- `command`–必填项。App Runner 用于在应用程序构建完成后运行应用程序的命令。

  语法

```
run:
  command: command
```

- network–可选。指定应用程序侦听的端口。其中包括以下内容：
  - port–可选。如果指定，则此为您的应用程序侦听的端口号。默认为 8080。
  - env–可选。如果指定，App Runner 将端口号传递给此环境变量中的容器，除了（而不是）在默认环境变量中传递相同的端口号外，PORT。换言之，如果指定env，App Runner 会在两个环境变量中传递端口号。

  语法

```
run:
  network:
    port: port-number
    env: env-variable-name
```

  Example

```
run:
  network:
    port: 8000
    env: MY_APP_PORT
```

- env–可选。定义运行阶段的自定义环境变量。定义为名称-值标量映射。您可以在运行时环境中按名称引用这些变量。

  语法

```
run:
  env:
    - name: name1
      value: value1
    - name: name2
      value: value2
    - …
```

  Example

```
run:
```

```
env:
  - name: MY_VAR_EXAMPLE
    value: "example"
```

应用程序运行者 API

这些区域有：AWS App Runner应用程序编程接口 (API) 是一个 RESTful API，用于向应用程序运行程序服务发出请求。您可以使用 API 来创建、列出、更新和删除 App Runner 资源。AWS 账户。

您可以在应用程序代码中直接调用 API，也可以使用其中的一个AWS开发工具包。对于命令行脚本，请使用AWS CLI来调用应用程序运行程序服务。有关 的更多信息AWS开发人员工具，请参阅要构建的工具AWS。

有关完整的 API 参考信息，请参阅AWS App RunnerAPI 参考。

应用程序运行者中的安全性

AWS 的云安全性的优先级最高。为了满足对安全性最敏感的组织的需求，我们打造了具有超高安全性的数据中心和网络架构。作为 AWS 客户，您也将从这些数据中心和网络架构受益。

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负责保护运行AWS服务AWS Cloud。AWS还向您提供可安全使用的服务。作为AWS 合规性计划的一部分，第三方审计人员将定期测试和验证安全性的有效性。要了解适用于AWS App Runner，请参阅AWS合规性计划范围内的服务。
- 云中的安全性— 您的责任由AWS服务。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

此文档将帮助您了解如何在使用 App Runner 时应用责任共担模型。以下主题说明如何配置 App Runner 以实现您的安全性和合规性目标。您还将了解如何使用其他AWS服务，以帮助您监控和保护您的应用程序运行程序资源。

主题

- App Runner 中的数据保护
- 适用于应用程序运行者的身份和访问管理
- 中的日志记录和监控
- 的应用运行者的合规性验证
- 应用程序运行者中的弹性
- AWS App Runner 中的基础设施安全性
- 应用程序运行中的配置和漏洞分析
- 适用于 App Runner 的安全最佳实践

App Runner 中的数据保护

这些区域有：AWS 责任共担模式适用于中的数据保护AWS App Runner。如本模型所述,AWS负责保护运行所有AWS云。您负责维护对托管在此基础设施上的内容的控制。此内容包括AWS您使用的服务。有关数据隐私的更多信息，请参阅数据隐私常见问题。有关欧洲数据保护的信息，请参阅AWS责任共担模式和 GDPR博客文章AWS安全博客。

出于数据保护目的，我们建议您保护AWS账户凭证并使用设置单独的用户账户AWS Identity and Access Management(IAM). 这仅向每个用户授予履行其工作职责所需的权限。我们还建议您通过以下方式保护您的数据：

- 对每个账户使用 Multi-Factor Authentication (MFA)。
- 使用 SSL/TLS 与 AWS 资源进行通信。建议使用 TLS 1.2 或更高版本。
- 使用 AWS CloudTrail 设置 API 和用户活动日志记录。
- 使用 AWS 加密解决方案以及 AWS 服务中的所有默认安全控制。
- 使用高级托管安全服务（例如 Amazon Macie），它有助于发现和保护存储在 Amazon S3 中的个人数据。
- 如果在通过命令行界面或 API 访问 AWS 时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 终端节点。有关可用的 FIPS 终端节点的更多信息，请参阅美国联邦信息处理标准 (FIPS) 第 140-2 版。

我们强烈建议您切勿将敏感的可识别信息（例如您客户的账号）放入自由格式字段（例如 Name (名称) 字段）。这包括使用 App Runner 或其他AWS服务使用控制台、API、AWS CLI，或者AWS开发工具包。您输入到 App Runner 或其他服务中的任何数据都可能被选取以包含在诊断日志中。当您向外部服务器提供 URL 时，请勿在 URL 中包含凭证信息来验证您对该服务器的请求。

有关其他应用程序 Runner 安全主题，请参阅。安全性。

主题

- 使用加密保护数据
- 互联网络流量隐私
- 将应用运行程序与 VPC 终端节点结合使用

使用加密保护数据

AWS App Runner从指定的存储库中读取应用程序源（源映像或源代码），并将其存储以供部署到您的服务。有关更多信息，请参阅体系结构和概念。

数据保护是指保护数据，而在运输中（当它往返于应用程序运行者时）和静态（而它存储在AWS数据中心）。

有关数据保护的更多信息，请参阅。the section called “数据保护”。

有关其他应用程序 Runner 安全主题，请参阅。安全性。

传输中加密

您可以通过两种方式在传输过程中实现数据保护：使用传输层安全 (TLS) 加密连接，或使用客户端加密（对象在发送之前先进行加密）。这两种方法都可有效地保护您的应用程序数据。为了保护连接，请在您的应用程序、其开发人员和管理员以及最终用户发送或接收任何对象时使用 TLS 对其进行加密。应用程序运行程序设置您的应用程序通过 TLS 接收流量。

客户端加密不是用于保护您提供给 App Runner 以部署的源映像或代码的有效方法。App Runner 需要访问您的应用程序源，因此无法对其进行加密。因此，请确保保护开发或部署环境与 App Runner 之间的连接。

静态加密和密钥管理

为了保护应用程序的静态数据，App Runner 将对应用程序源映像或源包的所有存储副本进行加密。当您创建 App Runner 服务时，可以提供客户主密钥 (CMK)。如果您提供了密钥，App Runner 将使用您提供的密钥对源进行加密。如果您未提供密钥策略，则 App Runner 将使用AWS托管 CMK。

有关 App Runner 服务创建参数的详细信息，请参阅CreateService。有关的信息AWS Key Management Service(AWS KMS），请参阅AWS Key Management Service开发人员指南。

互联网络流量隐私

App Runner 使用 Amazon Virtual Private Cloud (Amazon VPC) 在 App Runner 应用程序中的资源之间创建边界，并控制它们、本地网络和 Internet 之间的流量。有关 Amazon VPC 安全性的更多信息，请参阅。Amazon VPC 中的互联网络流量隐私中的Amazon VPC 用户指南。

有关使用 VPC 终端节点保护对应用 Runner 的请求，请参阅。the section called “VPC 终端节点”。

有关数据保护的更多信息，请参阅。the section called “数据保护”。

有关其他应用程序 Runner 安全主题，请参阅。安全性。

将应用运行程序与 VPC 终端节点结合使用

您的AWS应用程序可能集成AWS App Runner服务与其他AWS服务运行在Amazon Virtual Private Cloud (Amazon VPC)。您的应用程序的部分部分可能会从 VPC 内向 App Runner 发出请求。例如，您可以使用AWS CodePipeline以持续部署到您的应用程序运行程序服务。提高应用程序安全性的一种方法是将这些 App Runner 请求（以及请求发送到其他AWS服务 VPC。

VPC 终端节点 使您能够将 VPC 私密地连接到支持的 AWS 服务和 VPC 终端节点服务（由 AWS PrivateLink 提供支持），而无需互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。

VPC 中的资源不会使用公有 IP 地址与 App Runner 资源交互。VPC 和 App Runner 之间的流量不会脱离 Amazon 网络。有关 VPC 终端节点的完整信息，请参阅。VPC 终端节点中的AWS PrivateLink指南。

应用程序运行者支持AWS PrivateLink，这将提供与 App Runner 的私有连接并使流量都能进入公共 Internet。要使您的应用程序能够使用AWS PrivateLinkVPC，您可以配置一种称为接口 VPC 终端节点（接口端点）。有关更多信息，请参阅 。接口 VPC 终端节点 (AWS PrivateLink) 中的AWS PrivateLink指南。

> **Note**
>
> 您的 App Runner 服务中的 Web 应用程序在 App Runner 提供和配置的 VPC 中运行。此 VPC 是公共的 — 它已连接到公共互联网。App Runner 不支持在私有 VPC 中运行您的应用程序或为其创建 VPC 终端节点。

为应用程序运行程序设置 VPC 终端节点

要在 VPC 中为 App Runner 服务创建接口 VPC 终端节点，请遵循创建接口终端节点中的过程AWS PrivateLink指南。对于 Service Name (服务名称)，选择 `com.amazonaws.region.apprunner`。

VPC 网络隐私注意事项

> **Important**
>
> 对应用程序运行程序使用 VPC 终端节点并不能保证来自您 VPC 的所有流量都远离互联网。VPC 可能是公有的（互联网连接），您的解决方案的某些部分可能不会使用 VPC 终端节点来创建AWSAPI 调用。例如，AWS服务可能会使用其公有终端节点调用其他服务。如果 VPC 中的解决方案需要流量隐私，请阅读本节。

要确保 VPC 中的网络流量具有隐私性，请考虑以下事项：

- 启用 DNS 名— 部分应用程序可能仍然通过互联网向 App Runner 发送请求，使用`apprunner.region.amazonaws.com`公有终端节点。如果已为 VPC 配置公有 Internet 访问权限，则这些请求会成功，而无需向您发出任何指示。可以通过确保在终端节点创建期间启用 Enable

DNS name (启用 DNS 名称) 来防止这种情况（默认情况下为 true）。这会在您的 VPC 中添加一个 DNS 条目，该条目将公有服务终端节点映射到接口 VPC 终端节点。

- 为其他服务配置 VPC 终端节点— 您的解决方案可能会将请求发送到其他AWS服务。例如，AWS CodePipeline可能会将请求发送到AWS CodeBuild。也可为这些服务配置 VPC 终端节点，并在这些终端节点上启用 DNS 名。

使用终端节点策略控制通过 VPC 终端节点进行的访问

默认情况下，VPC 终端节点允许对与其关联的服务进行完全访问。当您为 App Runner 创建或修改 VPC 终端节点时，可以将终端节点策略到它。

终端策略是AWS Identity and Access Management(IAM) 资源策略，用于控制从终端节点对指定服务进行的访问。终端节点策略特定于某个终端节点。它独立于您的环境可能拥有的任何用户或实例 IAM 策略，并且不会覆盖或替换这些策略。有关创作和使用 VPC 终端节点策略的详细信息，请参阅。使用 VPC 终端节点控制对服务的访问中的AWS PrivateLink指南。

以下示例允许从 VPC 终端节点到应用程序运行程序的只读访问。这些是使用 VPC 终端节点时的最低访问权限。委托人可能通过其他策略具有其他权限。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apprunner:List*",
        "apprunner:Describe*"
      ],
      "Resource": "*"
    }
  ]
}
```

适用于应用程序运行者的身份和访问管理

AWS Identity and Access Management(IAM) 是一个AWS服务，可以帮助管理员安全地控制对AWS资源的费用。IAM 管理员可以控制哪些人身份验证(已登录) 和Autited（具有权限）来使用应用程序运行程序资源。IAM 是一个AWS服务，您可以免费使用。

有关其他应用程序运行者安全主题，请参阅安全性。

主题

Audience

如何使用AWS Identity and Access Management(IAM) 有所不同，具体取决于您在 App Runner 中完成的工作。

服务用户-如果您使用 App Runner 服务来完成工作，则您的管理员会为您提供所需的凭证和权限。当您使用更多 App Runner 功能来完成工作时，您可能需要额外权限。了解如何管理访问权限可帮助您向管理员请求适合的权限。如果您无法访问 App Runner 中的功能，请参阅排查应用运行者身份和权限的问题。

服务管理员— 如果您在公司负责管理 App Runner 资源，则您可能具有的完全访问权限。您有责任确定您的员工应访问哪些 App Runner 功能和资源。然后，您必须向 IAM 管理员提交请求以更改服务用户的权限。请查看该页面上的信息以了解 IAM 的基本概念。要了解有关您的公司如何将 IAM 与 App Runner 搭配使用的更多信息，请参阅应用程序运行者如何与 IAM 配合使用。

IAM 管理员-如果您是 IAM 管理员，您可能希望了解有关您可以如何编写策略以管理对 App Runner 的访问的详细信息。要查看您可在 IAM 中使用的 App Runner 基于身份的策略示例，请参阅。基于身份的策略示例。

使用身份进行身份验证

身份验证是您使用身份凭证登录 AWS 的方法。有关登录的更多信息，请使用AWS Management Console，请参阅登录到AWS Management Console作为 IAM 用户或根用户中的IAM 用户指南。

您必须是身份验证（登录到AWS）作为AWS账户根用户、IAM 用户或代入 IAM 角色。您还可以使用公司的单一登录身份验证方法，甚至使用 Google 或 Facebook 登录。在这些情况下，您的管理员以前

使用 IAM 角色设置了联合身份验证。在您使用来自其他公司的凭证访问 AWS 时，您间接地代入了角色。

要直接登录AWS Management Console中，将您的密码与您的根用户电子邮件地址或 IAM 用户名一起使用。您可以访问AWS以编程方式使用根用户或 IAM 用户访问密钥。AWS提供了开发工具包和命令行工具，以使用您的凭证对您的请求进行加密签名。如果您不使用 AWS 工具，则必须自行对请求签名。使用签名版本 4（用于对入站 API 请求进行验证的协议）完成此操作。有关验证请求的更多信息，请参阅签名版本 4 签名流程中的AWS一般参考。

无论使用何种身份验证方法，您可能还需要提供其他安全信息。例如，AWS 建议您使用多重身份验证 (MFA) 来提高账户的安全性。要了解更多信息，请参阅在中使用多重身份验证 (MFA)AWS中的IAM 用户指南。

AWS 账户根用户

当您首次创建 AWS 账户时，最初使用的是一个对账户中所有 AWS 服务和资源有完全访问权限的单点登录身份。此身份称为AWS账户根用户，可以通过使用您用于创建账户的电子邮件地址和密码登录来访问。强烈建议您不使用根用户执行日常任务，即使是管理任务。相反，请遵循仅使用根用户创建您的第一个 IAM 用户的最佳实践。然后请妥善保存根用户凭证，仅用它们执行少数账户和服务管理任务。

IAM 用户和组

一个IAM 用户是您的AWS账户，它具有某个人员或应用程序的特定权限。IAM 用户可能具有长期凭证，例如用户名和密码或一组访问密钥。要了解如何生成访问密钥，请参阅管理 IAM 用户的访问密钥中的IAM 用户指南。为 IAM 用户生成访问密钥时，请确保查看并安全保存密钥对。您以后无法找回秘密访问密钥，而是必须生成新的访问密钥对。

IAM 组 是一个指定一组 IAM 用户的身份。您不能使用组的身份登录。您可以使用组来一次性为多个用户指定权限。如果有大量用户，使用组可以更轻松地管理用户权限。例如，您可能具有一个名为 IAMAdmins 的组，并为该组授予权限以管理 IAM 资源。

用户与角色不同。用户唯一地与某个人员或应用程序关联，而角色旨在让需要它的任何人代入。用户具有永久的长期凭证，而角色提供临时凭证。要了解更多信息，请参阅何时创建 IAM 用户（而不是角色）中的IAM 用户指南。

IAM 角色

一个IAM 角色是您的AWS具有特定权限的账户。它类似于 IAM 用户，但与特定人员不关联。您可以在AWS Management Console通过切换角色。您可以调用 AWS CLI 或 AWS API 操作或使用自定义 URL 以代入角色。有关使用角色的方法的更多信息，请参阅使用 IAM 角色中的IAM 用户指南。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 临时 IAM 用户权限 – IAM 用户可以代入 IAM 角色，以暂时获得不同的权限以执行特定的任务。
- 联合身份用户访问— 您可以不创建 IAM 用户，而是使用来自AWS Directory Service、企业用户目录或 Web 身份提供商。它们被称为联合身份用户。AWS在通过身份提供商。有关联合身份用户的更多信息，请参阅联合身份用户和角色中的IAM 用户指南。
- 跨账户访问 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信委托人）访问您的账户中的资源。角色是授予跨账户访问权限的主要方式。但是，对于某些 AWS 服务，您可以将策略直接附加到资源（而不是使用角色作为代理）。要了解用于跨账户访问的角色和基于资源的策略之间的差别，请参阅IAM 角色与基于资源的策略有何不同中的IAM 用户指南。
- 跨服务访问— 一些AWS服务使用其他AWS服务。例如，当您在某个服务中进行调用时，该服务通常会在 Amazon EC2 中运行应用程序或在 Amazon S3 中存储对象。服务可能会使用发出调用的委托人的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 委托人权限— 当您使用 IAM 用户或角色在中执行操作时AWS，您将被视为一个委托人。策略向委托人授予权限。使用某些服务时，您可能会执行一个操作，此操作然后在不同服务中触发另一个操作。在这种情况下，您必须具有执行这两个操作的权限。要查看某个操作是否需要策略中的其他相关操作，请参阅。的操作、资源和条件键AWS App Runner中的服务授权参考。
  - 服务角色 – 服务角色是服务代表您在您的账户中执行操作而担任的 IAM 角色。服务角色只在您的账户内提供访问权限，不能用于为访问其他账户中的服务授权。IAM 管理员可以在 IAM 中创建、修改和删除服务角色。有关更多信息，请参阅 。创建向AWS服务中的IAM 用户指南。
  - 服务相关角色— 服务相关角色是一种服务角色，它与AWS服务。服务可以代入代表您执行操作的角色。服务相关角色显示在您的 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 Amazon EC2 上运行的应用程序— 您可以使用 IAM 角色管理在 EC2 实例上运行的应用程序的临时凭证，并使用AWS CLI或者AWSAPI 请求。这优先于在 EC2 实例中存储访问密钥。要将 AWS 角色分配给 EC2 实例并使其对该实例的所有应用程序可用，您可以创建一个附加到实例的实例配置文件。实例配置文件包含角色，并使 EC2 实例上运行的程序能够获得临时凭证。有关更多信息，请参阅 。使用 IAM 角色向在 Amazon EC2 实例上运行的应用程序授予权限中的IAM 用户指南。

要了解使用 IAM 角色还是 IAM 用户，请参阅何时创建 IAM 角色（而不是用户）中的IAM 用户指南。

使用策略管理访问

您可以在中控制访问AWS创建策略并将其附加到 IAM 身份或AWS资源的费用。策略是 AWS中的对象；在与标识或资源相关联时，策略定义它们的权限。您可以通过 root 用户或 IAM 用户身份登录，也可以代入 IAM 角色。当您随后提出请求时，AWS评估相关的基于身份或基于资源的策略。策略中的权

限确定是允许还是拒绝请求。大多数策略在 AWS 中存储为 JSON 文档。有关 JSON 策略文档的结构和内容的更多信息，请参阅。JSON 策略概述中的IAM 用户指南。

管理员可以使用AWSJSON 策略来指定谁可以访问哪些内容。也就是说，委托人可以执行操作关于什么resources，并根据条件。

每个 IAM 实体（用户或角色）最初没有任何权限。换言之，默认情况下，用户什么都不能做，甚至不能更改他们自己的密码。要为用户授予执行某些操作的权限，管理员必须将权限策略附加到用户。或者，管理员可以将用户添加到具有预期权限的组中。当管理员为某个组授予访问权限时，该组内的全部用户都会获得这些访问权限。

IAM 策略定义操作的权限，无关乎您使用哪种方法执行操作。例如，假设您有一个允许 `iam:GetRole` 操作的策略。具有该策略的用户可以从 AWS Management Console、AWS CLI 或 AWS API 获取角色信息。

基于身份的策略

基于身份的策略是可附加到身份（如 IAM 用户、用户组或角色）的 JSON 权限策略文档。这些策略控制用户和角色可在何种条件下对哪些资源执行哪些操作。要了解如何创建基于身份的策略，请参阅创建 IAM 策略中的IAM 用户指南。

基于身份的策略可以进一步归类为内联策略或托管策略。内联策略直接嵌入单个用户、组或角色中。托管策略是可以附加到 AWS 账户中的多个用户、组和角色的独立策略。托管策略包括 AWS 托管策略和客户托管策略。要了解如何在托管策略或内联策略之间选择，请参阅在托管策略与内联策略之间进行选择中的IAM 用户指南。

基于资源的策略

基于资源的策略是附加到资源的 JSON 策略文档。基于资源的策略的示例包括 IAM 角色信任策略 和 Amazon S3 存储桶策略。在支持基于资源的策略的服务中，服务管理员可以使用它们来控制对特定资源的访问。对于在其中附加策略的资源，策略定义指定委托人可以对该资源执行哪些操作以及在什么条件下执行。您必须在基于资源的策略中指定委托人。委托要可以包括帐户、用户、角色、联合身份用户或或AWS服务。

基于资源的策略是位于该服务中的内联策略。您不能使用AWS基于资源的策略中的托管策略。

访问控制列表 (ACL)

访问控制列表 (ACL) 控制哪些委托人（账户成员、用户或角色）有权访问资源。ACL 与基于资源的策略类似，尽管它们不使用 JSON 策略文档格式。

Amazon S3AWS WAF和 Amazon VPC 是支持 ACL 的服务示例。要了解有关 ACL 的更多信息，请参阅访问控制列表 (ACL) 概述中的Amazon Simple Storage Service 开发人员指南。

其他策略类型

AWS 支持额外的、不太常用的策略类型。这些策略类型可以设置更常用的策略类型向您授予的最大权限。

- 权限边界 – 权限边界是一个高级功能，用于设置基于身份的策略可以为 IAM 实体（IAM 用户或角色）授予的最大权限。您可为实体设置权限边界。这些结果权限是实体的基于身份的策略及其权限边界的交集。在 `Principal` 中指定用户或角色的基于资源的策略不受权限边界限制。任一项策略中的显式拒绝将覆盖允许。有关权限边界的更多信息，请参阅IAM 实体的权限边界中的IAM 用户指南。
- 服务控制策略 (SCP)— SCP 是指定在中的组织或组织单元 (OU) 的最大权限的 JSON 策略AWS Organizations。AWS Organizations是用于分组和集中管理多个AWS您的企业拥有的账户。如果在组织内启用了所有功能，则可对任意或全部账户应用服务控制策略 (SCP)。SCP 限制成员账户中实体的权限，包括每个AWS账户根用户。有关 Organizations 和 SCP 的更多信息，请参阅SCP 的工作方式中的AWS Organizations用户指南。
- 会话策略 – 会话策略是当您以编程方式为角色或联合身份用户创建临时会话时作为参数传递的高级策略。结果会话的权限是用户或角色的基于身份的策略和会话策略的交集。权限也可以来自基于资源的策略。任一项策略中的显式拒绝将覆盖允许。有关更多信息，请参阅 。会话策略中的IAM 用户指南。

多个策略类型

当多个类型的策略应用于一个请求时，生成的权限更加复杂和难以理解。要了解如何操作AWS确定在涉及多种策略类型时是否允许请求，请参阅策略评估逻辑中的IAM 用户指南。

应用程序运行者如何与 IAM 配合使用

在使用 IAM 管理对的访问AWS App Runner，您应该了解哪些 IAM 功能可用于应用程序运行程序。要获取 App Runner 和其他AWS服务与 IAM 配合使用，请参阅AWS使用的服务中的IAM 用户指南。

有关其他应用程序运行者安全主题，请参阅安全性。

主题

- 基于身份的策略
- 基于资源的策略
- 根据应用程序运行者标签进行授权

- 应用程序运行者用户权限
- 应用程序 IAM 角色

基于身份的策略

通过使用 IAM 基于身份的策略，您可以指定允许或拒绝的操作和资源以及允许或拒绝操作的条件。App Runner 支持特定的操作、资源和条件键。要了解在 JSON 策略中使用的所有元素，请参阅。IAM JSON 策略元素参考中的IAM 用户指南。

Actions

管理员可以使用AWSJSON 策略来指定谁可以访问哪些内容。也就是说，委托人可以执行操作关于什么resources，并根据条件。

JSON 策略的 `Action` 元素描述可用于在策略中允许或拒绝访问的操作。策略操作通常与关联的 AWS API 操作同名。有一些例外情况，例如没有匹配 API 操作的仅限权限 操作。还有一些操作要求在策略中执行多个操作。这些附加操作称为相关操作。

在策略中包含操作以授予执行相关操作的权限。

App Runner 中的策略操作在操作前使用以下前缀：`apprunner:`。 例如，要授予某人使用 Amazon EC2 运行 Amazon EC2 实例的权限`RunInstances`API 操作时，您可以包含`ec2:RunInstances`行动在他们的政策。策略语句必须包含 `Action` 或 `NotAction` 元素。App Runner 定义了一组自己的操作，以描述您可以使用该服务执行的任务。

要在单个语句中指定多项操作，请使用逗号将它们隔开，如下所示：

```
"Action": [
   "apprunner:CreateService",
   "apprunner:CreateConnection"
]
```

您也可以使用通配符 (*) 指定多个操作。例如，要指定以单词 `Describe` 开头的所有操作，请包括以下操作：

```
"Action": "apprunner:Describe*"
```

要查看应用程序运行者操作的列表，请参阅由 定义的操作AWS App Runner中的服务授权参考。

Resources

管理员可以使用AWSJSON 策略来指定谁可以访问哪些内容。也就是说，委托人可以执行操作关于什么resources，并根据条件。

Resource JSON 策略元素指定要向其应用操作的一个或多个对象。语句必须包含 Resource 或 NotResource 元素。作为最佳实践，请使用其 Amazon 资源名称 (ARN)指定资源。对于支持特定资源类型（称为资源级权限）的操作，您可以执行此操作。

对于不支持资源级权限的操作（如列出操作），请使用通配符 (*) 指示语句应用于所有资源。

```
"Resource": "*"
```

应用程序运行程序资源具有以下 ARN 结构：

```
arn:aws:apprunner:region:account-id:resource-type/resource-name[/resource-id]
```

有关 ARN 格式的更多信息，请参阅Amazon 资源名称 (ARN) 和AWS服务命名空间中的AWS一般参考。

例如，要指定my-service服务时，请使用以下 ARN：

```
"Resource": "arn:aws:apprunner:us-east-1:123456789012:service/my-service"
```

要指定属于特定账户的所有服务，请使用通配符 (*)：

```
"Resource": "arn:aws:apprunner:us-east-1:123456789012:service/*"
```

无法对特定资源执行某些 App Runner 操作，例如，用于创建资源的操作。在这些情况下，您必须使用通配符 (*)。

```
"Resource": "*"
```

要查看 App Runner 资源类型及其 ARN 的列表，请参阅由 定义的资源AWS App Runner中的服务授权参考。要了解您可以使用哪些操作指定每个资源的 ARN，请参阅。由 定义的操作AWS App Runner。

条件键

管理员可以使用AWSJSON 策略来指定谁可以访问哪些内容。也就是说，委托人可以执行操作关于什么resources，并根据条件。

在 Condition 元素（或 Condition 块）中，可以指定语句生效的条件。Condition 元素是可选的。您可以创建使用条件运算符（例如，等于或小于）的条件表达式，以使策略中的条件与请求中的值相匹配。

如果您在一个语句中指定多个 Condition 元素，或在单个 Condition 元素中指定多个键，则 AWS 使用逻辑 AND 运算评估它们。如果您为单个条件键指定多个值，则 AWS 使用逻辑 OR 运算来评估条件。在授予语句的权限之前必须满足所有的条件。

在指定条件时，您也可以使用占位符变量。例如，只有在使用 IAM 用户名标记 IAM 用户时，您才能为其授予访问资源的权限。有关更多信息，请参阅 。IAM 策略元素：变量和标签中的IAM 用户指南。

AWS支持全局条件键和特定于服务的条件键。要查看所有AWS全局条件键，请参阅AWS全局条件上下文键中的IAM 用户指南。

App Runner 支持使用某些全局条件键。要查看所有AWS全局条件键，请参阅AWS全局条件上下文键中的IAM 用户指南。

App Runner 定义了一组特定于服务的条件键。此外，App Runner 还支持基于标签的访问控制，这是使用条件键实现的。有关详细信息，请参阅 the section called “根据应用程序运行者标签进行授权”。

要查看 App Runner 条件键的列表，请参阅的条件键AWS App Runner中的服务授权参考。要了解您可以对哪些操作和资源使用条件键，请参阅。由 定义的操作AWS App Runner。

Examples

要查看 App Runner 基于身份的策略示例，请参阅基于身份的策略示例。

基于资源的策略

应用 Runner 不支持基于资源的策略。

根据应用程序运行者标签进行授权

您可以将标签附加到 App Runner 资源或将请求中的标签传递给 App Runner。要基于标签控制访问，您需要使用 `apprunner:ResourceTag/key-name`、`aws:RequestTag/key-name` 或 `aws:TagKeys` 条件键在策略的条件元素中提供标签信息。有关标记 App Runner 资源的更多信息，请参阅the section called “配置”。

要查看基于身份的策略（用于根据资源上的标签来限制对该资源的访问）的示例，请参阅基于标签控制对 App Runner 服务的访问。

应用程序运行者用户权限

要使用应用程序运行程序，IAM 用户需要对应用程序运行程序操作的权限。向用户授予权限的常见方法是将策略附加到 IAM 用户或组。有关管理用户权限的更多信息，请参阅更改 IAM 用户的权限中的IAM 用户指南。

App Runner 提供两个托管策略供您使用。

- `AWSAppRunnerReadOnlyAccess`— 授予列出和查看有关 App Runner 资源的详细信息的权限。
- `AWSAppRunnerFullAccess`— 授予对所有应用程序运行程序操作的权限。

要对用户权限进行更精细的控制，您可以创建自定义策略并将其附加到用户。有关详细信息，请参阅。创建 IAM 策略中的IAM 用户指南。

有关用户策略的示例，请参阅the section called “用户策略”。

AWSAppRunnerReadOnlyAccess

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apprunner:List*",
        "apprunner:Describe*"
      ],
      "Resource": "*"
    }
  ]
}
```

AWSAppRunnerFullAccess

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/apprunner.amazonaws.com/
AWSServiceRoleForAppRunner",
```

```
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "apprunner.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iam:PassedToService": "apprunner.amazonaws.com"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Administrative permission over AppRunner applications",
      "Effect": "Allow",
      "Action": "apprunner:*",
      "Resource": "*"
    }
  ]
}
```

应用程序 IAM 角色

一个IAM 角色是AWS 账户，具有特定权限。

服务相关角色

服务相关角色允许 AWS 服务访问其他服务中的资源以代表您完成操作。服务相关角色显示在您的 IAM 账户中，并归该服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。

应用程序 Runner 支持服务相关角色。有关创建或管理 App Runner 服务相关角色的信息，请参阅。the section called “使用服务相关角色”。

服务角色

此功能允许服务代表您代入服务角色。此角色允许服务访问其他服务中的资源以代表您完成操作。服务角色显示在您的 IAM 账户中，并归该账户所有。这意味着，IAM 管理员可以更改该角色的权限。但是，这样做可能会中断服务的功能。

应用程序运行程序支持几个服务角色。

访问角色

访问角色是 App Runner 用于访问您账户中的 Amazon Elastic Container Registry (Amazon ECR) 中的映像的角色。访问亚马逊 ECR 中的图片是必需的，而且亚马逊 ECR 公共不是必需的。在 Amazon ECR 中基于映像创建服务之前，请使用 IAM 创建服务角色并使用`AWSAppRunnerServicePolicyForECRAccess`托管策略。然后，您可以在调用CreateService中的 APIAuthenticationConfiguration结构（SourceConfiguration参数），或者在使用 App Runner 控制台创建服务时。

AWSAppRunnerServicePolicyForECRAccess

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchCheckLayerAvailability",
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:GetAuthorizationToken"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 如果您为访问角色创建自己的自定义策略，请确保指定`"Resource": "*"`(对于)`ecr:GetAuthorizationToken`action. 令牌可用于访问您可以访问的任何 Amazon ECR 注册表。

创建访问角色时，请务必添加声明 App Runner 服务主体的信任策略`build.apprunner.amazonaws.com`作为可信任的实体。

访问角色的信任策略

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

如果您使用 App Runner 控制台创建服务，则控制台可以自动为您创建访问角色，并为新服务选择该角色。控制台还会列出您帐户中的其他角色，如果您愿意，您可以选择其他角色。

实例角色

实例角色是一个可选角色，App Runner 使用该角色来提供AWS应用程序代码调用的服务操作。在创建 App Runner 服务之前，请使用 IAM 创建具有应用程序代码所需权限的服务角色。然后，您可以将此角色传递给CreateServiceAPI，或者在使用应用程序运行程序控制台创建服务时。

创建实例角色时，请务必添加声明 App Runner 服务主体的信任策略`tasks.apprunner.amazonaws.com`作为可信任的实体。

实例角色的信任策略

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

```
  ]
}
```

如果您使用 App Runner 控制台创建服务，则控制台会列出您帐户中的角色，您可以选择为此目的创建的角色。

有关创建服务的信息，请参阅the section called “Creation”。

> **Note**
>
> 实例角色应提供的权限完全取决于您的应用程序。您的代码可能不会调用任何AWSAPI，在这种情况下，您不需要为 App Runner 提供实例角色。
> 如果你的代码调用AWSAPI，我们无法预测它们是哪些调用。因此，我们不提供托管策略为实例角色。您必须在实例角色中明确包含所需的权限，或者创建自己的自定义策略并在实例角色中使用它。

基于身份的策略示例

默认情况下，IAM 用户和角色没有创建或修改权限AWS App Runner资源的费用。它们还无法使用 AWS Management Console、AWS CLI 或 AWS API 执行任务。IAM 管理员必须创建 IAM 策略，以便为用户和角色授予权限以对所需的指定资源执行特定的 API 操作。然后，管理员必须将这些策略附加到需要这些权限的 IAM 用户或组。

要了解如何使用这些示例 JSON 策略文档创建 IAM 身份策略，请参阅。在“JSON”选项卡上创建策略中的IAM 用户指南。

有关其他应用程序运行者安全主题，请参阅安全性。

主题

- 策略最佳实践
- 用户策略
- 基于标签控制对 App Runner 服务的访问

策略最佳实践

基于身份的策略非常强大。它们确定某个人是否可以创建、访问或删除您账户中的 App Runner 资源。这些操作可能会使 AWS 账户产生成本。创建或编辑基于身份的策略时，请遵循以下准则和建议：

- 入门AWS托管策略— 要快速开始使用应用程序运行程序，请使用AWS托管策略来为您的员工授予所需的权限。这些策略已在您的账户中提供，并由 AWS 维护和更新。有关更多信息，请参阅 。入门AWS托管策略中的IAM 用户指南。
- 授予最低权限 – 创建自定义策略时，仅授予执行任务所需的许可。最开始只授予最低权限，然后根据需要授予其他权限。这样做比起一开始就授予过于宽松的权限而后再尝试收紧权限来说更为安全。有关更多信息，请参阅 。授予最低权限中的IAM 用户指南。
- 为敏感操作启用 MFA – 为了提高安全性，要求 IAM 用户使用多重验证 (MFA) 访问敏感资源或 API 操作。有关更多信息，请参阅 。在中使用多重身份验证 (MFA)AWS中的IAM 用户指南。
- 使用策略条件来增强安全性 – 在切实可行的范围内，定义基于身份的策略在哪些情况下允许访问资源。例如，您可编写条件来指定请求必须来自允许的 IP 地址范围。您也可以编写条件，以便仅允许指定日期或时间范围内的请求，或者要求使用 SSL 或 MFA。有关更多信息，请参阅 。IAM JSON策略元素：Condition中的IAM 用户指南。

用户策略

要访问 App Runner 控制台，IAM 用户必须具有一组最低的权限。这些权限必须允许您列出和查看有关您的 App Runner 资源的详细信息。AWS 账户。如果创建比必需的最低权限更为严格的基于身份的策略，对于附加了该策略的用户，控制台将无法按预期正常运行。

App Runner 提供两个托管策略供您使用。

- `AWSAppRunnerReadOnlyAccess`— 授予列出和查看有关 App Runner 资源的详细信息的权限。
- `AWSAppRunnerFullAccess`— 授予对所有应用程序运行程序操作的权限。

要确保用户可以使用 App Runner 控制台，请至少附加`AWSAppRunnerReadOnlyAccess`托管策略提供给用户。您可以将`AWSAppRunnerFullAccess`托管策略，或者添加特定的其他权限，以允许用户创建、修改和删除资源。有关更多信息，请参阅 。向用户添加权限中的IAM 用户指南。

对于只需要调用 AWS CLI 或 AWS API 的用户，您无需为其提供最低控制台权限。相反，只允许访问与您希望允许用户执行的 API 操作相匹配的操作。

以下示例展示了自定义用户策略。您可以将它们用作定义自定义用户策略的起点。复制示例，或删除操作，缩小资源范围并添加条件。

示例：控制台和连接管理用户策略

此示例策略启用控制台访问，并允许创建和管理连接。它不允许应用程序 Runner 服务创建和管理。它可以附加到负责管理 App Runner 服务对源代码资产的访问权限的用户。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "apprunner:List*",
        "apprunner:Describe*",
        "apprunner:CreateConnection",
        "apprunner:DeleteConnection"
      ],
      "Resource": "*"
    }
  ]
}
```

示例：使用条件密钥的用户策略

本节中的示例演示了依赖于某些资源属性或操作参数的条件权限。

此示例策略允许创建 App Runner 服务，但拒绝使用名为`prod`。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCreateAppRunnerServiceWithNonProdConnections",
      "Effect": "Allow",
      "Action": "apprunner:CreateService",
      "Resource": "*",
      "Condition": {
        "ArnNotLike": {
          "apprunner:ConnectionArn": "arn:aws:apprunner:*:*:connection/prod/*"
        }
      }
    }
  ]
}
```

此示例策略允许更新名为`preprod`仅使用名为`preprod`。

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

```
  "Statement": [
    {
      "Sid": "AllowUpdatePreProdAppRunnerServiceWithPreProdASConfig",
      "Effect": "Allow",
      "Action": "apprunner:UpdateService",
      "Resource": "arn:aws:apprunner:*:*:service/preprod/*",
      "Condition": {
        "ArnLike": {
          "apprunner:AutoScalingConfigurationArn":
 "arn:aws:apprunner:*:*:autoscalingconfiguration/preprod/*"
        }
      }
    }
  ]
}
```

基于标签控制对 App Runner 服务的访问

您可以在基于身份的策略中使用条件，以根据标签控制对 App Runner 资源的访问。此示例显示如何创建允许删除 App Runner 服务的策略。但是，仅当服务标签 `Owner` 具有该用户的用户名的值时，才授予此权限。此策略还授予在控制台上完成此操作的必要权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListServicesInConsole",
      "Effect": "Allow",
      "Action": "apprunner:ListServices",
      "Resource": "*"
    },
    {
      "Sid": "DeleteServiceIfOwner",
      "Effect": "Allow",
      "Action": "apprunner:DeleteService",
      "Resource": "arn:aws:apprunner:*:*:service/*",
      "Condition": {
        "StringEquals": {"apprunner:ResourceTag/Owner": "${aws:username}"}
      }
    }
  ]
}
```

您可以将该策略附加到您账户中的 IAM 用户。如果名为`richard-roe`尝试删除 App Runner 服务时，服务必须标记为标记`Owner=richard-roe`或者`owner=richard-roe`。否则，他会被拒绝访问。条件标签键 `Owner` 匹配 `Owner` 和 `owner`，因为条件键名称不区分大小写。有关更多信息，请参阅 。IAM JSON 策略元素：Condition中的IAM 用户指南。

对应用程序运行者使用服务相关角色

AWS App Runner使用AWS Identity and Access Management(IAM)服务相关角色。服务相关角色是一种独特类型的 IAM 角色，它与 App Runner 直接相关。服务相关角色由 App Runner 预定义，并包含该服务调用其他权限AWS代表您服务。

您可以使用服务相关角色轻松设置 App Runner，因为您不必手动添加所需的权限。App Runner 定义其服务相关角色的权限，除非另外定义，否则只有 App Runner 可以代入该角色。定义的权限包括信任策略和权限策略，以及不能附加到任何其他 IAM 实体的权限策略。

只有在首先删除相关资源后，才能删除服务相关角色。这将保护您的 App Runner 资源，因为您无法无意中删除对资源的访问权限。

有关支持服务相关角色的其他服务的信息，请参阅与 IAM 配合使用的 AWS 服务并查找 Service-Linked 角色列为是的服务。选择 Yes (是)，可转到查看该服务的服务相关角色文档的链接。

有关其他应用程序运行者安全主题，请参阅安全性。

应用程序运行程序的服务相关角色权限

应用程序运行者使用名为的服务相关角色。AWSS 服务管理。

该角色允许应用程序运行员执行以下任务：

- 将日志推送到 Amazon CloudWatch Logs 日志组。
- 创建 Amazon CloudWatch Events 规则以订阅 Amazon Elastic Container Registry (Amazon ECR) 映像推送。

AWSServiceRoleForAPrunner 服务相关角色信任以下服务以代入该角色：

- `apprunner.amazonaws.com`

AWSServiceRoleForElastic Container 服务相关角色的权限策略包含应用程序 Runner 代表您完成操作所需的所有权限。

AppRunnerServiceRolePolicy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:PutRetentionPolicy"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:log-group:/aws/apprunner/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/apprunner/*:log-stream:*"
      ]
    },
    {
      "Sid": "AllowPutRuleForManagedRules",
      "Effect": "Allow",
      "Action": [
        "events: PutRule",
        "events: PutTargets",
        "events: DeleteRule",
        "events: RemoveTargets",
        "events: DisableRule",
        "events: EnableRule"
      ],
      "Resource": "arn:aws:events:*:*:rule/apprunner-*",
      "Condition": {
        "StringEquals": {
          "events:ManagedBy": "apprunner.amazonaws.com",
          "events:source": "aws.ecr",
          "events:detail-type": "ECR Image Action"
        }
      }
    }
```

```
  ]
}
```

您必须配置权限以允许 IAM 实体（例如，用户、组或角色）创建、编辑或删除服务相关角色。有关更多信息，请参阅 IAM 用户指南中的服务相关角色权限。

为应用程序运行者创建服务相关角色

您无需手动创建服务相关角色。当您在AWS Management Console，AWS CLI，或AWSAPI 中，应用 Runner 将为您创建服务相关角色。

如果您删除了此服务相关角色然后需要再次创建它，则可以使用相同的流程在您的账户中重新创建此角色。当您创建 App Runner 服务时，App Runner 将再次为您创建服务相关角色。

编辑应用程序运行者的服务相关角色

应用程序运行程序不允许您编辑 AWSServiceRoleForAPP 服务相关角色。创建服务相关角色后，您将无法更改角色的名称，因为可能有多种实体引用该角色。不过，您可以使用 IAM 编辑角色的说明。有关更多信息，请参阅 IAM 用户指南中的编辑服务相关角色。

删除应用程序运行者的服务相关角色

如果您不再需要使用某个需要服务相关角色的功能或服务，我们建议您删除该角色。这样您就没有未被主动监控或维护的未使用实体。但是，您必须先清除服务相关角色的资源，然后才能手动删除它。

在 App Runner 中，这意味着删除您账户中的所有 App Runner 服务。要了解有关删除 App Runner 服务的信息，请参阅the section called “Deletion”。

> **Note**
>
> 如果在您试图删除资源时 App Runner 服务正在使用该角色，则删除操作可能会失败。如果发生这种情况，则请等待几分钟后重试。

使用 IAM 手动删除服务相关角色

使用 IAM 控制台，AWS CLI，或AWSAPI 删除 AWS 服务相关角色。有关更多信息，请参阅 IAM 用户指南中的删除服务相关角色。

App Runner 服务相关角色支持的受支持区域

App Runner 支持在服务可用的所有区域中使用服务相关角色。有关更多信息，请参阅 。AWS App Runner终端节点和配额中的AWS一般参考。

适用于 AWS App Runner 的 AWS 托管策略

要将权限添加到用户、组和角色，可以更容易地使用AWS托管策略而不是自己编写策略。它需要时间和专业知识创建 IAM 客户托管策略，只为您的团队提供所需的权限。要快速开始，您可以使用我们的AWS托管策略。这些策略涵盖了常见的使用案例，可以在AWSaccount. 有关 的更多信息AWS托管策略，请参阅AWS托管策略中的IAM 用户指南。

AWS服务维护和更新AWS托管策略。不能更改AWS托管策略。服务偶尔会将其他权限添加到AWS托管策略来支持新功能。此类更新会影响策略附加到的所有身份（用户、组和角色）。服务最有可能更新AWS托管策略，当启动新功能或新操作可用时。服务不会从AWS托管策略，因此策略更新不会破坏您的现有权限。

此外,AWS支持跨多个服务的托管策略。例如，ReadOnlyAccess AWS托管策略提供对所有AWS服务和资源. 当服务启动新功能时，AWS为新操作和资源添加只读权限。有关工作职能策略的列表和说明，请参阅AWS工作职能的 托管策略中的IAM 用户指南。

应用程序运行者更新到AWS托管策略

查看有关的更新的详细信息AWS托管策略，因为此服务开始跟踪这些更改。有关此页面更改的自动警报，请订阅 “App Runner 文档历史记录” 页面上的 RSS 源。

变更	描述	日期
呼叫程序服务策略— 新的策略	应用程序运行者添加了一个新策略，允许应用程序运行者代表应用程序运行程序服务调用亚马逊 CloudWatAmazon CloudWatch Logs 和亚马逊 CloudWatch 事件。	2021 年 3 月 1 日

变更	描述	日期
AWSP 可读性访问— 新的策略	App Runner 添加了一个新策略，允许用户列出和查看有关 App Runner 资源的详细信息。	2021 年 3 月 1 日
AWSP 完全访问— 新的策略	App Runner 添加了一个新策略，允许用户执行所有 App Runner 操作。	2021 年 3 月 1 日
AWSP 管理服务策略预测访问— 新的策略	App Runner 添加了一个新策略，允许 App Runner 访问您的账户中的 Amazon Elastic Container Registry (Amazon ECR) 映像。	2021 年 3 月 1 日
应用程序运行者开始跟踪更改	应用程序运行者开始跟踪其AWS托管策略。	2021 年 3 月 1 日

排查应用运行者身份和权限的问题

使用以下信息可帮助您诊断和修复在使用时可能遇到的常见问题。AWS App Runner和 IAM。

有关其他应用程序运行者安全主题，请参阅安全性。

主题

- 我无权在 App Runner 中执行操作
- 我是管理员并希望允许其他人访问 App Runner
- 我想要允许我之外的用户AWS 账户访问我的应用程序运行者资源

我无权在 App Runner 中执行操作

如果 AWS Management Console 告诉您，您无权执行某个操作，请联系您的管理员寻求帮助。您的管理员是指为您提供AWS用户名和密码。

当名为的 IAM 用户时，会发生以下示例错误`marymajor`尝试使用控制台查看有关 App Runner 服务的详细信息，但不具有`apprunner:DescribeService`权限。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 apprunner:DescribeService on resource: my-example-service
```

在这种情况下，Mary 请求她的管理员来更新其策略，以允许她访问*my-example-service*资源使用apprunner:DescribeServiceaction.

我是管理员并希望允许其他人访问 App Runner

要允许其他人访问 App Runner，您必须为需要访问权限的人员或应用程序创建一个 IAM 实体（用户或角色）。他们（它们）将使用该实体的凭证访问 AWS。然后，您必须将策略附加到实体，以便在 App Runner 中为他们（它们）授予正确的权限。

要立即开始，请参阅创建您的第一个 IAM 委派用户和组中的IAM 用户指南。

我想要允许我之外的用户AWS 账户访问我的应用程序运行者资源

您可以创建一个角色，以便其他账户中的用户或您组织外的人员可以使用该角色来访问您的资源。您可以指定谁值得信赖，可以代入角色。对于支持基于资源的策略或访问控制列表 (ACL) 的服务，您可以使用这些策略向人员授予对您的资源的访问权。

要了解更多信息，请参阅以下内容：

- 要了解 App Runner 是否支持这些功能，请参阅应用程序运行者如何与 IAM 配合使用。
- 要了解如何提供对资源的访问AWS您拥有的帐户，请参阅向另一个 IAM 用户提供访问权限AWS您拥有的账户中的IAM 用户指南。
- 了解如何向第三方提供对资源的访问AWS帐户，请参阅提供对AWS第三方拥有的账户中的IAM 用户指南。
- 要了解如何通过身份联合提供访问权限，请参阅为经过外部身份验证的用户（联合身份验证）提供访问权限中的IAM 用户指南。
- 要了解使用角色和基于资源的策略进行跨账户访问之间的差别，请参阅IAM 角色与基于资源的策略有何不同中的IAM 用户指南。

中的日志记录和监控

监控是保持可靠性、可用性和性能的重要环节。AWS App Runner服务。从您的所有部分收集监控数据AWS解决方案可以更加轻松地调试故障（如果发生故障）。应用程序亚军集成了几个AWS工具来监控 App Runner 服务并响应潜在事件。

Amazon CloudWatch 警报

借助 Amazon CloudWatch 警报，您可以在指定的时间段内观看服务指标。如果指标在给定时间段数内超出给定阈值，您会收到通知。

App Runner 收集有关整个服务以及运行 Web 服务的实例（扩展单位）的各种指标。有关更多信息，请参阅指标（CloudWatch）。

应用程序日志

应用程序运行程序会收集您的应用程序代码的输出，并将其流式传输到 Amazon CloudWatch Logs。这个输出中的内容取决于你。例如，您可以包括对 Web 服务发出的请求的详细记录。这些日志记录可能在安全和访问权限审核方面十分有用。有关更多信息，请参阅日志（CloudWatch Logs）。

AWS CloudTrailAction Log

应用程序运行程序与AWS CloudTrail，提供用户、角色或AWS服务。CloudTrail 将应用程序运行者的所有 API 调用作为事件捕获。您可以在 CloudTrail 控制台中查看最近的事件，也可以创建跟踪，以便将 CloudTrail 事件持续传递到 Amazon Simple Storage Service (Amazon S3) 存储桶。有关更多信息，请参阅API 操作（CloudTrail）。

的应用运行者的合规性验证

作为多个 AWS 合规性计划的一部分，第三方审核员将评估 AWS App Runner 的安全性和合规性。其中包括 SOC、PCI、FedRAMP、HIPAA 及其他。

要了解应用程序运行者或其他AWS服务在特定的合规性计划范围内，请参阅AWS合规性计划范围内的服务。有关常规信息，请参阅 AWS 合规性计划。

您可以使用 AWS Artifact 下载第三方审计报告。有关更多信息，请参阅下载 AWS Artifact 中的报告。

您在使用 AWS 服务时的合规性责任由您数据的敏感性、贵公司的合规性目标以及适用的法律法规决定。AWS 提供以下资源来帮助满足合规性：

- 安全性与合规性快速入门指南— 这些部署指南讨论了架构注意事项，并提供了在AWS以安全性和法规遵从性为中心。
- HIPAA 安全性与合规性架构设计白皮书— 本白皮书介绍了公司如何使用AWS来创建符合 HIPAA 要求的应用程序。

> **Note**
>
> 并非所有服务都符合 HIPAA 的要求。

- AWS合规性资源— 此业务手册和指南集合可能适用于您的行业和位置。

- 使用规则评估资源中的AWS Config开发人员指南—AWS Config服务评估您的资源配置对内部实践、行业指南和法规的遵循情况。
- AWS Security Hub— 这AWS服务中安全状态的全面视图AWS，可帮助您检查是否符合安全行业标准和最佳实践。
- AWS Audit Manager— 这AWS服务可帮助您持续审核AWS使用，以简化您管理风险的方式，并遵守法规和行业标准。

有关其他应用程序运行者安全主题，请参阅安全性。

应用程序运行者中的弹性

这些区域有：AWS全球基础设施围绕AWS 区域和可用区。AWS 区域提供多个在物理上独立且隔离的可用区，这些可用区通过延迟低、吞吐量高且冗余性高的网络连接在一起。利用可用区，您可以设计和操作在可用区之间无中断地自动实现故障转移的应用程序和数据库。与传统的单个或多个数据中心基础设施相比，可用区具有更高的可用性、容错性和可扩展性。

有关 的更多信息AWS 区域和可用区，请参阅AWS全球基础设施。

AWS App Runner 代表您管理和自动执行 AWS 全球基础架构的使用。使用 App Runner 时，您将从这些可用性和容错机制中受益。AWS优惠。

有关其他应用程序运行者安全主题，请参阅安全性。

AWS App Runner 中的基础设施安全性

作为托管服务，AWS App Runner受AWS全局网络安全过程，请参阅Amazon Web Services：安全过程概述白皮书。

您可以使用AWS发布的 API 调用通过网络访问 App Runner。客户端必须支持传输层安全性 (TLS) 1.0 或更高版本。建议使用 TLS 1.2 或更高版本。客户端还必须支持具有完全向前保密 (PFS) 的密码套件，例如 Ephemeral Diffie-Hellman (DHE) 或 Elliptic Curve Ephemeral Diffie-Hellman (ECDHE)。大多数现代系统（如 Java 7 及更高版本）都支持这些模式。

此外，必须使用访问密钥 ID 和与 IAM 委托人关联的秘密访问密钥来对请求进行签名。或者，您可以使用 AWS Security Token Service (AWS STS) 生成临时安全凭证来对请求进行签名。

有关其他 App Runner 安全主题，请参阅安全性。

应用程序运行中的配置和漏洞分析

AWS 和我们的客户共担实现高水平的软件组件安全性和合规性的责任。有关更多信息，请参阅 AWS 责任共担模式。

有关其他应用 Runner 安全主题，请参阅安全性。

适用于 App Runner 的安全最佳实践

AWS App Runner 提供了您在开发和实施自己的安全策略时需要考虑的多项安全功能。以下最佳实践是一般准则，并不代表完整的安全解决方案。由于这些最佳实践可能不适合您的环境或不满足您的环境要求，因此将其视为有用的考虑因素，而不是惯例。

有关其他应用程序 Runner 安全主题，请参阅安全性。

预防性安全最佳实践

预防性安全控制措施试图在事件发生之前进行预防。

实施最低权限访问

应用程序运行者提供AWS Identity and Access Management的 (IAM) 托管策略IAM 用户和访问角色。这些托管策略指定了正确运行 App Runner 服务可能所需的所有权限。

您的应用程序可能不需要我们托管策略中的全部权限。您可以自定义它们，并仅授予用户和 App Runner 服务执行其任务所需的权限。这与用户策略特别相关，其中不同的用户角色可能具有不同的权限需求。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。

检测性安全最佳实践

侦探性安全控件会在发生安全违规后识别它们。它们可以帮助您发现潜在安全威胁或事件。

实施监控

在维护 App Runner 解决方案的可靠性、安全性、可用性和性能时，监控是一个重要部分。AWS提供了多种工具和服务来帮助您监控AWS服务。

以下是要监视的项目的一些示例：

- 适用于应用程序运行者的 Amazon CloudWatch 指标— 为关键 App Runner 指标和应用程序的自定义指标设置警报。有关详细信息，请参阅 指标（CloudWatch）。

- AWS CloudTrailentries— 跟踪可能影响可用性的操作，例如`PauseService`或者`DeleteConnection`。有关详细信息，请参阅 API 操作（CloudTrail）。

AWS术语表

最新的AWS术语，请参阅AWS术语表中的AWS一般参考。

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。