aws

开发人员指南

AWS App Runner

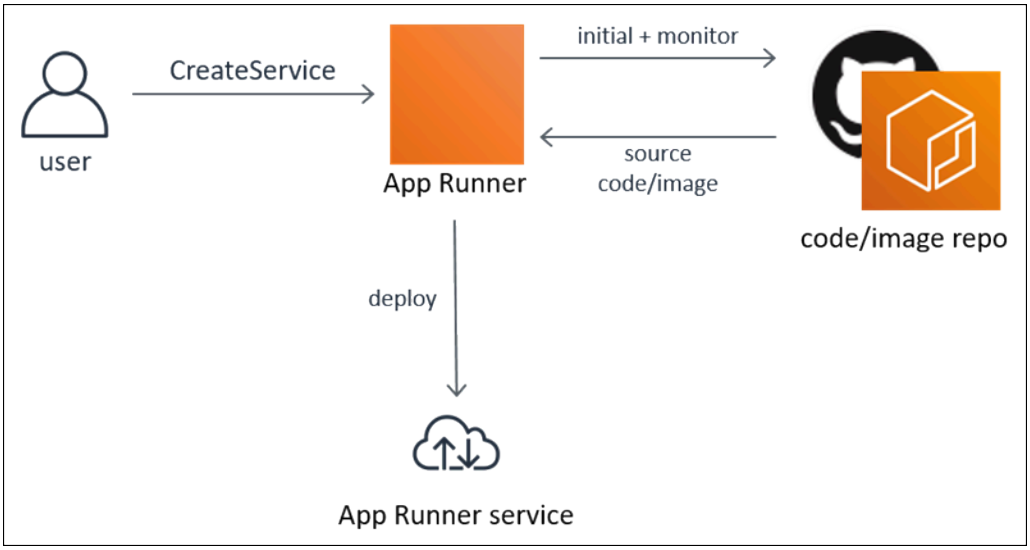
AWS App Runner: 开发人员指南

Table of Contents

什么是 AWS App Runner？

AWS App Runner 是一项 AWS 服务，它提供了一种快速、简单且经济实惠的方式，可将源代码或容器映像直接部署到 AWS 云中可扩展且安全的 Web 应用程序。您无需学习新技术、决定使用哪种计算服务，也不需要知道如何预置和配置 AWS 资源。

App 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。



App Runner 是为谁准备的？

如果您是开发人员，则可以使用 App Runner 来简化部署新版本代码或图像存储库的过程。

对于运营团队，每次将提交推送到代码存储库或将新的容器映像版本推送到镜像存储库时，App Runner 都支持自动部署。

访问应用程序运行器

您可以使用以下任何一个接口来定义和配置 App Runner 服务部署：

- App Runner 控制台 — 提供用于管理 App Runner 服务的 Web 界面。
- App Runner API — 提供用于执行应用程序运行器操作的 RESTful API。有关更多信息，请参阅 AWS App Runner API 参考。
- AWS 命令行界面 (AWS CLI) — 为包括亚马逊 VPC 在内的各种 AWS 服务提供命令，并在 Windows、macOS 和 Linux 上受支持。有关更多信息，请参阅 AWS Command Line Interface。

- AWS 软件开发工具包 — 提供特定语言的 API 并处理许多连接细节，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。

App Runner 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 App Runner 服务消耗的资源付费。当请求流量较低时，您的服务会缩减到更少的计算实例。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 自动缩放的更多信息，请参阅the section called “自动扩缩”。

有关定价信息，请参阅 AWS App Runner 定价。

接下来做什么

在以下主题中学习如何开始使用 App Runner：

- 设置— 完成使用 App Runner 的先决条件。
- 开始使用— 将你的第一个应用程序部署到 App Runner。

为 App Runner 进行设置

如果您是新 AWS 客户，请在开始使用之前完成本页列出的设置先决条件 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，请参阅以下参考材料：

- AWS Identity and Access Management (IAM)
- IAM 用户指南

注册获取 AWS 账户

如果您没有 AWS 账户，请完成以下步骤来创建一个。

报名参加 AWS 账户

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

   在注册时，将接到一通电话，要求使用电话键盘输入一个验证码。

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

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

创建具有管理访问权限的用户

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

保护你的 AWS 账户根用户

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

要获取使用根用户登录方面的帮助，请参阅《AWS 登录 用户指南》中的以根用户身份登录。

2. 为您的根用户启用多重身份验证 (MFA)。

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

创建具有管理访问权限的用户

1. 启用 IAM Identity Center

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的启用 AWS IAM Identity Center。

2. 在 IAM Identity Center 中，为用户授予管理访问权限。

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

以具有管理访问权限的用户身份登录

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

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

将访问权限分配给其他用户

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

   有关说明，请参阅《AWS IAM Identity Center 用户指南》中的创建权限集。

2. 将用户分配到一个组，然后为该组分配单点登录访问权限。

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

授权以编程方式访问

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

要向用户授予编程式访问权限，请选择以下选项之一。

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

接下来做什么

您已完成先决步骤。要将您的第一个应用程序部署到 App Runner，请参阅开始使用。

App Runner 入门

AWS App Runner 是一项 AWS 服务，它提供了一种快速、简单且经济实惠的方式，可将现有的容器镜像或源代码直接转换为中正在运行的 Web 服务 AWS Cloud。

本教程介绍如何使用 AWS App Runner 将应用程序部署到 App Runner 服务。它演示了配置源代码和部署、服务版本和服务运行时。它还展示了如何部署代码版本、更改配置和查看日志。最后，本教程展示了如何清理您在遵循本教程过程时创建的资源。

主题

先决条件

在开始本教程之前，请务必执行以下操作：

1. 完成中的设置步骤设置。
2. 决定您是要使用 GitHub 存储库还是 Bitbucket 存储库。
   - 要使用 Bitbucket，请先创建一个 Bitbucket 账户（如果你还没有）。如果你不熟悉 Bitbucket，请参阅 Bitbucket Cl oud 文档中的 Bitbuck et 入门。
   - 要使用 GitHub，请创建一个GitHub帐户（如果您还没有）。如果您不熟悉 GitHub，请参阅GitHub文档 GitHub中的入门。

> **Note**
>
> 您可以通过您的账户创建与多个存储库提供商的连接。因此，如果您想同时从 Bitbucket 存储库 GitHub和 Bitbucket 存储库进行部署，则可以重复此过程。下次通过创建新的 App Runner 服务并为其他存储库提供商创建新的账户连接。

3. 在您的仓库提供商账户中创建仓库。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。

python-hello示例存储库的文件

Example requirements.txt

```
pyramid==2.0
```

Example server.py

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
import os

def hello_world(request):
    name = os.environ.get('NAME')
    if name == None or len(name) == 0:
        name = "world"
    message = "Hello, " + name + "!\n"
    return Response(message)

if __name__ == '__main__':
    port = int(os.environ.get("PORT"))
    with Configurator() as config:
        config.add_route('hello', '/')
        config.add_view(hello_world, route_name='hello')
        app = config.make_wsgi_app()
    server = make_server('0.0.0.0', port, app)
    server.serve_forever()
```

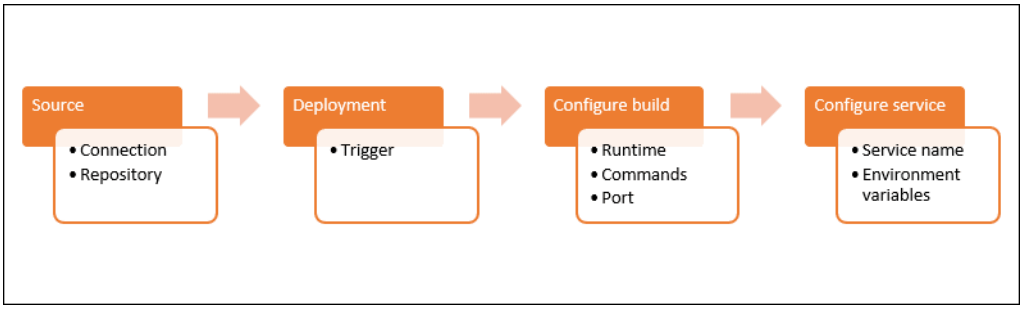
步骤 1：创建 App Runner 服务

在此步骤中，您将基于您在 GitHub 或 Bitbucket 上创建的示例源代码存储库创建 App Runner 服务。the section called “先决条件”该示例包含一个简单的 Python 网站。以下是创建服务的主要步骤：

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：

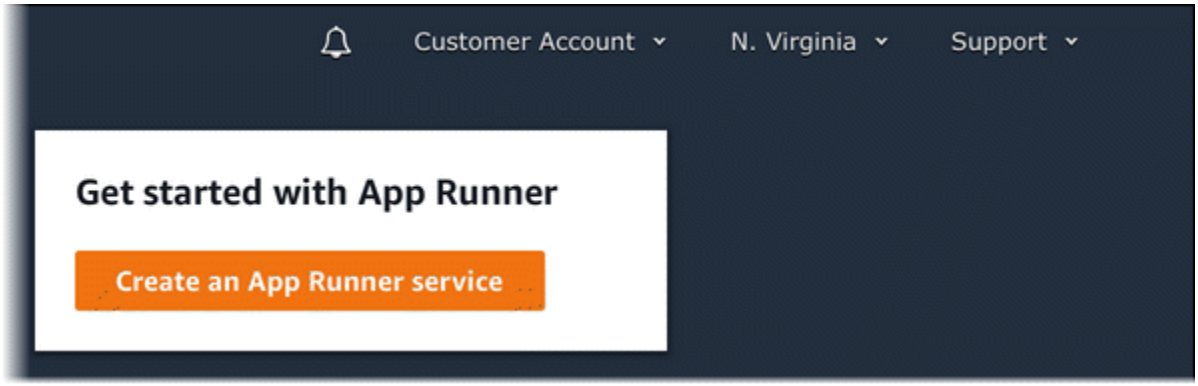


基于源代码存储库创建 App Runner 服务

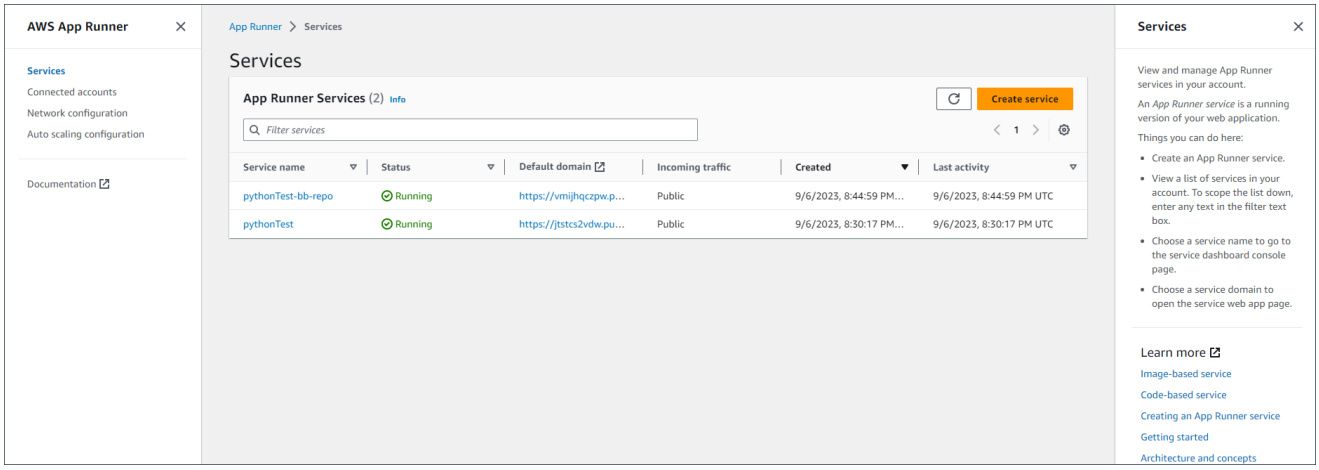
1. 配置您的源代码。

   a. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。

   b. 如果还 AWS 账户 没有任何 App Runner 服务，则会显示主机主页。选择创建 App Runner 服务。



   如果 AWS 账户 已有服务，则会显示包含您的服务列表的 “服务” 页面。选择 Create service。



c. 在源代码和部署页面的源代码部分中，对于存储库类型，选择源代码存储库。

d. 选择提供商类型。选择其中一个GitHub或 Bitbucket。

e. 接下来选择新增。如果出现提示，请提供您的 GitHub 或 Bitbucket 凭证。

f. 根据您之前选择的提供者类型选择下一组步骤。

> **Note**
>
> 以下为 GitHub 账户安装 AWS 连接器的 GitHub 步骤是一次性步骤。您可以重复使用该连接，根据此账户中的存储库创建多个 App Runner 服务。当您已有连接时，请选择该连接并跳至存储库选择。
>
> 这同样适用于您的 Bitbucket 账户的 AWS 连接器。如果您同时使用两者 GitHub 和 Bitbucket 作为 App Runner 服务的源代码存储库，则需要为每个提供商安装一个 AWS 连接器。然后，您可以重复使用每个连接器来创建更多 App Runner 服务。

- 对于 GitHub，请按照以下步骤操作。

  i. 在下一个屏幕上，输入连接名称。

  ii. 如果这是您第一次在 App Runner 中使用 GitHub ，请选择安装另一个。

  iii. 在 “AWS 连接器 GitHub” 对话框中，如果出现提示，请选择您的 GitHub 帐户名。

  iv. 如果系统提示授权 AWS 连接器 GitHub，请选择授权 AWS 连接。

  v. 在 “安装 AWS 连接器 GitHub” 对话框中，选择 “安装”。

  您的账户名称显示为选定的GitHub 账户/组织。现在，您可以在账户中选择存储库。

  vi. 在 “存储库” 中，选择您创建的示例存储库`python-hello`。对于 Bran ch，选择仓库的默认分支名称（例如，main）。

vii. 将源目录保留为默认值。该目录默认为存储库根目录。在前面的 “先决条件” 步骤中，您已将源代码存储在存储库根目录中。

- 对于 Bitbucket，请按照以下步骤操作。

  i. 在下一个屏幕上，输入连接名称。

  ii. 如果这是您第一次在 App Runner 中使用 Bitbucket，请选择安装另一个。

  iii. 在AWS CodeStar 请求访问权限对话框中，您可以选择您的工作空间并授予对 Bitbucket 集成的访问权限。 AWS CodeStar 选择您的工作空间，然后选择 “授予访问权限”。

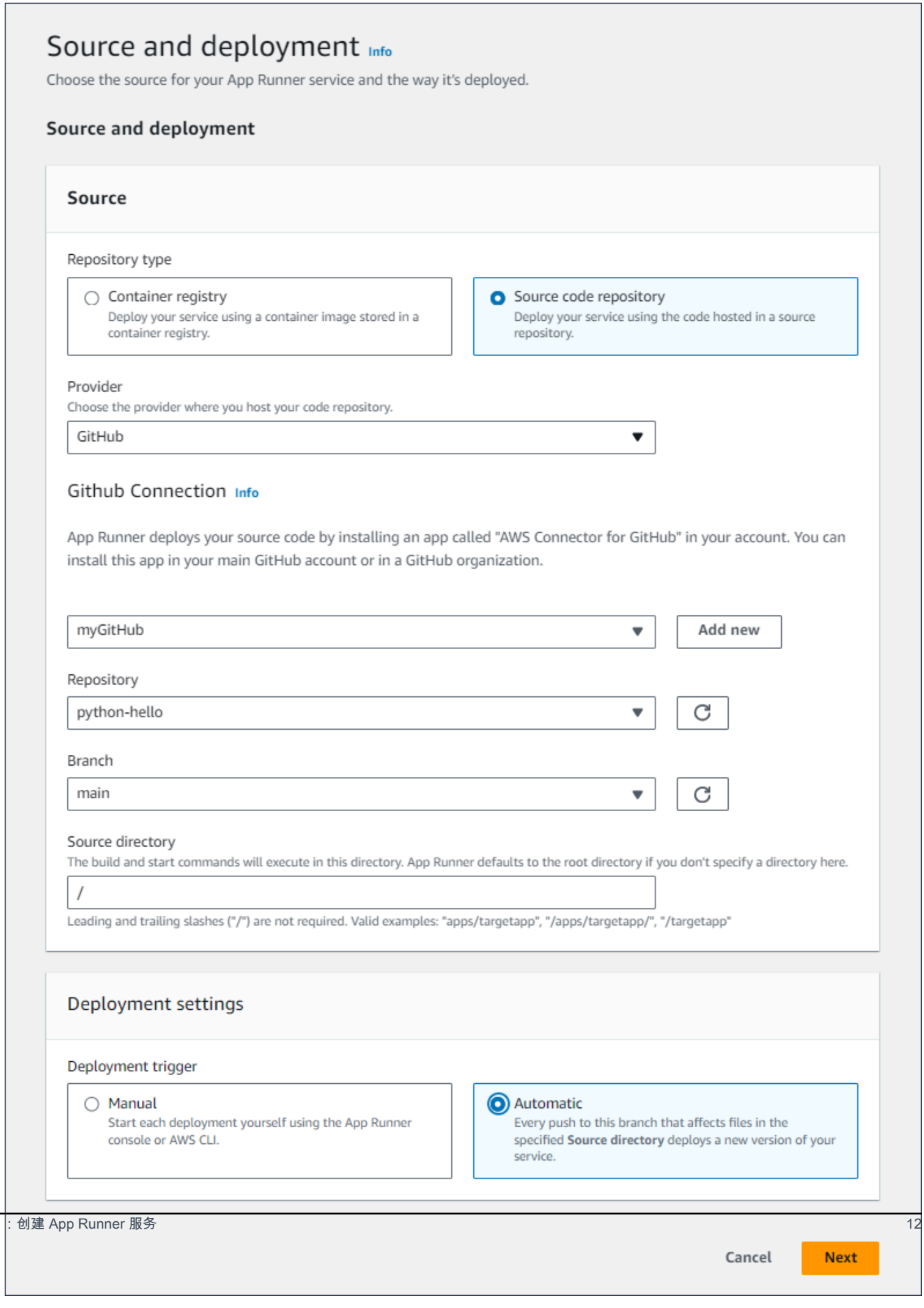
  iv. 接下来，您将被重定向到 AWS 控制台。确认 Bitbucket 应用程序已设置为正确的 Bitbucket 工作空间，然后选择 “下一步”。

  v. 在 “存储库” 中，选择您创建的示例存储库`python-hello`。对于 Bran ch，选择仓库的默认分支名称（例如，main）。

  vi. 将源目录保留为默认值。该目录默认为存储库根目录。在前面的 “先决条件” 步骤中，您已将源代码存储在存储库根目录中。

2. 配置您的部署：在 “部署设置” 部分中，选择 “自动”，然后选择 “下一步”。

> **Note**
>
> 通过自动部署，每次向存储库源目录提交的新内容都会自动部署服务的新版本。


Source and deployment Info
Choose the source for your App Runner service and the way it's deployed.
Source and deployment
Source
Repository type
Container registry
Deploy your service using a container image stored in a container registry.
Source code repository
Deploy your service using the code hosted in a source repository.
Provider
Choose the provider where you host your code repository.
GitHub
Github Connection Info
App Runner deploys your source code by installing an app called "AWS Connector for GitHub" in your account. You can install this app in your main GitHub account or in a GitHub organization.
myGitHub
Add new
Repository
python-hello
Branch
main
Source directory
The build and start commands will execute in this directory. App Runner defaults to the root directory if you don't specify a directory here.
/
Leading and trailing slashes ("/") are not required. Valid examples: "apps/targetapp", "/apps/targetapp/", "/targetapp"
Deployment settings
Deployment trigger
Manual
Start each deployment yourself using the App Runner console or AWS CLI.
Automatic
Every push to this branch that affects files in the specified Source directory deploys a new version of your service.
Cancel
Next

3. 配置应用程序构建。

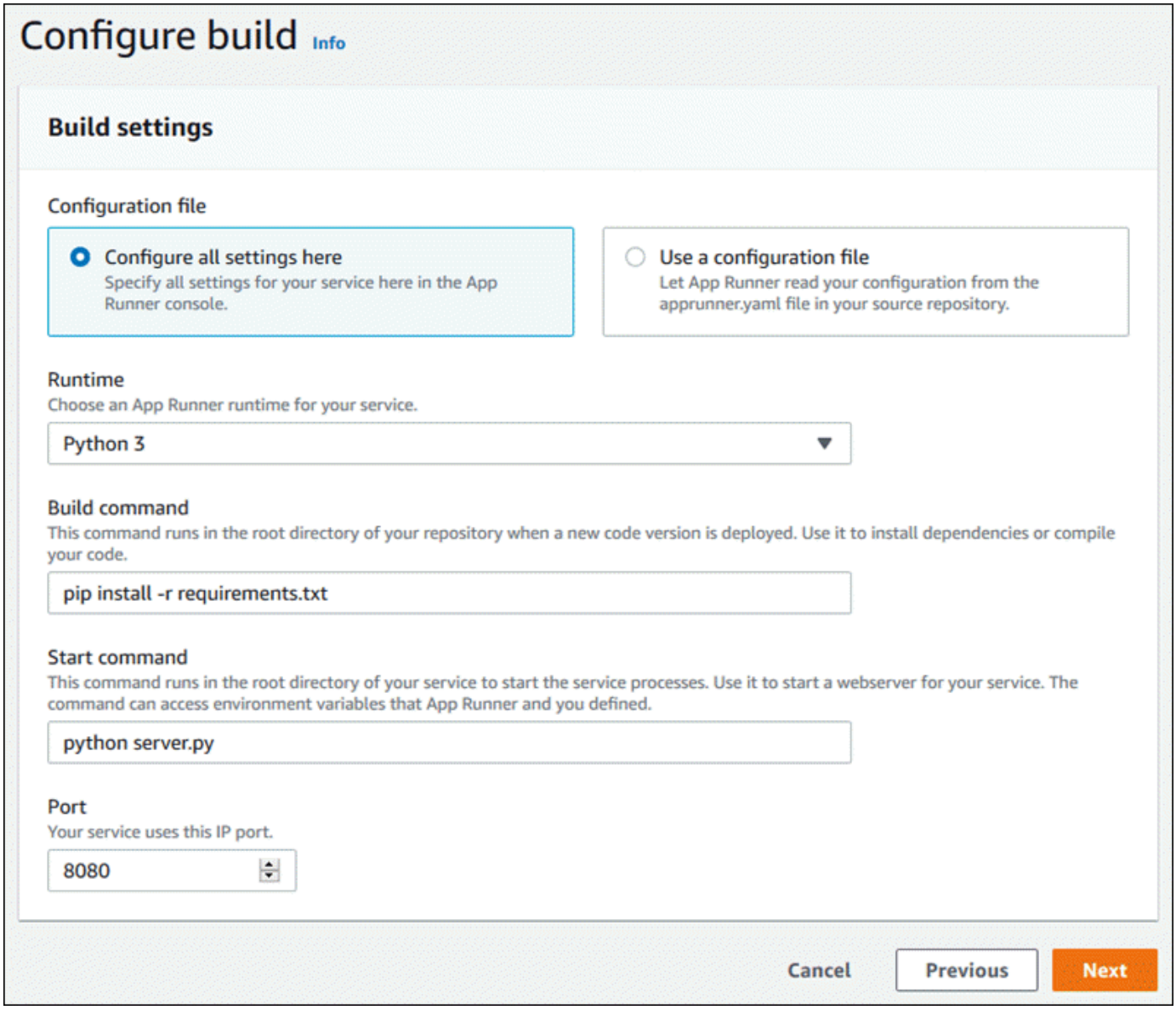
   a. 在 “配置构建” 页面上，对于 “配置文件”，选择 “在此处配置所有设置”。

   b. 提供以下编译设置：

      - 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. 选择下一步。

> **Note**
>
> Python 3 运行时使用基本的 Python 3 镜像和你的示例 Python 代码构建 Docker 镜像。然后，它会启动一项运行该镜像的容器实例的服务。



4. 配置您的服务。

   a. 在配置服务页面的服务设置部分，输入服务名称。

   b. 在 “环境变量” 下，选择 “添加环境变量”。为环境变量提供以下值。

      - 来源-选择纯文本
      - 环境变量名称 — **NAME**
      - 环境变量值-任意名称（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. 选择下一步。

Configure service Info

Service settings

Service name

python-test

Virtual CPU & memory

1 vCPU | 2 GB

Environment variables — *optional* Info

Add environment variables in plain text or reference them from Secrets Manager and SSM Parameter Store. Update IAM Policies using the IAM Policy template given below to securely reference secrets and configurations as environment variables.

No environment variables have been configured.

Add environment variable

You can add up to 50 more items.

▶ IAM policy templates

▶ 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

▶ Networking Info

Configure the way your service communicates with other applications, services, and resources.

▶ Observability

Configure observability tooling.

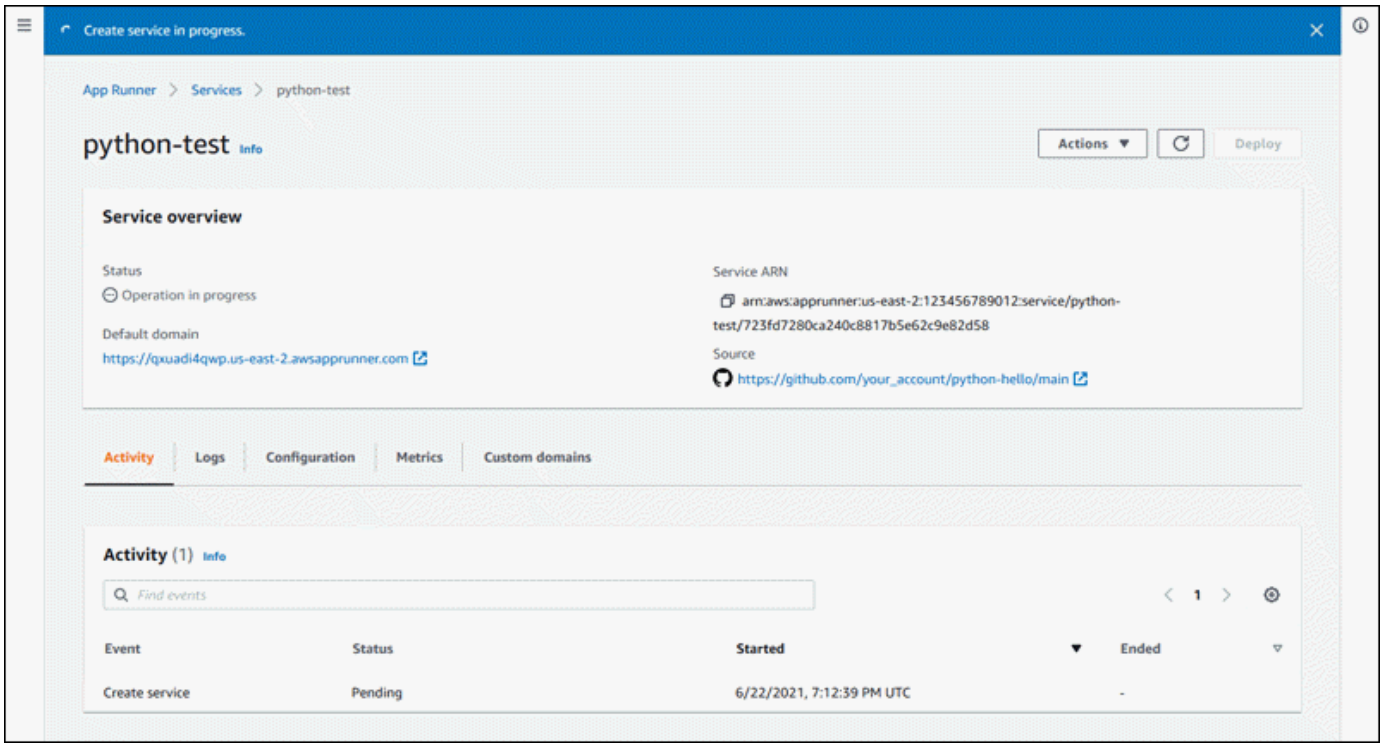
▼ Tags Info

Use tags to search and filter your resources, track your AWS costs, and control access permissions.

Tags — *optional*

5. 在“查看并创建”页面上，验证您输入的所有详细信息，然后选择“创建并部署”。

   如果成功创建了服务，控制台将显示服务控制面板，其中包含新服务的服务概述。



6. 验证您的服务是否正在运行。

   a. 在服务仪表板页面上，等到服务状态变为“正在运行”。

   b. 选择默认域名值，即您的服务网站的网址。

> **Note**
>
> 为了增强 App Runner 应用程序的安全性，*.awsapprunner.com 域已在公共后缀列表 (PSL) 中注册。为了进一步提高安全性，如果您需要在 App Runner 应用程序的默认域名中设置敏感 Cookie，我们建议您使用带__Host-前缀的 Cookie。这将有助于保护您的域，防范跨站点请求伪造 (CSRF) 攻击。要了解更多信息，请参阅 Mozilla 开发者网络中的 Set-Cookie 页面。

   屏幕上会显示一个网页：你好，**####**！

第 2 步：更改您的服务代码

在此步骤中，您将对存储库源目录中的代码进行更改。App Runner CI/CD 功能会自动生成变更并将其部署到您的服务。

更改您的服务代码

1. 导航到您的示例存储库。
2. 编辑名为的文件`server.py`。
3. 在分配给变量的表达式中`message`，将文本更改`Hello`为`Good morning`。
4. 保存您的更改并将其提交到存储库。
5. 以下步骤说明了更改 GitHub 存储库中的服务代码。
    a. 导航到您的示例 GitHub 存储库。
    b. 选择文件名`server.py`以导航到该文件。
    c. 选择 “编辑此文件”（铅笔图标）。
    d. 在分配给变量的表达式中`message`，将文本更改`Hello`为`Good morning`。

e. 选择提交更改。

6. 开始为你的 App Runner 服务部署新的提交。在服务控制面板页面上，服务状态更改为操作进行中。

   等待部署结束。在服务仪表板页面上，服务状态应更改回正在运行。

7. 验证部署是否成功：刷新显示服务网页的浏览器选项卡。

   页面现在显示修改后的消息：早上好，**####**！

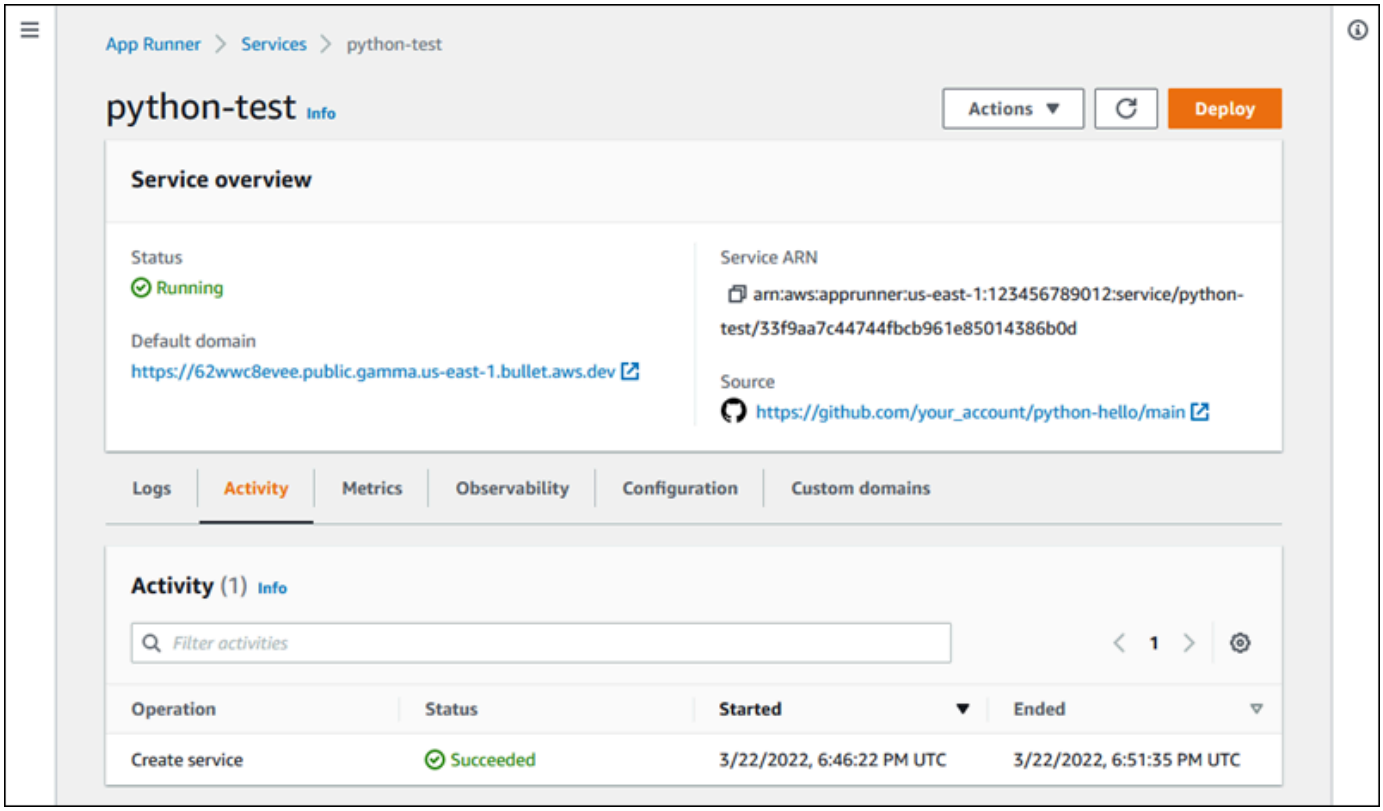
步骤 3：更改配置

在此步骤中，您将对**NAME**环境变量值进行更改，以演示服务配置的更改。

更改环境变量值

1. 打开 App Runner 控制台，然后在 “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 App Runner 服务。

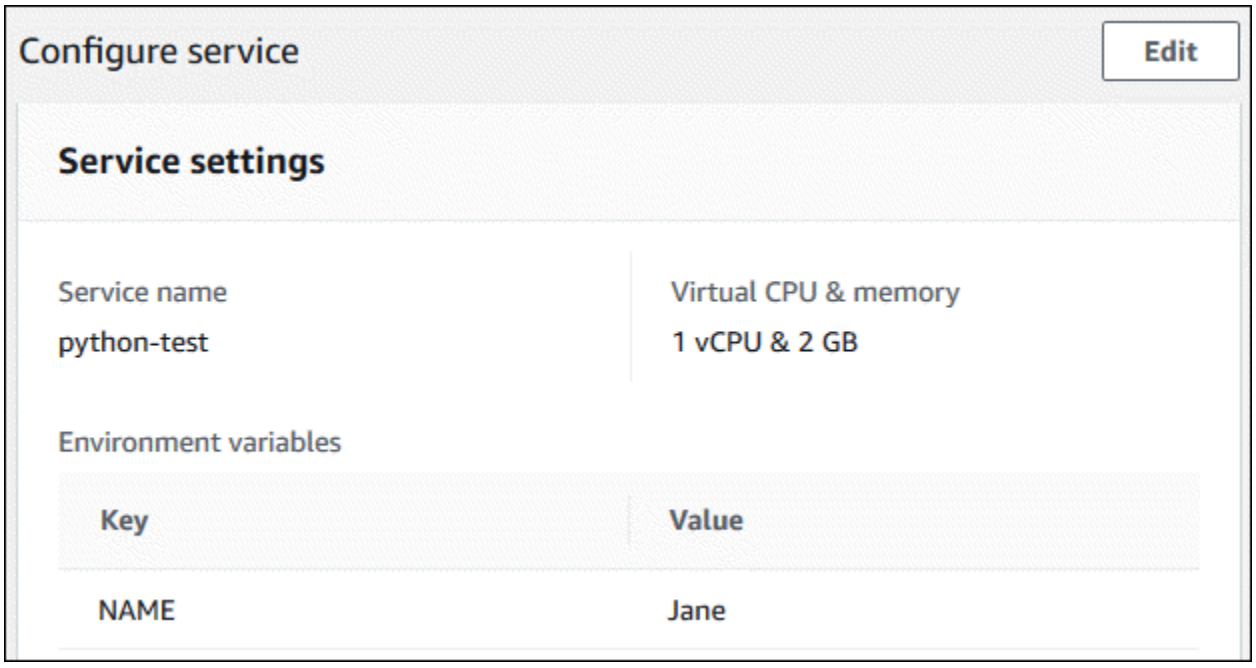
   控制台显示带有服务概述的服务仪表板。



3. 在服务仪表板页面上，选择配置选项卡。

   控制台分几个部分显示您的服务配置设置。

4. 在 “配置服务” 部分中，选择编辑。



5. 对于带有密钥的环境变量**NAME**，请将该值更改为其他名称。

6. 选择 Apply changes。

   App Runner 启动更新过程。在服务控制面板页面上，服务状态更改为操作进行中。

7. 等待更新结束。在服务仪表板页面上，服务状态应更改回正在运行。

8. 验证更新是否成功：刷新显示服务网页的浏览器选项卡。

   页面现在显示修改后的名称：早上好，**###**！

步骤 4：查看服务日志

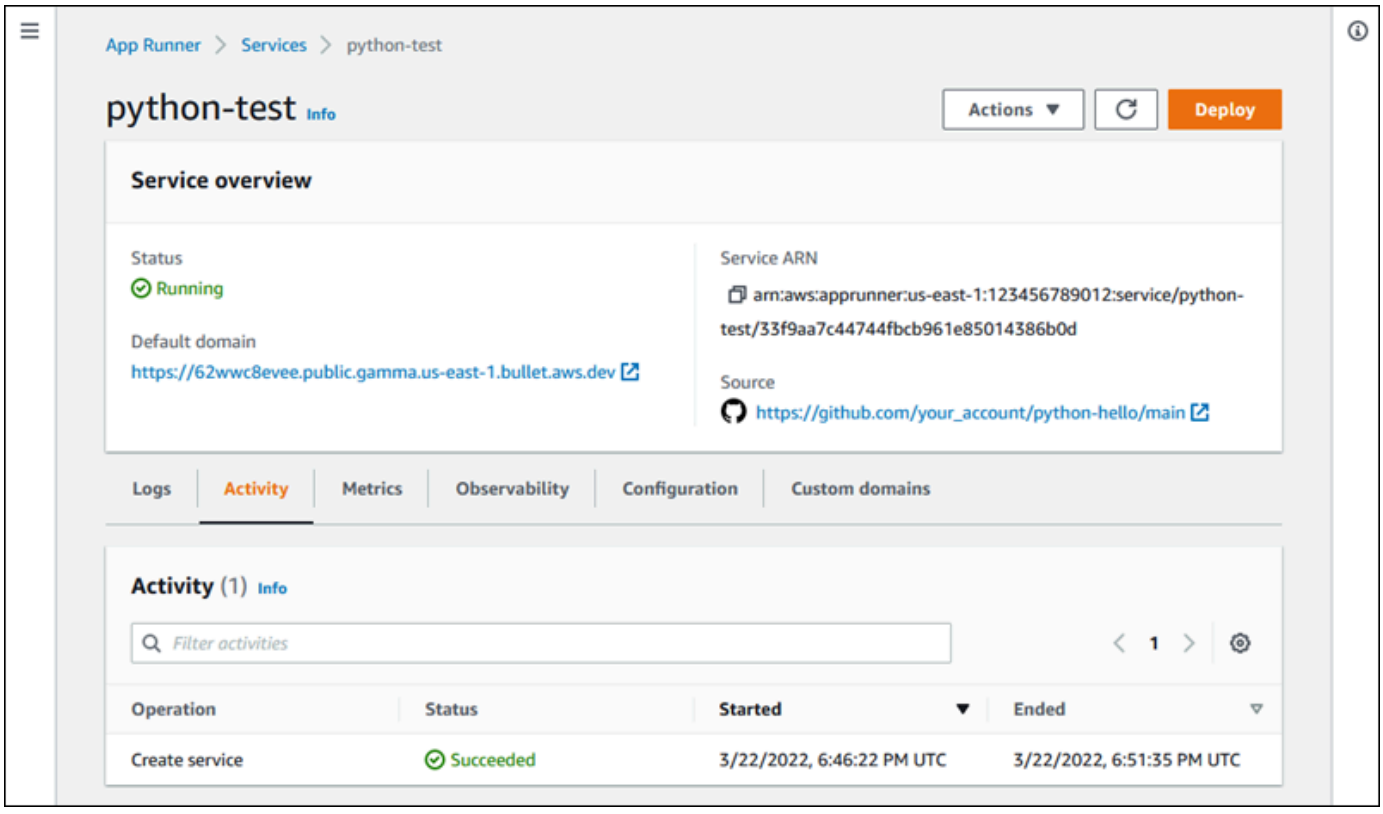
在此步骤中，您将使用 App Runner 控制台来查看 App Runner 服务的日志。App Runner 将日志流式传输到 Amazon CloudWatch CloudWatch 日志（日志），并将其显示在服务的控制面板上。有关 App Runner 日志的信息，请参阅the section called “日志（CloudWatch 日志）”。

查看服务日志

1. 打开 App Runner 控制台，然后在 “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 App Runner 服务。

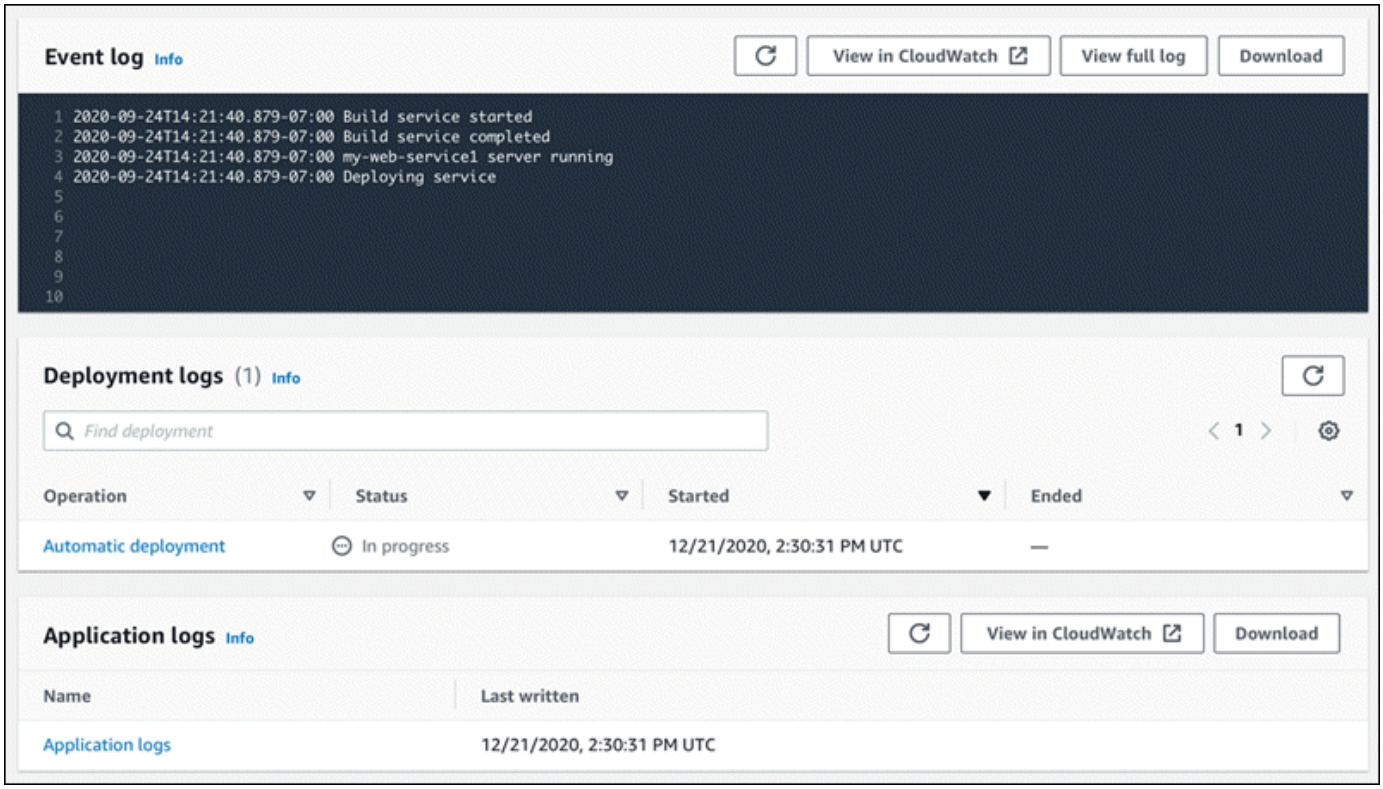
   控制台显示带有服务概述的服务仪表板。



3. 在服务仪表板页面上，选择日志选项卡。

   控制台分几个部分显示几种类型的日志：

   - 事件日志-App Runner 服务生命周期中的活动。控制台显示最新事件。
   - 部署日志-源存储库部署到您的 App Runner 服务。控制台显示每个部署的单独日志流。
   - 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合并到一个日志流中。



4. 要查找特定的部署，请输入搜索词，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. 选择查看 CloudWatch以打开 CloudWatch 控制台，并使用其全部功能浏览您的 App Runner 服务日志。对于部署日志流，请先选择一个日志流。

> **Note**
>
> 如果您想查看特定实例的应用程序日志，而不是合并的应用程序日志，则 CloudWatch 控制台特别有用。

第 5 步：清理

现在，您已经学习了如何创建 App Runner 服务、查看日志和进行一些更改。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. 在服务仪表板页面上，选择操作，然后选择删除服务。
2. 在确认对话框中，输入所需的文本，然后选择删除。

   结果：控制台导航到 “服务” 页面。您刚刚删除的服务显示的状态为 “删除”。不久之后，它从列表中消失了。

还可以考虑删除您在本教程中创建的 GitHub 和 Bitbucket 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，请在以下主题中了解更多信息：

- 架构和概念— 与 App Runner 相关的架构、主要概念和 AWS 资源。
- 基于图像的服务和 基于代码的服务 — App Runner 可以部署的两种类型的应用程序源。
- 为 App Runner 开发— 开发或迁移要部署到 App Runner 的应用程序代码时应了解的事项。
- 应用程序运行器控制台— 使用 App Runner 控制台管理和监控您的服务。
- 管理您的服务— 管理 App Runner 服务的生命周期。
- 可观察性— 通过监控指标、读取日志、处理事件、跟踪服务操作调用以及跟踪 HTTP 调用等应用程序事件，了解您的 App Runner 服务操作。
- App Runner 配置文件— 一种基于配置的方法，用于为 App Runner 服务的构建和运行时行为指定选项。

- 应用程序运行器 API— 使用 App Runner 应用程序编程接口 (API) 创建、读取、更新和删除 App Runner 资源。
- 安全性— 在使用 App Runner 和其他服务时，您可以通过不同的方式来确保云安全。 AWS

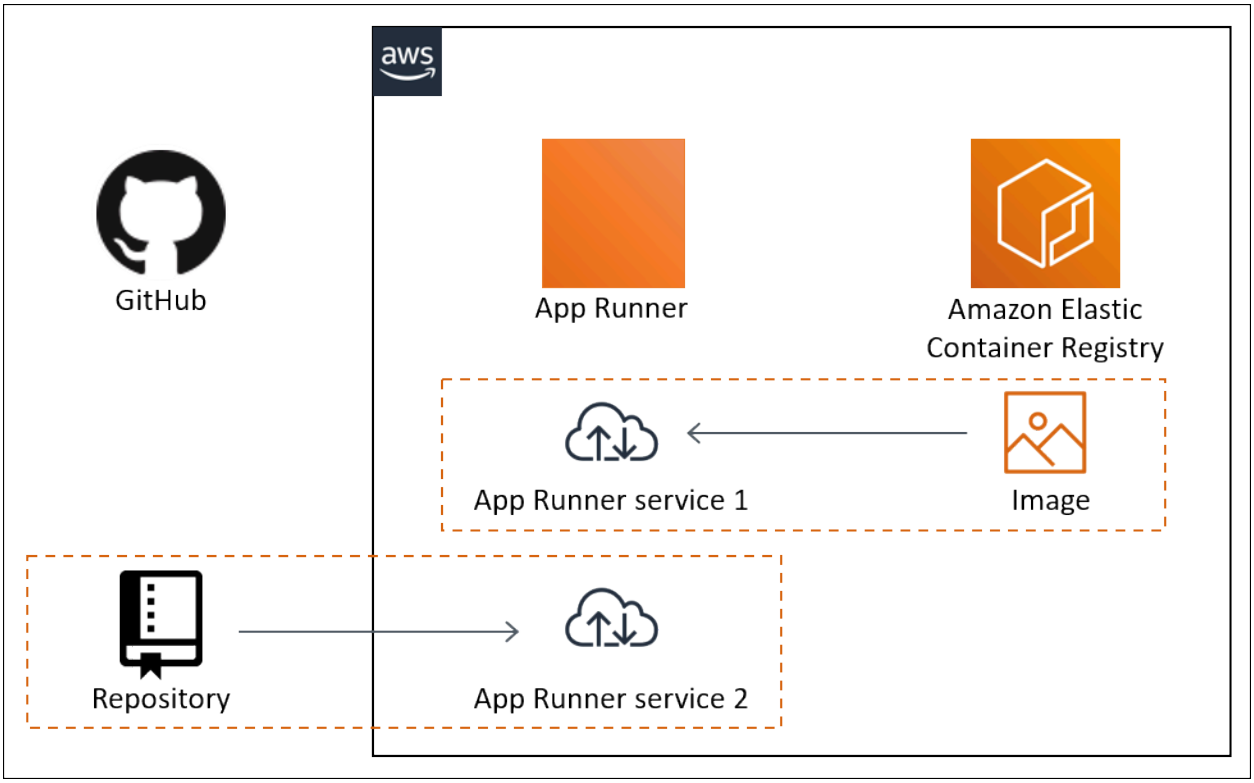
App Runner 架构和概念

AWS App Runner 从存储库中获取源代码或源图像，并在中为您创建和维护正在运行的 Web 服务 AWS Cloud。通常，您只需要调用一个 App Runner 操作即可创建您的服务。CreateService

使用源映像存储库，您可以提供一个 ready-to-use 容器镜像，App Runner 可以部署该镜像来运行您的 Web 服务。使用源代码存储库，您可以提供用于构建和运行 Web 服务的代码和说明，并以特定的运行时环境为目标。App Runner 支持多个编程平台，每个平台都有一个或多个平台主要版本的托管运行时。

此时，App Runner 可以从 Bitbucket 或存储GitHub库中检索您的源代码，也可以从您的 Amazon Elastic Con tainer Registry (Amazon ECR) Re gistry 中检索您的源图像。 AWS 账户

下图显示了 App Runner 服务架构的概述。在图中，有两个示例服务：一种部署来自 Amazon ECR 的源代码 GitHub，另一种部署来自 Amazon ECR 的源映像。同样的流程也适用于 Bitbucket 存储库。



应用程序运行器概念

以下是与 App Runner 中运行的 Web 服务相关的关键概念：

- App Runner 服务 — App Runner 用于根据源代码存储库或容器映像部署和管理应用程序的 AWS 资源。App Runner 服务是应用程序的运行版本。有关创建服务的更多信息，请参阅the section called “创建”。
- 源类型-您为部署 App Runner 服务提供的源存储库类型：源代码或源图像。
- 存储库提供商-包含您的应用程序源（例如 Bitbucket 或 Amazon ECR）的存储库服务。GitHub
- App Runner 连接 — 一种 AWS 资源，允许 App Runner 访问存储库提供商帐户（例如， GitHub 账户或组织）。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镜像。App Runner 为不同的编程平台和版本提供了各种托管运行时。有关更多信息，请参阅 基于代码的服务。
- 部署-将源存储库的某个版本（代码或图像）应用于 App Runner 服务的操作。服务的首次部署是在创建服务时进行的。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 服务配置为按照存储库中显示的应用程序的每个版本自动构建（对于源代码）和部署。这可以是源代码存储库中的新提交，也可以是源图像存储库中的新图像版本。
  - 手动部署 — 您明确启动的 App Runner 服务部署。
- 自定义域 — 您与 App Runner 服务关联的域名。您的 Web 应用程序的用户可以使用此域而不是默认的 App Runner 子域来访问您的 Web 服务。有关更多信息，请参阅 the section called “自定义域名”。

> **Note**
>
> 为了增强 App Runner 应用程序的安全性，*.awsapprunner.com 域已在公共后缀列表 (PSL) 中注册。为了进一步提高安全性，如果您需要在 App Runner 应用程序的默认域名中设置敏感 Cookie，我们建议您使用带__Host-前缀的 Cookie。这将有助于保护您的域，防范跨站点请求伪造 (CSRF) 攻击。要了解更多信息，请参阅 Mozilla 开发者网络中的 Set-Cookie 页面。

- 维护 — App Runner 偶尔会在运行 App Runner 服务的基础架构上执行的活动。维护进行时，服务状态会暂时更改为OPERATION_IN_PROGRESS（控制台中正在进行操作），持续几分钟。在此期间，您的服务上的操作（例如部署、配置更新、暂停/恢复或删除）将被阻止。几分钟后，当服务状态恢复为时，请重试该操作RUNNING。

> **Note**
>
> 如果您的操作失败，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，并且会继续处理请求。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存在问题，它就会迁移您的服务。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 支持的配置

配置 App Runner 服务时，需要指定要分配给服务的虚拟 CPU 和内存配置。您需要根据您选择的计算配置付费。有关定价的更多信息，请参阅 AWS Resource Groups 定价。

下表提供了有关 App Runner 支持的 vCPU 和内存配置的信息：

CPU	内存
0.25 个 vCPU	0.5 GB
0.25 个 vCPU	1 GB
0.5 个 vCPU	1 GB
1 个 vCPU	2 GB
1 个 vCPU	3 GB
1 个 vCPU	4 GB
2 个 vCPU	4 GB
2 个 vCPU	6 GB
4 个 vCPU	8 GB
4 个 vCPU	10 GB

CPU	内存
4 个 vCPU	12 GB

应用程序运行器资源

使用 App Runner 时，可以在中创建和管理几种类型的资源 AWS 账户。这些资源用于访问您的代码和管理您的服务。

下表概述了这些资源：

资源名称	描述
Service	表示应用程序的运行版本。本指南的其余大部分内容描述了服务类型、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 服务提供对存储在第三方提供商处的私有存储库的访问权限。作为单独的资源存在，用于在多个服务之间共享。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 服务提供用于控制应用程序自动缩放的设置。作为单独的资源存在，用于在多个服务之间共享。有关自动扩展的更多信息，请参阅the section called “自动扩缩”。 ARN：arn:aws:apprunner:*region*:*account-id*:autoscalingconfiguration/*config-name*[/*config-revision*[/*config-id*]]
ObservabilityConfiguration	为您的 App Runner 服务配置其他应用程序可观察性功能。作为单独的资源存在，用于在多个服务之间共享。有关可观测性配置的更多信息，请参阅the section called “可观测性配置”。

资源名称	描述
	ARN：arn:aws:apprunner: *region*:*account-id* :observability configuration/ *config-name* [/*config-revision* [/*config-id*]]
VpcConnector	为您的 App Runner 服务配置 VPC 设置。作为单独的资源存在，用于在多个服务之间共享。有关 VPC 功能的更多信息，请参阅the section called “传出流量”。 ARN：arn:aws:apprunner: *region*:*account-id* :vpcconnector/ *connector-name* [/*connector-revision* [/*connector-id*]]
VpcIngressConnection	它是一种用于配置传入流量的 AWS App Runner 资源。它在 VPC 接口终端节点和 App Runner 服务之间建立连接，使您的 App Runner 服务只能从 Amazon VPC 中访问。有关 VPC 功能的更多信息IngressConnection，请参阅the section called “启用私有终端节点”。 ARN：arn:aws:apprunner: *region*:*account-id* :vpcingressconnection/ *vpc-ingress-connection-name* [/*connector-id*]]

应用程序运行器资源配额

AWS 对每个 AWS 区域账户中的 AWS 资源使用施加一些配额（也称为限制）。下表列出了与 App Runner 资源相关的配额。配额也列在AWS App Runner 终端节点中，配额列在中AWS 一般参考。

资源配额	描述	默认值	可调节？
Services	您可以在账户中为每项服务创建的最大数量 AWS 区域。	30	✓ 是
Connections	您可以在账户中为每个连接创建的最大连接数 AWS 区域。您可以将单个连接用于多个服务。	10	✓ 是

资源配额		描述	默认值	可调节？
Auto scaling configurations	名字	您在账户中为每个名称创建的 auto Scaling 配置中可以拥有的最大唯一名称数 AWS 区域。您可以将单个自动扩展配置用于多个服务。	10	✓ 是
	每个名字的修订次数	您可以在账户中为每个 AWS 区域 唯一名称创建的 auto Scaling 配置修订的最大数量。您可以在多个服务中使用单个 auto Scaling 配置修订版。	5	X 否
Observability configurations	名字	在账户中为每个 AWS 区域可观测性配置创建的唯一名称的最大数量。您可以将单个可观察性配置用于多个服务。	10	✓ 是
	每个名字的修订次数	您可以在账户中为每个 AWS 区域 唯一名称创建的可观测性配置修订的最大数量。您可以在多个服务中使用单个可观测性配置修订版。	10	X 否
VPC connectors		您可以在账户中为每个连接器创建的最大 VPC 连接器数量 AWS 区域。您可以将单个 VPC 连接器用于多个服务。	10	✓ 是
VPC Ingress Connection		您可以在账户中为每 AWS 区域个 VPC 入口连接创建的最大数量。您可以使用单个 VPC 入口连接来访问多个 App Runner 服务。	1	X 否

大多数配额都是可调整的，您可以申请增加配额。有关更多信息，请参阅《Service Quotas 用户指南》中的请求增加配额。

基于源图像的 App Runner 服务

您可以使用基于两种根本不同的服务源 AWS App Runner 来创建和管理服务：源代码和源图像。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，它会自动构建（对于源代码）并将新版本部署到您的 App Runner 服务。

本章讨论基于源图像的服务。有关基于源代码的服务的信息，请参见基于代码的服务。

源镜像是存储在镜像存储库中的公共或私有容器镜像。你将 App Runner 指向一个镜像，它就会启动一个基于该镜像运行容器的服务。无需构建阶段。相反，您提供一张 ready-to-deploy 图片。

图像存储库提供商

App Runner 支持以下镜像存储库提供商：

- 亚马逊 Elastic Container Registry (Amazon ECR) — 存储私有镜像。 AWS 账户
- Amazon Elastic Container 公共注册表（Amazon ECR Public）— 存储可公开读取的图像。

提供商用例

- 在您的账户中使用存储在 Amazon ECR 中的 AWS 图片
- 使用存储在不同 AWS 账户的 Amazon ECR 中的图片
- 使用存储在 Amazon ECR Public 中的图像

在您的账户中使用存储在 Amazon ECR 中的 AWS 图片

Amazon ECR 将图像存储在存储库中。有私有和公共存储库。要将您的映像从私有存储库部署到 App Runner 服务，App Runner 需要获得从 Amazon ECR 读取您的图像的权限。要向 App Runner 授予该权限，您需要为 App Runner 提供访问角色。这是一个 AWS Identity and Access Management (IAM) 角色，具有必要的 Amazon ECR 操作权限。使用 App Runner 控制台创建服务时，您可以选择账户中的现有角色。或者，您可以使用 IAM 控制台创建新的自定义角色。或者，您可以选择让 App Runner 控制台根据托管策略为您创建角色。

当你使用 App Runner API 或时 AWS CLI，你需要完成一个分为两步的过程。首先，您使用 IAM 控制台创建访问角色。您可以使用 App Runner 提供的托管策略，也可以输入自己的自定义权限。然后，在创建服务期间使用 CreateServiceAPI 操作提供访问角色。

有关 App Runner 服务创建的信息，请参阅the section called “创建”。

使用存储在不同 AWS 账户的 Amazon ECR 中的图片

创建 App Runner 服务时，您可以使用存储在 Amazon ECR 存储库中的映像，该存储库属于您的服务所在 AWS 账户以外的账户。在使用跨账户图片时，除了上一节中列出的关于同账号图片的注意事项外，还有一些其他注意事项需要记住。

- 跨账户存储库应附有政策。存储库策略为您的访问角色提供了读取存储库中图像的权限。为此，请使用以下政策。*access-role-arn*替换为您的访问角色的 Amazon 资源名称 (ARN)。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "access-role-arn"
      },
      "Action": [
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:GetDownloadUrlForLayer"
      ]
    }
  ]
}
```

  有关将存储库策略附加到 Amazon ECR 存储库的信息，请参阅 Amazon El astic Contain er Registry 用户指南中的设置存储库策略声明。
- App Runner 不支持在与您的服务所在账户不同的账户中自动部署 Amazon ECR 映像。

使用存储在 Amazon ECR Public 中的图像

Amazon ECR Publi c 存储可公开阅读的图像。以下是 Amazon ECR 和 Amazon ECR Public 之间的主要区别，在 App Runner 服务的背景下，您应该注意这些区别：

- Amazon ECR 公开图片是公开可读的。在基于 Amazon ECR 公共镜像创建服务时，您无需提供访问角色。存储库不需要附加任何策略。
- App Runner 不支持自动（持续）部署 Amazon ECR 公共镜像。

直接从 Amazon ECR Public 启动服务

您可以直接启动托管在 Amazon ECR 公共库中的兼容 Web 应用程序的容器映像，作为在 App Runner 上运行的 Web 服务。浏览图库时，请在图库页面上查找 Launch with App Runner 以获取图片。带有此选项的图像与 App Runner 兼容。有关图库的更多信息，请参阅 Amazon ECR 公共用户指南中的使用 Amazon ECR 公共画廊。

将图库图片作为 App Runner 服务启动

1. 在图像的图库页面上，选择使用 App Runner 启动。

   结果：App Runner 控制台将在新的浏览器选项卡中打开。控制台显示创建服务向导，其中已预先填写了大部分所需的新服务详细信息。
2. 如果要在控制台显示的 AWS 区域以外的区域中创建服务，请选择控制台标题上显示的区域。然后，选择另一个区域。
3. 在“端口”中，输入图像应用程序侦听的端口号。通常，你可以在图片库页面上找到它。
4. （可选）更改任何其他配置详细信息。
5. 选择“下一步”，查看设置，然后选择“创建并部署”。

图片示例

App Runner 团队在 Amazon ECR 公共图库中维护hello-app-runner示例图片。您可以使用此示例开始创建基于图像的 App Runner 服务。有关更多信息，请参阅hello-app-runner。

基于源代码的 App Runner 服务

您可以使用基于两种根本不同的服务源 AWS App Runner 来创建和管理服务：源代码和源图像。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，它会自动构建（对于源代码）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。有关基于源图像的服务的信息，请参阅基于图像的服务。

源代码是 App Runner 为您构建和部署的应用程序代码。将 App Runner 指向代码存储库中的源目录，然后选择与编程平台版本相对应的合适运行时。App Runner 根据运行时的基本映像和您的应用程序代码构建映像。然后，它会启动一个基于此镜像运行容器的服务。

App Runner 提供了便捷的特定于平台的托管运行时。这些运行时中的每一个都根据你的源代码构建一个容器镜像，并在你的镜像中添加语言运行时依赖关系。您无需提供容器配置和构建说明，例如 Dockerfile。

本章的副主题讨论 App Runner 支持的各种平台，即为不同的编程环境和版本提供托管运行时的托管平台。

主题

源代码存储库提供者

App Runner 通过从源代码存储库中读取源代码来部署您的源代码。App Runner 支持两个源代码存储库提供程序：GitHub和 Bitbucket。

从您的源代码存储库提供商部署

要将源代码从源代码存储库部署到 App Runner 服务，App Runner 需要与该服务建立连接。使用 App Runner 控制台创建服务时，您需要提供连接详细信息和源目录，供 App Runner 部署源代码。

连接

作为服务创建过程的一部分，您需要提供连接详细信息。当你使用 App Runner API 或时 AWS CLI，连接是一种单独的资源。首先，使用 CreateConnectionAPI 操作创建连接。然后，在创建服务期间使用 CreateServiceAPI 操作提供连接的 ARN。

源目录

创建服务时，还会提供源目录。默认情况下，App Runner 使用存储库的根目录作为源目录。源目录是源代码存储库中存储应用程序源代码和配置文件的位置。生成和启动命令也从源目录执行。当您使用 App Runner API 或创建或更新服务时，您需要在CreateService和 UpdateServiceAPI 操作中提供源目录。 AWS CLI 有关更多信息，请参阅下面的源目录部分。

有关创建 App Runner 服务的更多信息，请参阅the section called “创建”。有关 App Runner 连接的更多信息，请参阅the section called “连接”。

源目录

创建 App Runner 服务时，您可以提供源目录以及存储库和分支。将 “源目录” 字段的值设置为存储应用程序源代码和配置文件的存储库目录路径。App Runner 从您提供的源目录路径执行生成和启动命令。

将源目录路径的值输入为从根存储库目录开始的绝对路径。如果未指定值，则默认为存储库顶级目录，也称为存储库根目录。

除了顶级存储库目录之外，您还可以选择提供不同的源目录路径。这支持 monorepo 存储库架构，这意味着多个应用程序的源代码存储在一个存储库中。要通过单个 monorepo 创建和支持多个 App Runner 服务，请在创建每个服务时指定不同的源目录。

> **Note**
>
> 如果您为多个 App Runner 服务指定相同的源目录，则这两个服务将分别部署和运行。

如果您选择使用`apprunner.yaml`配置文件来定义服务参数，请将其放在存储库的源目录文件夹中。

如果 “部署” 触发器选项设置为 “自动”，则您在源目录中提交的更改将触发自动部署。 只有源目录路径中的更改才会触发自动部署。重要的是要了解源目录的位置如何影响自动部署的范围。有关更多信息，请参阅中的自动部署部署方法。

> **Note**
>
> 如果您的 App Runner 服务使用 PHP 托管的运行时，并且您想要指定默认根存储库以外的源目录，那么使用正确的 PHP 运行时版本非常重要。有关更多信息，请参阅 使用 PHP 平台。

应用程序运行器托管平台

App Runner 托管平台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编程语言或运行时环境的版本轻松构建和运行容器。使用托管运行时时，App Runner 从托管运行时映像开始。此镜像基于 Amazon Linux Docker 镜像，包含语言运行时包以及一些工具和常用的依赖包。App Runner 使用此托管运行时映像作为基础映像，并添加您的应用程序代码来构建 Docker 映像。然后，它会部署此映像以在容器中运行您的 Web 服务。

在使用 App Runner 控制台或 CreateServiceAPI 操作创建服务时，可以为 App Runner 服务指定运行时。您也可以将运行时指定为源代码的一部分。在包含在代码存储库中的 A pp Runner 配置文件中使用`runtime`关键字。托管运行时的命名约定是。 <language-name><major-version>

每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。如果您的应用程序需要托管运行时的特定版本，则可以使用 App Runner 配置文件中的`runtime-version`关键字进行指定。您可以锁定到任何级别的版本，包括主要版本或次要版本。App Runner 仅对服务的运行时进行较低级别的更新。

托管运行时版本和 App Runner 版本

App Runner 现在为您的应用程序提供了更新的构建流程。它目前正在为在托管运行时 Python 3.11 和 Node.js 18 上运行的服务调用新版本，上次发布于 2023 年 12 月 29 日。修改后的构建过程更快、更

高效。它还会创建占用空间较小的最终映像，其中仅包含运行应用程序所需的源代码、构建工件和运行时。

我们将较新的构建过程称为修订后的 App Runner 版本，将原始构建过程称为最初的 App Runner 版本。为了避免对早期版本的运行时平台进行重大更改，App Runner 仅将修订后的版本应用于特定的运行时版本，通常是新发布的主要版本。

我们在`apprunner.yaml`配置文件中引入了一个新组件，以使修订后的版本向后兼容，适用于非常具体的用例，还可以更灵活地配置应用程序的构建。这是可选`pre-run`参数。我们将在以下各节中说明何时使用此参数以及有关构建的其他有用信息。

下表说明了 App Runner 版本的哪个版本适用于特定的托管运行时版本。我们将继续更新此文档，让您随时了解我们当前的运行时间。

平台	原始版本	修改后的版本
Python – 发布信息	• Python 3.8 • Python 3.7	• Python 3.11 (!)
Node.js – 发布信息	• Node.js 16 • Node.js 14 • Node.js 12	• Node.js 18
Corretto — 发布信息	• Corretto 11 • Corretto 8	
.NET – 发布信息	• .NET 6	
PHP – 发布信息	• PHP 8.1	
Ruby – 发布信息	• Ruby 3.1	
Go – 发布信息	• Go 1	

Important

Python 3.11 — 我们对使用 Python 3.11 托管运行时的服务的构建配置提出了具体建议。有关更多信息，请参阅 Python 平台主题特定运行时版本的标注中的。

有关 App Runner 版本和迁移的更多信息

当您将应用程序迁移到使用修订版本的较新运行时时，可能需要稍微修改构建配置。

为了提供迁移注意事项的背景信息，我们将首先描述原始 App Runner 版本和修订版的高级流程。接下来我们将有一节介绍您的服务的特定属性，这些属性可能需要一些配置更新。

最初的 App Runner 版本

最初的 App Runner 应用程序构建过程利用了该 AWS CodeBuild 服务。初始步骤基于该 CodeBuild 服务精心策划的图像。接下来是 Docker 构建流程，该过程使用适用的 App Runner 托管运行时映像作为基础映像。

一般步骤如下：

1. 在 CodeBuild精心策划的图像中运行`pre-build`命令。

   这些`pre-build`命令是可选的。它们只能在`apprunner.yaml`配置文件中指定。
2. 使用在 CodeBuild 上一步中的同一张图像上运行`build`命令。

   这些`build`命令是必需的。它们可以在 App Runner 控制台、App Runner API 或`apprunner.yaml`配置文件中指定。
3. 运行 Docker 版本，根据您的特定平台和运行时版本的 App Runner 托管运行时映像生成映像。
4. 从我们在步骤 2 中生成的图像中复制该`/app`目录。目标是基于我们在步骤 3 中生成的 App Runner 托管运行时映像的图像。
5. 在生成的 App Runner 托管运行时映像上再次运行`build`命令。我们再次运行构建命令，从步骤 4 中复制到的`/app`目录中的源代码生成构建工件。稍后，App Runner 将部署此映像，以便在容器中运行您的 Web 服务。

   这些`build`命令是必需的。它们可以在 App Runner 控制台、App Runner API 或`apprunner.yaml`配置文件中指定。
6. 运行步骤 2 中 CodeBuild 镜像中的`post-build`命令。

   这些`post-build`命令是可选的。它们只能在`apprunner.yaml`配置文件中指定。

构建完成后，App Runner 会部署步骤 5 中生成的 App Runner 托管运行时映像，以便在容器中运行您的 Web 服务。

修订后的 App Runner 版本

修订后的构建过程比上一节中描述的原始构建过程更快、更高效。它消除了先前版本版本中出现的生成命令的重复。它还会创建占用空间较小的最终映像，其中仅包含运行应用程序所需的源代码、构建工件和运行时。

此构建过程使用 Docker 多阶段构建。一般处理步骤如下：

1. 构建阶段 — 启动 docker 构建过程，该过程在 App Runner 构建映像之上执行`pre-build`和`build`命令。
   a. 将应用程序源代码复制到`/app`目录中。

      > **Note**
      >
      > 在 Docker 构建的每个阶段，此`/app`目录都被指定为工作目录。

   b. 运行 `pre-build` 命令。

      这些`pre-build`命令是可选的。它们只能在`apprunner.yaml`配置文件中指定。

   c. 运行`build`命令。

      这些`build`命令是必需的。它们可以在 App Runner 控制台、App Runner API 或`apprunner.yaml`配置文件中指定。
2. 打包阶段 — 生成最终的客户容器镜像，该镜像也基于 App Runner 运行映像。
   a. 将前一个生成阶段的`/app`目录复制到新的运行映像。这包括您的应用程序源代码和前一阶段的构建工件。
   b. 运行`pre-run`命令。如果您需要使用`build`命令修改`/app`目录之外的运行时映像，请将相同或必需的命令添加到`apprunner.yaml`配置文件的这一段。

      这是为支持修订后的 App Runner 版本而引入的新参数。

      这些`pre-run`命令是可选的。它们只能在`apprunner.yaml`配置文件中指定。

      > **注意**
      >
      > - 只有修订后的版本支持这些`pre-run`命令。如果您的服务使用的是使用原始版本的运行时版本，请不要将它们添加到配置文件中。

- 如果您不需要使用build命令修改/app目录之外的任何内容，则无需指定pre-run命令。

3. 生成后阶段 — 此阶段从 “构建” 阶段恢复并运行post-build命令。
   a. 在/app目录中运行post-build命令。

      这些post-build命令是可选的。它们只能在apprunner.yaml配置文件中指定。

构建完成后，App Runner 会部署运行映像，以便在容器中运行您的 Web 服务。

> **Note**
>
> 配置生成过程apprunner.yaml时，不要被误导到的 “运行” 部分中的env条目。尽管步骤 2 (b) 中引用的pre-run命令参数位于 “运行” 部分，但不要使用 “运行” 部分中的env参数来配置构建。这些pre-run命令仅引用配置文件的 Build 部分中定义的env变量。有关更多信息，请参阅 App Runner 配置文件一章跑步部分中的。

考虑迁移时需要考虑的服务要求

如果您的应用程序环境具有这两个要求中的任何一个，则需要通过添加pre-run命令来修改构建配置。

- 如果你需要使用build命令修改/app目录之外的任何内容。
- 如果您需要运行两次build命令来创建所需的环境。这是一个非常不寻常的要求。绝大多数版本都不会这样做。

在**/app**目录之外进行修改

- 修订后的 App Runner 版本假设您的应用程序在/app目录之外没有依赖关系。
- 您在apprunner.yaml文件、App Runner API 或 App Runner 控制台中提供的命令必须在/app目录中生成构件。
- 您可以修改pre-buildbuild、和post-build命令以确保所有构建工件都在/app目录中。
- 如果您的应用程序需要在编译版本中进一步修改为服务生成的映像，则可以在/app目录之外使用新pre-run命令apprunner.yaml。有关更多信息，请参阅 使用配置文件设置 App Runner 服务选项。

运行**build**命令两次

- 最初的 App Runner 版本运行build命令两次，第一次是在步骤 2 中，然后在步骤 5 中再次运行。修订后的 App Runner 版本纠正了这种冗余性，并且只运行一次build命令。如果您的应用程序不寻常地要求build命令运行两次，则修订后的 App Runner 版本提供了使用pre-run参数再次指定和执行相同命令的选项。这样做会保留相同的双重构建行为。

使用 Python 平台

P AWS App Runner ython 平台提供托管运行时。每个运行时都可以轻松构建和运行基于 Python 版本的 Web 应用程序的容器。当你使用 Python 运行时时，App Runner 从托管的 Python 运行时镜像开始。此镜像基于 Amazon Linux Docker 镜像，包含某个 Python 版本的运行时包以及一些工具和常用的依赖包。App Runner 使用此托管运行时映像作为基础映像，并添加您的应用程序代码来构建 Docker 映像。然后，它会部署此映像以在容器中运行您的 Web 服务。

在使用 App Runner 控制台或 CreateServiceAPI 操作创建服务时，可以为 App Runner 服务指定运行时。您也可以将运行时指定为源代码的一部分。在包含在代码存储库中的 A pp Runner 配置文件中使用runtime关键字。托管运行时的命名约定是。 <language-name><major-version>

有关有效的 Python 运行时名称和版本，请参见the section called “发布信息”。

每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。如果您的应用程序需要托管运行时的特定版本，则可以使用 App Runner 配置文件中的runtime-version关键字进行指定。您可以锁定到任何级别的版本，包括主要版本或次要版本。App Runner 仅对服务的运行时进行较低级别的更新。

Python 运行时的版本语法：*major*[.*minor*[.*patch*]]

例如：3.8.5

以下示例演示了版本锁定：

- 3.8— 锁定主要版本和次要版本。App Runner 仅更新补丁版本。
- 3.8.5— 锁定到特定的补丁版本。App Runner 不会更新你的运行时版本。

主题

- Python 运行时配置
- 特定运行时版本的标注

- Python 运行时示例
- Python 运行时发布信息

Python 运行时配置

选择托管运行时时，还必须至少配置生成和运行命令。您可以在创建或更新 App Runner 服务时对其进行配置。您可以使用以下方法之一来执行此操作：

- 使用 App Runner 控制台-在创建过程或配置选项卡的 “配置构建” 部分中指定命令。
- 使用 App Runner API-调用CreateService或 UpdateServiceAPI 操作。使用CodeConfigurationValues数据类型的`BuildCommand`和`StartCommand`成员来指定命令。
- 使用配置文件-在最多三个构建阶段中指定一个或多个构建命令，并指定一个用于启动应用程序的运行命令。还有其他可选的配置设置。

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 是在创建时直接获取配置设置还是从配置文件中获取配置设置。

特定运行时版本的标注

> **Note**
>
> App Runner 现在基于以下运行时版本为应用程序运行更新的构建流程：Python 3.11 和 Node.js 18。如果您的应用程序在其中一个运行时版本上运行，请参阅托管运行时版本和 App Runner 版本，了解有关修订后的构建过程的更多信息。使用所有其他运行时版本的应用程序不受影响，它们将继续使用原始的构建过程。

Python 3.11（修订的 App Runner 版本）

在 apprunner.yaml 中为托管 P ython 3.11 运行时使用以下设置。

- 将 “顶部” 部分的`runtime`密钥设置为 `python311`

  Example

  ```
  runtime: python311
  ```

- 使用代`pip3`替`pip`来安装依赖关系。

- 请改用python3解释器python。
- 将pip3安装程序作为pre-run命令运行。Python 在/app目录之外安装依赖项。由于 App Runner 运行适用于 Python 3.11 的修订版 App Runner 版本，因此通过apprunner.yaml文件的 “构建” 部分中的命令安装在/app目录之外的任何内容都将丢失。有关更多信息，请参阅 修订后的 App Runner 版本。

  Example

```
run:
  runtime-version: 3.11
  pre-run:
    - pip3 install pipenv
    - pipenv install
    - python3 copy-global-files.py
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
```

有关更多信息，另请参阅本主题后面的 Python 3.11 扩展配置文件示例。

Python 运行时示例

以下示例显示了用于构建和运行 Python 服务的 App Runner 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。

> **Note**
>
> ############## 3.7.7 # 3. 11#您可以将其替换为要使用的版本。有关支持的最新 Python 运行时版本，请参阅the section called “发布信息”。

最小的 Python 配置文件

此示例显示了可以与 Python 托管运行时配合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 “配置文件示例”。

Python 3.11 使用pip3和python3命令。有关更多信息，请参阅本主题后面的 Python 3.11 扩展配置文件示例。

Example apprunner.yaml

```
version: 1.0
```

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

扩展的 Python 配置文件

此示例展示了在 Python 托管运行时中使用所有配置密钥的情况。

> **Note**
>
> 这些示例中使用的运行时版本为 *3.7.7*。您可以将其替换为要使用的版本。有关支持的最新 Python 运行时版本，请参阅the section called “发布信息”。
> Python 3.11 使用`pip3`和`python3`命令。有关更多信息，请参阅本主题后面的 Python 3.11 扩展配置文件示例。

Example apprunner.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"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

扩展 Python 配置文件 — Python 3.11（使用修改后的版本）

此示例显示了在 Python 3.11 托管运行时中使用所有配置密钥的情况。`apprunner.yaml`此示例包括一`pre-run`节，因为此版本的 Python 使用修订后的 App Runner 版本。

只有修订后的 App Runner 版本支持该`pre-run`参数。如果您的应用程序使用原始 App Runner 版本支持的运行时版本，请不要在配置文件中插入此参数。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

> **Note**
>
> 这些示例中使用的运行时版本是 *`3.11`*。您可以将其替换为要使用的版本。有关支持的最新 Python 运行时版本，请参阅the section called “发布信息”。

Example apprunner.yaml

```
version: 1.0
runtime: python311
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/DOC-EXAMPLE-BUCKET/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip3 install pipenv
```

```
      - pipenv install
    post-build:
      - python3 manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.11
  pre-run:
    - pip3 install pipenv
    - pipenv install
    - python3 copy-global-files.py
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

完整的 Python 应用程序源码

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

```
pyramid==2.0
```

Example server.py

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
```

```
import os

def hello_world(request):
    name = os.environ.get('NAME')
    if name == None or len(name) == 0:
        name = "world"
    message = "Hello, " + name + "!\n"
    return Response(message)

if __name__ == '__main__':
    port = int(os.environ.get("PORT"))
    with Configurator() as config:
        config.add_route('hello', '/')
        config.add_view(hello_world, route_name='hello')
        app = config.make_wsgi_app()
    server = make_server('0.0.0.0', port, app)
    server.serve_forever()
```

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install -r requirements.txt
run:
  command: python server.py
```

Python 运行时发布信息

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

支持的运行时版本 — 经过修订的 App Runner 版本

运行时名称	次要版本	包含的套餐
Python 3.11 (python311)	3.11.9	SQLite 3.46.0
	3.11.8	SQLite 3.45.2
	3.11.7	SQLite 3.44.2

注意

- Python 3.11 — 我们对使用 Python 3.11 托管运行时的服务的构建配置提出了具体建议。有关更多信息，请参阅 Python 平台主题特定运行时版本的标注中的。
- App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

支持的运行时版本-原始 App Runner 版本

运行时名称	次要版本	包含的套餐
Python 3 (python3)	3.8.16	SQLite 3.46.0
	3.7.16	SQLite 3.46.0
	3.8.15	SQLite 3.40.0
	3.8.5	SQLite 3.39.4
	3.7.15	SQLite 3.40.0
	3.7.10	SQLite 3.40.0

Note

App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

使用 Node.js 平台

AWS App Runner Node.js 平台提供托管运行时。使用基于 Node.js 版本的 Web 应用程序，每个运行时都可以轻松构建和运行容器。当你使用 Node.js 运行时时，App Runner 从托管的 Node.js 运行时镜像开始。此镜像基于亚马逊 Linux Docker 镜像，包含某个 Node.js 版本的运行时包和一些工具。App

Runner 使用此托管运行时映像作为基础映像，并添加您的应用程序代码来构建 Docker 映像。然后，它会部署此映像以在容器中运行您的 Web 服务。

在使用 App Runner 控制台或 CreateServiceAPI 操作创建服务时，可以为 App Runner 服务指定运行时。您也可以将运行时指定为源代码的一部分。在包含在代码存储库中的 A pp Runner 配置文件中使用`runtime`关键字。托管运行时的命名约定是。 <language-name><major-version>

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

每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。如果您的应用程序需要托管运行时的特定版本，则可以使用 App Runner 配置文件中的`runtime-version`关键字进行指定。您可以锁定到任何级别的版本，包括主要版本或次要版本。App Runner 仅对服务的运行时进行较低级别的更新。

Node.js 运行时的版本语法：*`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例如：`12.21.0`

以下示例演示了版本锁定：

- `12.21`— 锁定主要版本和次要版本。App Runner 仅更新补丁版本。
- `12.21.0`— 锁定到特定的补丁版本。App Runner 不会更新你的运行时版本。

主题

- Node.js 运行时配置
- 特定运行时版本的标注
- Node.js 运行时示例
- Node.js 运行时发布信息

Node.js 运行时配置

选择托管运行时时，还必须至少配置生成和运行命令。您可以在创建或更新 App Runner 服务时对其进行配置。您可以使用以下方法之一来执行此操作：

- 使用 App Runner 控制台-在创建过程或配置选项卡的 “配置构建” 部分中指定命令。
- 使用 App Runner API — 调用CreateService或 UpdateServiceAPI 操作。使用CodeConfigurationValues数据类型的`BuildCommand`和`StartCommand`成员来指定命令。

- 使用配置文件-在最多三个构建阶段中指定一个或多个构建命令，并指定一个用于启动应用程序的运行命令。还有其他可选的配置设置。

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 是在创建时直接获取配置设置还是从配置文件中获取配置设置。

具体而言，对于 Node.js 运行时，您还可以使用源存储库根目录`package.json`中命名的 JSON 文件配置构建和运行时。使用此文件，您可以配置 Node.js 引擎版本、依赖包和各种命令（命令行应用程序）。诸如 npm 或 yarn 之类的 Package 管理器将此文件解释为其命令的输入。

例如：

- npm install安装中`dependencies`和`devDependencies`节点定义的软件包`package.json`。
- npm start或者npm run start运行中`scripts/start`节点定义的命令`package.json`。

下面是一个 `package.json` 示例文件。

package.j

```
{
  "name": "node-js-getting-started",
  "version": "0.3.0",
  "description": "A sample Node.js app using Express 4",
  "engines": {
    "node": "12.21.0"
  },
  "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"
  }
}
```

有关更多信息package.json，请参阅 npm Doc s 网站上创建 package.json 文件。

提示

- 如果您的package.json文件定义了start命令，则可以将其用作 App Runner 配置文件中的run命令，如下例所示。

  Example

  package.j

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

  apprunner.yaml

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

- 当你在开发环境npm install中运行时，npm 会创建该文件package-lock.json。此文件包含 npm 刚安装的软件包版本的快照。此后，当 npm 安装依赖项时，它会使用这些确切的版本。如果你安装 yarn，它会创建一个yarn.lock文件。将这些文件提交到您的源代码存储库中，以确保您的应用程序与您开发和测试的依赖项版本一起安装。
- 您也可以使用 App Runner 配置文件来配置 Node.js 版本并启动命令。执行此操作时，这些定义会覆盖中的定义package.json。中的nodepackage.json版本与 App Runner 配置文件中的runtime-version值发生冲突会导致 App Runner 构建阶段失败。

特定运行时版本的标注

Node.js 18（修改后的 App Runner 版本）

App Runner 现在基于以下运行时版本为应用程序运行更新的构建流程：Python 3.11 和 Node.js 18。如果您的应用程序在其中一个运行时版本上运行，请参阅托管运行时版本和 App Runner 版本，了解有关修订后的构建过程的更多信息。使用所有其他运行时版本的应用程序不受影响，它们将继续使用原始的构建过程。

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 App Runner 配置文件。

> **Note**
>
> *############## 12.21.0 # 18.19. 0#*您可以将其替换为要使用的版本。有关支持的最新的 Node.js 运行时版本，请参阅the section called “发布信息”。

Node.js 的最小配置文件

此示例显示了一个可以与 Node.js 托管运行时配合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 “配置文件示例”。

Example apprunner.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

此示例显示了在 Node.js 托管运行时中使用所有配置密钥的情况。

> **Note**
>
> 这些示例中使用的运行时版本为 *12.21.0*。您可以将其替换为要使用的版本。有关支持的最新的 Node.js 运行时版本，请参阅the section called “发布信息”。

Example apprunner.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: 12.21.0
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Node.js 扩展配置文件 — Node.js 18（使用修订后的版本）

此示例显示了如何在 Node.js 托管运行时中使用所有配置密钥`apprunner.yaml`。此示例包括一`pre-run`节，因为此版本的 Node.js 使用的是修订后的 App Runner 版本。

只有修订后的 App Runner 版本支持该`pre-run`参数。如果您的应用程序使用原始 App Runner 版本支持的运行时版本，请不要在配置文件中插入此参数。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

> **Note**
>
> 这些示例中使用的运行时版本为 *18.19.0*。您可以将其替换为要使用的版本。有关支持的最新的 Node.js 运行时版本，请参阅the section called “发布信息”。

Example apprunner.yaml

```
version: 1.0
runtime: nodejs18
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: 18.19.0
  pre-run:
    - node copy-global-files.js
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

带有 Grunt 的 Node.js 应用程序

此示例说明如何配置使用 Grunt 开发的 Node.js 应用程序。Grunt 是一个命令行 JavaScript 任务运行器。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。您可以通过将`Gruntfile.js`文件包含在源存储库的根目录中来配置 Grunt。

Example package.j

```
{
  "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 apprunner.yaml

> **Note**
>
> 这些示例中使用的运行时版本为 *12.21.0*。您可以将其替换为要使用的版本。有关支持的最新的 Node.js 运行时版本，请参阅the section called “发布信息”。

```
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: 12.21.0
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
```

Node.js 运行时发布信息

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

支持的运行时版本 — 修订的 App Runner 版本

运行时名称	次要版本	内含套餐
Node.js 18 (nodejs18)	18.20.3	npm 10.7.0，yarn 1.22.22
	18.20.2	npm 10，yarn *
	18.19.1	npm 10，yarn *
	18.19.0	npm 10，yarn *

> **Note**
>
> App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

支持的运行时版本-原始 App Runner 版本

运行时名称	次要版本	内含套餐
Node.js 16 (nodejs16)	16.20.2	npm 8.19.4，yarn 1.22.22

运行时名称	次要版本	内含套餐
	16.20.1	npm 8.19.4，yarn *
	16.20.0	npm 8.19.4，yarn *
	16.19.1	npm 8.19.4，yarn *
	16.19.0	npm 8.19.4，yarn *
	16.18.1	npm 8.19.4，yarn *
	16.17.1	npm 8.19.4，yarn *
	16.17.0	npm 8.19.4，yarn *
Node.js 14 (nodejs14)	14.21.3	npm 6.14.18，yarn 1.22.22
	14.21.2	npm 6.14.18，yarn *
	14.21.1	npm 6.14.18，yarn *
	14.20.1	npm 6.14.18，yarn *
	14.19.0	npm 6.14.18，yarn *
Node.js 12 (nodejs12)	12.22.12	npm 6.14.16，yarn 1.22.22
	12.21.0	npm 6.14.16，yarn *

> **Note**
>
> App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

使用 Java 平台

AWS App Runner Java 平台提供托管运行时。使用基于 Java 版本的 Web 应用程序，每个运行时都可以轻松构建和运行容器。当你使用 Java 运行时时，App Runner 从托管 Java 运行时镜像开始。此镜像基于 Amazon Linux Docker 镜像，包含一个 Java 版本的运行时包和一些工具。App Runner 使用此托管运行时映像作为基础映像，并添加您的应用程序代码来构建 Docker 映像。然后，它会部署此映像以在容器中运行您的 Web 服务。

在使用 App Runner 控制台或 CreateServiceAPI 操作创建服务时，可以为 App Runner 服务指定运行时。您也可以将运行时指定为源代码的一部分。在包含在代码存储库中的 A pp Runner 配置文件中使用`runtime`关键字。托管运行时的命名约定是。 <language-name><major-version>

目前，所有支持的 Java 运行时都基于 Amazon Corretto。有关有效的 Java 运行时名称和版本，请参见the section called “发布信息”。

每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。如果您的应用程序需要托管运行时的特定版本，则可以使用 App Runner 配置文件中的`runtime-version`关键字进行指定。您可以锁定到任何级别的版本，包括主要版本或次要版本。App Runner 仅对服务的运行时进行较低级别的更新。

Amazon Corretto 运行时的版本语法：

运行时	语法	示例
corretto11	`11.0[.`*`openjdk-update`* `[.`*`openjdk-build`* `[.`*`corretto-specific-revision`* `]]]`	`11.0.13.08.1`
corretto8	`8[.`*`openjdk-update`* `[.`*`openjdk-build`* `[.`*`corretto-specific-revision`* `]]]`	`8.312.07.1`

以下示例演示了版本锁定：

- `11.0.13`— 锁定 Open JDK 更新版本。App Runner 仅更新 Open JDK 和 Amazon Corretto 较低级别的版本。
- `11.0.13.08.1`— 锁定到特定版本。App Runner 不会更新你的运行时版本。

主题

Java 运行时配置

选择托管运行时时，还必须至少配置生成和运行命令。您可以在创建或更新 App Runner 服务时对其进行配置。您可以使用以下方法之一来执行此操作：

- 使用 App Runner 控制台-在创建过程或配置选项卡的 “配置构建” 部分中指定命令。
- 使用 App Runner API — 调用CreateService或 UpdateServiceAPI 操作。使用CodeConfigurationValues数据类型的`BuildCommand`和`StartCommand`成员来指定命令。
- 使用配置文件-在最多三个构建阶段中指定一个或多个构建命令，并指定一个用于启动应用程序的运行命令。还有其他可选的配置设置。

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 是在创建时直接获取配置设置还是从配置文件中获取配置设置。

Java 运行时示例

以下示例显示了用于构建和运行 Java 服务的 App Runner 配置文件。最后一个示例是完整的 Java 应用程序的源代码，您可以将其部署到 Corretto 11 运行时服务。

> **Note**
>
> 这些示例中使用的运行时版本是 *`11.0.13.08.1`*。您可以将其替换为要使用的版本。有关支持的最新 Java 运行时版本，请参阅the section called “发布信息”。

最小 Corretto 11 配置文件

此示例显示了可以与 Corretto 11 托管运行时配合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。

Example apprunner.yaml

```
version: 1.0
```

```
runtime: corretto11
build:
  commands:
    build:
      - mvn clean package
run:
  command: java -Xms256m -jar target/MyApp-1.0-SNAPSHOT.jar .
```

扩展的 Corretto 11 配置文件

此示例说明如何在 Corretto 11 托管运行时中使用所有配置密钥。

> **Note**
>
> 这些示例中使用的运行时版本是 *11.0.13.08.1*。您可以将其替换为要使用的版本。有关支持的最新 Java 运行时版本，请参阅the section called “发布信息”。

Example apprunner.yaml

```
version: 1.0
runtime: corretto11
build:
  commands:
    pre-build:
      - yum install some-package
      - scripts/prebuild.sh
    build:
      - mvn clean package
    post-build:
      - mvn clean test
  env:
    - name: M2
      value: "/usr/local/apache-maven/bin"
    - name: M2_HOME
      value: "/usr/local/apache-maven/bin"
run:
  runtime-version: 11.0.13.08.1
  command: java -Xms256m -jar target/MyApp-1.0-SNAPSHOT.jar .
  network:
    port: 8000
    env: APP_PORT
  env:
```

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

完整的 Corretto 11 应用程序源代码

此示例显示了可以部署到 Corretto 11 运行时服务的完整 Java 应用程序的源代码。

Example src/main/java/com/ /.java HelloWorld HelloWorld

```
package com.HelloWorld;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorld {

    @RequestMapping("/")
    public String index(){
        String s = "Hello World";
        return s;
    }
}
```

Example src/main/java/com/ /Main.java HelloWorld

```
package com.HelloWorld;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Main {

    public static void main(String[] args) {

        SpringApplication.run(Main.class, args);
    }
}
```

Example apprunner.yaml

```
version: 1.0
```

```
runtime: corretto11
build:
  commands:
    build:
      - mvn clean package
run:
  command: java -Xms256m -jar target/HelloWorldJavaApp-1.0-SNAPSHOT.jar .
  network:
    port: 8080
```

Example pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.1.RELEASE</version>
    <relativePath/>
  </parent>
  <groupId>com.HelloWorld</groupId>
  <artifactId>HelloWorldJavaApp</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <java.version>11</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-rest</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
```

```
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
        <configuration>
          <release>11</release>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Java 运行时版本信息

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

支持的运行时版本-原始 App Runner 版本

运行时名称	次要版本	内含套餐
Corretto 11 (corretto11)	11.0.23.9.1	Maven 3.9.8、Gradle 6.9.4
	11.0.22.7.1	Maven 3.9.6，Gradle 6.9.4
	11.0.21.9.1	Maven 3.9.6，Gradle 6.9.4
	11.0.21.9.1	Maven 3.9.5，Gradle 6.9.4
	11.0.20.8.1	Maven 3.9.3、Gradle 6.9.4

运行时名称	次要版本	内含套餐
	11.0.19.7.1	Maven 3.9.3、Gradle 6.9.4
	11.0.18.10.1	Maven 3.9.1，Gradle 6.9.4
	11.0.17.8.1	Maven 3.8.6、Gradle 6.9.3
	11.0.16.9.1	Maven 3.8.6，Gradle 6.9.2
	11.0.13.08.1	Maven 3.6.3、Gradle 6.5
Corretto 8 (corretto8)	8.412.08.1	Maven 3.9.8、Gradle 6.9.4
	8.402.08.1	Maven 3.9.6，Gradle 6.9.4
	8.392.08.1	Maven 3.9.6，Gradle 6.9.4
	8.382.05.1	Maven 3.9.4、Gradle 6.9.4
	8.372.07.1	Maven 3.9.3、Gradle 6.9.4
	8.362.08.1	Maven 3.9.1，Gradle 6.9.4
	8.352.08.1	Maven 3.8.6、Gradle 6.9.3
	8.342.07.4	Maven 3.8.6，Gradle 6.9.2
	8.312.07.1	Maven 3.6.3、Gradle 6.5

Note

App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

使用 .NET 平台

AWS App Runner .NET 平台提供托管运行时。每个运行时都可以轻松构建和运行带有基于.NET 版本的 Web 应用程序的容器。当你使用.NET 运行时时，App Runner 从托管的.NET 运行时镜像开始。此镜像基于 Amazon Linux Docker 镜像，包含适用于.NET 版本的运行时包以及一些工具和常用的依赖包。App Runner 使用此托管运行时映像作为基础映像，并添加您的应用程序代码来构建 Docker 映像。然后，它会部署此映像以在容器中运行您的 Web 服务。

在使用 App Runner 控制台或 CreateServiceAPI 操作创建服务时，可以为 App Runner 服务指定运行时。您也可以将运行时指定为源代码的一部分。在包含在代码存储库中的 A pp Runner 配置文件中使用`runtime`关键字。托管运行时的命名约定是。 <language-name><major-version>

有关有效的.NET 运行时名称和版本，请参阅the section called “发布信息”。

每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。如果您的应用程序需要托管运行时的特定版本，则可以使用 App Runner 配置文件中的`runtime-version`关键字进行指定。您可以锁定到任何级别的版本，包括主要版本或次要版本。App Runner 仅对服务的运行时进行较低级别的更新。

.NET 运行时的版本语法：*`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例如：`6.0.9`

以下示例演示了版本锁定：

- `6.0`— 锁定主要版本和次要版本。App Runner 仅更新补丁版本。
- `6.0.9`— 锁定到特定的补丁版本。App Runner 不会更新你的运行时版本。

主题

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

.NET 运行时配置

选择托管运行时时，还必须至少配置生成和运行命令。您可以在创建或更新 App Runner 服务时对其进行配置。您可以使用以下方法之一来执行此操作：

- 使用 App Runner 控制台-在创建过程或配置选项卡的 “配置构建” 部分中指定命令。
- 使用 App Runner API — 调用CreateService或 UpdateServiceAPI 操作。使用CodeConfigurationValues数据类型的`BuildCommand`和`StartCommand`成员来指定命令。
- 使用配置文件-在最多三个构建阶段中指定一个或多个构建命令，并指定一个用于启动应用程序的运行命令。还有其他可选的配置设置。

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 是在创建时直接获取配置设置还是从配置文件中获取配置设置。

NET 运行时示例

以下示例显示了用于构建和运行.NET 服务的 App Runner 配置文件。最后一个示例是完整的.NET 应用程序的源代码，您可以将其部署到.NET 运行时服务。

> **Note**
>
> 这些示例中使用的运行时版本为 *`6.0.9`*。您可以将其替换为要使用的版本。有关支持的最新.NET 运行时版本，请参阅the section called “发布信息”。

最小的.NET 配置文件

此示例显示了可在.NET 托管运行时中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 “配置文件示例”。

Example apprunner.yaml

```
version: 1.0
runtime: dotnet6
build:
  commands:
    build:
      - dotnet publish -c Release -o out
run:
  command: dotnet out/HelloWorldDotNetApp.dll
```

扩展的.NET 配置文件

此示例显示了在.NET 托管运行时中使用所有配置密钥的情况。

> **Note**
>
> 这些示例中使用的运行时版本为 *6.0.9*。您可以将其替换为要使用的版本。有关支持的最新.NET 运行时版本，请参阅the section called “发布信息”。

Example apprunner.yaml

```
version: 1.0
runtime: dotnet6
build:
  commands:
    pre-build:
      - scripts/prebuild.sh
    build:
      - dotnet publish -c Release -o out
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 6.0.9
  command: dotnet out/HelloWorldDotNetApp.dll
  network:
    port: 5000
    env: APP_PORT
  env:
    - name: ASPNETCORE_URLS
      value: "http://*:5000"
```

完整的.NET 应用程序源代码

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

> **Note**
>
> - 运行以下命令创建一个简单的.NET 6 Web 应用程序： `dotnet new web --name HelloWorldDotNetApp -f net6.0`
> - 将添加到已创建的.NET 6 Web 应用程序中。`apprunner.yaml`

Example HelloWorldDotNetApp

```
version: 1.0
runtime: dotnet6
build:
  commands:
    build:
      - dotnet publish -c Release -o out
run:
  command: dotnet out/HelloWorldDotNetApp.dll
  network:
    port: 5000
    env: APP_PORT
  env:
    - name: ASPNETCORE_URLS
      value: "http://*:5000"
```

.NET 运行时版本信息

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

支持的运行时版本-原始 App Runner 版本

运行时名称	次要版本	内含套餐
.NET 6 (dotnet6)	6.0.31	.NET SDK 6.0.423
	6.0.30	.NET SDK 6.0.422
	6.0.29	.NET SDK 6.0.421
	6.0.28	.NET SDK 6.0.420
	6.0.26	.NET SDK 6.0.418
	6.0.25	.NET SDK 6.0.417
	6.0.24	.NET SDK 6.0.416
	6.0.22	.NET SDK 6.0.414
	6.0.21	.NET SDK 6.0.413

运行时名称	次要版本	内含套餐
	6.0.20	.NET SDK 6.0.412
	6.0.19	.NET SDK 6.0.411
	6.0.16	.NET SDK 6.0.408
	6.0.15	.NET SDK 6.0.407
	6.0.14	.NET SDK 6.0.406
	6.0.13	.NET SDK 6.0.405
	6.0.12	.NET SDK 6.0.404
	6.0.11	.NET SDK 6.0.403
	6.0.10	.NET SDK 6.0.402
	6.0.9	.NET SDK 6.0.401

> **Note**
>
> App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

使用 PHP 平台

P AWS App Runner HP 平台提供托管运行时。您可以使用每个运行时来构建和运行带有基于 PHP 版本的 Web 应用程序的容器。当你使用 PHP 运行时时，App Runner 从托管的 PHP 运行时镜像开始。此镜像基于 Amazon Linux Docker 镜像，包含一个 PHP 版本的运行时包和一些工具。App Runner 使用此托管运行时映像作为基础映像，并添加您的应用程序代码来构建 Docker 映像。然后，它会部署此映像以在容器中运行您的 Web 服务。

在使用 App Runner 控制台或 CreateServiceAPI 操作创建服务时，可以为 App Runner 服务指定运行时。您也可以将运行时指定为源代码的一部分。在包含在代码存储库中的 A pp Runner 配置文件中使用`runtime`关键字。托管运行时的命名约定是。 <language-name><major-version>

有关有效的 PHP 运行时名称和版本，请参见the section called “发布信息”。

每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。如果您的应用程序需要托管运行时的特定版本，则可以使用 App Runner 配置文件中的`runtime-version`关键字进行指定。您可以锁定到任何级别的版本，包括主要版本或次要版本。App Runner 仅对服务的运行时进行较低级别的更新。

PHP 运行时的版本语法：*`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例如：`8.1.10`

以下是版本锁定的示例：

- `8.1`— 锁定主要版本和次要版本。App Runner 仅更新补丁版本。
- `8.1.10`— 锁定到特定的补丁版本。App Runner 不会更新你的运行时版本。

> **Important**
>
> 如果您想将 App Runner 服务的代码存储库源目录指定到默认存储库根目录以外的位置，则您的 PHP 托管运行时版本必须是 PHP `8.1.22` 或更高版本。之前的 PHP 运行时版本`8.1.22`只能使用默认的根源目录。

主题

- PHP 运行时配置
- 兼容性
- PHP 运行时示例
- PHP 运行时版本信息

PHP 运行时配置

选择托管运行时时，还必须至少配置生成和运行命令。您可以在创建或更新 App Runner 服务时对其进行配置。您可以使用以下方法之一来执行此操作：

- 使用 App Runner 控制台-在创建过程或配置选项卡的 “配置构建” 部分中指定命令。
- 使用 App Runner API — 调用CreateService或 UpdateServiceAPI 操作。使用CodeConfigurationValues数据类型的`BuildCommand`和`StartCommand`成员来指定命令。
- 使用配置文件-在最多三个构建阶段中指定一个或多个构建命令，并指定一个用于启动应用程序的运行命令。还有其他可选的配置设置。

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 是在创建时直接获取配置设置还是从配置文件中获取配置设置。

兼容性

您可以使用以下 Web 服务器之一在 PHP 平台上运行 App Runner 服务：

- Apache HTTP Server
- NGINX

Apache HTTP ServerNGINX并且与 PHP-FPM 兼容。您可以使用以下任一方法NGINX来启动Apache HTTP Server和：

- S@@ upervisord-有关运行的更多信息 supervisord，请参阅运行 supervisord。
- 启动脚本

有关如何使用 Ap ache HTTP 服务器或 NGINX 在 PHP 平台上配置 App Runn er 服务的示例，请参阅。the section called “完整的 PHP 应用程序源码”

文件结构

`index.php`必须安装在 Web 服务器`root`目录下的`public`文件夹中。

> **Note**
>
> 我们建议将`startup.sh`或`supervisord.conf`文件存储在 Web 服务器的根目录中。确保该`start`命令指向存储`startup.sh`或`supervisord.conf`文件的位置。

如果您正在使用，以下是文件结构的示例supervisord。

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

以下是使用启动脚本时的文件结构示例。

```
/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

我们建议您将这些文件结构存储在为 App Runner 服务指定的代码存储库源目录中。

```
/<sourceDirectory>/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

> **Important**
>
> 如果您想将 App Runner 服务的代码存储库源目录指定到默认存储库根目录以外的位置，则您的 PHP 托管运行时版本必须是 PHP 8.1.22 或更高版本。之前的 PHP 运行时版本8.1.22只能使用默认的根源目录。
>
> 每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。默认情况下，您的服务将使用最新的运行时，除非您在 App Runner 配置文件中使用runtime-version关键字指定版本锁定。

PHP 运行时示例

以下是用于构建和运行 PHP 服务的 App Runner 配置文件的示例。

最小 PHP 配置文件

以下示例是可以与 PHP 托管运行时一起使用的最小配置文件。有关最小配置文件的更多信息，请参阅the section called “配置文件示例”。

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
  command: ./startup.sh
```

扩展 PHP 配置文件

以下示例使用了 PHP 托管运行时的所有配置密钥。

> **Note**
>
> 这些示例中使用的运行时版本是 *8.1.10*。您可以将其替换为要使用的版本。有关支持的最新的 PHP 运行时版本，请参阅the section called “发布信息”。

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - echo example build command for PHP
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 8.1.10
  command: ./startup.sh
  network:
    port: 5000
    env: APP_PORT
```

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

完整的 PHP 应用程序源码

以下示例是 PHP 应用程序源代码，您可以使用这些源代码通过Apache HTTP Server或将其部署到 PHP 运行时服务NGINX。这些示例假设您使用默认文件结构。

Apache HTTP Server使用运行 PHP 平台 supervisord

Example 文件结构

> **Note**
>
> - 该supervisord.conf文件可以存储在存储库中的任何地方。确保该start命令指向supervisord.conf文件的存储位置。
> - index.php必须安装在root目录下的public文件夹中。

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

Example superisord.conf

```
[supervisord]
nodaemon=true

[program:httpd]
command=httpd -DFOREGROUND
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

```
[program:php-fpm]
command=php-fpm -F
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - PYTHON=python2 amazon-linux-extras install epel
      - yum -y install supervisor
run:
  command: supervisord
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

Apache HTTP Server使用运行 PHP 平台 startup script

Example 文件结构

> **Note**
>
> - 该startup.sh文件可以存储在存储库中的任何地方。确保该start命令指向startup.sh文件的存储位置。
> - index.php必须安装在root目录下的public文件夹中。

```
/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

Example startup.sh

```
#!/bin/bash

set -o monitor

trap exit SIGCHLD

# Start apache
httpd -DFOREGROUND &

# Start php-fpm
php-fpm -F &

wait
```

> **Note**
>
> - 在将startup.sh文件提交到 Git 存储库之前，请务必将其保存为可执行文件。chmod +x startup.sh用于为您的startup.sh文件设置执行权限。

- 如果未将startup.sh文件另存为可执行文件，请在apprunner.yaml文件中以build命令chmod +x startup.sh形式输入。

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
  command: ./startup.sh
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

NGINX使用运行 PHP 平台 supervisord

Example 文件结构

Note

- 该supervisord.conf文件可以存储在存储库中的任何地方。确保该start命令指向supervisord.conf文件的存储位置。
- index.php必须安装在root目录下的public文件夹中。

```
/
## public/
#  ## index.php
## apprunner.yaml
## supervisord.conf
```

Example superisord.conf

```
[supervisord]
nodaemon=true

[program:nginx]
command=nginx -g "daemon off;"
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0

[program:php-fpm]
command=php-fpm -F
autostart=true
autorestart=true
stdout_logfile=/dev/stdout
stdout_logfile_maxbytes=0
stderr_logfile=/dev/stderr
stderr_logfile_maxbytes=0
```

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - PYTHON=python2 amazon-linux-extras install epel
      - yum -y install supervisor
run:
  command: supervisord
  network:
    port: 8080
```

```
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
</html>
```

NGINX使用运行 PHP 平台 startup script

Example 文件结构

> **Note**
>
> - 该startup.sh文件可以存储在存储库中的任何地方。确保该start命令指向startup.sh文件的存储位置。
> - index.php必须安装在root目录下的public文件夹中。

```
/
## public/
#  ## index.php
## apprunner.yaml
## startup.sh
```

Example startup.sh

```
#!/bin/bash

set -o monitor

trap exit SIGCHLD
```

```
# Start nginx
nginx -g 'daemon off;' &

# Start php-fpm
php-fpm -F &

wait
```

> **Note**
>
> - 在将startup.sh文件提交到 Git 存储库之前，请务必将其保存为可执行文件。chmod +x startup.sh用于为您的startup.sh文件设置执行权限。
> - 如果未将startup.sh文件另存为可执行文件，请在apprunner.yaml文件中以build命令chmod +x startup.sh形式输入。

Example apprunner.yaml

```
version: 1.0
runtime: php81
build:
  commands:
    build:
      - echo example build command for PHP
run:
  command: ./startup.sh
  network:
    port: 8080
    env: APP_PORT
```

Example index.php

```
<html>
<head> <title>First PHP App</title> </head>
<body>
<?php
    print("Hello World!");
    print("<br>");
?>
</body>
```

```
</html>
```

PHP 运行时版本信息

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

支持的运行时版本-原始 App Runner 版本

运行时名称	次要版本	包含的套餐
PHP 8.1 (php81)	8.1.29	
	8.1.28	
	8.1.27	
	8.1.26	
	8.1.24	
	8.1.22	
	8.1.21	
	8.1.20	
	8.1.19	
	8.1.17	
	8.1.16	
	8.1.14	
	8.1.13	
	8.1.12	
	8.1.10	

> **Note**
>
> App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

使用 Ruby 平台

AWS App Runner Ruby 平台提供托管运行时。每个运行时都可以轻松地使用基于 Ruby 版本的 Web 应用程序构建和运行容器。当你使用 Ruby 运行时时，App Runner 会从托管的 Ruby 运行时镜像开始。此镜像基于 Amazon Linux Docker 镜像，包含某个 Ruby 版本的运行时包和一些工具。App Runner 使用此托管运行时映像作为基础映像，并添加您的应用程序代码来构建 Docker 映像。然后，它会部署此映像以在容器中运行您的 Web 服务。

在使用 App Runner 控制台或 CreateServiceAPI 操作创建服务时，可以为 App Runner 服务指定运行时。您也可以将运行时指定为源代码的一部分。在包含在代码存储库中的 A pp Runner 配置文件中使用`runtime`关键字。托管运行时的命名约定是。 <language-name><major-version>

有关有效的 Ruby 运行时名称和版本，请参见the section called “发布信息”。

每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。如果您的应用程序需要托管运行时的特定版本，则可以使用 App Runner 配置文件中的`runtime-version`关键字进行指定。您可以锁定到任何级别的版本，包括主要版本或次要版本。App Runner 仅对服务的运行时进行较低级别的更新。

Ruby 运行时的版本语法：*`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例如：`3.1.2`

以下示例演示了版本锁定：

- `3.1`— 锁定主要版本和次要版本。App Runner 仅更新补丁版本。
- `3.1.2`— 锁定到特定的补丁版本。App Runner 不会更新你的运行时版本。

主题

- Ruby 运行时配置
- Ruby 运行时示例

- Ruby 运行时发布信息

Ruby 运行时配置

选择托管运行时时，还必须至少配置生成和运行命令。您可以在创建或更新 App Runner 服务时对其进行配置。您可以使用以下方法之一来执行此操作：

- 使用 App Runner 控制台-在创建过程或配置选项卡的 “配置构建” 部分中指定命令。
- 使用 App Runner API-调用CreateService或 UpdateServiceAPI 操作。使用CodeConfigurationValues数据类型的`BuildCommand`和`StartCommand`成员来指定命令。
- 使用配置文件-在最多三个构建阶段中指定一个或多个构建命令，并指定一个用于启动应用程序的运行命令。还有其他可选的配置设置。

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 是在创建时直接获取配置设置还是从配置文件中获取配置设置。

Ruby 运行时示例

以下示例显示了用于构建和运行 Ruby 服务的 App Runner 配置文件。

最小 Ruby 配置文件

此示例显示了一个可以与 Ruby 托管运行时配合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 “配置文件示例”。

Example apprunner.yaml

```
version: 1.0
runtime: ruby31
build:
  commands:
    build:
      - bundle install
run:
  command: bundle exec rackup --host 0.0.0.0 -p 8080
```

扩展的 Ruby 配置文件

此示例显示了在 Ruby 托管运行时中使用所有配置密钥的情况。

> **Note**
>
> 这些示例中使用的运行时版本是 *3.1.2*。您可以将其替换为要使用的版本。有关支持的最新的 Ruby 运行时版本，请参阅the section called “发布信息”。

Example apprunner.yaml

```
version: 1.0
runtime: ruby31
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - bundle install
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.1.2
  command: bundle exec rackup --host 0.0.0.0 -p 4567
  network:
    port: 4567
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

完整的 Ruby 应用程序源码

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

Example server.rb

```
# server.rb
require 'sinatra'

get '/' do
  'Hello World!'
```

```
end
```

Example config.ru

```
# config.ru

require './server'

run Sinatra::Application
```

Example Gemfile

```
# Gemfile
source 'https://rubygems.org (https://rubygems.org/)'

gem 'sinatra'
gem 'puma'
```

Example apprunner.yaml

```
version: 1.0
runtime: ruby31
build:
  commands:
    build:
      - bundle install
run:
  command: bundle exec rackup --host 0.0.0.0 -p 4567
  network:
    port: 4567
    env: APP_PORT
```

Ruby 运行时发布信息

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

支持的运行时版本-原始 App Runner 版本

运行时名称	次要版本	包含的套餐
Ruby 3.1 (ruby31)	3.1.6	SQLite 3.46.0

运行时名称	次要版本	包含的套餐
	3.1.4	SQLite 3.46.0
	3.1.3	SQLite 3.41.0
	3.1.2	SQLite 3.39.4

> **Note**
>
> App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

使用 Go 平台

G AWS App Runner o 平台提供托管运行时。每个运行时都可以轻松构建和运行基于 Go 版本的 Web 应用程序的容器。当你使用 Go 运行时时，App Runner 从托管 Go 运行时镜像开始。此镜像基于 Amazon Linux Docker 镜像，包含某个 Go 版本的运行时包和一些工具。App Runner 使用此托管运行时映像作为基础映像，并添加您的应用程序代码来构建 Docker 映像。然后，它会部署此映像以在容器中运行您的 Web 服务。

在使用 App Runner 控制台或 CreateServiceAPI 操作创建服务时，可以为 App Runner 服务指定运行时。您也可以将运行时指定为源代码的一部分。在包含在代码存储库中的 A pp Runner 配置文件中使用`runtime`关键字。托管运行时的命名约定是。 <language-name><major-version>

有关有效的 Go 运行时名称和版本，请参阅the section called “发布信息”。

每次部署或服务更新时，App Runner 都会将服务的运行时更新到最新版本。如果您的应用程序需要托管运行时的特定版本，则可以使用 App Runner 配置文件中的`runtime-version`关键字进行指定。您可以锁定到任何级别的版本，包括主要版本或次要版本。App Runner 仅对服务的运行时进行较低级别的更新。

Go 运行时的版本语法：*`major`*`[.`*`minor`*`[.`*`patch`*`]]`

例如：`1.18.7`

以下示例演示了版本锁定：

- `1.18`— 锁定主要版本和次要版本。App Runner 仅更新补丁版本。
- `1.18.7`— 锁定到特定的补丁版本。App Runner 不会更新你的运行时版本。

主题

Go 运行时配置

选择托管运行时时，还必须至少配置生成和运行命令。您可以在创建或更新 App Runner 服务时对其进行配置。您可以使用以下方法之一来执行此操作：

- 使用 App Runner 控制台-在创建过程或配置选项卡的 “配置构建” 部分中指定命令。
- 使用 App Runner API-调用CreateService或 UpdateServiceAPI 操作。使用CodeConfigurationValues数据类型的`BuildCommand`和`StartCommand`成员来指定命令。
- 使用配置文件-在最多三个构建阶段中指定一个或多个构建命令，并指定一个用于启动应用程序的运行命令。还有其他可选的配置设置。

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 是在创建时直接获取配置设置还是从配置文件中获取配置设置。

Go 运行时示例

以下示例显示了用于构建和运行 Go 服务的 App Runner 配置文件。

最小 Go 配置文件

此示例显示了可以与 Go 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 “配置文件示例”。

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
    build:
```

```
      - go build main.go
run:
  command: ./main
```

扩展 Go 配置文件

此示例显示了在 Go 托管运行时中使用所有配置密钥的情况。

> **Note**
>
> 这些示例中使用的运行时版本是 *1.18.7*。您可以将其替换为要使用的版本。有关支持的最新 Go 运行时版本，请参阅the section called “发布信息”。

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
     pre-build:
      - scripts/prebuild.sh
    build:
      - go build main.go
    post-build:
      - scripts/postbuild.sh
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 1.18.7
  command: ./main
  network:
    port: 3000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

完整的 Go 应用程序源码

这些示例显示了可以部署到 Go 运行时服务的完整 Go 应用程序的源代码。

Example main.go

```
package main
import (
    "fmt"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        fmt.Fprint(w, "<h1>Welcome to App Runner</h1>")
    })
    fmt.Println("Starting the server on :3000...")
    http.ListenAndServe(":3000", nil)
}
```

Example apprunner.yaml

```
version: 1.0
runtime: go1
build:
  commands:
    build:
      - go build main.go
run:
  command: ./main
  network:
    port: 3000
    env: APP_PORT
```

Go 运行时版本信息

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

支持的运行时版本-原始 App Runner 版本

运行时名称	次要版本	包含的套餐
Go 1 (go1)	1.18.10	
	1.18.9	

运行时名称	次要版本	包含的套餐
	1.18.8	
	1.18.7	

> **Note**
>
> App Runner 为最近发布的特定主要运行时提供了修订后的构建流程。因此，您将在本文档的某些部分中看到对修订后的 App Runner 版本和原始 App Runner 版本的引用。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

为 App Runner 开发应用程序代码

本章讨论了在开发或迁移要部署到的应用程序代码时应考虑的运行时信息和开发指南 AWS App Runner。

运行时信息

无论您是提供容器镜像还是 App Runner 为您构建容器镜像，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（如果有的话）无关。为方便起见，我们为各种编程平台提供特定于平台的托管运行时，以简化应用程序构建过程和抽象图像创建。
- 网络请求 — App Runner 为容器实例提供对 HTTP 1.0 和 HTTP 1.1 的支持。有关配置服务的更多信息，请参阅the section called “配置”。您无需实现对 HTTPS 安全流量的处理。App Runner 会将所有传入的 HTTP 请求重定向到相应的 HTTPS 您无需配置任何设置即可重定向 HTTP Web 请求。App Runner 会在向您的应用程序容器实例传递请求之前终止 TLS。

  > **Note**
  >
  > - HTTP 请求总共有 120 秒的请求超时限制。120 秒包括应用程序读取请求（包括正文）和完成 HTTP 响应的写入所花费的时间。
  > - 请求读取和响应超时限制取决于您使用的应用程序。这些应用程序可能有自己的内部超时，例如 Python 的 HTTP 服务器 Gunicorn 有 30 秒的默认超时限制。在这种情况下，应用程序的超时限制会覆盖 App Runner 的 120 秒超时限制。
  > - 您无需配置 TLS 密码套件或任何其他参数，因为 App Runner 是一项完全托管的服务，可以为您管理 TLS 终止。

- 无状态应用程序-当前 App Runner 不支持有状态的应用程序。因此，App Runner 不保证状态持续时间超过处理单个传入 Web 请求的持续时间。
- 存储 — App Runner 会根据传入的流量自动向上或向下扩展 App Runner 应用程序的实例。您可以为 App Runner 应用程序配置自动缩放选项。由于处理 Web 请求的当前活动实例数量取决于传入的流量，因此 App Runner 无法保证这些文件在处理单个请求后仍能持续存在。因此，App Runner 将容器实例中的文件系统实现为临时存储，这意味着文件是瞬态的。例如，当您暂停和恢复 App Runner 服务时，文件不会保留。

App Runner 为您提供 3 GB 的临时存储空间，并将 3 GB 临时存储空间的一部分用于实例上的拉取、压缩和未压缩容器映像。剩余的临时存储空间可供您的 App Runner 服务使用。但是，由于其无国籍性质，这不是永久存储。

> **Note**
>
> 在某些情况下，存储文件确实会跨请求保留。例如，如果下一个请求落在同一个实例上，则存储文件将保留。在某些情况下，跨请求存储文件的持久性可能很有用。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些文件）。这可能会加快 future 请求的处理速度，但不能保证速度的提高。您的代码不应假设在之前的请求中下载的文件仍然存在。
>
> 要使用高吞吐量、低延迟的内存数据存储来保证缓存，请使用诸如 Amazon ElastiCache 之类的服务。

- 环境变量-默认情况下，App Runner 使`PORT`环境变量在您的容器实例中可用。您可以使用端口信息配置变量值，也可以添加自定义环境变量和值。您也可以将存储在AWS Secrets Manager或AWS Systems Manager 参数存储中的敏感数据引用为环境变量。有关创建环境变量的更多信息，请参阅参考环境变量。
- 实例角色-如果您的应用程序代码使用 AWS 服务 API 或其中一个 AWS 软件开发工具包调用任何服务，请使用 AWS Identity and Access Management (IAM) 创建实例角色。然后，在创建 App Runner 服务时将其附加到该服务。在您的实例角色中包含您的代码所需的所有 AWS 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。您的代码应假设任何状态都不会持续超过处理单个传入的 Web 请求的持续时间。
- 删除临时文件-创建文件时，它们存储在文件系统中，占用服务的部分存储分配。为避免 out-of-storage 出错，请勿长时间保留临时文件。在做出文件缓存决策时，请平衡存储大小和请求处理速度。
- 实例启动 — App Runner 提供五分钟的实例启动时间。您的实例必须在其配置的侦听端口上侦听请求，并在启动后的五分钟内恢复正常。在启动期间，将根据您的 vCPU 配置为 App Runner 实例分配虚拟 CPU (vCPU)。有关可用 vCPU 配置的更多信息，请参阅。the section called “App Runner 支持的配置”

实例成功启动后，它将进入空闲状态并等待请求。您根据实例启动持续时间付费，每次启动实例的最低费用为一分钟。有关定价的信息，请参阅 AWS App Runner 定价。

使用 App Runner 控制台

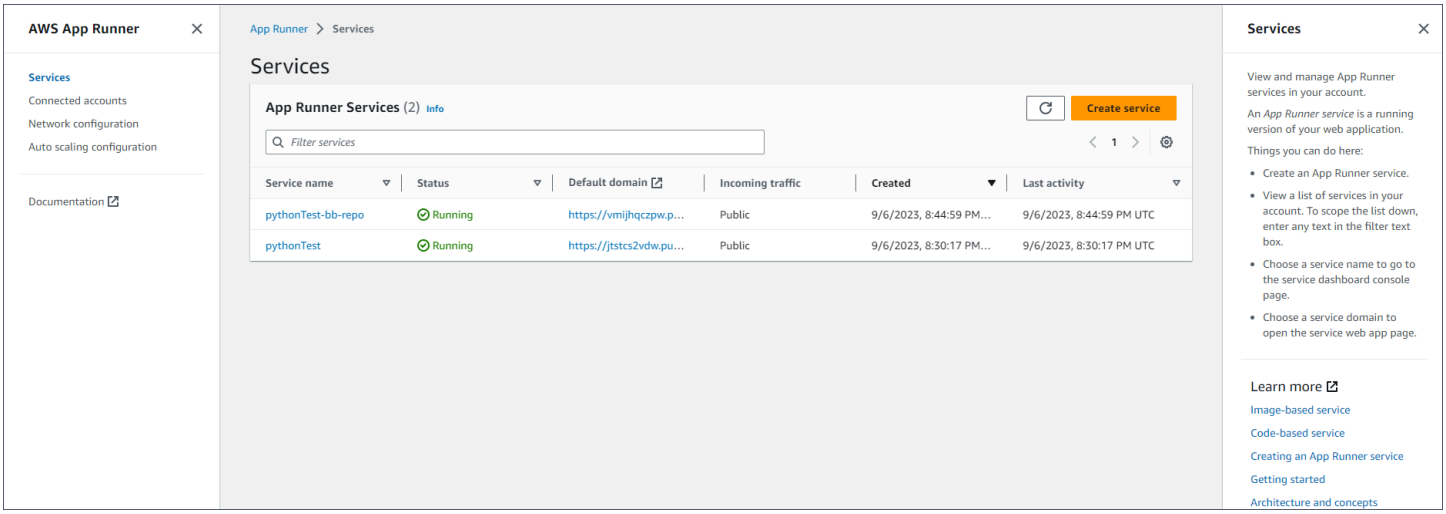
使用 AWS App Runner 控制台创建、管理和监控您的 App Runner 服务和相关资源，例如关联账户。您可以查看现有服务、创建新服务以及配置服务。您可以查看 App Runner 服务的状态以及查看日志、监控活动和跟踪指标。您也可以导航到您的服务网站或源存储库。

以下各节描述了控制台的布局和功能，并为您指出了相关信息。

整体控制台布局

App Runner 控制台有三个区域。从左到右：

- 导航窗格-可以折叠或展开的侧面窗格。使用它来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任务。
- 帮助窗格-侧面窗格提供更多信息。将其展开以获取有关您所在页面的帮助。或者在控制台页面上选择任何 “信息” 链接以获取上下文帮助。



“服务” 页面

服务页面列出了您账户中的 App Runner 服务。您可以使用筛选器文本框缩小列表的范围。

前往 “服务” 页面

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 在导航窗格中，选择服务。

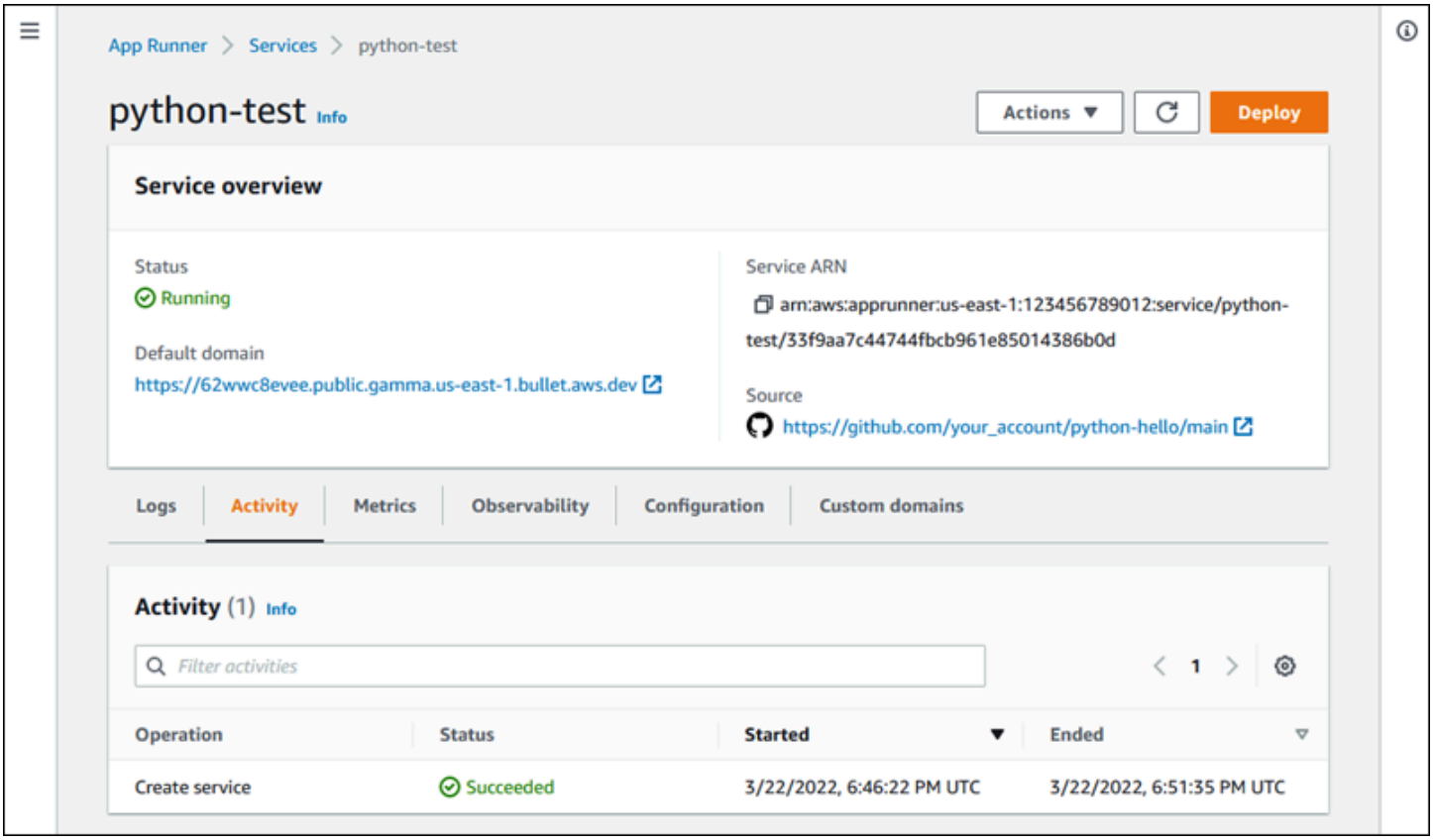
你可以在这里做的事情：

- 创建 App Runner 服务。有关更多信息，请参阅 the section called “创建”。
- 选择服务名称以转到服务控制台页面。
- 选择服务域以打开服务 Web 应用程序页面。

服务仪表板页面

您可以从服务仪表板页面查看有关 App Runner 服务的信息并对其进行管理。在页面顶部，您可以看到服务名称。

要进入服务控制面板，请导航到服务页面（参见上一节），然后选择您的 App Runner 服务。



服务概述部分提供有关 App Runner 服务和您的应用程序的基本详细信息。你可以在这里做的事情：

- 查看服务详情，例如状态、运行状况和 ARN。
- 导航到默认域，即 App Runner 为在您的服务中运行的 Web 应用程序提供的域。这是 App Runner 拥有的`awsapprunner.com`域中的一个子域名。
- 导航到部署到该服务的源存储库。

- 开始将源存储库部署到您的服务。
- 暂停、恢复和删除您的服务。

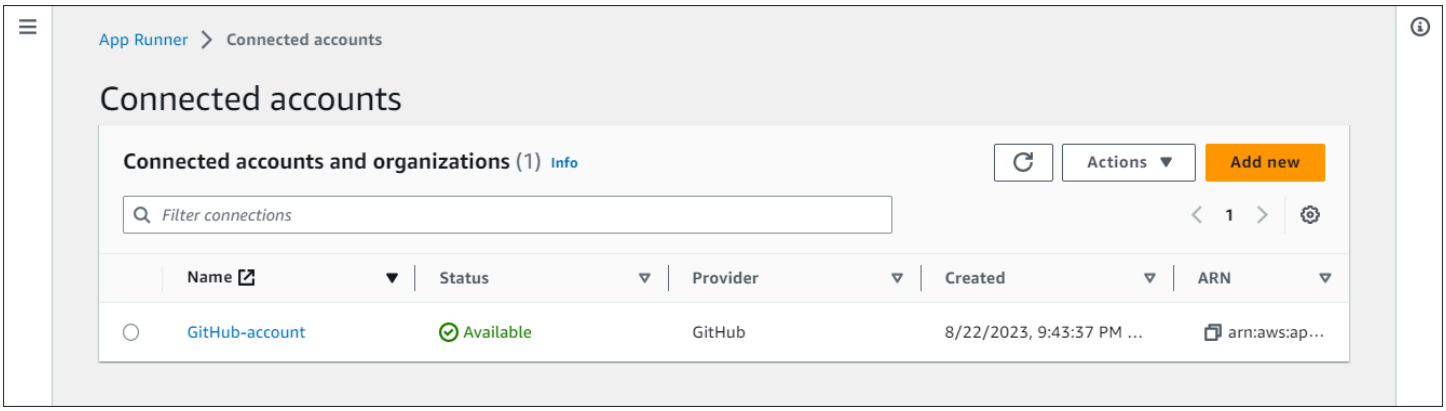
服务概述下方的选项卡用于服务管理和可观察性。

关联账户页面

关联账户页面列出了 App Runner 与您账户中源代码存储库提供商的连接。您可以使用筛选器文本框缩小列表的范围。有关关联账户的更多信息，请参阅the section called “连接”。

前往 “关联账户” 页面

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 在导航窗格中，选择已关联账户。



你可以在这里做的事情：

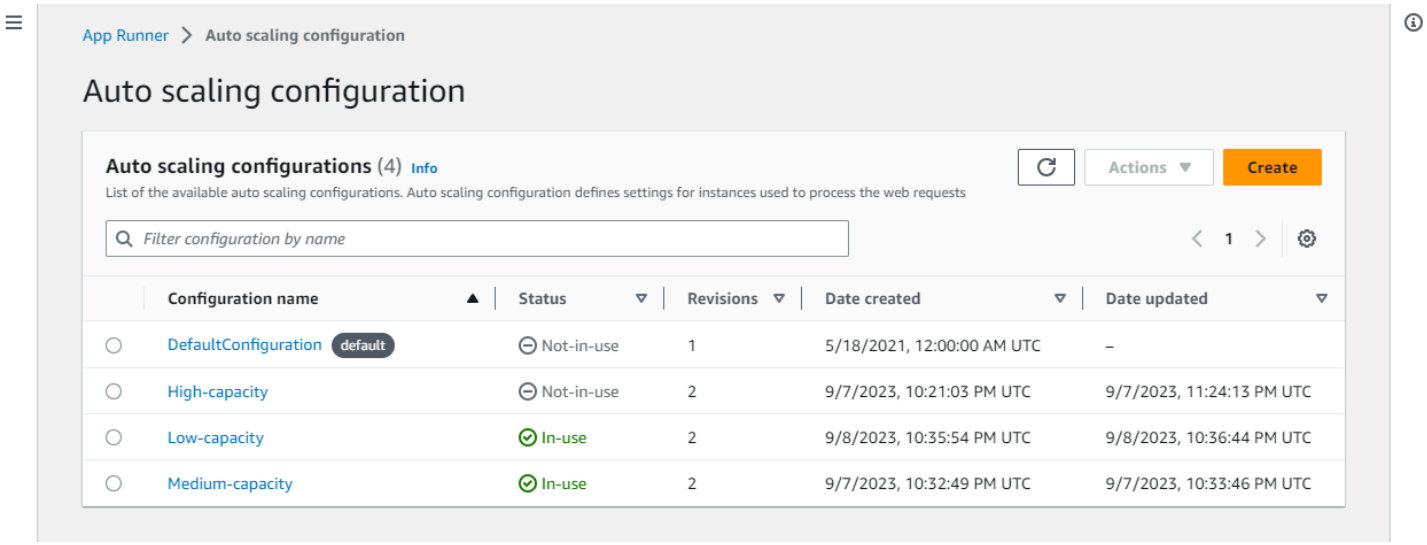
- 查看您账户中存储库提供商的连接列表。要缩小列表范围，请在筛选器文本框中输入任意文本。
- 选择连接名称以转到相关的提供商帐户或组织。
- 选择一个连接以完成刚刚建立的连接（作为创建服务的一部分）的握手，或者删除该连接。

自动伸缩配置页面

A uto Scaling 配置页面列出了您在账户中设置的自动伸缩配置。您可以配置一些参数来调整自动缩放行为，并将它们保存为不同的配置，以便稍后分配给一个或多个 App Runner 服务。您可以使用筛选器文本框缩小列表的范围。有关 auto Scaling 配置的更多信息，请参阅管理服务的自动缩放。

要进入自动缩放配置页面

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 在导航窗格中，选择自动伸缩配置。



你可以在这里做的事情：

- 查看您账户中的现有 auto Scaling 配置列表。
- 创建新的 auto Scaling 配置或对现有配置进行修订。
- 将 auto Scaling 配置设置为您创建的新服务的默认配置。
- 删除配置。
- 选择配置名称以导航到 “自动缩放修订版” 面板以管理修订版。

管理你的 App Runner 服务

本章介绍如何管理您的 AWS App Runner 服务。在本章中，您将学习如何管理服务的生命周期：创建、配置和删除服务，将新的应用程序版本部署到您的服务，以及通过暂停和恢复服务来控制 Web 服务的可用性。您还将学习如何管理服务的其他方面，例如连接和 auto Scaling。

主题

创建 App Runner 服务

AWS App Runner 自动从容器映像或源代码存储库过渡到可自动扩展的正在运行的 Web 服务。您可以将 App Runner 指向您的源图像或代码，只需指定少量必需的设置。App Runner 可根据需要构建您的应用程序，预置计算资源，然后部署您的应用程序以在这些资源上运行。

创建服务时，App Runner 会创建一个服务资源。在某些情况下，您可能需要提供连接资源。如果您使用 App Runner 控制台，则控制台会隐式创建连接资源。有关 App Runner 资源类型的更多信息，请参阅the section called “应用程序运行器资源”。这些资源类型都有与您的账户关联的配额 AWS 区域。有关更多信息，请参阅 the section called “应用程序运行器资源配额”。

根据来源类型和提供商的不同，创建服务的过程会有细微的差异。本主题介绍了创建这些源类型的不同过程，以便您可以按照适合自己情况的方式进行操作。有关使用代码示例开始基本过程的信息，请参见开始使用。

先决条件

在创建 App Runner 服务之前，请务必完成以下操作：

- 完成中的设置步骤设置。
- 确保您的应用程序源已准备就绪。您可以使用 Bitbuck et 中的GitHub代码存储库或亚马逊弹性容器注册表 (Amazon ECR) Container Registry 中的容器镜像来创建 App Runn er 服务。

创建服务

本节介绍两种 App Runner 服务类型的创建过程：基于源代码和基于容器镜像。

> **Note**
>
> 如果您为服务创建出站流量 VPC 连接器，则随后的服务启动过程将出现一次性延迟。您可以在创建新服务时或之后通过服务更新为新服务设置此配置。有关更多信息，请参阅一次性延迟本指南的 “与 App Runner 联网” 一章。

从代码仓库创建服务

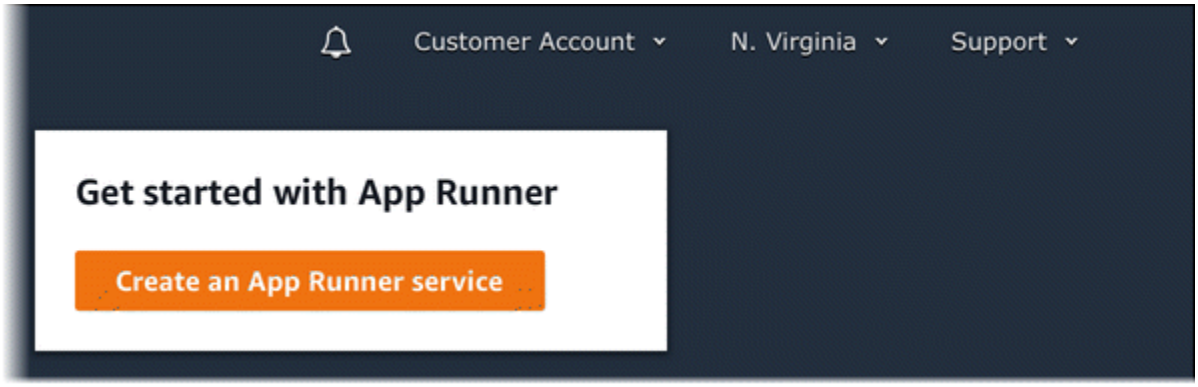
以下各节介绍当您的源代码是GitHub或 Bitbuck et 中的代码存储库时如何创建 App Runner 服务。使用代码存储库时，App Runner 必须连接到提供商组织或帐户。因此，您需要帮助建立这种连接。有关 App Runner 连接的更多信息，请参阅the section called “连接”。

创建服务时，App Runner 会生成一个 Docker 镜像，其中包含您的应用程序代码和依赖项。然后，它会启动一项运行该镜像的容器实例的服务。

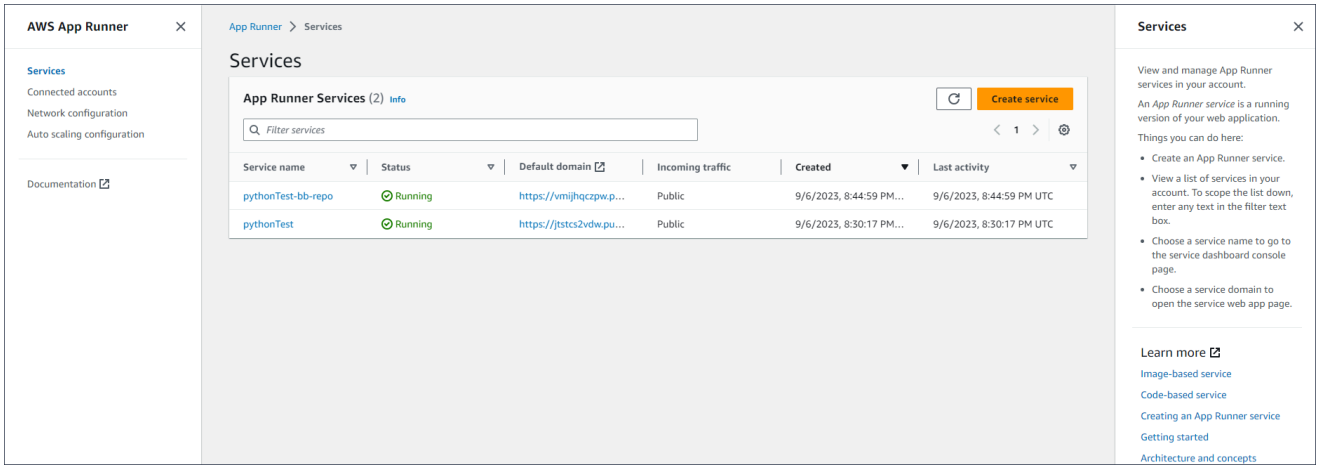
使用 App Runner 控制台从代码创建服务

使用控制台创建 App Runner 服务

1. 配置您的源代码。
   a. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
   b. 如果还 AWS 账户 没有任何 App Runner 服务，则会显示主机主页。选择创建 App Runner 服务。



如果 AWS 账户 已有服务，则会显示包含您的服务列表的 “服务” 页面。选择 Create service。



c. 在 “源代码和部署” 页面的 “源” 部分中，为 “存储库类型” 选择 “源代码存储库”。

d. 选择提供商类型。选择其中一个GitHub或 Bitbucket。

e. 接下来，为该提供商选择您之前使用过的帐户或组织，或者选择新增。然后，完成提供您的代码存储库凭据并选择要连接的账户或组织的过程。

f. 在 “存储库” 中，选择包含您的应用程序代码的存储库。

g. 对于 Bran ch，选择要部署的分支。

h. 在源目录中，输入源存储库中用于存储应用程序代码和配置文件的目录。

> **Note**
>
> 生成和启动命令从您指定的源目录执行。App Runner 将路径视为从根目录开始的绝对路径。如果您未在此处指定值，则该目录默认为存储库根目录。

2. 配置您的部署。

a. 在 “部署设置” 部分，选择 “手动” 或 “自动”。

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

b. 选择下一步。

Source and deployment Info

Choose the source for your App Runner service and the way it's deployed.

Source and deployment

Source

Repository type

- ○ Container registry
  Deploy your service using a container image stored in a container registry.
- ◉ Source code repository
  Deploy your service using the code hosted in a source repository.

Provider

Choose the provider where you host your code repository.

GitHub

Github Connection Info

App Runner deploys your source code by installing an app called "AWS Connector for GitHub" in your account. You can install this app in your main GitHub account or in a GitHub organization.

myGitHub | Add new

Repository

python-hello

Branch

main

Source directory

The build and start commands will execute in this directory. App Runner defaults to the root directory if you don't specify a directory here.

/

Leading and trailing slashes ("/") are not required. Valid examples: "apps/targetapp", "/apps/targetapp/", "/targetapp"

Deployment settings

Deployment trigger

- ○ Manual
  Start each deployment yourself using the App Runner console or AWS CLI.
- ◉ Automatic
  Every push to this branch that affects files in the specified **Source directory** deploys a new version of your service.

Cancel Next

3. 配置应用程序构建。

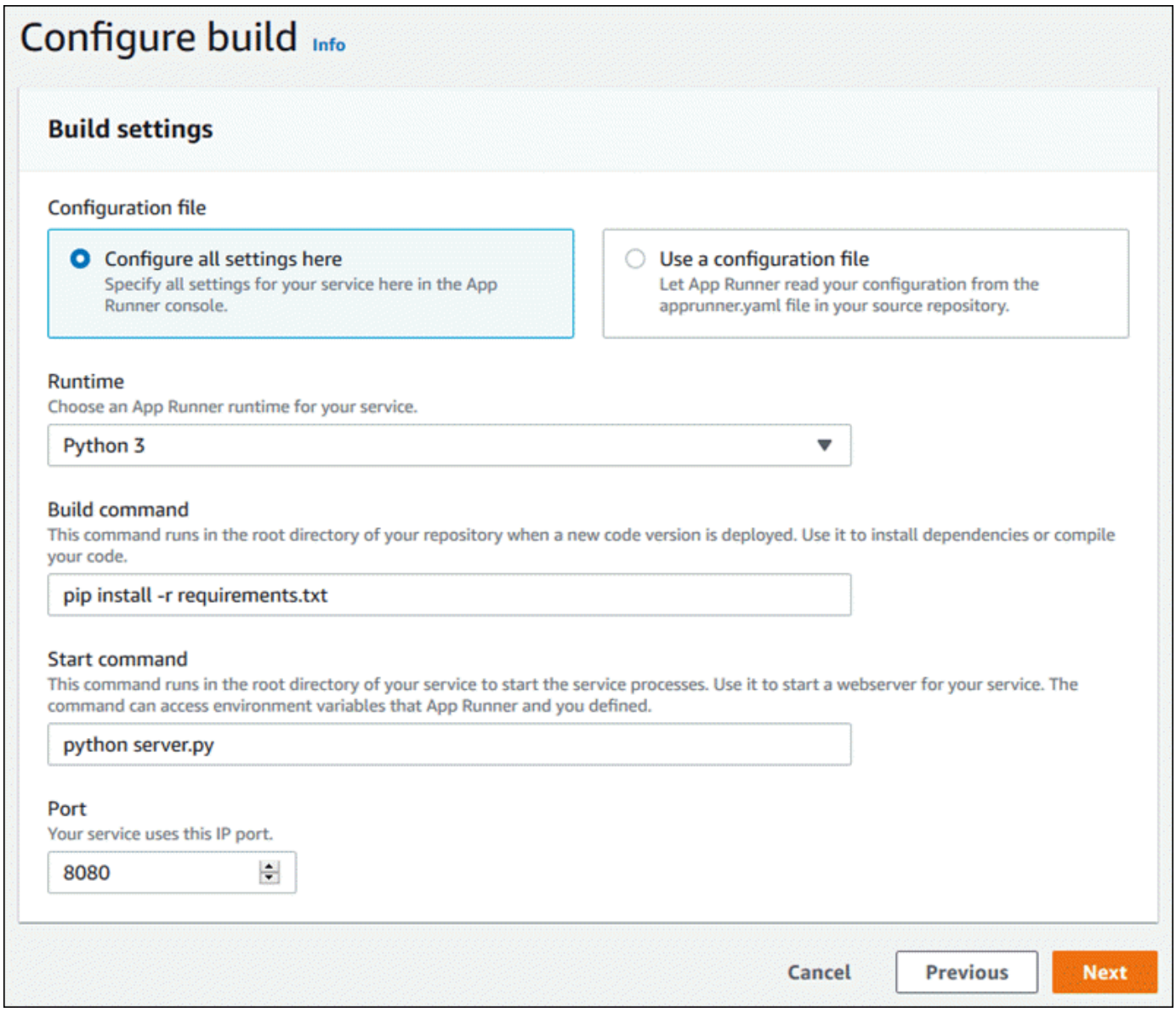
   a. 在 “配置构建” 页面上，对于配置文件，如果您的存储库中不包含 App Runner 配置文件，请选择在此处配置所有设置，或者如果包含 App Runner 配置文件，则选择使用配置文件。

   > **Note**
   >
   > App Runner 配置文件是一种将编译配置作为应用程序源的一部分进行维护的方法。当你提供一个值时，App Runner 会从文件中读取一些值，但不允许你在控制台中设置这些值。

   b. 提供以下编译设置：

   - 运行时-为您的应用程序选择特定的托管运行时。
   - 生成命令-输入从源代码构建应用程序的命令。这可能是您的代码中提供的特定语言工具或脚本。
   - 启动命令-输入启动 Web 服务的命令。
   - 端口-输入您的 Web 服务监听的 IP 端口。

   c. 选择下一步。



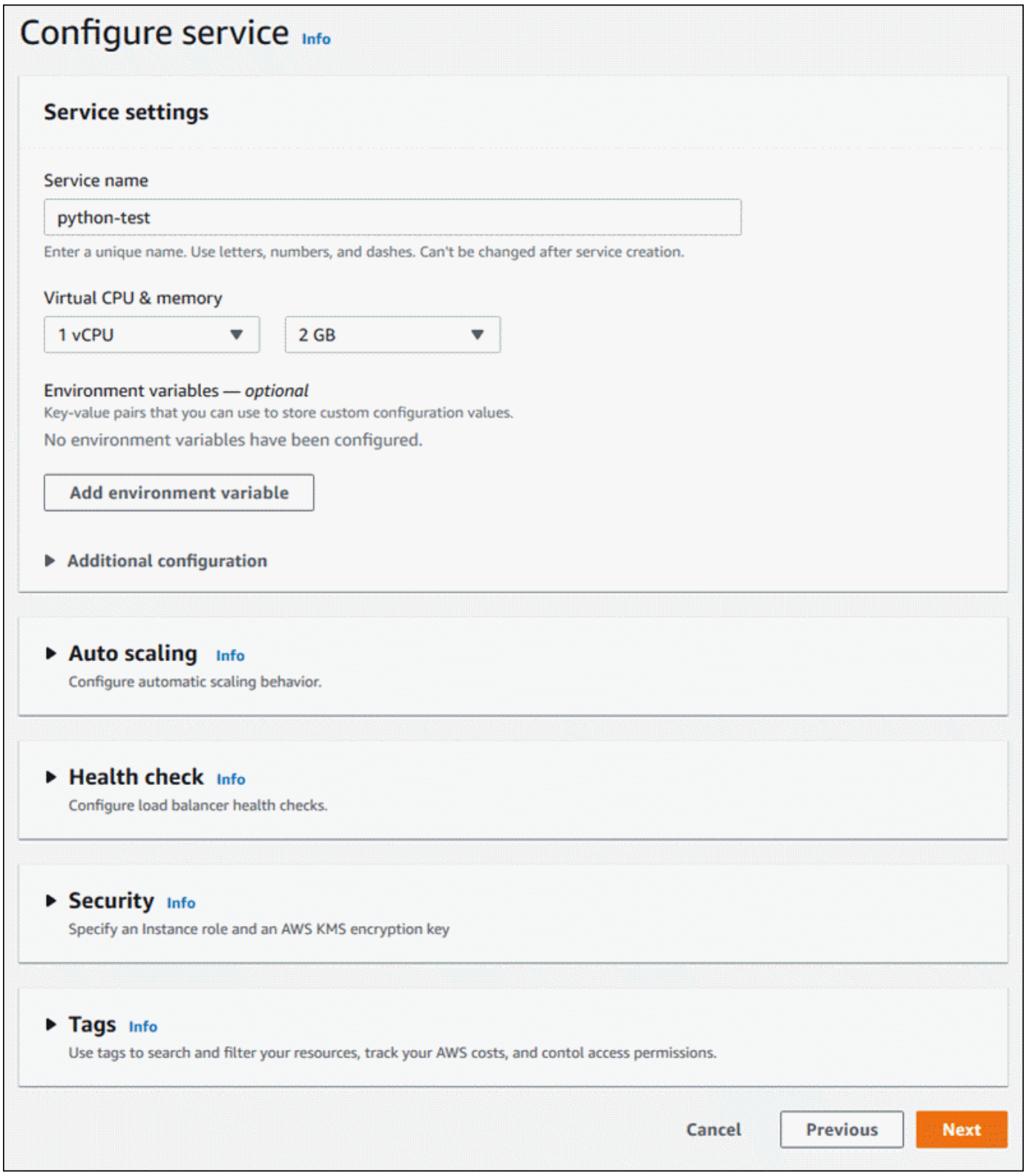
4. 配置您的服务。

   a. 在配置服务页面的服务设置部分，输入服务名称。

   > **Note**
   >
   > 所有其他服务设置要么是可选的，要么具有控制台提供的默认设置。

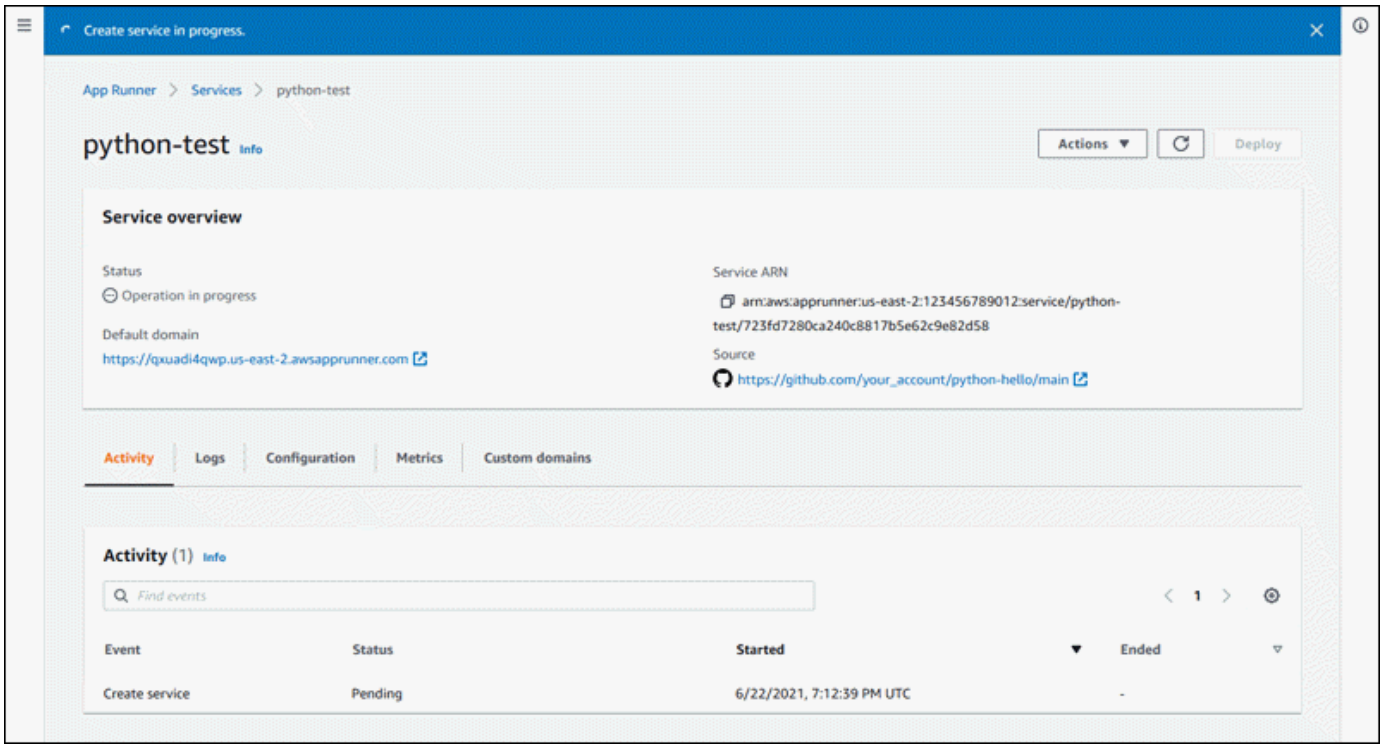
   b. （可选）更改或添加其他设置以满足您的应用程序要求。

   c. 选择下一步。



5. 在“查看并创建”页面上，验证您输入的所有详细信息，然后选择“创建并部署”。

结果：如果成功创建服务，控制台将显示服务仪表板，其中包含新服务的服务概述。



6. 验证您的服务是否正在运行。

   a. 在服务仪表板页面上，等到服务状态变为 “正在运行”。

   b. 选择默认域值。这是您的服务网站的网址。

   c. 使用您的网站并验证其是否正常运行。

使用 App Runner API 从代码创建服务或 AWS CLI

要使用 App Runner API 或创建服务 AWS CLI，请调用 `CreateService` API 操作。有关更多信息和示例，请参阅CreateService。如果这是您首次使用特定组织或账户为源代码存储库（GitHub 或 Bitbucket）创建服务，请先调用CreateConnection。这会在 App Runner 与存储库提供商的组织或帐户之间建立连接。有关 App Runner 连接的更多信息，请参阅the section called “连接”。

如果调用返回成功响应，且显示了 S ervic e 对象`"Status": "CREATING"`，则您的服务将开始创建。

有关调用的示例，请参阅《AWS App Runner API 参考》中的 “创建源代码存储库服务”

从 Amazon ECR 镜像创建服务

以下各节介绍当您的来源是存储在 A mazon ECR 中的容器映像时，如何创建 App R unner 服务。亚马逊 ECR 是一个。 AWS 服务因此，要基于 Amazon ECR 映像创建服务，您需要为 App Runner 提供一个包含必要的 Amazon ECR 操作权限的访问角色。

> **Note**
>
> 存储在 Amazon ECR Public 中的图像是公开的。因此，如果您的图像存储在 Amazon ECR Public 中，则不需要访问角色。

创建服务时，App Runner 会启动一项服务，该服务会运行您提供的映像的容器实例。在这种情况下，没有构建阶段。

有关更多信息，请参阅 基于图像的服务。

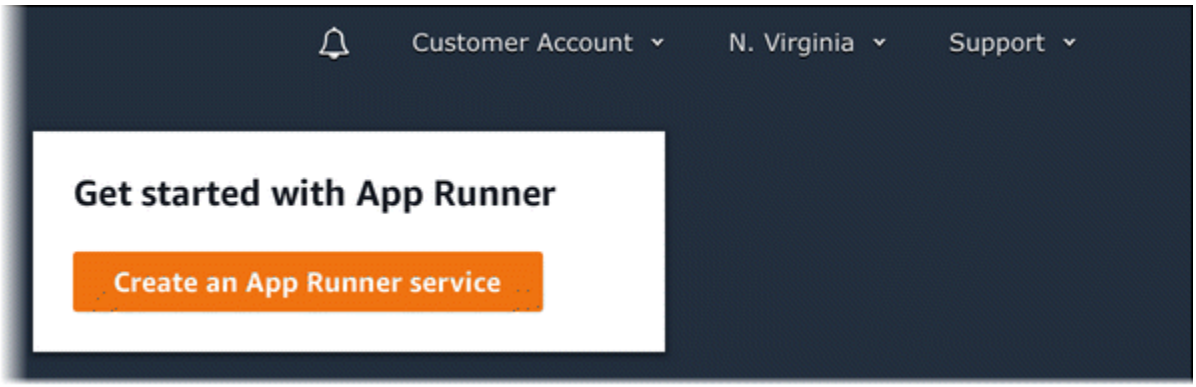
使用 App Runner 控制台从图像创建服务

使用控制台创建 App Runner 服务

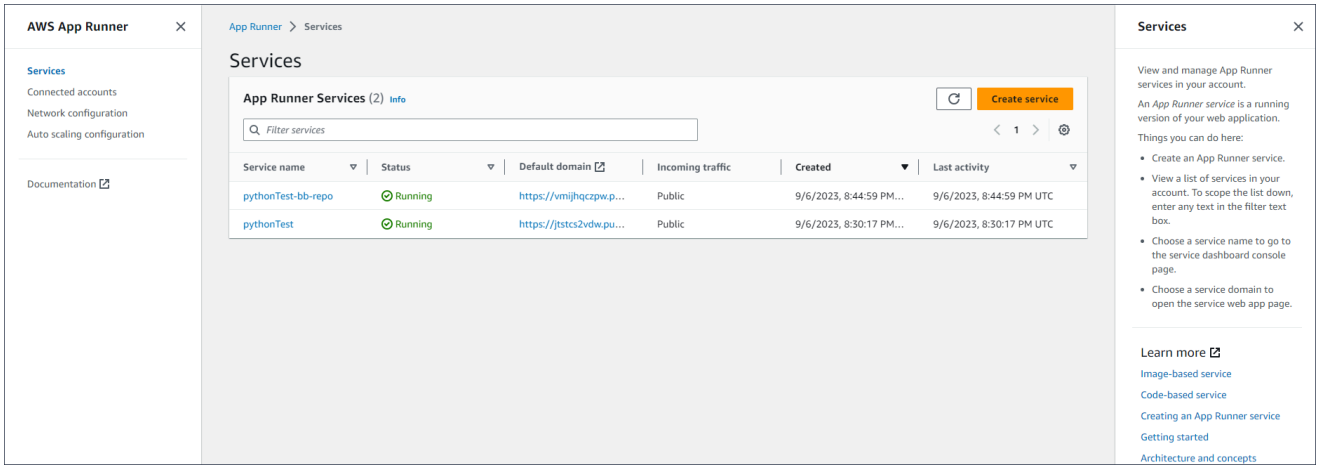
1. 配置您的源代码。

   a. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。

   b. 如果还 AWS 账户 没有任何 App Runner 服务，则会显示主机主页。选择创建 App Runner 服务。



   如果 AWS 账户 已有服务，则会显示包含您的服务列表的 “服务” 页面。选择 Create service。



c. 在源和部署页面的来源部分中，对于存储库类型，选择容器注册表。

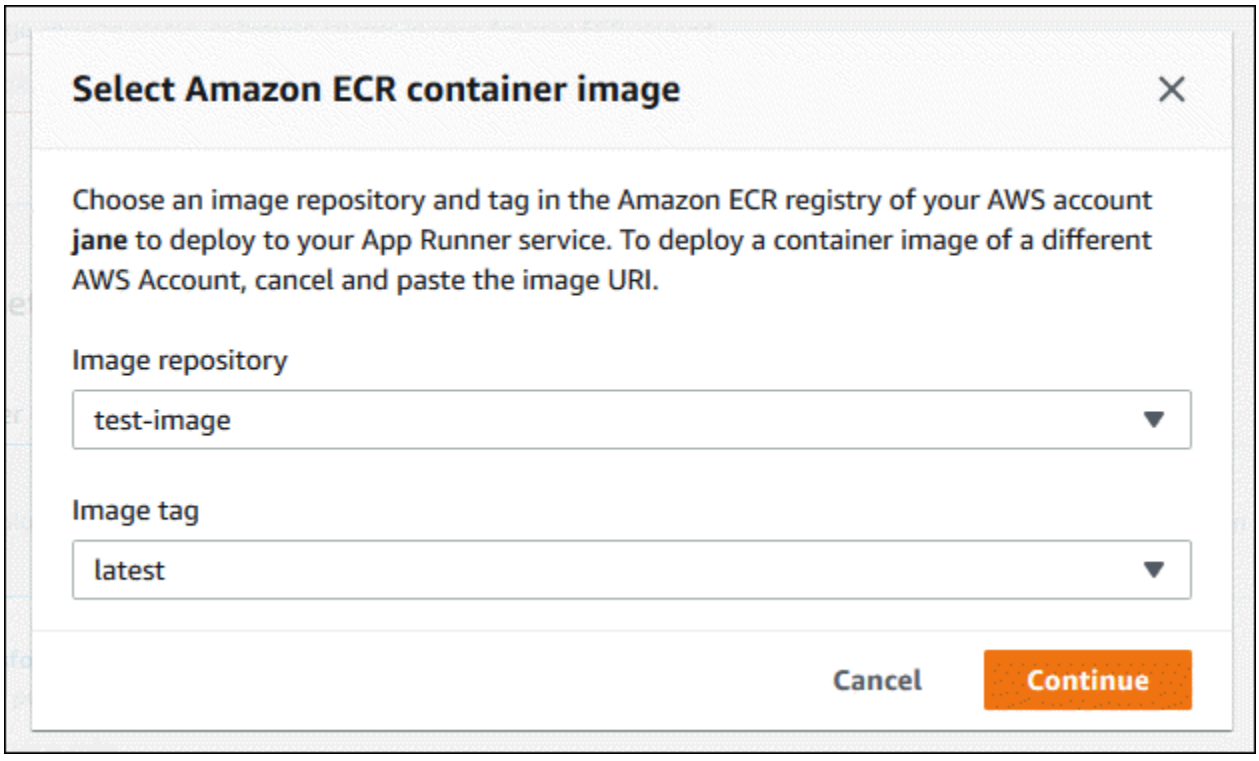
d. 对于提供商，请选择存储图像的提供商：

- 亚马逊 ECR — 存储在亚马逊 ECR 中的私有镜像。
- Amazon ECR Pub lic — 存储在 Amazon ECR Public 中的公开可读图片。

e. 对于容器镜像 URI，请选择浏览。

f. 在选择 Amazon ECR 容器镜像对话框中，对于镜像存储库，选择包含您的镜像的存储库。

g. 对于图像标签，选择要部署的特定映像标签（例如，最新），然后选择继续。



2. 配置您的部署。

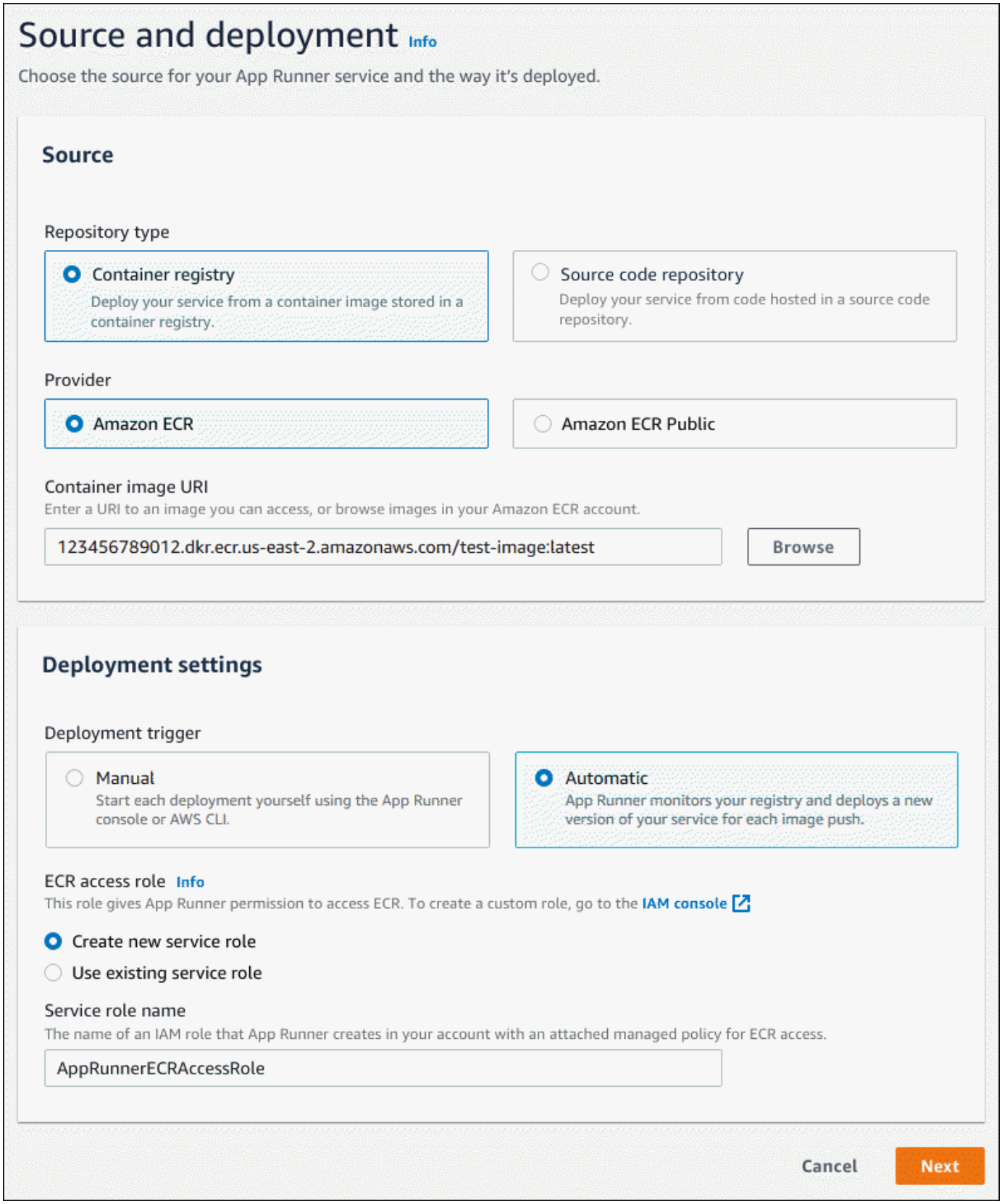
   a. 在 “部署设置” 部分，选择 “手动” 或 “自动”。

   > **Note**
   >
   > App Runner 不支持自动部署 Amazon ECR 公共镜像，也不支持自动部署 Amazon ECR 存储库中与您的服务所在 AWS 账户不同的账户。

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

   b. [Amazon ECR 提供商] 对于 ECR 访问角色，请选择账户中的现有服务角色或选择创建新角色。如果您使用的是手动部署，则还可以在部署时选择使用 IAM 用户角色。

   c. 选择下一步。


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

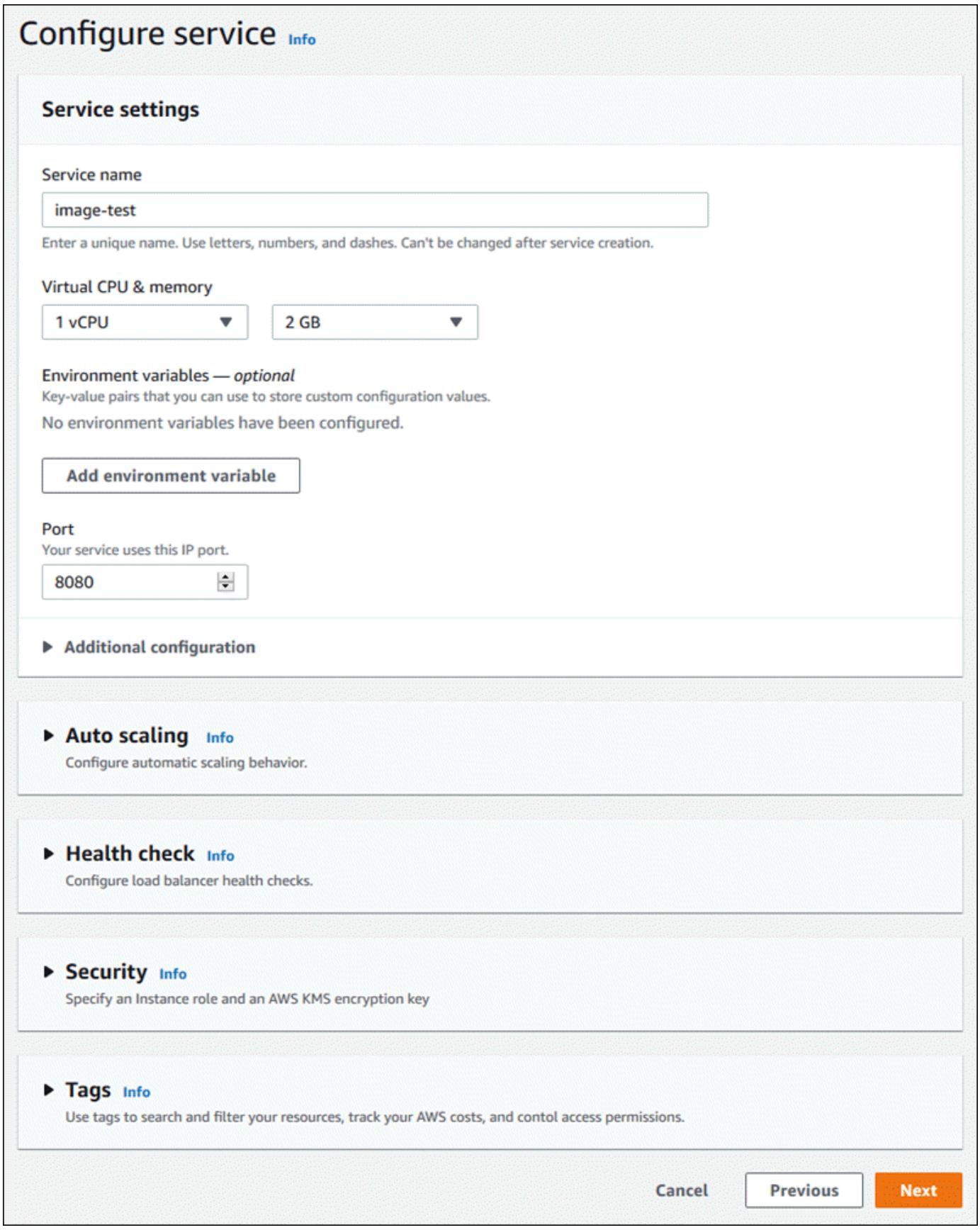
3. 配置您的服务。

   a. 在配置服务页面的服务设置部分，输入服务名称和您的服务网站监听的 IP 端口。

   > **Note**
   >
   > 所有其他服务设置要么是可选的，要么具有控制台提供的默认设置。

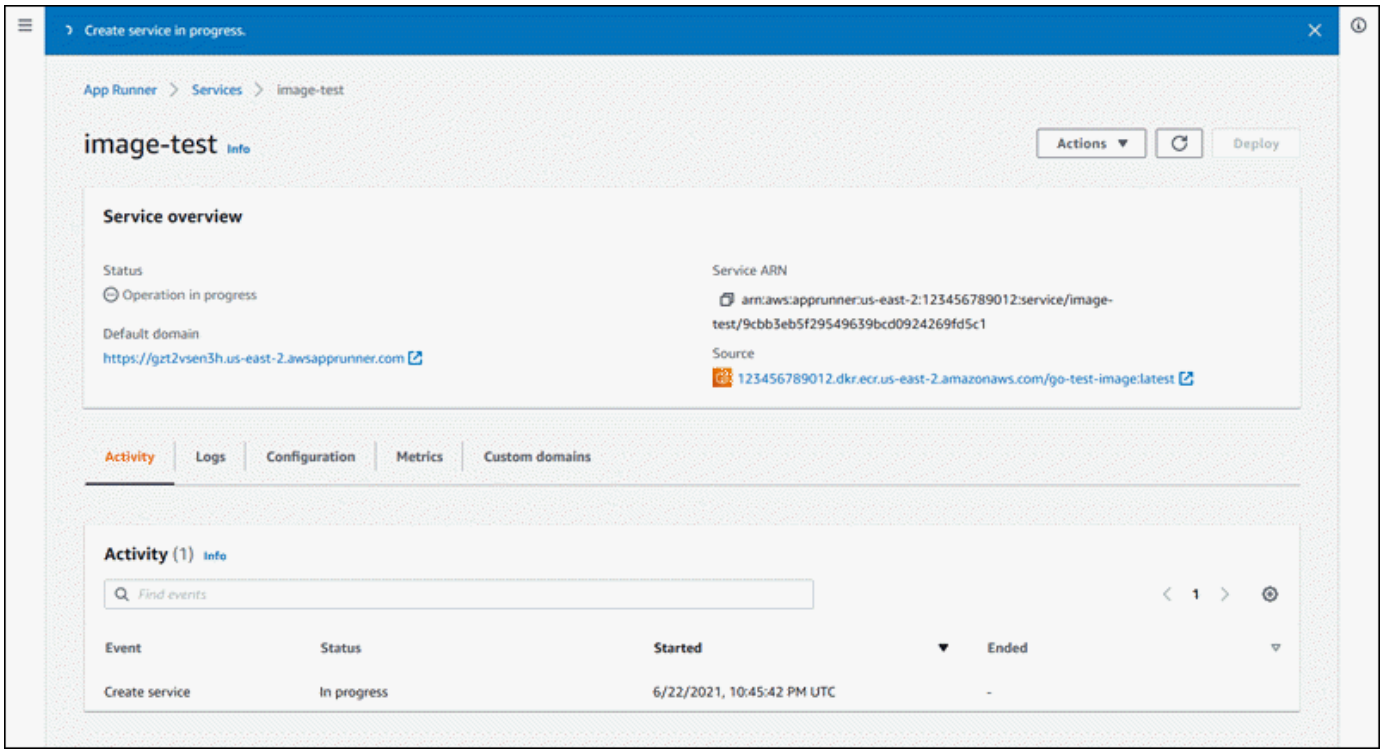
   b. （可选）更改或添加其他设置以满足应用程序的需求。

   c. 选择下一步。


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“查看并创建”页面上，验证您输入的所有详细信息，然后选择“创建并部署”。

   结果：如果成功创建服务，控制台将显示服务控制面板，其中包含新服务的服务概述。



5. 验证您的服务是否正在运行。

   a. 在服务仪表板页面上，等到服务状态变为“正在运行”。

   b. 选择默认域值。这是您的服务网站的网址。

   c. 使用您的网站并验证其是否正常运行。

使用 App Runner API 从图像创建服务或 AWS CLI

要使用 App Runner API 或创建服务 AWS CLI，请调用 CreateServiceAPI 操作。

如果调用返回成功响应，且显示了 Service 对象，则您的服务创建就开始`"Status": "CREATING"`了。

有关调用的示例，请参阅《AWS App Runner API 参考》中的“创建源图像存储库服务”

正在重建失败的 App Runner 服务

如果您在创建 App Runner 服务时收到“创建失败”错误，则可以执行以下操作之一。

- 按照中的the section called “创建服务失败”步骤确定错误的原因。
- 如果您在源代码或配置中发现错误，请进行必要的更改，然后重新构建您的服务。
- 如果 App Runner 的临时问题导致您的服务失败，请在不对源或配置进行任何更改的情况下重建失败的服务。

您可以通过 App Runner 控制台或 App Runner API 重建失败的服务，或者 AWS CLI。

使用 App Runner 控制台重建失败的 App Runner 服务

Rebuild with updates

创建服务可能由于多种原因而失败。发生这种情况时，重要的是要在重建服务之前确定并纠正问题的根本原因。有关更多信息，请参阅 the section called “创建服务失败”。

使用更新重建失败的服务

1. 前往服务页面上的 “配置” 选项卡，然后选择 “编辑”。

   该页面将打开一个摘要面板，其中显示了所有更新的列表。
2. 进行所需的更改并在摘要面板中进行查看。
3. 选择 “保存并重建”。

   您可以在服务页面的 “日志” 选项卡上监控进度。

Rebuild without updates

如果临时问题导致服务创建失败，则无需修改服务源或配置设置即可重建服务。

在不进行更新的情况下重建失败的服务

- 选择服务页面右上角的 “重建”。

  您可以在服务页面的 “日志” 选项卡上监控进度。
- 如果您的服务无法再次创建，请按照中的故障排除说明进行操作the section called “创建服务失败”。进行必要的更改，然后重新构建您的服务。

使用 App Runner API 重建失败的 App Runner 服务或 AWS CLI

Rebuild with updates

要重建失败的服务，请执行以下操作：

1. 按照中的the section called “创建服务失败”说明查找错误原因。
2. 对源存储库的分支或映像或导致错误的配置进行必要的更改。
3. 使用新的源代码存储库或源图像存储库参数调用 UpdateServiceAPI 操作进行重建。App Runner 从源代码存储库中检索最新的提交。

Example 使用更新进行重建

在以下示例中，正在更新基于图像的服务的源配置。的值更改`Port`为`80`。

更新基于图像的 App Runner 服务的`input.json`文件

```
{
  "ServiceArn": "arn:aws:apprunner:us-east-1:123456789012:service/python-
app/8fe1e10304f84fd2b0df550fe98a71fa",
  "SourceConfiguration": {
    "ImageRepository": {
      "ImageConfiguration": {
        "Port": "80"
      }
    }
  }
}
```

调用 `UpdateService` API 操作。

```
aws apprunner update-service
--cli-input-json file://input.json
```

Rebuild without updates

要使用 App Runner API 重建失败的服务 AWS CLI，或者，请在不对服务来源或配置进行任何更改的情况下调用 UpdateServiceAPI 操作。仅当您的服务创建因App Runner暂时出现问题而失败时，才选择在不进行更新的情况下进行重建。

将新的应用程序版本部署到 App Runner

在中创建服务时 AWS App Runner，需要配置应用程序源，即容器映像或源存储库。App Runner 会配置资源来运行您的服务，并将您的应用程序部署到这些资源中。

本主题介绍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。这可以是镜像仓库中的新镜像版本，也可以是代码仓库中的新提交。App Runner 提供了两种部署到服务的方法：自动和手动。

部署方法

App Runner 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

如果您想让服务实现持续集成和部署 (CI/CD) 行为，请使用自动部署。App Runner 会监控您的图像或代码存储库中是否有更改。

图像存储库 — 每当您将新的图像版本推送到图像存储库或向代码存储库推送新的提交时，App Runner 都会自动将其部署到您的服务，而无需您采取进一步的行动。

代码存储库 — 每当您将新的提交推送到代码存储库并在源目录中进行更改时，App Runner 都会部署您的整个存储库。由于只有源目录中的更改才会触发自动部署，因此了解源目录位置如何影响自动部署的范围非常重要。

- 顶级目录（存储库根目录）-这是创建服务时为源目录设置的默认值。如果您的源目录设置为该值，则意味着整个存储库都在源目录中。因此，在这种情况下，您推送到源存储库的所有提交都将触发部署。
- 任何不是存储库根目录的目录路径（非默认）-由于只有在源目录中推送的更改才会触发自动部署，因此任何推送到存储库但不在源目录中的更改都不会触发自动部署。因此，必须使用手动部署来部署推送到源目录之外的更改。

> **Note**
>
> App Runner 不支持自动部署 Amazon ECR 公共镜像，也不支持自动部署 Amazon ECR 存储库中与您的服务所在 AWS 账户不同的账户。

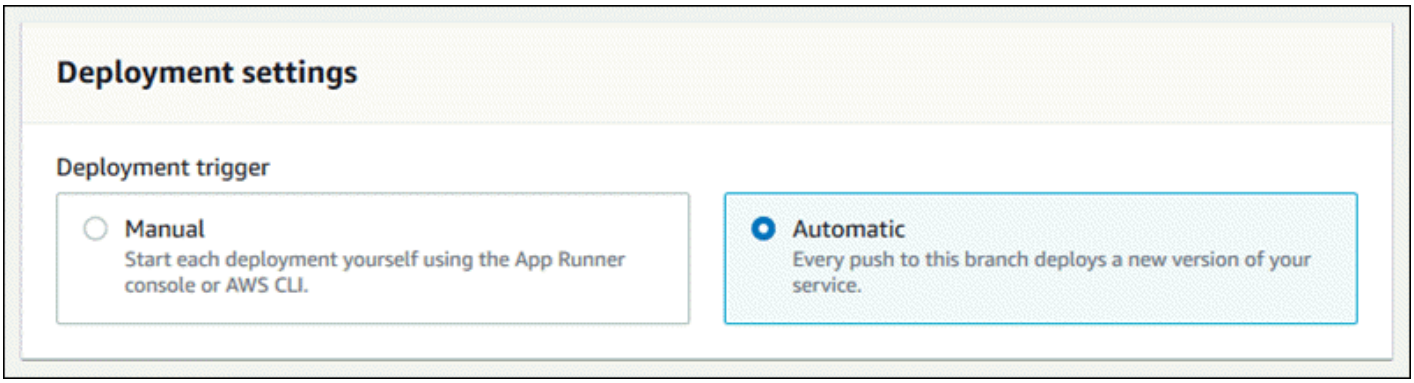
手动部署

如果要明确启动对服务的每次部署，请使用手动部署。如果您为服务配置的存储库有要部署的新版本，则可以启动部署。有关更多信息，请参阅 the section called “手动部署”。

> **Note**
>
> 当您运行手动部署时，App Runner 会部署来自完整存储库的源代码。

您可以通过以下方式为服务配置部署方法：

- 控制台-对于您正在创建的新服务或现有服务，在源和部署配置页面的部署设置部分，选择手动或自动。



- API 或 AWS CLI — 在调用CreateService或UpdateService操作时，将SourceConfiguration参数的`AutoDeploymentsEnabled`成员设置为，`False`用于手动部署或`True`自动部署。

> **比较自动部署和手动部署**
>
> 自动部署和手动部署的结果都是一样的：两种方法都部署了完整的存储库。
> 这两种方法的区别在于触发机制：
>
> - 手动部署是由从控制台进行部署、调用或调用 App Runner API 来触发的。 AWS CLI接下来的手动部署章节提供了这些操作的程序。
> - 源目录内容的更改会触发自动部署。

手动部署

使用手动部署，您需要明确启动对服务的每次部署。当您准备好部署应用程序映像或代码的新版本时，可以参考以下章节，了解如何使用控制台和 API 执行部署。

> **Note**
>
> 当您运行手动部署时，App Runner 会部署来自完整存储库的源代码。

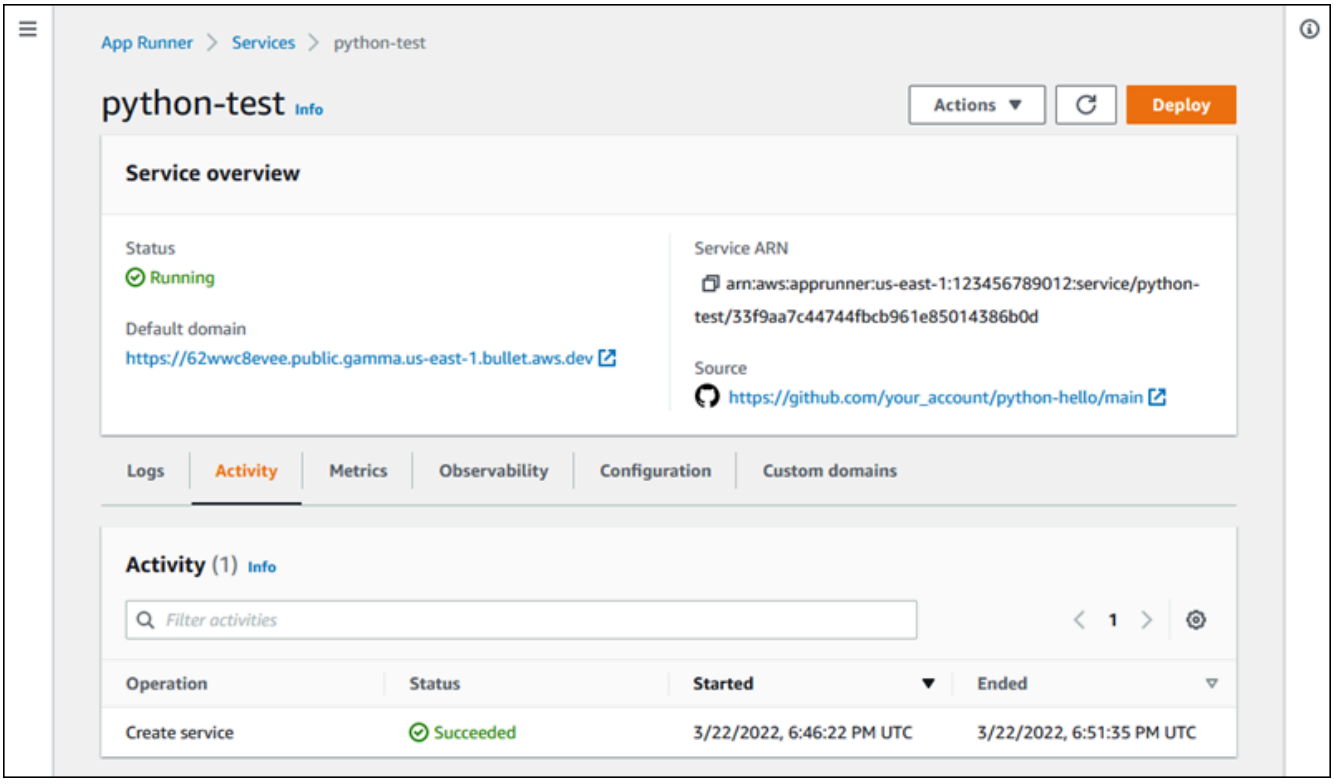
使用以下方法之一部署应用程序版本：

App Runner console

使用 App Runner 控制台进行部署

1. 打开 App Runner 控制台，然后在 “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 App Runner 服务。

   控制台显示带有服务概述的服务仪表板。



3. 选择部署。

结果：开始部署新版本。在服务控制面板页面上，服务状态更改为操作进行中。

4. 等待部署结束。在服务仪表板页面上，服务状态应更改回正在运行。
5. 要验证部署是否成功，请在服务控制面板页面上选择默认域值，即指向服务网站的网址。检查您的 Web 应用程序或与之交互，并验证您的版本更改。

> **Note**
>
> 为了增强 App Runner 应用程序的安全性，*.awsapprunner.com 域已在公共后缀列表 (PSL) 中注册。为了进一步提高安全性，如果您需要在 App Runner 应用程序的默认域名中设置敏感 Cookie，我们建议您使用带__Host-前缀的 Cookie。这将有助于保护您的域，防范跨站点请求伪造 (CSRF) 攻击。要了解更多信息，请参阅 Mozilla 开发者网络中的 Set-Cookie 页面。

App Runner API or AWS CLI

要使用 App Runner API 或进行部署 AWS CLI，请调用 StartDeploymentAPI 操作。唯一要传递的参数是您的服务 ARN。您在创建服务时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，则部署即开始。

配置 App Runner 服务

创建 AWS App Runner 服务时，可以设置各种配置值。创建服务后，您可以更改其中的一些配置设置。其他设置只能在创建服务时应用，之后无法更改。本主题讨论如何使用 App Runner API、App Runner 控制台和 App Runner 配置文件配置服务。

主题

- 使用 App Runner API 配置您的服务或 AWS CLI
- 使用 App Runner 控制台配置您的服务
- 使用 App Runner 配置文件配置您的服务
- 为您的服务配置可观测性
- 使用可共享资源配置服务设置
- 为您的服务配置运行状况检查

使用 App Runner API 配置您的服务或 AWS CLI

API 定义了在创建服务之后可以更改哪些设置。以下列表讨论了相关的操作、类型和限制。

- UpdateServiceaction-可以在创建后调用以更新某些配置设置。
  - 可以更新-您可以更新SourceConfigurationInstanceConfiguration、和HealthCheckConfiguration参数中的设置。但是，在中SourceConfiguration，您无法将源类型从代码切换到图像或其他方式。您提供的存储库参数必须与创建服务时提供的存储库参数相同。要么是要CodeRepository么ImageRepository。

    您还可以更新与该服务关联的单独配置资源的以下 ARN：

    - AutoScalingConfigurationArn
    - VpcConnectorArn
  - 无法更新-您无法更改CreateService操作中可用的ServiceName和EncryptionConfiguration参数。它们在创建后无法更改。该UpdateService操作不包括这些参数。
  - API 与文件 — 您可以将CodeConfiguration类型的ConfigurationSource参数（作为源代码存储库的一部分SourceConfiguration）设置为Repository。在这种情况下，App Runner 会忽略中的配置设置CodeConfigurationValues，并从存储库中的配置文件中读取这些设置。如果设置为 ConfigurationSourceAPI，App Runner 会从 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。
- TagResourceaction — 可以在创建服务后调用，以向服务添加标签或更新现有标签的值。
- UntagResourceaction — 可以在创建服务后调用，以从服务中移除标签。

> **Note**
>
> 如果您为服务创建出站流量 VPC 连接器，则随后的服务启动过程将出现一次性延迟。您可以在创建新服务时或之后通过服务更新为新服务设置此配置。有关更多信息，请参阅一次性延迟本指南的“与 App Runner 联网”一章。

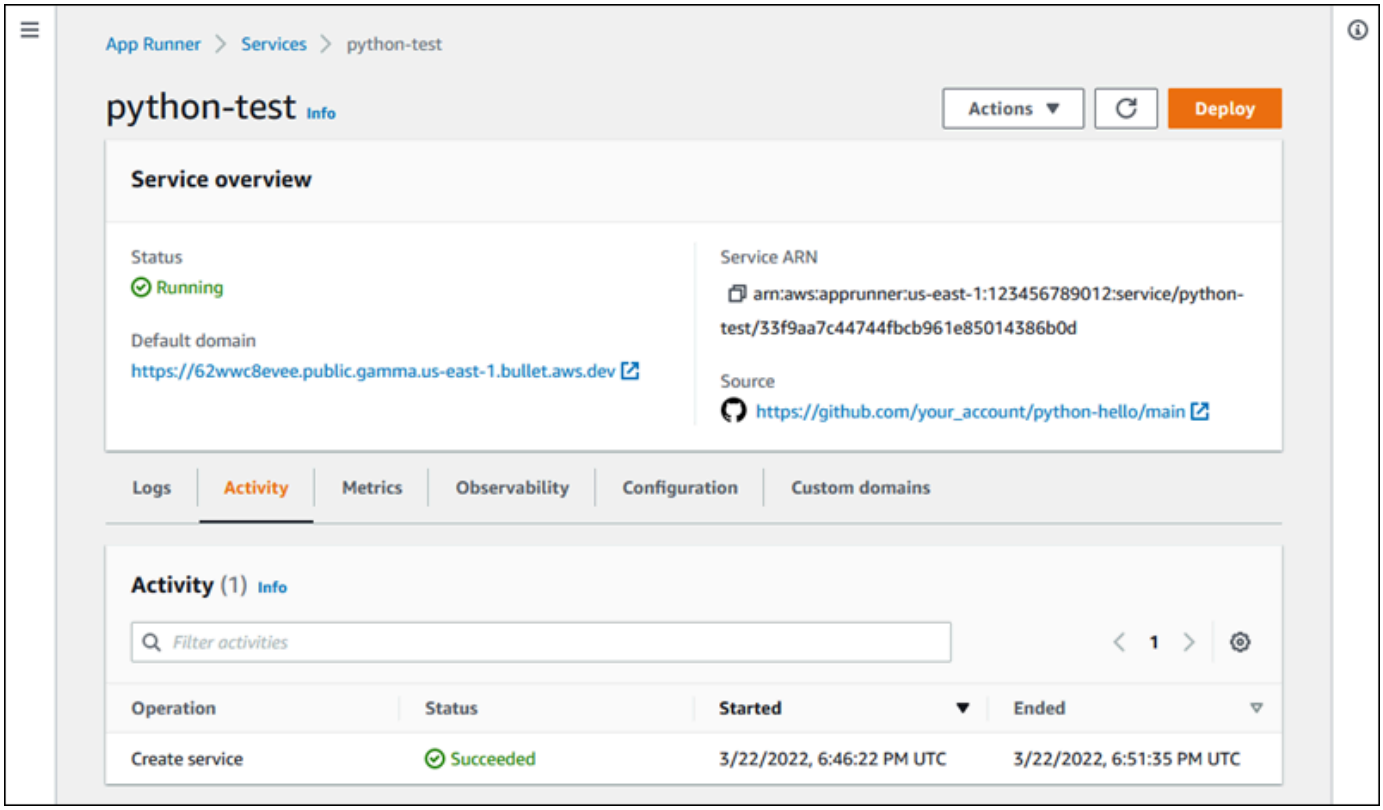
使用 App Runner 控制台配置您的服务

控制台使用 App Runner API 来应用配置更新。API 施加的更新规则（如上一节所定义）决定了您可以使用控制台配置的内容。创建服务期间可用的某些设置以后无法修改。此外，如果您决定使用配置文件，则其他设置会隐藏在控制台中，App Runner 会从文件中读取这些设置。

配置您的服务

1. 打开 App Runner 控制台，然后在 “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 App Runner 服务。

   控制台显示带有服务概述的服务仪表板。



3. 在服务仪表板页面上，选择配置选项卡。

   结果：控制台分几个部分显示服务的当前配置设置：源和部署、配置构建和配置服务。
4. 要更新任何类别的设置，请选择 “编辑”。
5. 在配置编辑页面上，进行任何所需的更改，然后选择保存更改。

> **Note**
>
> 如果您为服务创建出站流量 VPC 连接器，则随后的服务启动过程将出现一次性延迟。您可以在创建新服务时或之后通过服务更新为新服务设置此配置。有关更多信息，请参阅一次性延迟本指南的 “与 App Runner 联网” 一章。

使用 App Runner 配置文件配置您的服务

创建或更新 App Runner 服务时，您可以指示 App Runner 从您作为源存储库一部分提供的配置文件中读取一些配置设置。通过这样做，你可以在源代码控制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高级设置，您无法使用控制台或 API 进行设置。有关更多信息，请参阅 App Runner 配置文件。

> **Note**
>
> 如果您为服务创建出站流量 VPC 连接器，则随后的服务启动过程将出现一次性延迟。您可以在创建新服务时或之后通过服务更新为新服务设置此配置。有关更多信息，请参阅一次性延迟本指南的 “与 App Runner 联网” 一章。

为您的服务配置可观测性

AWS App Runner 与多项 AWS 服务集成，为您的 App Runner 服务提供广泛的可观察性工具套件。有关更多信息，请参阅 可观察性。

App Runner 支持启用某些可观察性功能并使用名为的可共享资源来配置其行为。ObservabilityConfiguration您可以在创建或更新服务时提供可观测性配置资源。当你创建新的 App Runner 服务时，App Runner 控制台会为你创建一个服务。提供可观测性配置是可选的。如果您不提供可观测性配置，App Runner 会提供默认的可观测性配置。

您可以跨多个 App Runner 服务共享单个可观察性配置，以确保它们具有相同的可观察性行为。有关更多信息，请参阅 the section called “配置资源”。

您可以使用可观测性配置配置配置以下可观测性功能：

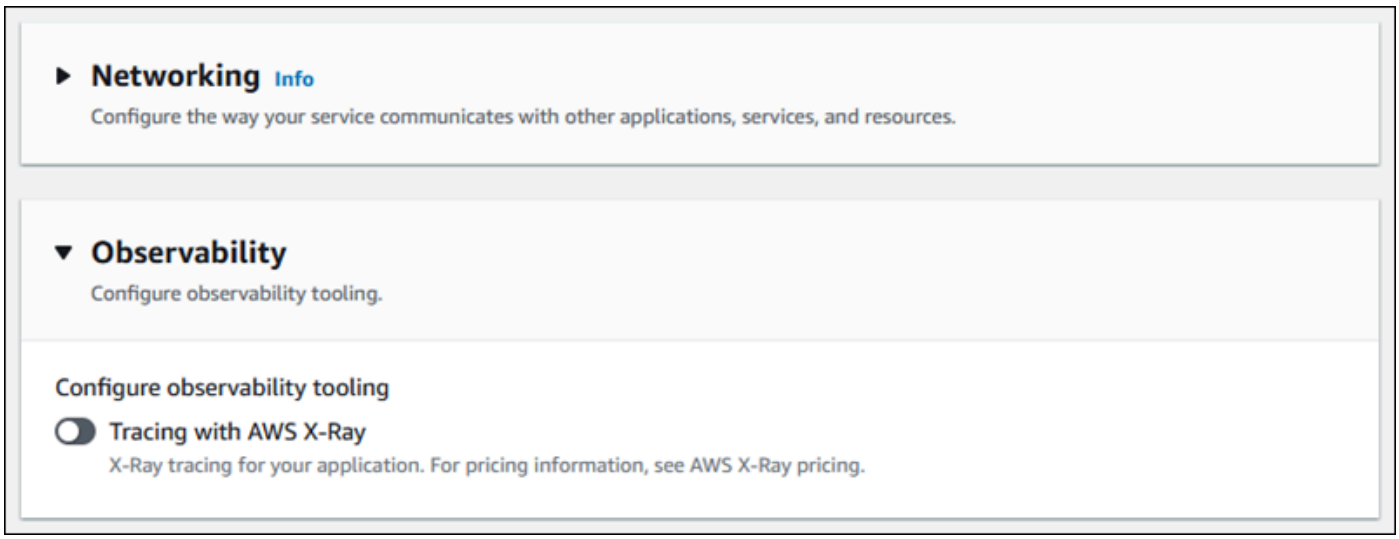
- 跟踪配置-用于跟踪您的应用程序所处理的请求和它发出的下游调用的设置。有关跟踪的更多信息，请参阅 the section called “追踪 (X-Ray)”。

管理可观察性

使用以下方法之一管理您的 App Runner 服务的可观察性：

App Runner console

使用 App Runner 控制台创建服务时，或者稍后更新其配置时，可以为服务配置可观察性功能。在控制台页面上查找 “可观察性配置” 部分。



App Runner API or AWS CLI

当您调用CreateService或 UpdateServiceApp Runner API 操作时，您可以使用`ObservabilityConfiguration`参数对象来启用可观测性功能并为服务指定可观测性配置资源。

使用以下 App Runner API 操作来管理您的可观测性配置资源。

- CreateObservabilityConfiguration— 创建新的可观测性配置或对现有可观测性配置的修订。
- ListObservabilityConfigurations— 返回与您的 AWS 账户关联的可观测性配置列表以及摘要信息。
- DescribeObservabilityConfiguration— 返回可观测性配置的完整描述。
- DeleteObservabilityConfiguration— 删除可观测性配置。您可以删除特定的修订版或最新的活动修订版。如果您达到可观测性配置配额，则可能需要删除不必要的可观测性配置。 AWS 账户

使用可共享资源配置服务设置

对于某些功能，跨 AWS App Runner 服务共享配置是有意义的。例如，您可能希望一组服务具有相同的 auto Scaling 行为。或者，您可能需要为所有服务设置相同的可观测性设置。App Runner 允许您使用单独的可共享资源来共享设置。您创建一个为某项功能定义一组配置设置的资源，然后将此配置资源的亚马逊资源名称 (ARN) 提供给一个或多个 App Runner 服务。

App Runner 为以下功能实现了可共享的配置资源：

- 自动扩缩

- 可观察性
- VPC 访问

每项功能的文档页面都提供了有关可用设置和管理程序的信息。

使用单独配置资源的功能具有一些共同的设计特征和注意事项。

- 修订版-某些配置资源可能有修订版。自动缩放和可观察性是两个使用修订版的配置资源的示例。在这些情况下，每个配置都有一个名称和一个数字修订版。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。您可以针对不同的场景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，以微调特定场景的设置。

  使用名称创建的第一个配置的修订版号为 1。具有相同名称的后续配置将获得连续的修订版号（以 2 开头）。您可以将 App Runner 服务与特定的配置版本或最新版本的配置相关联。
- 共享 — 您可以跨多个 App Runner 服务共享单个配置资源。如果您想在这些服务之间保持相同的配置，这很有用。特别是，如果您的资源支持修订版，则可以将多个服务配置为使用最新版本的配置。为此，您可以只指定配置名称，而不指定修订版。更新服务时，您以这种方式配置的任何服务都会收到配置更新。有关配置更改的更多信息，请参阅the section called “配置”。
- 资源管理-您可以使用 App Runner 创建和删除配置。您无法直接更新配置。相反，对于支持修订版的资源，您可以为现有配置名称创建新的修订版以有效地更新配置。

  > **Note**
  >
  > 对于自动缩放，您可以使用 App Runner 控制台和 App Runner API 创建配置和多个修订版。App Runner 控制台和 App Runner API 也可以删除配置和修订版。有关更多详细信息，请参阅管理 App Runner 自动缩放。
  > 对于其他配置类型，例如可观察性配置，您只能使用 App Runner 控制台创建具有单个修订版的配置。要创建更多修订版和删除配置，必须使用 App Runner API。
- 资源配额 — 您可以为每个配置资源设置的唯一配置名称和修订版本数量的配额 AWS 区域。如果达到这些配额，则必须先删除配置名称或至少删除其中的一些修订版，然后才能创建更多版本。对于自动缩放配置修订版，您可以使用 App Runner 控制台或 App Runner API 将其删除。有关更多详细信息，请参阅管理 App Runner 自动缩放。您必须使用 App Runner API 来删除其他资源。有关限额的更多信息，请参阅the section called “应用程序运行器资源配额”。
- 无资源成本 — 创建配置资源不会产生额外费用。您可能会为该功能本身承担费用（例如，在打开 X-Ray 跟踪时，您需要支付正常 AWS X-Ray 费用），但不会为为 App Runner 服务配置该功能的 App Runner 配置资源付费。

为您的服务配置运行状况检查

AWS App Runner 通过执行运行状况检查来监控服务的运行状况。默认运行状况检查协议为 TCP。App Runner 会对分配给你的服务的域执行 ping 操作。您也可以将运行状况检查协议设置为 HTTP。App Runner 向你的 Web 应用程序发送运行状况检查 HTTP 请求。

您可以配置一些与运行状况检查相关的设置。下表描述了运行状况检查设置及其默认值。

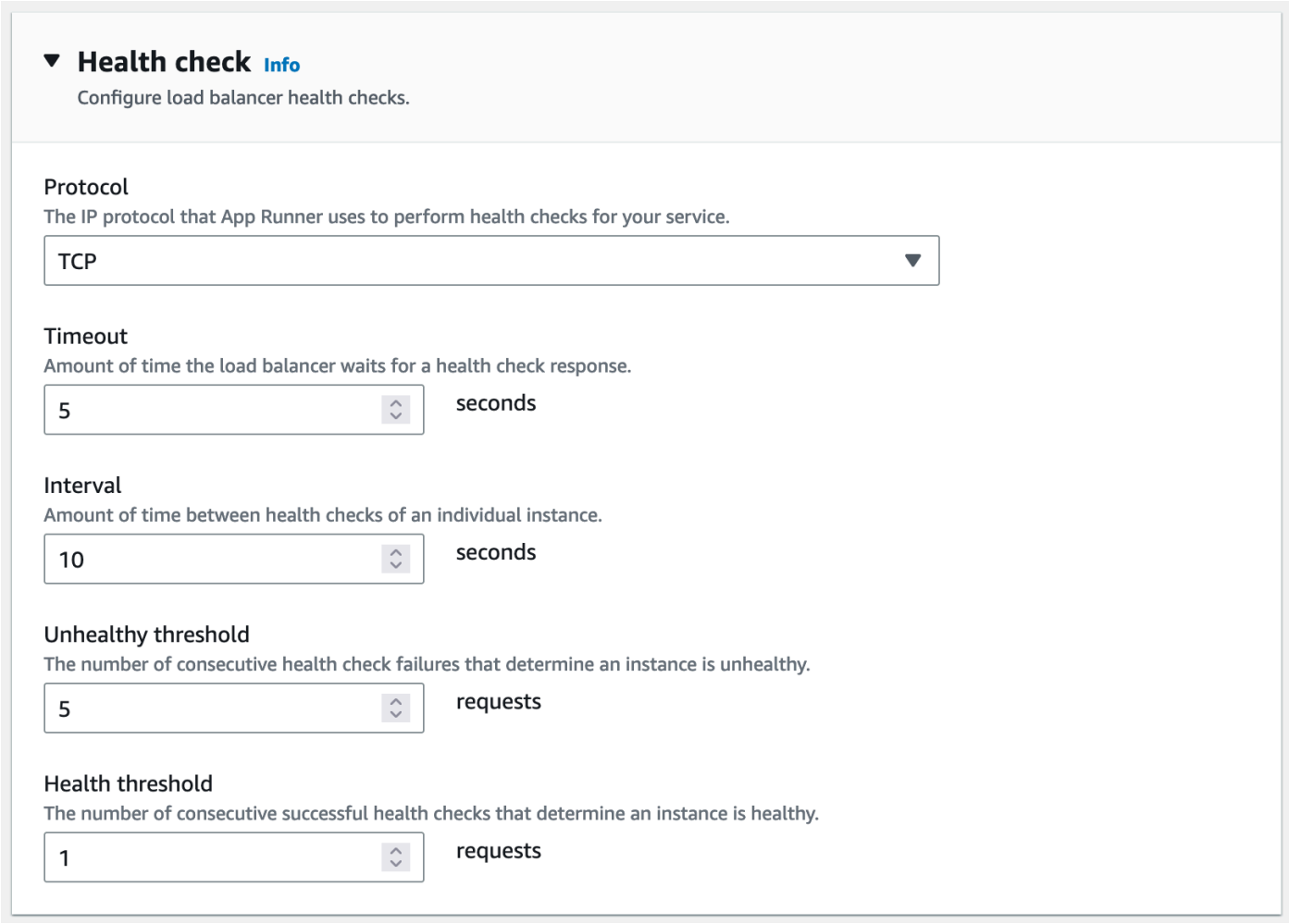
设置	描述	默认
协议	App Runner 用于对您的服务执行运行状况检查的 IP 协议。 如果您将协议设置为`TCP`，则 App Runner 会在应用程序正在监听的端口上对分配给您的服务的默认域执行 ping 操作。 如果您将协议设置为`HTTP`，App Runner 会向配置的路径发送运行状况检查请求。	`TCP`
路径	App Runner 向其发送 HTTP 运行状况检查请求的网址。仅适用于 HTTP 检查。	`/`
Interval	运行状况检查之间的时间间隔（以秒为单位）。	5
超时	在确定运行状况检查响应失败之前等待的时间（以秒为单位）。	2
健康阈值	App Runner 确定服务运行状况正常之前必须成功的连续检查次数。	1
不健康阈值	App Runner 确定服务运行状况不正常之前必须失败的连续检查次数。	5

配置运行状况检查

使用以下方法之一为您的 App Runner 服务配置运行状况检查：

App Runner console

使用 App Runner 控制台创建 App Runner 服务时，或者稍后更新其配置时，可以配置运行状况检查设置。有关完整的控制台程序，请参阅the section called “创建”和the section called “配置”。在这两种情况下，请在控制台页面上查找 Health check 配置部分。



App Runner API or AWS CLI

当您调用CreateService或 UpdateServiceAPI 操作时，您可以使用`HealthCheckConfiguration`参数来指定运行状况检查设置。

有关参数结构的信息，请参阅 AWS App Runner API 参考HealthCheckConfiguration中的。

管理 App Runner 连接

在中创建服务时 AWS App Runner，需要配置应用程序源，即容器映像或与提供者一起存储的源存储库。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中。当您创建访问存储在您的 AWS 账户代码的服务时，App Runner 不需要建立连接。

App Runner 将连接信息保存在名为连接的资源中。App Runner 控制台和本指南还将连接称为关联帐户。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。以下是有关连接的一些重要信息：

- 提供商 — App Runner 目前需要与GitHub或 Bitbucket 的连接资源。
- 共享 — 您可以使用连接资源创建使用相同存储库提供者帐户的多个 App Runner 服务。
- 资源管理-在 App Runner 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 — 每个连接资源都有与您 AWS 账户 关联的固定配额 AWS 区域。如果您达到此配额，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 App Runner 控制台或 API 删除连接，如下一节所述the section called “管理连接”。有关更多信息，请参阅 the section called “应用程序运行器资源配额”。

管理连接

使用以下方法之一管理您的 App Runner 连接：

App Runner console

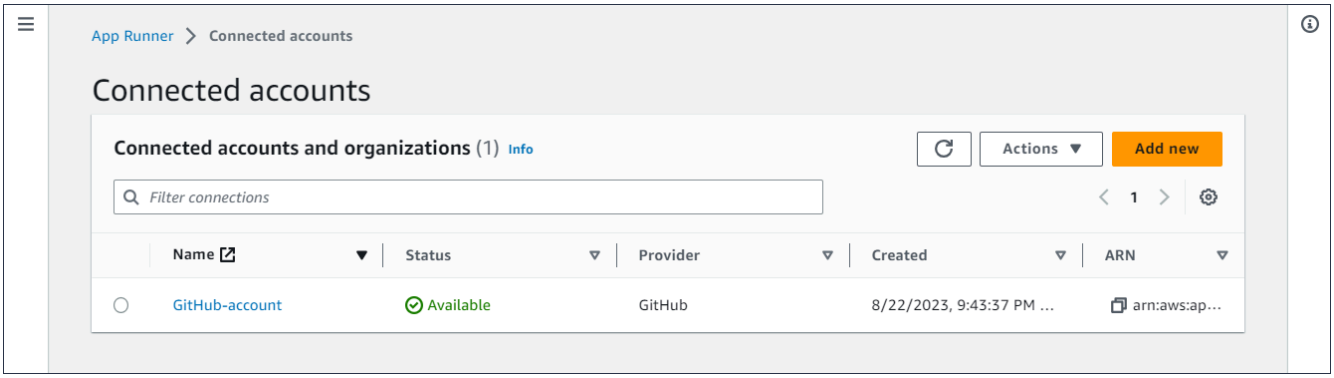
当您使用 App Runner 控制台创建服务时，您需要提供连接详细信息。您不必显式创建连接资源。在控制台中，您可以选择连接之前连接过的 GitHub 或 Bitbucket 账户，也可以选择连接到新账户。必要时，App Runner 会为您创建连接资源。对于新连接，某些提供商要求您先完成身份验证握手，然后才能使用该连接。控制台将引导您完成此过程。

控制台还有一个用于管理现有连接的页面。如果您在创建服务时未完成身份验证握手，则可以完成连接的身份验证握手。您也可以删除不再使用的连接。以下过程说明如何管理存储库提供程序连接。

管理账户中的连接

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 在导航窗格中，选择已关联账户。

   然后，控制台会显示您账户中存储库提供商连接的列表。



3. 现在，您可以对列表中的任何连接执行以下操作之一：

   - 打开 GitHub /Bitbucket 账户或组织 — 选择连接的名称。
   - 完成身份验证握手-选择连接，然后从 “操作” 菜单中选择 “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，然后从 “操作” 菜单中选择 “删除”。按照删除提示中的说明进行操作。

App Runner API or AWS CLI

您可以使用以下 App Runner API 操作来管理您的连接。

- CreateConnection— 创建与存储库提供者帐户的连接。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。此过程将在上一节中介绍。
- ListConnections— 返回与您的关联的 App Runner 连接列表 AWS 账户。
- DeleteConnection— 删除连接。如果您已达到自己的连接配额，则可能需要删除不必要的连接 AWS 账户。

管理 App Runner 自动缩放

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，特别是实例。自动扩展可在流量繁忙时提供足够的请求处理，并在流量减慢时降低成本。

自动缩放配置

您可以配置一些参数来调整服务的自动缩放行为。App Runner 在名AutoScalingConfiguration为的可共享资源中维护自动缩放设置。在将独立的 auto Scaling 配置分配给服务之前，您可以创建和维护这些配置。在它们与服务关联后，您可以继续维护配置。在创建新服务或配置现有服务时，您也可以选择创建新的 auto Scaling 配置。创建新的 auto Scaling 配置后，您可以将其与服务关联并继续创建或配置服务的过程。

命名和修订

auto Scaling 配置具有名称和数字修订版。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。您可以为不同的 auto Scaling 场景使用不同的配置名称，例如高可用性或低成本。对于每个名称，您可以添加多个修订版，以微调特定场景的设置。每个配置最多可以有 10 个唯一的 auto scaling 配置名称和最多 5 个修订版。如果您达到限制并需要创建更多内容，则可以删除一个，然后再创建一个。App Runner 不允许您删除已设置为默认配置或正在由活动服务使用的配置。有关限额的更多信息，请参阅the section called “应用程序运行器资源配额”。

设置默认配置

创建或更新 App Runner 服务时，可以提供自动缩放配置资源。提供 auto Scaling 配置是可选的。如果您不提供自动缩放配置，App Runner 会提供带有推荐值的默认自动缩放配置。自动缩放配置功能使您可以选择设置自己的默认自动缩放配置，而不是使用 App Runner 提供的默认配置。将另一个 auto Scaling 配置指定为默认配置后，该配置将自动作为默认配置分配给您将来创建的新服务。新的默认指定不会影响先前为现有服务设置的关联。

使用 auto Scaling 配置服务

您可以跨多个 App Runner 服务共享单个自动缩放配置，以确保这些服务具有相同的自动缩放行为。有关使用 App Runner 控制台或 App Runner API 配置自动缩放配置的更多信息，请参阅本主题后面的部分。有关可共享资源的更多一般信息，请参阅the section called “配置资源”。

可配置的设置

您可以配置以下 auto 缩放设置：

- 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 会扩大服务规模。
- 最大大小-您的服务可以扩展到的最大实例数。这是可以同时处理您的服务流量的最大实例数。
- 最小大小 — App Runner 可以为您的服务配置的最小实例数。该服务始终有至少此数量的预配置实例。其中一些实例会主动处理流量。其余部分是经济实惠的计算容量储备的一部分，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。您只需为活动子集的 CPU 使用量付费。

> **Note**
>
> vCPU 资源数量决定了 App Runner 可以为您的服务提供的实例数量。这是服务中驻留的 Fargate 按需 vCPU 资源数量的可调整配额值。 AWS Fargate (Fargate) 要查看您账户的

vCPU 配额设置或申请增加配额，请使用中的 Service Quotas 控制台。 AWS Management Console有关更多信息，请参阅AWS Fargate Amazon 弹性容器服务开发人员指南中的服务配额。

管理服务的自动缩放

使用以下方法之一管理 App Runner 服务的自动缩放：

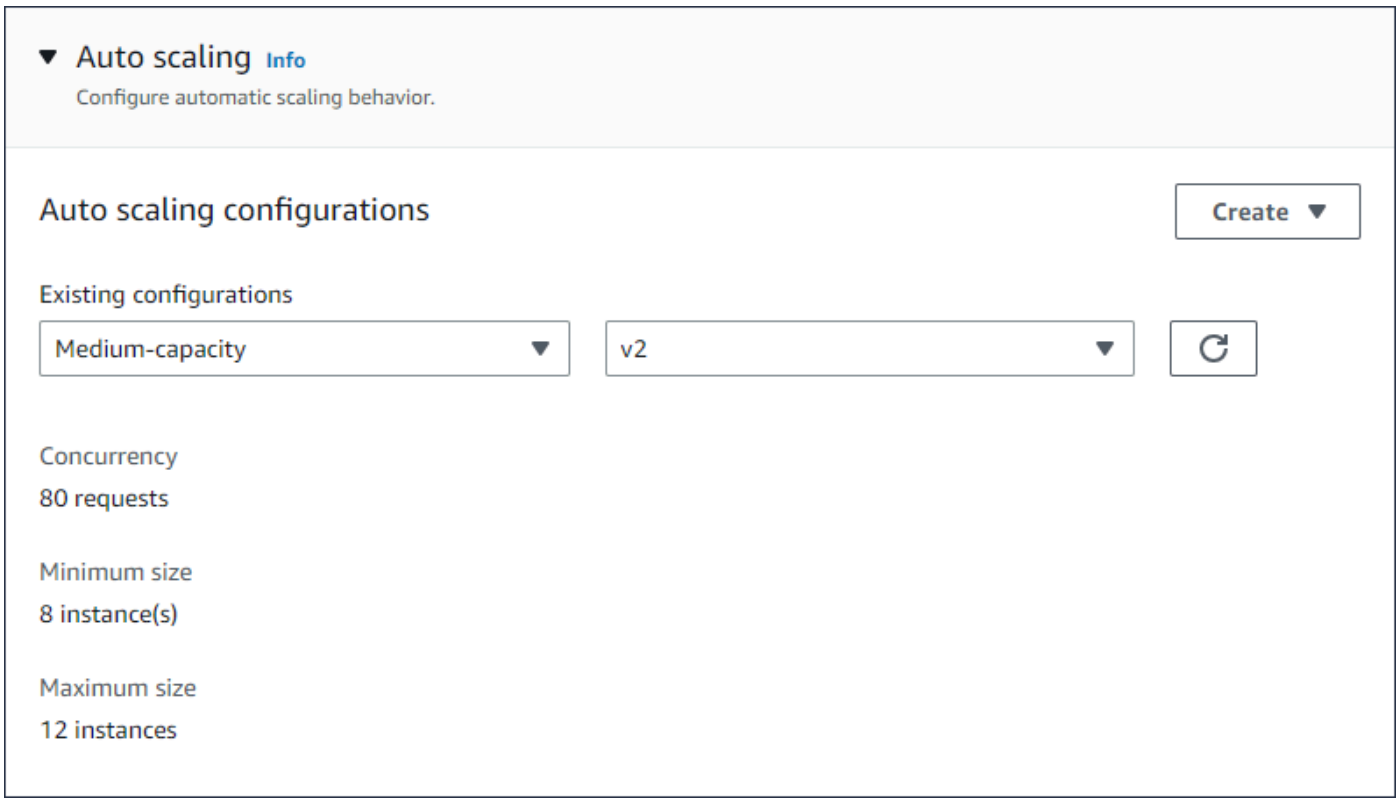
App Runner console

使用 App Runner 控制台创建服务或更新服务配置时，可以指定自动缩放配置。

> **Note**
>
> 当您更改与服务关联的 auto Scaling 配置或修订版时，您的服务将被重新部署。

A uto Scaling 配置页面提供了多个选项来为您的服务配置自动缩放。

- 要分配现有配置和修订版-从现有配置下拉列表中选择一个值。最新的修订版本将默认显示在旁边的下拉列表中。如果存在您想要选择的其他修订版本，请从修订版下拉列表中进行选择。将显示修订版本的配置值。
- 要创建和分配新的自动缩放配置，请从 “创建” 菜单中选择 “创建新 ASC”。这将启动添加自定义 auto Scaling 配置页面。输入配置名称和 auto scaling 参数的值。然后选择添加。App Runner 会为您创建新的自动缩放配置资源，然后将您返回到自动缩放部分，并选择并显示新配置。
- 要创建和分配新修订版，请先从现有配置下拉列表中选择配置名称。然后从 “创建” 菜单中选择 “创建 ASC 修订版”。这将启动添加自定义 auto Scaling 配置页面。输入 auto 缩放参数的值。然后选择添加。App Runner 会为您创建新的自动缩放配置修订版，然后将您返回到自动缩放部分，并选择并显示新版本。



App Runner API or AWS CLI

当您调用CreateService或 UpdateServiceApp Runner API 操作时，您可以使用`AutoScalingConfigurationArn`参数为您的服务指定自动缩放配置资源。

下一节将提供管理 auto Scaling 配置资源的指南。

管理 auto Scaling 配置资源

使用以下方法之一管理账户的 App Runner 自动缩放配置和修订：

App Runner console

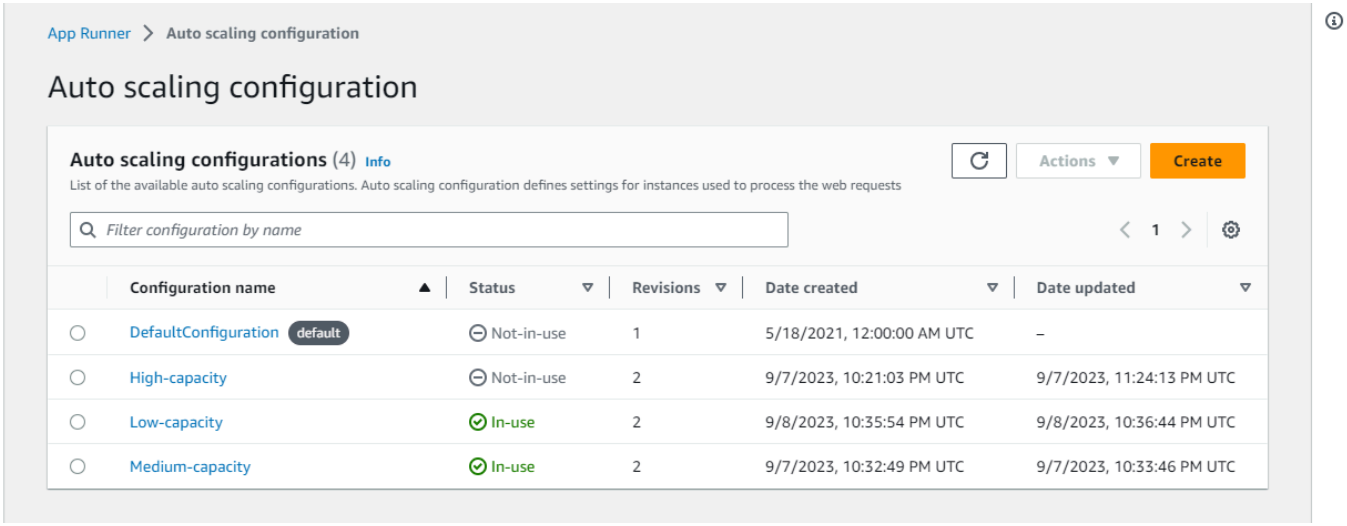
管理 auto Scaling 配置

A uto Scaling 配置页面列出了您在账户中设置的自动伸缩配置。您可以在此页面上创建和管理自动缩放配置，然后将其分配给一个或多个 App Runner 服务。

您可以在此页面上执行以下任一操作：

- 创建新的 auto Scaling 配置。
- 为现有 auto Scaling 配置创建新版本。

- 删除 auto 伸缩配置。
- 将 auto 缩放配置设置为默认配置。



管理账户中的 auto Scaling 配置

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 在导航窗格中，选择自动伸缩配置。控制台会显示您账户中的 auto Scaling 配置列表。

   现在，您可以执行以下任一操作。

   - 要创建新的 auto Scaling 配置，请按照以下步骤操作。

     a. 在自动伸缩配置页面上，选择创建。

        将显示 “创建 auto Scaling 配置” 页面。

     b. 输入配置名称、并发性、最小大小和最大大小的值。

     c. （可选）如果要添加标签，请选择 “自动新建标签”。然后在出现的字段中输入名称和值（可选）。

     d. 选择创建。

   - 要为现有 auto Scaling 配置创建新修订版，请按照以下步骤操作。

     a. 在 Auto Scaling 配置页面上，选择需要新修订版的配置旁边的单选按钮。然后从 “操作” 菜单中选择 “创建修订”。

        将显示 “创建修订” 页面。

     b. 在 “并发”、“最小大小” 和 “最大大小” 中输入值。

c. （可选）如果要添加标签，请选择“自动新建标签”。然后在出现的字段中输入名称和值（可选）。

d. 选择创建。

- 要删除 auto Scaling 配置，请按照以下步骤操作。

  a. 在 Auto Scaling 配置页面上，选择需要删除的配置旁边的单选按钮。

  b. 从“操作”菜单中选择“删除”。

  c. 要继续删除，请在确认对话框中选择“删除”。否则，请选择取消。

  > **Note**
  >
  > App Runner 会验证您的删除选项是否未设置为默认值，或者您的删除选项是否已被任何活动服务使用。

- 要将 auto Scaling 配置设置为默认配置，请按照以下步骤操作。

  a. 在 Auto Scaling 配置页面上，选择需要设置为默认配置旁边的单选按钮。

  b. 从“操作”菜单中选择“设为默认值”。

  c. 将显示一个对话框，通知您 App Runner 将使用最新版本作为您创建的所有新服务的默认配置。选择“确认”继续。否则请选择“取消”。

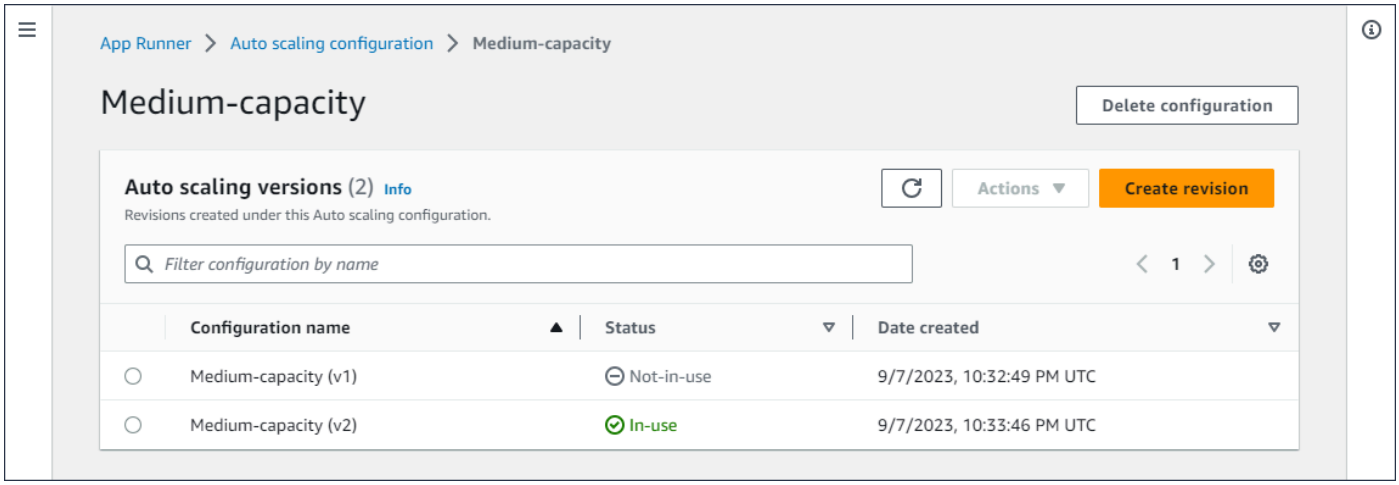
  > **Note**
  >
  > - 当你将 auto Scaling 配置设置为默认配置时，它会自动作为默认配置分配给你将来创建的新服务。
  > - 新的默认指定不会影响先前为现有服务设置的关联。
  > - 如果指定的默认 auto Scaling 配置有修订版，则 App Runner 会将其最新版本指定为默认版本。

管理修订

控制台还有一个用于创建和管理现有自动缩放修订版的页面，名为 Auto sc aling 修订版。在 A uto Scaling 配置页面上选择配置名称即可访问此页面。

您可以从 Auto Scaling 修订版页面执行以下任一操作：

- 创建新的 auto Scaling 修订版。
- 将 auto Scaling 配置版本设置为默认版本。
- 删除修订版。
- 删除整个 auto scaling 配置，包括所有关联的修订版。
- 查看修订版的配置详细信息。
- 查看与修订版相关的服务列表。
- 更改所列服务的修订版。



管理账户中的自动缩放修订

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 在导航窗格中，选择自动伸缩配置。控制台会显示您账户中的 auto Scaling 配置列表。本管理 auto Scaling 配置节前面的一组步骤包括此页面的屏幕图像。
3. 现在，您可以深入研究特定的 auto scaling 配置，以查看和管理其所有修订版。在 Auto Scaling 配置窗格的配置名称列下，选择一个自动伸缩配置名称。选择实际名称，而不是单选按钮。这将在 A uto Scaling 修订版页面上将您导航到该配置的所有修订版列表。
4. 现在，您可以执行以下任一操作。
   - 要为现有 auto Scaling 配置创建新修订版，请按照以下步骤操作。
     a. 在 “自动缩放修订版” 页面上，选择 “创建修订版”。

        将显示 “创建修订” 页面。
     b. 输入并发度、最小大小和最大大小的值。

c. （可选）如果要添加标签，请选择“自动新建标签”。然后在出现的字段中输入名称和值（可选）。

d. 选择创建。

- 要删除整个 auto Scaling 配置，包括所有关联的修订版，请按照以下步骤操作。

  a. 选择页面右上角的删除配置。

  b. 要继续删除，请在确认对话框中选择“删除”。否则，请选择取消。

  > **Note**
  >
  > App Runner 会验证您的删除选项是否未设置为默认值，或者您的删除选项是否已被任何活动服务使用。

- 要将 auto Scaling 版本设置为默认版本，请按照以下步骤操作。

  a. 选择需要设置为默认版本的版本旁边的单选按钮。

  b. 从“操作”菜单中选择“设为默认值”。

  > **Note**
  >
  > - 当你将 auto Scaling 配置设置为默认配置时，它会自动作为默认配置分配给你将来创建的新服务。
  > - 新的默认指定不会影响先前为现有服务设置的关联。

- 要查看修订版的配置详细信息，请执行以下步骤。

  - 选择修订版旁边的单选按钮。

    修订版的配置详细信息（包括 ARN）显示在下方的拆分面板中。请参阅本过程末尾的屏幕图像。

- 要查看与修订相关的服务列表，请执行以下步骤。

  - 选择修订版旁边的单选按钮。

    服务面板显示在下方的拆分面板中，位于修订配置详细信息下方。该面板列出了使用此 auto Scaling 配置修订版的所有服务。请参阅本过程末尾的屏幕图像。

- 要更改所列服务的修订版，请按照以下步骤操作。

a. 如果您尚未选择修订版旁边的单选按钮，请选择该按钮。

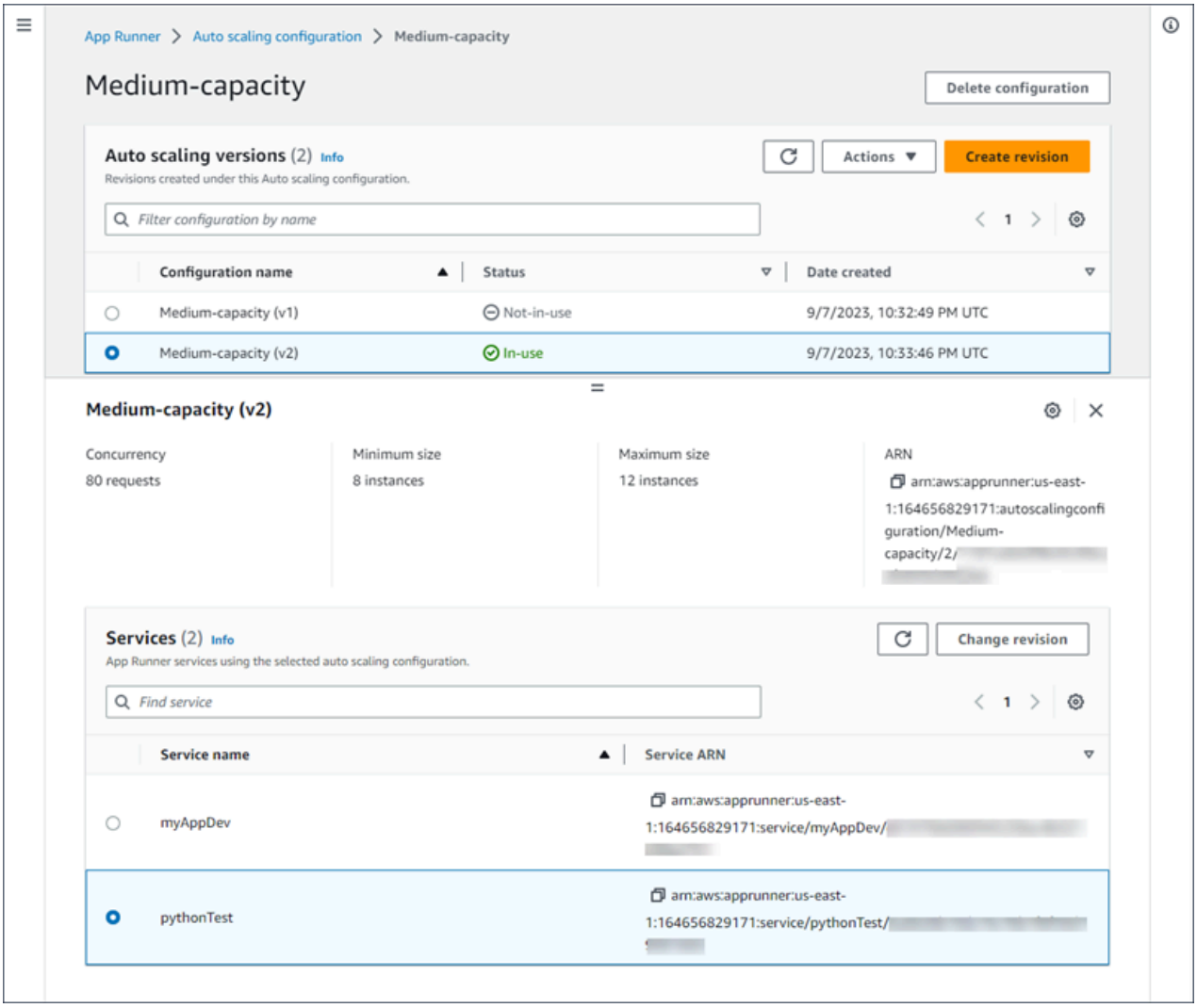
   服务面板显示在下方的拆分面板中，位于修订配置详细信息下方。该面板列出了使用此 auto Scaling 配置修订版的所有服务。请参阅本过程末尾的屏幕图像。

b. 在 “服务” 面板上，选择要修改的服务旁边的单选按钮。然后选择 “更改修订版”。

c. 将显示 “更改 ASC 修订版” 面板。从下拉列表中的可用版本中进行选择。只有您之前选择的 auto Scaling 配置的修订版可用。如果您需要更改为不同的 auto Scaling 配置，请按照上一节中的步骤进行操作the section called “管理服务的自动缩放”。

   选择 “更新” 以继续进行更改。否则请选择 “取消”。

> **Note**
>
> 当您更改与服务关联的修订版时，您的服务将被重新部署。
> 必须在此面板上选择 “刷新” 才能看到更新的关联。
> 要查看正在进行的活动和服务重新部署的状态，请使用面板痕迹导航到 App Runner > 服务，选择服务，然后从服务概述面板中查看 “日志” 选项卡。



App Runner API or AWS CLI

使用以下 App Runner API 操作来管理您的自动缩放配置资源。

- CreateAutoScalingConfiguration— 创建新的 auto scaling 配置或对现有配置的修订。
- UpdateDefaultAutoScalingConfiguration— 将 auto 缩放配置设置为默认配置。现有的默认 auto Scaling 配置将自动设置为非默认配置。
- ListAutoScalingConfigurations— 返回与您的 AWS 账户关联的 auto Scaling 配置列表以及摘要信息。
- ListServicesForAutoScalingConfiguration— 使用自动缩放配置返回相关的 App Runner 服务的列表。

- DescribeAutoScalingConfiguration— 返回 auto 缩放配置的完整描述。
- DeleteAutoScalingConfiguration— 删除 auto 缩放配置。您可以删除顶级 auto Scaling 配置、一个配置的特定修订版或与顶级配置关联的所有修订版。使用可选`DeleteAllRevisions`参数删除所有修订版。如果您达到的 auto Scaling 配置资源配额 AWS 账户，则可能需要删除不必要的自动缩放配置。

管理 App Runner 服务的自定义域名

创建 AWS App Runner 服务时，App Runner 会为其分配一个域名。这是 App Runner 拥有的`awsapprunner.com`域中的一个子域名。您可以使用域名来访问您的服务中运行的 Web 应用程序。

> **Note**
>
> 为了增强 App Runner 应用程序的安全性，*.awsapprunner.com 域已在公共后缀列表 (PSL) 中注册。为了进一步提高安全性，如果您需要在 App Runner 应用程序的默认域名中设置敏感 Cookie，我们建议您使用带`__Host-`前缀的 Cookie。这将有助于保护您的域，防范跨站点请求伪造 (CSRF) 攻击。要了解更多信息，请参阅 Mozilla 开发者网络中的 Set-Cookie 页面。

如果您拥有域名，则可以将其关联到您的 App Runner 服务。App Runner 验证您的新域名后，除了 App Runner 域之外，您还可以使用您的域名来访问您的应用程序。您最多可以关联五个自定义域名。

> **Note**
>
> 您可以选择添加域名的`www`子域名。但是，目前只有 API 支持此功能。App Runner 控制台不支持包含您的`www`域名的子域名。

> **Note**
>
> AWS App Runner 不支持使用 Route 53 私有托管区域。私有托管区域可为 Amazon VPC 流量自定义域名解析。有关私有托管区域的更多信息，请参阅 Route 53 文档中的使用私有托管区域。

将自定义域名与您的服务关联（链接）

将自定义域与服务关联时，必须将 CNAME 记录和 DNS 目标记录添加到您的 DNS 服务器。以下各节提供有关 CNAME 记录和 DNS 目标记录以及如何使用它们的信息。

> **Note**
>
> 如果您使用 Amazon Route 53 作为 DNS 提供商，App Runner 会自动为您的自定义域配置所需的证书验证和 DNS 记录，以链接到您的 App Runner 网络应用程序。当您使用 App Runner 控制台将自定义域名链接到您的服务时，就会发生这种情况。以下管理自定义域名主题提供了更多信息。

CNAME 记录

当您将自定义域与服务关联时，App Runner 会为您提供一组用于证书验证的证书验证记录。您必须将这些证书验证记录添加到您的域名系统 (DNS) 服务器。将 App Runner 提供的证书验证记录添加到您的 DNS 服务器。这样，App Runner 就可以验证您是否拥有或控制该域名。

> **Note**
>
> 要自动续订自定义域名证书，请确保不要从 DNS 服务器中删除证书验证记录。有关如何解决与证书续订相关的问题的信息，请参阅the section called “续订自定义域名证书”。

App Runner 使用 ACM 来验证域。如果您在 DNS 记录中使用 CAA 记录，请确保至少有一条 CAA 记录引用。`amazon.com`否则，ACM 无法验证域名并成功创建您的域。

如果您收到与 CAA 相关的错误，请查看以下链接以了解如何解决这些错误：

- 证书颁发机构授权 (CAA) 问题
- 如何解决在颁发或续订 ACM 证书时出现的 CAA 错误？
- 自定义域名

> **Note**
>
> 如果您使用 Amazon Route 53 作为 DNS 提供商，App Runner 会自动为您的自定义域配置所需的证书验证和 DNS 记录，以链接到您的 App Runner 网络应用程序。当您使用 App Runner

控制台将自定义域名链接到您的服务时，就会发生这种情况。以下管理自定义域名主题提供了更多信息。

DNS 目标记录

将 DNS 目标记录添加到您的 DNS 服务器以定向 App Runner 域。如果您选择此选项，则为自定义域名添加一条记录，为www子域添加另一条记录。然后，在 App Runner 控制台中等待自定义域名状态变为 “激活”。这通常需要几分钟，但可能最多需要 24-48 小时（1—2 天）。当您的自定义域名通过验证后，App Runner 会开始将流量从该域路由到您的 Web 应用程序。

> **Note**
>
> 为了更好地兼容 App Runner 服务，我们建议您使用 Amazon Route 53 作为 DNS 提供商。如果您不使用 Amazon Route 53 来管理您的公有 DNS 记录，请联系您的 DNS 提供商以了解如何添加记录。
>
> 如果您使用 Amazon Route 53 作为 DNS 提供商，则可以为子域名添加别名记录或别名记录。对于根域，请确保使用别名记录。

您可以从 Amazon Route 53 或其他提供商处购买域名。要使用亚马逊 Route 53 购买域名，请参阅亚马逊 R oute 53 开发者指南中的注册新域名。

有关如何在 Route 53 中配置 DNS 目标的说明，请参阅 Amazon Route 53 开发者指南中的将流量路由到您的资源。

有关如何在其他注册商（例如 Shopify GoDaddy、Hover 等）上配置 DNS 目标的说明，请参阅他们关于添加 DNS 目标记录的特定文档。

指定要与 App Runner 服务关联的域

您可以通过以下方式指定要与 App Runner 服务关联的域：

- 根域 — DNS 有一些固有的限制，可能会阻止您为根域名创建 CNAME 记录。例如，如果您的域名是`example.com`，则可以创建别名记录，将流量路由`acme.example.com`到您的 App Runner 服务。但是，您无法创建将流量路由`example.com`到您的 App Runner 服务的别名记录。要创建根域，请确保添加别名记录。

  别名记录特定于 Route 53，与 CNAME 记录相比，它具有以下优点：

  - Route 53 为您提供了更大的灵活性，因为可以为根域或子域创建别名记录。例如，如果您的域名是example.com，则可以创建一条记录，将请求路由acme.example.com到您的example.com App Runner 服务。
  - 它更具成本效益。这是因为 Route 53 不对使用别名记录路由流量的请求收费。
- 子域名-例如，login.example.com或admin.login.example.com。您也可以选择将www子域关联为同一操作的一部分。您可以为子域名添加 CNAME 或别名记录。
- 通配符-例如，*.example.com。在这种情况下，您不能使用该www选项。您只能将通配符指定为根域名的直接子域，并且只能将其指定为根域名的直接子域。这些不是有效的规范：login*.example.com，*.login.example.com。此通配符规范将所有直接子域名关联起来，并且不关联根域名本身。根域必须通过单独的操作进行关联。

更具体的域名关联会覆盖不太具体的域名关联。例如，login.example.com覆盖*.example.com。使用更具体的协会的证书和别名记录。

以下示例显示了如何使用多个自定义域名关联：

1. example.com与您的服务的主页关联。启用www以进行关联www.example.com。
2. login.example.com与您的服务的登录页面关联。
3. *.example.com与自定义 “未找到” 页面关联。

取消关联（取消关联）自定义域名

您可以取消自定义域名与 App Runner 服务的关联（取消关联）。当您取消链接某个域时，App Runner 会停止将流量从该域路由到您的 Web 应用程序。

> **Note**
>
> 您必须删除与 DNS 服务器取消关联的域的记录。

App Runner 在内部创建用于跟踪域有效性的证书。这些证书存储在 AWS Certificate Manager (ACM) 中。App Runner 不会在域名与您的服务解除关联后的 7 天内或服务被删除后的 7 天内删除这些证书。

管理自定义域名

使用以下方法之一管理您的 App Runner 服务的自定义域名：

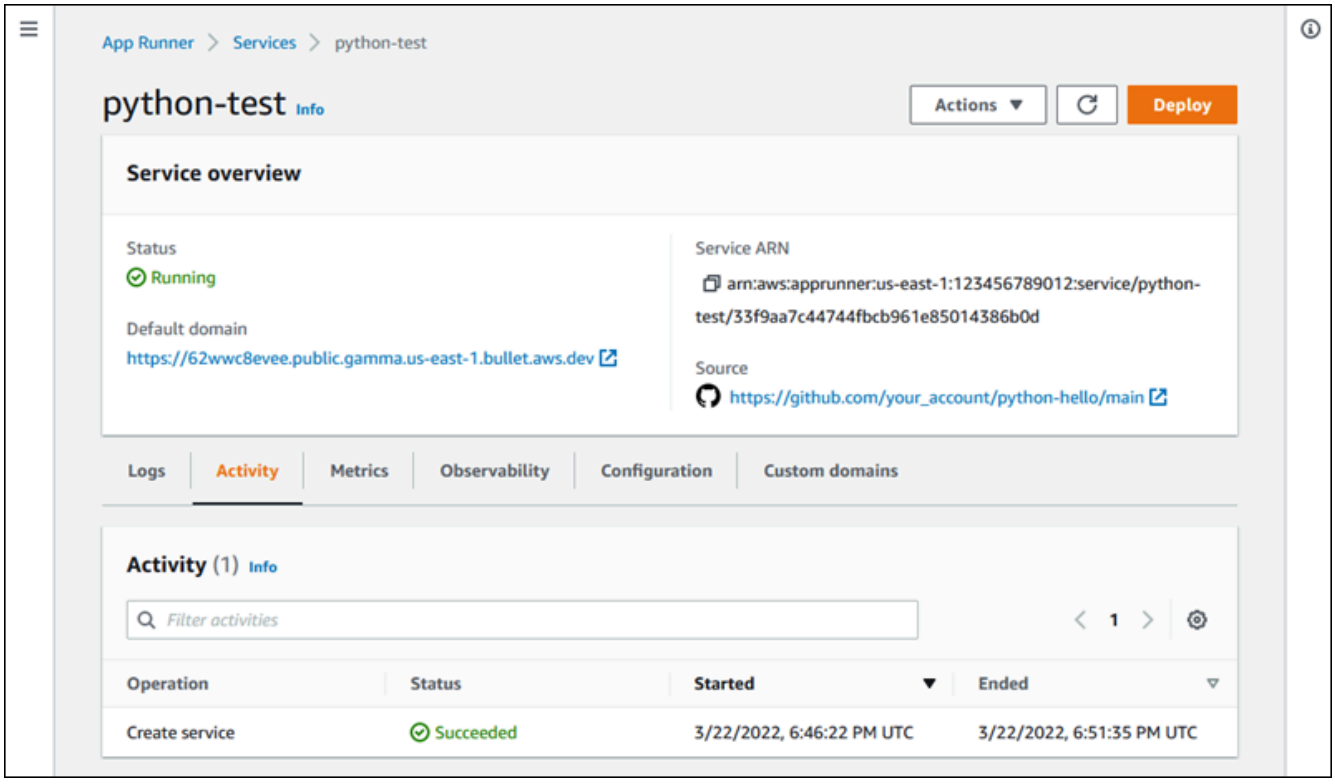
> **Note**
>
> 为了更好地兼容 App Runner 服务，我们建议您使用 Amazon Route 53 作为 DNS 提供商。如果您不使用 Amazon Route 53 来管理您的公有 DNS 记录，请联系您的 DNS 提供商以了解如何添加记录。
> 如果您使用 Amazon Route 53 作为 DNS 提供商，则可以为子域名添加别名记录或别名记录。对于根域，请确保使用别名记录。

App Runner console

使用 App Runner 控制台关联（链接）自定义网域

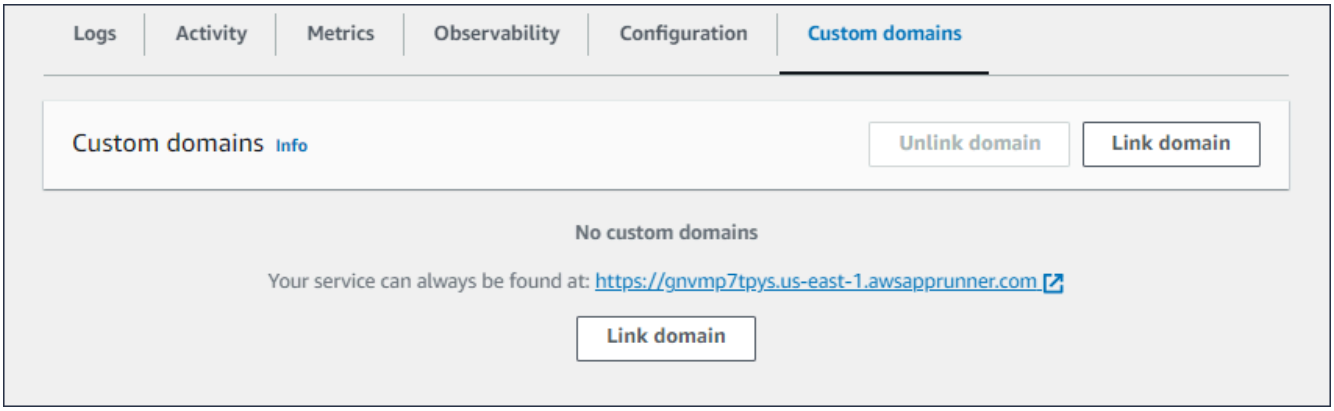
1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的 App Runner 服务。

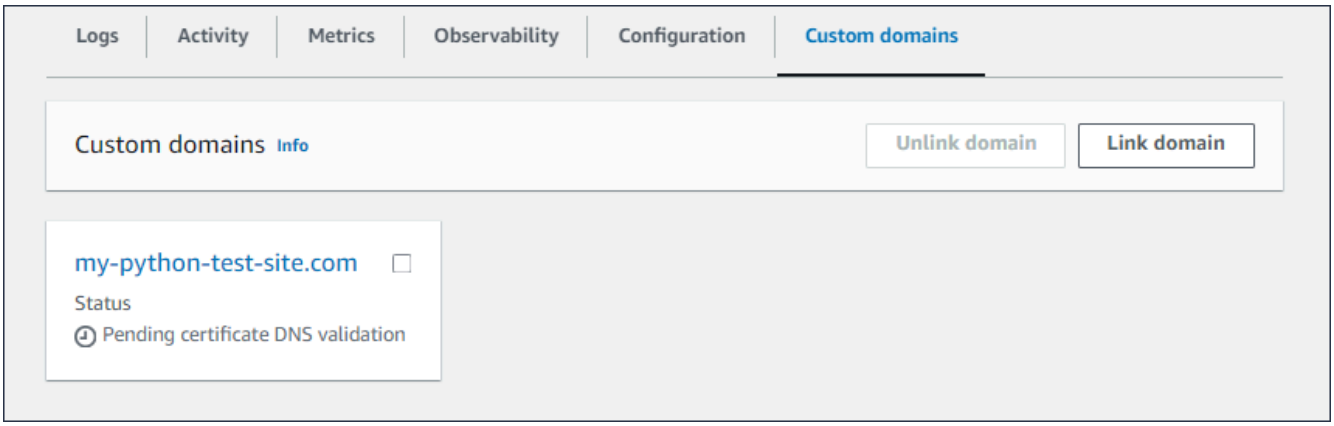
   控制台显示带有服务概述的服务仪表板。



3. 在服务控制面板页面上，选择自定义域名选项卡。

   控制台显示与您的服务关联的自定义域名，或者不显示自定义域名。

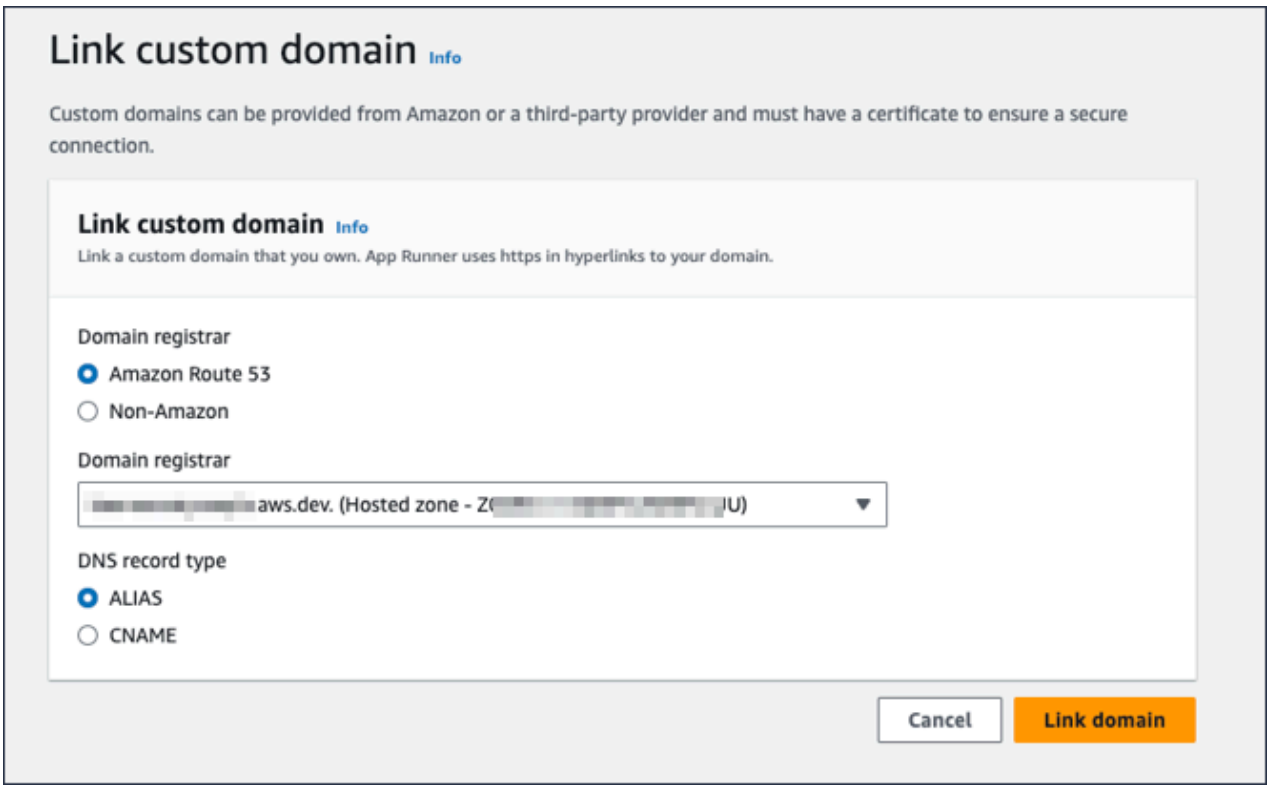




4. 在自定义域名选项卡上，选择关联域名。
5. 将显示 “链接自定义域名” 页面。
   - 如果您的自定义域名是在亚马逊 Route 53 上注册的，请为域名注册商选择亚马逊 Route 53。
     a. 从下拉列表中选择域名。此列表显示您的 Route 53 域名的名称和托管区域 ID。

     > **Note**
     > 您必须首先使用与管理其他 App Runner 资源相同的 AWS 账户，使用亚马逊 Route 53 服务创建 Route 53 域。

     b. 选择 D NS 记录类型。
     c. 选择链接域名。

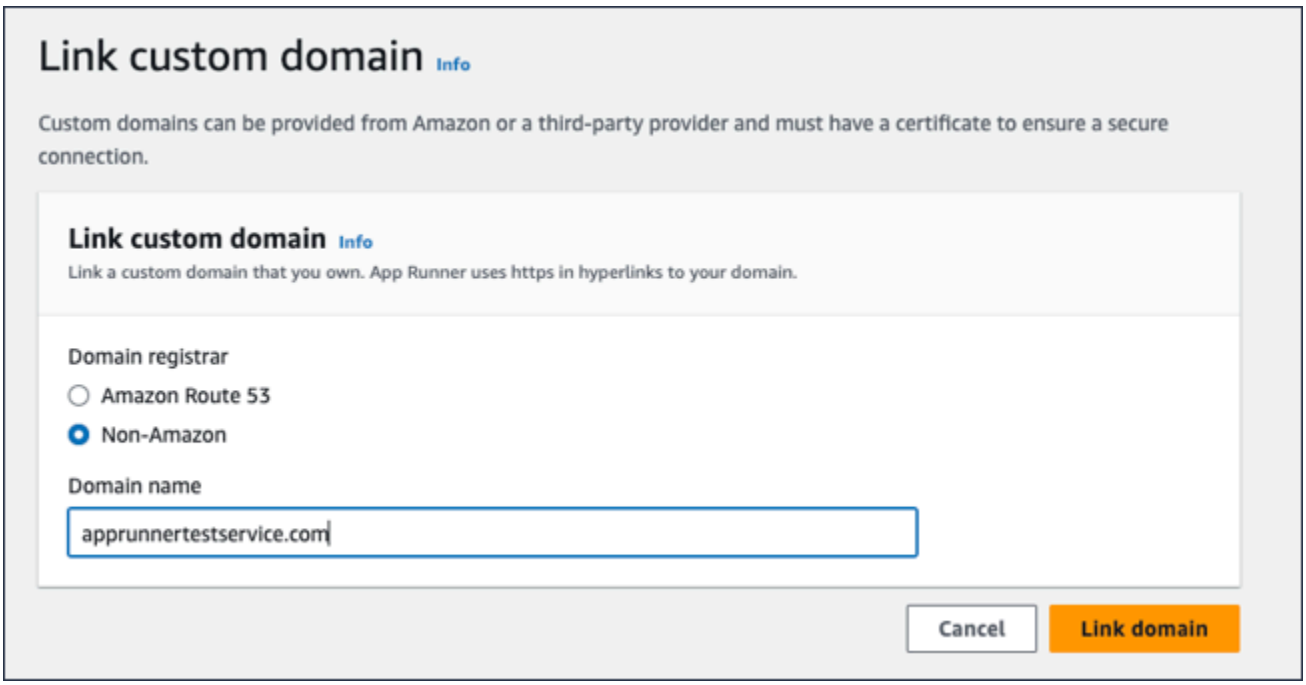


> **Note**
>
> 如果 App Runner 显示错误消息，指出自动配置尝试失败，则可以继续手动配置 DNS 记录。如果以前取消了同一个域名与服务的关联，但没有指向该服务之后被删除的 DNS 提供商记录，则可能会出现此问题。在这种情况下，App Runner 被禁止自动覆盖这些记录。要完成 DNS 配置，请跳过此过程中的其余步骤，然后按照中的说明进行操作配置 Amazon Route 53 别名记录。

- 如果您的自定义域名是在其他域名注册商处注册的，请为域名注册商选择非—亚马逊。

  a. 输入域名。

  b. 选择链接域名。



6. 将显示 “配置 DNS” 页面。

   - 如果 Amazon Route 53 是您的 DNS 提供商，则此步骤是可选的。

     此时，App Runner 已自动为你的 Route 53 域配置了所需的证书验证和 DNS 记录。

     > **Note**
     >
     > 如果此前未将同一个域名与服务取消链接，但之后没有指向该服务删除的 DNS 提供商记录，那么 App Runner 尝试的自动配置可能会失败。要解决此问题并完成 DNS 关联，请继续执行 “配置 DNS” 页面上的步骤 (1) 和 (2)，将当前目标和证书记录复制到 DNS 提供商。

   - 复制证书验证记录和 DNS 目标记录，并将它们添加到您的 DNS 服务器。然后，App Runner 可以验证您是否拥有或控制该域名。

     > **Note**
     >
     > 要自动续订您的自定义域名证书，请确保不要从 DNS 服务器上删除证书验证记录。

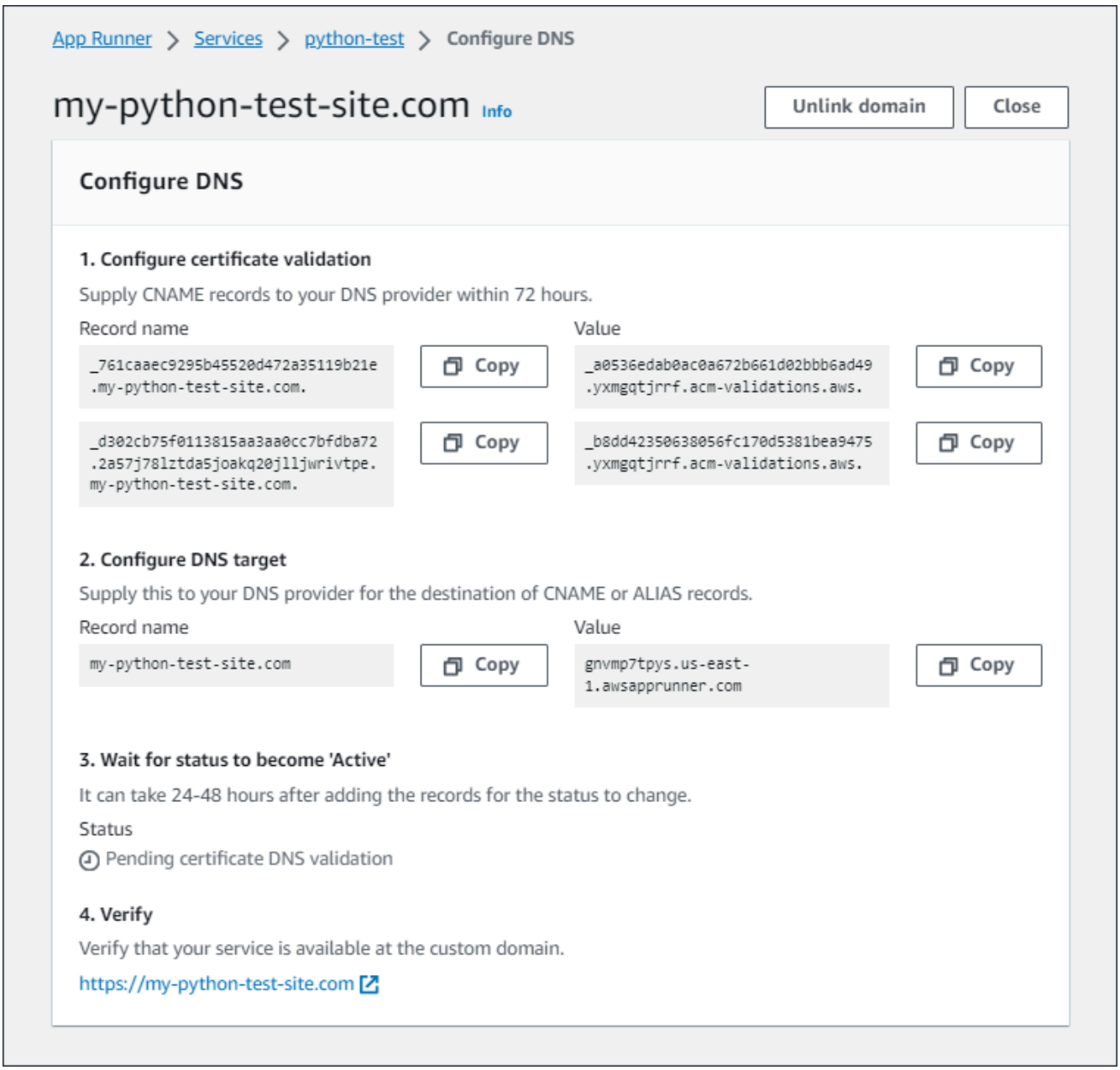
- 有关配置证书验证的更多信息，请参阅《AWS Certificate Manager 用户指南》中的 DNS 验证。
- 有关如何使用 Amazon Route 53 别名记录配置 DNS 目标的信息，请参阅the section called “配置 Amazon Route 53 别名记录”。

- 如果您使用的是 Amazon Route 53 以外的 DNS 提供商，请按照以下步骤操作。

  - 复制证书验证记录和 DNS 目标记录，并将它们添加到您的 DNS 服务器。然后，App Runner 可以验证您是否拥有或控制该域名。

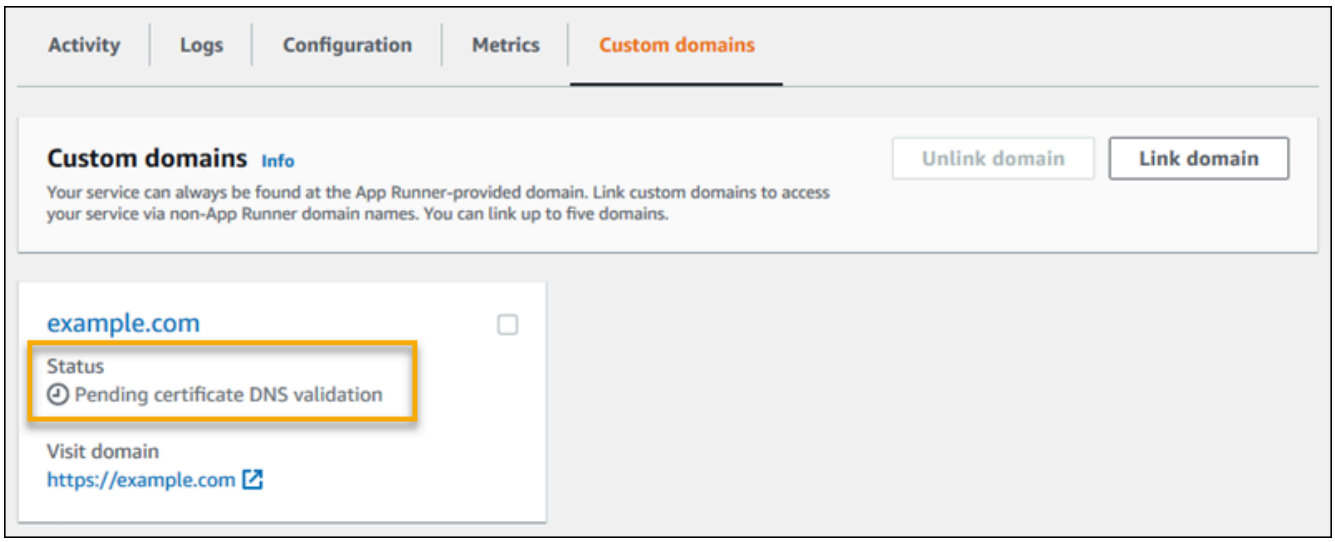
    > **Note**
    >
    > 要自动续订您的自定义域名证书，请确保不要从 DNS 服务器上删除证书验证记录。

    - 有关配置证书验证的更多信息，请参阅《AWS Certificate Manager 用户指南》中的 DNS 验证。
    - 有关如何在其他注册商（例如 Shopify GoDaddy、Hover 等）上配置 DNS 目标的说明，请参阅他们关于添加 DNS 目标的特定文档。

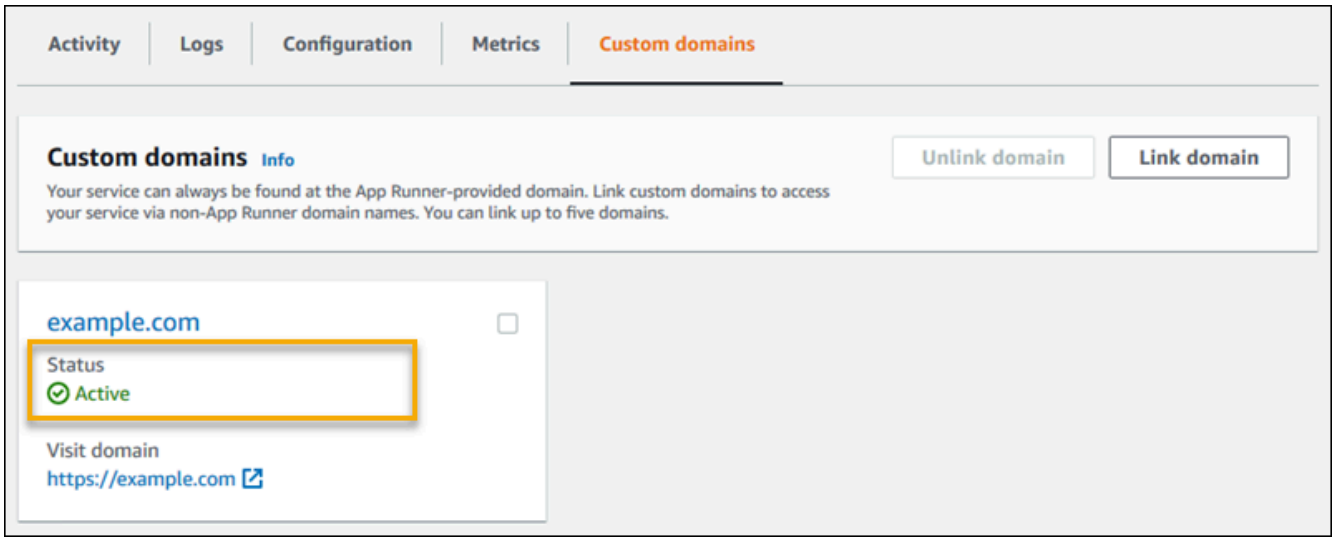


7. 选择 “关闭”

控制台再次显示仪表板。自定义域名选项卡上有一个新磁贴，显示您刚刚关联的域名处于 “待处理证书 DNS 验证” 状态。



8. 当域状态更改为“活动”时，请通过浏览该域来验证该域是否可以路由流量。



> **Note**
>
> 有关如何解决与自定义域名相关的错误的说明，请参阅the section called “自定义域名”。

使用 App Runner 控制台取消关联（取消关联）自定义域

1. 在自定义域名选项卡上，选择要取消关联的域名的磁贴，然后选择取消关联域名。
2. 在取消链接域对话框中，通过选择取消链接域来验证操作。

> **Note**
>
> 您必须删除与 DNS 服务器取消关联的域的记录。

App Runner API or AWS CLI

要使用 App Runner API 或将自定义网域与您的服务相关联 AWS CLI，请调用 AssociateCustomDomainAPI 操作。调用成功后，将返回一个描述与您的服务关联的自定义域的CustomDomain对象。该对象显示CREATING状态并包含CertificateValidationRecord对象列表。该调用还会返回可用于配置 DNS 目标的目标别名。这些记录可以添加到您的 DNS 中。

要使用 App Runner API 取消自定义网域与您的服务的关联 AWS CLI，请调用 DisassociateCustomDomainAPI 操作。调用成功后，将返回一个CustomDomain对象，该对象描述了与您的服务取消关联的自定义域。该对象显示DELETING状态。

主题

- 为您的目标 DNS 配置 Amazon Route 53 别名记录

为您的目标 DNS 配置 Amazon Route 53 别名记录

> **Note**
>
> 如果 Amazon Route 53 是您的 DNS 提供商，则无需遵循此程序。在这种情况下，App Runner 会自动为你的 Route 53 域配置所需的证书验证和 DNS 记录，以链接到你的 App Runner Web 应用程序。
> 如果 App Runner 的自动配置尝试失败，请按照以下步骤完成 DNS 配置。如果之前取消了同一个域名与服务的关联，但之后没有指向该服务的 DNS 提供商记录，则会阻止 App Runner 自动覆盖这些记录。此过程说明了如何将它们手动复制到您的 Route 53 DNS。

您可以使用 Amazon Route 53 作为您的 DNS 提供商，将流量路由到您的 App Runner 服务。它是一项高度可用且可扩展的域名系统 (DNS) Web 服务。Amazon Route 53 记录包含控制流量如何路由到您的 App Runner 服务的设置。您可以创建 CNAME 记录或别名记录。有关别名记录和别名记录的比较，请参阅 A mazon Route 53 开发者指南中的在别名和非别名记录之间进行选择。

> **Note**
>
> 亚马逊 Route 53 目前支持 2022 年 8 月 1 日之后创建的服务的别名记录。

Amazon Route 53 console

配置 Amazon Route 53 别名记录

1. 登录 AWS Management Console 并打开 Route 53 控制台。
2. 在导航窗格中，选择 Hosted zones（托管区域）。
3. 选择要用于将流量路由到 App Runner 服务的托管区域的名称。
4. 选择创建记录。
5. 指定以下值：
   - 路由策略：选择适用的路由策略。有关更多信息，请参阅选择路由策略。
   - 记录名称：输入您要用于将流量路由到您的 App Runner 服务的域名。默认值为托管区域的名称。例如，如果托管区域的名称为，`example.com`并且您想使用该名称将流量路由`acme.example.com`到您的环境，请输入`acme`。
   - 值/将流量路由到：选择 App Runner 应用程序的别名，然后选择终端节点所在的区域。选择要将流量路由到的应用程序的域名。
   - 记录类型：接受默认的 A — IPv4 地址。
   - 评估目标生命值：接受默认值 “是”。
6. 选择创建记录。

您创建的 Route 53 别名记录将在 60 秒内传播到所有 Route 53 服务器上。当使用您的别名记录传播 Route 53 服务器时，您可以使用您创建的别名记录的名称将流量路由到您的 App Runner 服务。

有关如何在 DNS 更改传播时间过长时进行故障排除的信息，请参阅为什么我的 DNS 更改需要这么长时间才在 Route 53 和公共解析器中传播？ 。

Amazon Route 53 API or AWS CLI

使用亚马逊 Route 53 API 配置亚马逊 Route 53 别名记录或 AWS CLI 调用 ChangeResourceRecordSetsAPI 操作。要了解 Route 53 的目标托管区域 ID，请参阅服务终端节点。

暂停和恢复 App Runner 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。App Runner 会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，可以恢复 App Runner 服务。App Runner 将预置新的计算容量，为其部署应用程序，然后运行该应用程序。您的应用程序源不会重新部署，也无需构建。相反，App Runner 会使用你当前部署的版本进行恢复。您的应用程序会保留其 App Runner 域。

Important

- 当您暂停服务时，您的应用程序会失去其状态。例如，您的代码使用的任何临时存储都将丢失。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，则在恢复服务之前无法部署新版本。

  因此，我们建议您保持服务运行并回滚到上一个稳定的应用程序版本。
- 当您恢复服务时，App Runner 会部署在您暂停服务之前使用的最后一个应用程序版本。如果您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，假设您在镜像仓库中有新的镜像版本或者在代码仓库中有新的提交。这些版本不会自动部署。

  要部署更新的版本，请在恢复 App Runner 服务后执行手动部署或向源存储库中添加其他版本。

暂停和删除对比

暂停您的 App Runner 服务以暂时将其禁用。只有计算资源会被终止，您存储的数据（例如，带有应用程序版本的容器映像）保持不变。恢复服务很快 — 您的应用程序已准备好部署到新的计算资源。您的 App Runner 域名保持不变。

删除您的 App Runner 服务以将其永久删除。您存储的数据已删除。如果您需要重新创建服务，App Runner 需要重新获取源代码，如果是代码存储库，还需要构建它。您的 Web 应用程序将获得一个新的 App Runner 域。

当您的服务暂停时

当您暂停服务且服务处于 “已暂停” 状态时，它对操作请求（包括 API 调用或控制台操作）的响应会有所不同。服务暂停后，您仍然可以执行 App Runner 操作，这些操作不会以影响其运行时的方式修改服务的定义或配置。换句话说，如果某项操作改变了正在运行的服务的行为、规模或其他特征，则无法对已暂停的服务执行该操作。

以下列表提供了有关您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。

您可以对已暂停的服务执行的操作

- *List**和*Describe**操作-仅读取信息的操作。
- *DeleteService*— 您可以随时删除服务。
- *TagResource*，*UntagResource*— 标签与服务关联，但不是其定义的一部分，也不会影响其运行时行为。

您无法对已暂停的服务执行的操作

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

暂停并恢复您的服务

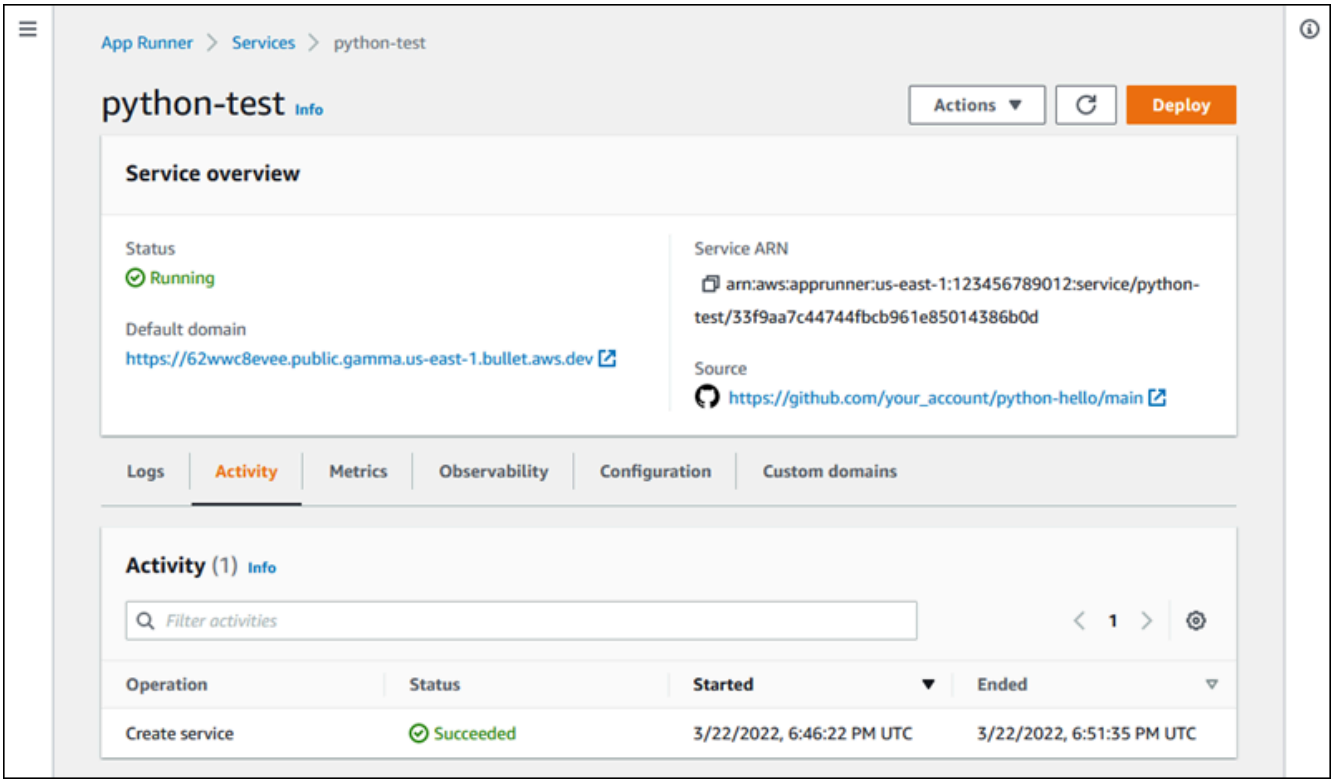
使用以下方法之一暂停和恢复 App Runner 服务：

App Runner console

使用 App Runner 控制台暂停服务

1. 打开 App Runner 控制台，然后在 “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 App Runner 服务。

   控制台显示带有服务概述的服务仪表板。



3. 选择 “操作”，然后选择 “暂停”。

   在服务控制面板页面上，服务状态更改为操作进行中，然后更改为已暂停。您的服务现已暂停。

使用 App Runner 控制台恢复服务

1. 选择 “操作”，然后选择 “恢复”。

   在服务控制面板页面上，服务状态更改为操作进行中。

2. 等待服务恢复。在服务仪表板页面上，服务状态更改回正在运行。
3. 要验证恢复服务是否成功，请在服务控制面板页面上选择 App Runner 域值。这是您的服务网站的网址。验证您的 Web 应用程序是否正常运行。

App Runner API or AWS CLI

要使用 App Runner API 或暂停服务 AWS CLI，请调用 PauseServiceAPI 操作。如果调用返回成功响应并显示`"Status": "OPERATION_IN_PROGRESS"`服务对象，则 App Runner 会开始暂停您的服务。

要使用 App Runner API 恢复服务 AWS CLI，或者请调用 ResumeServiceAPI 操作。如果调用返回成功响应并显示"Status": "OPERATION_IN_PROGRESS"服务对象，则 App Runner 会开始恢复您的服务。

删除 App Runner 服务

当您想要终止 AWS App Runner 服务中运行的 Web 应用程序时，可以删除该服务。删除服务会停止正在运行的 Web 服务、移除底层资源并删除您的关联数据。

出于以下一个或多个原因，您可能需要删除 App Runner 服务：

- 你不再需要该 Web 应用程序了 — 例如，它已停用，或者它是你用完的开发版本。
- 您已达到 App Runner 服务配额 — 您想在同一个服务中创建一项新服务， AWS 区域 并且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 “应用程序运行器资源配额”。
- 安全或隐私注意事项 — 你希望 App Runner 删除它为你的服务存储的数据。

暂停和删除对比

暂停您的 App Runner 服务以暂时将其禁用。只有计算资源会被终止，您存储的数据（例如，带有应用程序版本的容器映像）保持不变。恢复服务很快 — 您的应用程序已准备好部署到新的计算资源。您的 App Runner 域名保持不变。

删除您的 App Runner 服务以将其永久删除。您存储的数据已删除。如果您需要重新创建服务，App Runner 需要重新获取源代码，如果是代码存储库，还需要构建它。您的 Web 应用程序将获得一个新的 App Runner 域。

App Runner 会删除什么？

当您删除服务时，App Runner 会删除一些关联项目，而不会删除其他项目。以下列表提供了详细信息。

App Runner 删除的项目：

- 容器映像 — 您部署的映像或 App Runner 根据您的源代码构建的映像的副本。它使用 App Runner 拥有的 AWS 账户 内部存储在亚马逊弹性容器注册表 (Amazon ECR) Container Registry 中。
- 服务配置-与您的 App Runner 服务关联的配置设置。它们使用 App Runner 拥有的 AWS 账户 内部存储在亚马逊 DynamoDB 中。

App Runner 无法删除的项目：

- 连接-您的连接可能与您的服务相关联。App Runner 连接是一种单独的资源，可以在多个 App Runner 服务之间共享。如果您不再需要该连接，则可以明确将其删除。有关更多信息，请参阅 the section called “连接”。
- 自定义域证书 — 如果您将自定义域链接到 App Runner 服务，App Runner 会在内部创建用于跟踪域有效性的证书。它们存储在 AWS Certificate Manager (ACM) 中。App Runner 不会在域名与您的服务取消关联后的七天内或该服务被删除后的七天内删除证书。有关更多信息，请参阅 the section called “自定义域名”。

删除您的服务

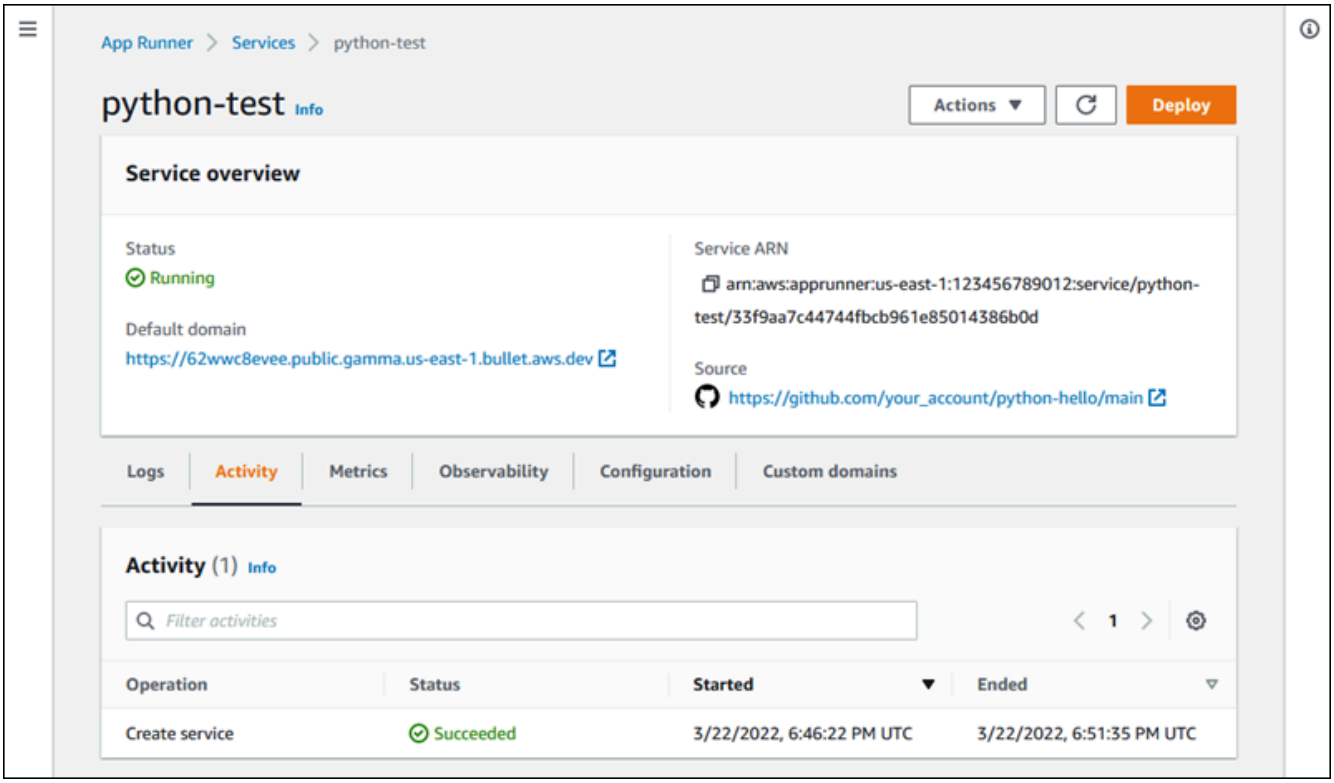
使用以下方法之一删除您的 App Runner 服务：

App Runner console

使用 App Runner 控制台删除您的服务

1. 打开 App Runner 控制台，然后在 “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 App Runner 服务。

   控制台显示带有服务概述的服务仪表板。



3. 选择操作，然后选择删除。

   控制台会将您带到 “服务” 页面。已删除的服务将显示操作进行中状态，然后该服务将从列表中消失。您的服务现已删除。

App Runner API or AWS CLI

要使用 App Runner API 或删除您的服务 AWS CLI，请调用 DeleteServiceAPI 操作。如果调用返回成功响应并显示`"Status": "OPERATION_IN_PROGRESS"`服务对象，则 App Runner 会开始删除您的服务。

引用环境变量

借助 App Runner，您可以在创建服务或更新服务时将机密和配置作为服务中的环境变量引用。

您可以将非敏感配置数据（例如超时和重试次数）以纯文本形式引用为键值对。您在纯文本中引用的配置数据未经过加密，其他人可以在 App Runner 服务配置和应用程序日志中看到这些数据。

> **Note**
>
> 出于安全考虑，请勿在 App Runner 服务中以纯文本形式引用任何敏感数据。

将敏感数据引用为环境变量

App Runner 支持在服务中安全地将敏感数据引用为环境变量。考虑将要引用的敏感数据存储在我们的 P AWS Systems Manager arameter St AWS Secrets Managerore 中。然后，您可以从 App Runner 控制台或通过调用 API 在服务中安全地将它们作为环境变量引用。这可以有效地将密钥和参数管理与应用程序代码和服务配置分开，从而提高在 App Runner 上运行的应用程序的整体安全性。

> **Note**
>
> App Runner 不会因为将 Secrets Manager 和 SSM Parameter Store 引用为环境变量而向你收取费用。但是，使用 Secrets Manager 和 SSM Parameter Store 需要支付标准价格。
> 有关定价的更多信息，请参阅以下内容：
>
> - AWS Secrets Manager 定价
> - AWS SSM 参数存储定价

以下是将敏感数据作为环境变量引用的过程：

1. 将敏感数据（例如 API 密钥、数据库凭据、数据库连接参数或应用程序版本）作为机密或参数存储在任一 AWS Secrets Manager 或 Paramet AWS Systems Manager er Store 中。
2. 更新您的实例角色的 IAM 策略，这样 App Runner 就可以访问存储在 Secrets Manager 和 SSM Parameter Store 中的机密和参数。有关更多信息，请参阅 权限。

3. 通过指定名称并提供其 Amazon 资源名称 (ARN)，将密钥和参数安全地引用为环境变量。您可以在创建服务或更新服务配置时添加环境变量。您可以使用以下选项之一来添加环境变量：
   - 应用程序运行器控制台
   - 应用程序运行器 API
   - `apprunner.yaml` 配置文件

   > **Note**
   >
   > 在创建或更新 `PORT` App Runner 服务时，不能将环境变量指定为名称。它是 App Runner 服务的预留环境变量。

   有关如何引用密钥和参数的更多信息，请参阅管理环境变量。

> **Note**
>
> 由于 App Runner 仅存储对机密和参数 ARN 的引用，因此其他人无法在 App Runner 服务配置和应用程序日志中看到敏感数据。

注意事项

- 请务必更新您的实例角色，使其具有访问参数存储中 AWS Secrets Manager 或其中的密钥和 AWS Systems Manager 参数的相应权限。有关更多信息，请参阅 权限。
- 确保 AWS Systems Manager Parameter Store AWS 账户 与您要启动或更新的服务位于同一位置。目前，您无法跨账户引用 SSM 参数存储参数。
- 轮换或更改密钥和参数值时，它们不会在您的 App Runner 服务中自动更新。重新部署你的 App Runner 服务，因为 App Runner 在部署期间仅提取机密和参数。
- 您还可以选择通过 App Runner 服务中的 SDK 直接调用 AWS Secrets Manager 和 P AWS Systems Manager arameter Store。
- 为避免错误，在将它们引用为环境变量时，请确保以下几点：
  - 您可以指定密钥的正确 ARN。
  - 您可以指定参数的正确名称或 ARN。

权限

要启用引用存储在 AWS Secrets Manager 或 SSM Parameter Store 中的密钥和参数，请向您的实例角色的 IAM 策略添加相应的权限，以访问 Secrets Manager 和 SSM Parameter Store。

> **Note**
>
> 未经您的许可，App Runner 无法访问您账户中的资源。您可以通过更新您的 IAM 政策来提供权限。

您可以使用以下策略模板在 IAM 控制台中更新您的实例角色。您可以修改这些策略模板以满足您的特定要求。有关更新实例角色的更多信息，请参阅 IAM 用户指南中的修改角色。

> **Note**
>
> 创建环境变量时，您也可以从 App Runner 控制台复制以下模板。

将以下模板复制到您的实例角色中，以添加引用密钥的权限AWS Secrets Manager。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "kms:Decrypt*"
      ],
      "Resource": [
        "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:<secret_name>",
        "arn:aws:kms:<region>:<aws_account_id>:key/<key_id>"
      ]
    }
  ]
}
```

将以下模板复制到您的实例角色，以添加从 P AWS Systems Managerarameter Store 引用参数的权限。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameters"
      ],
      "Resource": [
        "arn:aws:ssm:<region>:<aws_account_id>:parameter/<parameter_name>"
      ]
    }
  ]
}
```

管理您的环境变量

使用以下方法之一管理 App Runner 服务的环境变量：

- the section called “应用程序运行器控制台”
- the section called “应用程序运行器 API 或 AWS CLI”

应用程序运行器控制台

在 App Runner 控制台上创建服务或更新服务时，可以添加环境变量。

添加环境变量

添加环境变量

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 根据您是在创建还是更新服务，执行以下步骤之一：
   - 如果您要创建新服务，请选择创建 App Runner 服务，然后转到配置服务。
   - 如果您要更新现有服务，请选择要更新的服务，然后转到该服务的配置选项卡。
3. 转到环境变量-在 “服务设置” 下可选。
4. 根据您的要求选择以下任一选项：
   - 从环境变量源中选择纯文本，然后分别在环境变量名称和环境变量值下输入其键值对。

> **Note**
>
> 如果要引用非敏感数据，请选择纯文本。这些数据未经加密，在 App Runner 服务配置和应用程序日志中对其他人可见。

- 从环境变量源中选择 Secr ets Manager，以引用 AWS Secrets Manager 作为环境变量存储在服务中的密钥。分别在环境变量名称和环境变量值下提供您所引用的密钥的环境变量名称和亚马逊资源名称 (ARN)。
- 从环境变量源中选择 SSM 参数存储，将存储在 SSM 参数存储中的参数作为服务中的环境变量引用。分别在环境变量名称和环境变量值下提供您所引用的参数的环境变量名称和 ARN。

> **Note**
>
> - 在创建或更新 PORT App Runner 服务时，不能将环境变量指定为名称。它是 App Runner 服务的保留环境变量。
> - 如果 SSM 参数存储参数与您要启动的服务 AWS 区域 相同，则可以指定完整的 Amazon 资源名称 (ARN) 或参数名称。如果参数位于不同的区域，则需要指定完整的 ARN。
> - 确保你所引用的参数与你正在启动或更新的服务位于同一个账户中。目前，您无法跨账户引用 SSM 参数存储参数。

5. 选择添加环境变量以引用另一个环境变量。
6. 展开 IAM 策略模板以查看和复制为 AWS Secrets Manager 和 SSM Parameter Store 提供的 IAM 策略模板。只有在您尚未使用所需权限更新实例角色的 IAM 策略时，才需要执行此操作。有关更多信息，请参阅 权限。

移除环境变量

在删除环境变量之前，请确保更新您的应用程序代码以反映相同的内容。如果应用程序代码未更新，则您的 App Runner 服务可能会失败。

移除环境变量

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 转到要更新的服务的配置选项卡。
3. 转到环境变量-在 “服务设置” 下可选。

4. 选择要移除的环境变量旁边的 “移除”。您会收到一条确认删除的消息。
5. 选择删除。

应用程序运行器 API 或 AWS CLI

您可以引用 Secrets Manager 和 SSM Parameter Store 中存储的敏感数据，方法是将它们作为环境变量添加到服务中。

> **Note**
>
> 更新您的实例角色的 IAM 策略，这样 App Runner 就可以访问存储在 Secrets Manager 和 SSM Parameter Store 中的机密和参数。有关更多信息，请参阅 权限。

将密钥和配置作为环境变量引用

1. 在 Secrets Manager 或 SSM 参数存储区中创建密钥或配置。

   以下示例说明如何使用 SSM 参数存储区创建密钥和参数。

   Example 创建密钥-请求

   以下示例说明如何创建代表数据库凭证的密钥。

   ```
   aws secretsmanager create-secret \
   –name DevRdsCredentials \
   –description "Rds credentials for development account." \
   –secret-string "{\"user\":\"diegor\",\"password\":\"EXAMPLE-PASSWORD\"}"
   ```

   Example 创建密钥-响应

   ```
   arn:aws:secretsmanager:<region>:<aws_account_id>:secret:DevRdsCredentials
   ```

   Example 创建配置-请求

   以下示例说明如何创建表示 RDS 连接字符串的参数。

   ```
   aws systemsmanager put-parameter \
   –name DevRdsConnectionString \
   ```

```
–value "mysql2://dev-mysqlcluster-rds.com:3306/diegor" \
–type "String" \
–description "Rds connection string for development account."
```

Example 创建配置-响应

```
arn:aws:ssm:<region>:<aws_account_id>:parameter/DevRdsConnectionString
```

2. 通过将存储在 Secrets Manager 和 SSM Parameter Store 中的密钥和配置添加为环境变量来引用它们。您可以在创建或更新 App Runner 服务时添加环境变量。

   以下示例说明如何在基于代码和基于图像的 App Runner 服务上将密钥和配置作为环境变量引用。

   Example 基于图像的 App Runner 服务的输入.json 文件

```
{
  "ServiceName": "example-secrets",
  "SourceConfiguration": {
    "ImageRepository": {
      "ImageIdentifier": "<image-identifier>",
      "ImageConfiguration": {
        "Port": "<port>",
        "RuntimeEnvironmentSecrets": {

 "Credential1":"arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
          "Credential2":"arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
        }
      },
      "ImageRepositoryType": "ECR_PUBLIC"
    }
  },
  "InstanceConfiguration": {
    "Cpu": "1 vCPU",
    "Memory": "3 GB",
    "InstanceRoleArn": "<instance-role-arn>"
  }
}
```

   Example 基于图像的 App Runner 服务-请求

```
aws apprunner create-service \
```

```
--cli-input-json file://input.json
```

Example 基于图像的 App Runner 服务-响应

```
{
...
      "ImageRepository": {
         "ImageIdentifier":"<image-identifier>",
         "ImageConfiguration":{
            "Port": "<port>",
            "RuntimeEnvironmentSecrets":{
               "Credential1":
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
               "Credential2": "arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
            },
            "ImageRepositoryType":"ECR"
      }
   },
   "InstanceConfiguration": {
        "CPU": "1 vCPU",
        "Memory": "3 GB",
        "InstanceRoleArn: "<instance-role-arn>"
   }
...
}
```

Example 基于代码的 App Runner 服务的输入.json 文件

```
{
  "ServiceName": "example-secrets",
  "SourceConfiguration": {
    "AuthenticationConfiguration": {
      "ConnectionArn": "arn:aws:apprunner:us-east-1:123456789012:connection/my-
github-connection/XXXXXXXXXX"
    },
    "AutoDeploymentsEnabled": false,
    "CodeRepository": {
      "RepositoryUrl": "<repository-url>",
      "SourceCodeVersion": {
        "Type": "BRANCH",
        "Value": "main"
```

```
      },
      "CodeConfiguration": {
        "ConfigurationSource": "API",
        "CodeConfigurationValues": {
          "Runtime": "<runtime>",
          "BuildCommand": "<build-command>",
          "StartCommand": "<start-command>",
          "Port": "<port>",
          "RuntimeEnvironmentSecrets": {

 "Credential1":"arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX",
            "Credential2":"arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
          }
        }
      }
    }
  },
  "InstanceConfiguration": {
    "Cpu": "1 vCPU",
    "Memory": "3 GB",
    "InstanceRoleArn": "<instance-role-arn>"
  }
}
```

Example 基于代码的 App Runner 服务-请求

```
aws apprunner create-service \
--cli-input-json file://input.json
```

Example 基于代码的 App Runner 服务-响应

```
{
...
   "SourceConfiguration":{
      "CodeRepository":{
         "RepositoryUrl":"<repository-url>",
         "SourceCodeVersion":{
            "Type":"Branch",
            "Value":"main"
         },
         "CodeConfiguration":{
```

```
            "ConfigurationSource":"API",
            "CodeConfigurationValues":{
               "Runtime":"<runtime>",
               "BuildCommand":"<build-command>",
               "StartCommand":"<start-command>",
               "Port":"<port>",
               "RuntimeEnvironmentSecrets":{
                  "Credential1" :
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXX",
                  "Credential2" : "arn:aws:ssm:<region>:<aws_account_id>:parameter/
<parameter-name>"
               }
            }
         }
      },
      "InstanceConfiguration": {
        "CPU": "1 vCPU",
        "Memory": "3 GB",
        "InstanceRoleArn: "<instance-role-arn>"
   }
...
}
```

3. apprunner.yaml模型已更新，以反映新增的秘密。

   以下是更新后的apprunner.yaml模型的示例。

   Example **apprunner.yaml**

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - python -m pip install flask
run:
  command: python app.py
  network:
    port: 8080
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
```

```
      value-from:
 "arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXXX"
    - name: my-parameter
      value-from: "arn:aws:ssm:<region>:<aws_account_id>:parameter/<parameter-
name>"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

与应用运行器联网

本章介绍您的 AWS App Runner 服务的网络配置。

从本章中，您将学到以下内容：

- 如何为私有和公共端点配置传入流量。有关更多信息，请参阅为传入流量设置网络配置。
- 如何配置您的传出流量以访问在 Amazon VPC 中运行的其他应用程序。有关更多信息，请参阅为出站流量启用 VPC 访问权限。

> **Note**
>
> App Runner 目前仅支持公共传入流量的双栈（IPv4 和 IPv6）地址类型。对于传出流量和私人传入流量，仅支持 IPv4。

主题

术语

为了了解如何自定义网络流量以满足您的需求，让我们来了解一下本章中使用的以下术语。

一般条款

要了解与亚马逊虚拟私有云 (VPC) Amazon Virtual Private Cloud 关联需要什么，让我们了解以下术语：

- VPC：Amazon VPC 是一个逻辑上隔离的虚拟网络，可让您完全控制自己的虚拟网络环境，包括资源放置、连接和安全。它是一个虚拟网络，与您在自己的数据中心中运行的传统网络非常相似。
- VPC 接口终端节点：VPC 接口终端节点（一种 AWS PrivateLink 资源）将 VPC 连接到终端节点服务。创建 VPC 接口终端节点，将流量发送到使用 Network Load Balancer 来分配流量的终端节点服务。发往端点服务的流量使用 DNS 进行解析。

- 区域：每个区域都是一个单独的地理区域，您可以在其中托管 App Runner 服务。
- 可用区：可用区是 AWS 区域内的隔离位置。它是一个或多个具有冗余电源、网络和连接的独立数据中心。可用区可为生产级应用程序提供高可用性、容错能力和可扩展性。
- 子网：子网是您的 VPC 中的 IP 地址范围。子网必须位于单个可用区中。您可以将 AWS 资源启动到指定的子网。对必须连接互联网的资源使用公有子网，而对将不会连接到互联网的资源使用私有子网。
- 安全组：安全组控制允许访问和离开与其关联的资源的流量。安全组提供了额外的安全层来保护每个子网中的 AWS 资源，使您可以更好地控制网络流量。创建 VPC 时，它带有一个默认安全组。您可以为每个 VPC 创建额外的安全组。您只能将安全组与为其创建的 VPC 内的资源相关联。
- 双堆栈：双堆栈是一种支持来自 IPv4 和 IPv6 端点的网络流量的地址类型。

特定于配置传出流量的术语

VPC 连接器

VPC 连接器是一种应用程序运行器资源，它使应用程序运行器服务能够访问在私有 Amazon VPC 中运行的应用程序。

特定于配置传入流量的术语

要了解如何使您的服务只能在 Amazon VPC 内以私密方式访问，让我们了解以下术语：

- VPC 入口连接：VPC 入口连接是一种 App Runner 资源，它为传入流量提供应用程序运行器终端节点。当您在 App Runner 控制台上为传入流量选择私有终端节点时，App Runner 会在后台分配 VPC 入口连接资源。VPC 入口连接资源将您的 App Runner 服务连接到 Amazon VPC 的 VPC 接口终端节点。

  > **Note**
  >
  > 如果您使用的是 App Runner API，则不会自动创建 VPC 入口连接资源。

- 私有终端节点：私有终端节点是一个 App Runner 控制台选项，您可以选择该选项将传入的网络流量配置为只能从 Amazon VPC 内访问。

为传入流量设置网络配置

您可以将服务配置为接收来自私有或公共端点的传入流量。

公共端点是默认配置。它会向来自公共互联网的任何传入流量开放您的服务。它还允许您灵活地为服务选择互联网协议版本 4 (IPv4) 或双栈（IPv4 和 IPv6）地址类型。

私有终端节点仅允许来自 Amazon VPC 的流量访问您的 App Runner 服务。这是通过为 App Runner 服务设置 VPC 接口终端节点（一种 AWS PrivateLink 资源）来实现的。因此，在 Amazon VPC 和您的 App Runner 服务之间创建私有连接。

> **Note**
>
> App Runner 目前仅支持公共端点的双堆栈（IPv4 和 IPv6）地址类型。对于私有终端节点，仅支持 IPv4。

以下是为传入流量设置网络配置时所涉及的主题：

- 如何配置您的传入流量，使您的服务只能在 Amazon VPC 内私下使用。有关更多信息，请参阅为传入流量启用私有端点。
- 如何配置您的服务以接收来自双栈地址类型的互联网流量。有关更多信息，请参阅为公共传入流量启用双堆栈。

标头

使用 App Runner，您可以访问进入应用程序的流量的原始源 IPv4 和 IPv6 地址。通过为原始源 IP 地址分配`X-Forwarded-For`请求标头来保留这些地址。这使您的应用程序能够在需要时获取原始的源 IP 地址。

> **Note**
>
> 如果您的服务配置为使用私有端点，则`X-Forwarded-For`请求标头不能用于访问原始源 IP 地址。如果使用，它会检索假值。

为传入流量启用私有端点

默认情况下，当您创建 AWS App Runner 服务时，该服务可通过 Internet 进行访问。但是，您也可以将您的 App Runner 服务设为私有，并且只能从亚马逊虚拟私有云（亚马逊 VPC）中访问。

将 App Runner 服务私有化后，您可以完全控制传入流量，从而增加一层额外的安全性。这在各种用例中都很有用，包括运行内部 API、企业 Web 应用程序或仍在开发中、需要更高隐私和安全级别或需要满足特定合规性要求的应用程序。

> **Note**
>
> 如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WA F Web ACL。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。因此，与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。
>
> 要了解有关基础设施安全和安全组的更多信息，包括最佳实践，请参阅 Amazon VPC 用户指南中的以下主题：使用安全组控制网络流量和控制流向 AWS 资源的流量。

当您的 App Runner 服务为私有服务时，您可以从 Amazon VPC 中访问您的服务。不需要互联网网关、NAT 设备或 VPN 连接。

> **Note**
>
> App Runner 目前仅支持公共传入流量的双栈（IPv4 和 IPv6）地址类型。对于传出流量和私人传入流量，仅支持 IPv4。

注意事项

- 在为 App Runner 设置 VPC 接口终端节点之前，请查看AWS PrivateLink 指南中的注意事项。
- App Runner 不支持 VPC 终端节点策略。默认情况下，允许通过 VPC 接口终端节点对 App Runner 进行完全访问。或者，您可以将安全组与终端节点网络接口关联，以控制通过 VPC 接口终端节点到 App Runner 的流量。
- 如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WA F Web ACL。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。因此，与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。
- 启用私有终端节点后，只能从您的 VPC 访问您的服务，并且无法通过互联网访问您的服务。
- 为了获得更高的可用性，建议您在可用区中为 VPC 接口终端节点选择至少两个不同的子网。我们不建议只使用一个子网。
- 您可以使用相同的 VPC 接口终端节点访问一个 VPC 中的多个 App Runner 服务。

有关本节中使用的术语的信息，请参阅术语。

权限

以下是启用私有终端节点所需的权限列表：

- ec2: CreateTags
- ec2: CreateVpcEndpoint
- ec2: ModifyVpcEndpoint
- ec2: DeleteVpcEndpoints
- ec2: DescribeSubnets
- ec2: DescribeVpcEndpoints
- ec2: DescribeVpcs

VPC 接口终端节点

VPC 接口终端节点是一种将 Amazon VPC 连接到终端节点服务的AWS PrivateLink资源。您可以通过传递 VPC 接口终端节点来指定您希望通过哪个 Amazon VPC 访问您的 AppRunner 服务。要创建 VPC 接口终端节点，请指定以下内容：

- 用于启用连接的 Amazon VPC。
- 添加安全组。默认情况下，会向 VPC 接口终端节点分配安全组。您可以选择关联自定义安全组，以进一步控制传入的网络流量。
- 添加子网。为确保更高的可用性，建议您为每个可用区选择至少两个子网来访问 App Runner 服务。将在您为 VPC 接口终端节点启用的每个子网中创建一个网络接口终端节点。这些是请求者管理的网络接口，用作发往 App Runner 的流量的入口点。请求者托管式网络接口是 AWS 服务代表您在 VPC 中创建的网络接口。
- 如果您使用的是 API，请添加 App Runner VPC 接口终端节点`Servicename`。例如，

```
com.amazonaws.region.apprunner.requests
```

您可以使用以下 AWS 服务之一创建 VPC 接口终端节点：

- App Runner 控制台 有关更多信息，请参阅管理私有终端节点。
- 亚马逊 VPC 控制台或 API，以及 AWS Command Line Interface (AWS CLI)。有关更多信息，请参阅AWS PrivateLink 指南 AWS PrivateLink中的AWS 服务 直通访问。

> **Note**
>
> 您需要根据AWS PrivateLink 定价为您使用的每个 VPC 接口终端节点付费。因此，为了提高成本效益，您可以使用相同的 VPC 接口终端节点访问一个 VPC 内的多个 App Runner 服务。但是，为了实现更好的隔离，可以考虑为每个 App Runner 服务关联不同的 VPC 接口终端节点。

VPC 入口连接

VPC 入口连接是一种 App Runner 资源，用于为传入流量指定应用程序运行器端点。当您在 App Runner 控制台上为传入流量选择私有终端节点时，App Runner 会在幕后分配 VPC 入口连接资源。选择此选项仅允许来自 Amazon VPC 的流量访问您的 App Runner 服务。VPC 入口连接资源将您的 App Runner 服务连接到 Amazon VPC 的 VPC 接口终端节点。只有在使用 API 操作为传入流量配置网络设置时，才能创建 VPC 入口连接资源。有关如何创建 VPC 入口连接资源的更多信息，请参阅 AWS App Runner API 参考CreateVpcIngressConnection中的。

> **Note**
>
> App Runner 的一个 VPC 入口连接资源可以连接到 Amazon VPC 的一个 VPC 接口终端节点。此外，您只能为每个 App Runner 服务创建一个 VPC 入口连接资源。

私有端点

私有终端节点是一个 App Runner 控制台选项，如果您只想接收来自 Amazon VPC 的传入流量，则可以选择该选项。在 App Runner 控制台上选择 “私有终端节点” 选项，您可以通过配置 VPC 接口终端节点将服务连接到 VPC。在幕后，App Runner 会为您配置的 VPC 接口终端节点分配 VPC 入口连接资源。

> **Note**
>
> 私有终端节点仅支持 IPv4 网络流量。

Summary

仅允许来自 Amazon VPC 的流量访问您的 App Runner 服务，从而使您的服务私有化。为此，您需要使用 App Runner 或 Amazon VPC 为选定的亚马逊 VPC 创建 VPC 接口终端节点。在 App Runner 控

制台上，当您为传入流量启用私有终端节点时，可以创建 VPC 接口终端节点。然后，App Runner 会自动创建 VPC 入口连接资源并连接到 VPC 接口终端节点和您的 App Runner 服务。这将创建一个私有服务连接，确保只有来自所选 VPC 的流量才能访问您的 App Runner 服务。

管理私有端点

使用以下方法之一管理传入流量的私有端点：

- the section called “应用程序运行器控制台”
- the section called “应用程序运行器 API 或 AWS CLI”

> **Note**
>
> 如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WA F Web ACL。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。因此，与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。
> 要了解有关基础设施安全和安全组的更多信息，包括最佳实践，请参阅 Amazon VPC 用户指南中的以下主题：使用安全组控制网络流量和控制流向 AWS 资源的流量。

应用程序运行器控制台

使用 App Runner 控制台创建服务时，或者稍后更新其配置时，可以选择配置传入流量。

要配置您的传入流量，请选择以下选项之一。

- 公共终端节点：使所有服务都能通过互联网访问您的服务。默认情况下，选择公共端点。
- 私有终端节点：使您的 App Runner 服务只能在 Amazon VPC 内访问。

> **Note**
>
> 目前，App Runner 仅支持公共端点的 IPv6。托管在亚马逊虚拟私有云（亚马逊 VPC）中的 App Runner 服务不支持 IPv6 终端节点。如果您将使用双栈公共终端节点的服务更新为私有终端节点，则您的 App Runner 服务将默认仅支持来自 IPv4 端点的流量，并且无法接收来自 IPv6 端点的流量。

启用私有终端节点

通过将私有终端节点与您要访问的 Amazon VPC 的 VPC 接口终端节点关联来启用该终端节点。您可以创建新的 VPC 接口终端节点，也可以选择现有的 VPC 接口终端节点。

创建 VPC 接口终端节点

1. 打开 App Runner 控制台，然后在“区域”列表中，选择您的 AWS 区域。
2. 转到“配置服务”下的“网络”部分。
3. 对于传入的网络流量，选择私有端点。使用 VPC 接口终端节点连接到 VCP 的选项打开。
4. 选择创建新端点。创建新 VPC 接口终端节点对话框打开。
5. 输入 VPC 接口终端节点的名称。
6. 从可用下拉列表中选择所需的 VPC 接口终端节点。
7. 从下拉列表中选择安全组。添加安全组可为 VPC 接口终端节点提供额外的安全层。建议选择两个或多个安全组。如果您未选择安全组，App Runner 会向 VPC 接口终端节点分配默认安全组。确保安全组规则不会阻止想要与您的 App Runner 服务通信的资源。安全组规则必须允许与您的 App Runner 服务进行交互的资源。

   > **Note**
   >
   > 如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WA F Web ACL。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。因此，与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。
   >
   > 要了解有关基础设施安全和安全组的更多信息，包括最佳实践，请参阅 Amazon VPC 用户指南中的以下主题：使用安全组控制网络流量和控制流向 AWS 资源的流量。

8. 从下拉列表中选择所需的子网。建议您为每个可用区至少选择两个子网，以便从中访问 App Runner 服务。
9. （可选）选择添加新标签并输入标签键和标签值。
10. 选择创建。配置服务页面打开，顶部栏上显示成功创建 VPC 接口终端节点的消息。

选择现有 VPC 接口终端节点

1. 打开 App Runner 控制台，然后在“区域”列表中，选择您的 AWS 区域。
2. 转到“配置服务”下的“网络”部分。

3. 对于传入的网络流量，选择私有端点。使用 VPC 接口终端节点连接到 VPC 的选项打开。将显示可用的 VPC 接口终端节点列表。
4. 选择 VPC 接口终端节点下列出的所需 VPC 接口终端节点。
5. 选择 “下一步” 创建您的服务。App Runner 启用私有端点。

> **Note**
>
> 创建服务后，如果需要，您可以选择编辑与 VPC 接口终端节点关联的安全组和子网。

要查看私有终端节点的详细信息，请转到您的服务并展开 “配置” 选项卡下的 “网络” 部分。它显示了与私有终端节点关联的 VPC 和 VPC 接口终端节点的详细信息。

更新 VPC 接口终端节点

创建 App Runner 服务后，您可以编辑与私有终端节点关联的 VPC 接口终端节点。

> **Note**
>
> 您无法更新终端节点名称和 VPC 字段。

更新 VPC 接口终端节点

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 前往您的服务，然后在左侧面板上选择网络配置。
3. 选择传入流量以查看与相应服务关联的 VPC 接口终端节点。
4. 选择要编辑的 VPC 接口终端节点。
5. 选择编辑。用于编辑 VPC 接口终端节点的对话框打开。
6. 选择所需的安全组和子网，然后单击 “更新”。显示 VPC 接口终端节点详细信息的页面打开，顶部栏上显示成功更新 VPC 接口终端节点的消息。

> **Note**
>
> 如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WA F Web ACL。这是因为我们目前不支持将请求源 IP 数据转发

到与 WAF 关联的 App Runner 私有服务。因此，与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。
要了解有关基础设施安全和安全组的更多信息，包括最佳实践，请参阅 Amazon VPC 用户指南中的以下主题：使用安全组控制网络流量和控制流向 AWS 资源的流量。

删除 VPC 接口终端节点

如果您不希望私密访问您的 App Runner 服务，则可以将传入流量设置为公开。更改为公共会移除私有终端节点，但不会删除 VPC 接口终端节点

删除 VPC 接口终端节点

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 前往您的服务，然后在左侧面板上选择网络配置。
3. 选择传入流量以查看与相应服务关联的 VPC 接口终端节点。

> **Note**
> 在删除 VPC 接口终端节点之前，请通过更新您的服务将其从其连接的所有服务中删除。

4. 选择删除。

   如果有服务连接到 VPC 接口终端节点，则您会收到 “无法删除 VPC 接口终端节点” 消息。如果没有服务连接到 VPC 接口终端节点，您将收到一条确认删除的消息。
5. 选择删除。网络配置页面将打开传入流量，顶部栏上显示成功删除 VPC 接口终端节点的消息。

应用程序运行器 API 或 AWS CLI

您可以在 App Runner 上部署只能从 Amazon VPC 中访问的应用程序。

有关将您的服务设为私有所需的权限的信息，请参阅the section called “权限”。

> **Note**
> 目前，App Runner 仅支持公共端点的 IPv6。托管在亚马逊虚拟私有云（亚马逊 VPC）中的 App Runner 服务不支持 IPv6 终端节点。如果您将使用双栈公共终端节点的服务更新为私有

终端节点，则您的 App Runner 服务将默认仅支持来自 IPv4 端点的流量，并且无法接收来自 IPv6 端点的流量。

创建与 Amazon VPC 的私有服务连接

1. 创建 VPC 接口终端节点（一种 AWS PrivateLink 资源）以连接到 App Runner。为此，请指定要与应用程序关联的子网和安全组。以下是创建 VPC 接口终端节点的示例。

   > **Note**
   >
   > 如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WA F Web ACL。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。因此，与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。
   >
   > 要了解有关基础设施安全和安全组的更多信息，包括最佳实践，请参阅 Amazon VPC 用户指南中的以下主题：使用安全组控制网络流量和控制流向 AWS 资源的流量。

   Example

   ```
   aws ec2 create-vpc-endpoint
    --vpc-endpoint-type: Interface
    --service-name: com.amazonaws.us-east-1.apprunner.requests
    --subnets: subnet1, subnet2
    --security-groups: sg1
   ```

2. 通过 CLI 使用CreateService或 UpdateServiceApp Runner API 操作引用 VPC 接口终端节点。将您的服务配置为不可公开访问。`False`在`NetworkConfiguration`参数的`IngressConfiguration`成员中设置`IsPubliclyAccessible`为。以下是引用 VPC 接口终端节点的示例。

   Example

   ```
   aws apprunner create-service
    --network-configuration: ingress-configuration=<ingress_configuration>
    --service-name: com.amazonaws.us-east-1.apprunner.requests
    --source-configuration: <source_configuration>
    # Ingress Configuration
    {
   ```

```
 "IsPubliclyAccessible": False
 }
```

3. 调用 `create-vpc-ingress-connection` API 操作为 App Runner 创建 VPC 入口连接资源，并将其与您在上一步中创建的 VPC 接口终端节点相关联。它会返回一个用于访问指定 VPC 中的服务的域名。以下是创建 VPC 入口连接资源的示例。

   Example 请求

```
aws apprunner create-vpc-ingress-connection
 --service-arn: <apprunner_service_arn>
 --ingress-vpc-configuration: {"VpcId":<vpc_id>, "VpceId": <vpce_id>}
 --vpc-ingress-connection-name: <vic_connection_name>
```

   Example 响应

```
{
   "VpcIngressConnectionArn": <vpc_ingress_connection_arn>,
   "VpcIngressConnectionName": <vic_connection_name>,
   "ServiceArn": <apprunner_service_arn>,
   "Status": "PENDING_CREATION",
   "AccountId": <connection_owner_id>,
   "DomainName": <domain_name_associated_with_vpce>,
   "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
   "CreatedAt": <date_created>
}
```

更新 VPC 入口连接

您可以更新 VPC 入口连接资源。VPC 入口连接必须处于以下状态之一才能更新：

- AVAILABLE
- 创建失败
- 更新失败

以下是更新 VPC 入口连接资源的示例。

Example 请求

```
aws apprunner update-vpc-ingress-connection
```

```
      --vpc-ingress-connection-arn: <vpc_ingress_connection_arn>
```

Example 响应

```
{
    "VpcIngressConnectionArn":  <vpc_ingress_connection_arn>,
    "VpcIngressConnectionName":  <vic_connection_name>,
    "ServiceArn":  <apprunner_service_arn>,
    "Status": "FAILED_UPDATE",
    "AccountId":  <connection_owner_id>,
    "DomainName":  <domain_name_associated_with_vpce>,
    "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
    "CreatedAt":  <date_created>
}
```

删除 VPC 入口连接

如果您不再需要与 Amazon VPC 的私有连接，则可以删除 VPC 入口连接资源。

VPC 入口连接必须处于以下状态之一才能删除：

- AVAILABLE
- 创建失败
- 更新失败
- 删除失败

以下是删除 VPC 入口连接的示例

Example 请求

```
aws apprunner delete-vpc-ingress-connection
      --vpc-ingress-connection-arn: <vpc_ingress_connection_arn>
```

Example 响应

```
{
   "VpcIngressConnectionArn": <vpc_ingress_connection_arn>,
   "VpcIngressConnectionName": <vic_connection_name>,
   "ServiceArn": <apprunner_service_arn>,
```

```
    "Status": "PENDING_DELETION",
    "AccountId": <connection_owner_id>,
    "DomainName": <domain_name_associated_with_vpce>,
    "IngressVpcConfiguration": {"VpcId":<vpc_id>, "VpceId":<vpce_id>},
    "CreatedAt": <date_created>,
    "DeletedAt": <date_deleted>
}
```

使用以下 App Runner API 操作来管理服务的私有入站流量。

- CreateVpcIngressConnection— 创建新的 VPC 入口连接资源。当您想要将 App Runner 服务与 Amazon VPC 端点相关联时，App Runner 需要此资源。
- ListVpcIngressConnections— 返回与您的 AWS 账户关联的 AWS App Runner VPC 入口连接终端节点列表。
- DescribeVpcIngressConnection— 返回 AWS App Runner VPC 入口连接资源的完整描述。
- UpdateVpcIngressConnection— 更新 AWS App Runner VPC 入口连接资源。
- DeleteVpcIngressConnection— 删除与 App Runner 服务关联的 App Runner VPC 入口连接资源。

有关使用 App Runner API 的更多信息，请参阅 App Runner API 参考指南。

为公共传入流量启用 IPv6

如果您希望您的服务接收来自 IPv6 地址或 IPv4 和 IPv6 地址的传入网络流量，请为公共终端节点选择双栈地址类型。创建新应用程序时，可以在配置服务 > 网络部分下找到此设置。有关如何使用 App Runner 控制台或 App Runner API 启用 IPv6 的更多信息，请参阅the section called “管理公共端点的双堆栈”。

有关在上采用 IPv6 的更多信息 AWS，请参阅开启 IPv6 AWS。

App Runner 仅支持公共 App Runner 服务端点的双堆栈。对于所有 App Runner 私有服务，仅支持 IPv4。

> **Note**
>
> 当您将 IP 地址类型设置为双堆栈并将网络配置从公共端点更改为私有端点时，App Runner 会自动将您的地址类型更改为 IPv4。这是因为 App Runner 仅支持公共端点的 IPv6。

了解有关 IPv4 与 IPv6 的背景信息

IPv4 网络层通常用于通过互联网路由网络流量，它使用 32 位地址方案。此地址空间有限，使用大量网络设备可能会耗尽。因此，网络地址转换 (NAT) 通常用于通过单个公共网络地址路由多个 IPv4 地址。

IPv6 是互联网协议的最新版本，它以 IPv4 为基础，采用 128 位寻址方案扩展了地址空间。使用 IPv6，您可以构建连接设备数量几乎不受限制的网络。由于网络地址数量庞大，IPv6 不需要 NAT。

IPv4 和 IPv6 端点彼此不兼容，因为 IPv4 端点无法接收传入的 IPv6 流量，反之亦然。双堆栈提供了一种便捷的解决方案，可以同时支持 IPv4 和 IPv6 网络流量。

管理公共传入流量的双堆栈

使用以下方法之一管理公共传入流量的双栈地址类型：

- the section called “应用程序运行器控制台”
- the section called “应用程序运行器 API 或 AWS CLI”

应用程序运行器控制台

当你使用 App Runner 控制台创建服务时，或者稍后更新服务配置时，你可以为传入的互联网流量选择双栈地址类型。

要启用双栈地址，请键入

1. 创建或更新服务时，请展开 “配置服务” 下的 “网络” 部分。
2. 对于传入的网络流量，选择公共端点。公共端点 IP 地址类型选项打开。
3. 展开公共终端节点 IP 地址类型以查看以下 IP 地址类型。
   - IPv4
   - 双堆栈（IPv4 和 IPv6）

   > **Note**
   >
   > 如果您没有展开公共终端节点 IP 地址类型进行选择，则 App Runner 会将 IPv4 分配为默认配置。

4. 选择双堆栈（IPv4 和 IPv 6）。

5. 如果要创建服务，请选择 “下一步”，然后选择 “创建并部署”。否则，如果您要更新服务，请选择保存更改。

   部署服务后，您的应用程序开始接收来自 IPv4 和 IPv6 端点的网络流量。

> **Note**
>
> 目前，App Runner 仅支持公共端点的 IPv6。托管在亚马逊虚拟私有云（亚马逊 VPC）中的 App Runner 服务不支持 IPv6 终端节点。如果您将使用双栈公共终端节点的服务更新为私有终端节点，则您的 App Runner 服务将默认仅支持来自 IPv4 端点的流量，并且无法接收来自 IPv6 端点的流量。

更改地址类型

1. 按照步骤更新服务并导航到 “网络”。
2. 导航到传入网络流量下的公共端点 IP 地址类型，然后选择所需的地址类型。
3. 选择保存更改。您的服务已根据您的选择进行更新。

应用程序运行器 API 或 AWS CLI

当您调用CreateService或 UpdateServiceApp Runner API 操作时，请使用`NetworkConfiguration`参数的`IpAddressType`成员来指定地址类型。您可以指定的支持的值为`IPv4`和`DUAL_STACK`。指定`DUAL_STACK`是否希望您的服务接收来自 IPv4 和 IPv6 端点的互联网流量。如果您未为指定任何值`IpAddressType`，则默认情况下将应用 IPv4。

以下是创建以双堆栈作为 IP 地址的服务的示例。此示例调用一个`input.json`文件。

Example 请求创建支持双堆栈的服务

```
aws apprunner create-service \
 --cli-input-json file://input.json
```

Example **`input.json`** 的内容

```
{
  "ServiceName": "example-service",
  "SourceConfiguration": {
```

```
    "ImageRepository": {
      "ImageIdentifier": "public.ecr.aws/aws-containers/hello-app-runner:latest",
      "ImageConfiguration": {
        "Port": "8000"
      },
      "ImageRepositoryType": "ECR_PUBLIC"
    },
    "NetworkConfiguration": {
      "IpAddressType": "DUAL_STACK"
    }
  }
}
```

Example 响应

```
{
  "Service": {
    "ServiceName": "example-service",
    "ServiceId": "<service-id>",
    "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/example-
service/<service-id>",
    "ServiceUrl": "1234567890.us-east-2.awsapprunner.com",
    "CreatedAt": "2023-10-16T12:30:51.724000-04:00",
    "UpdatedAt": "2023-10-16T12:30:51.724000-04:00",
    "Status": "OPERATION_IN_PROGRESS",
    "SourceConfiguration": {
      "ImageRepository": {
        "ImageIdentifier": "public.ecr.aws/aws-containers/hello-app-runner:latest",
        "ImageConfiguration": {
          "Port": "8000"
        },
        "ImageRepositoryType": "ECR_PUBLIC"
      },
      "AutoDeploymentsEnabled": false
    },
    "InstanceConfiguration": {
      "Cpu": "1024",
      "Memory": "2048"
    },
    "HealthCheckConfiguration": {
      "Protocol": "TCP",
      "Path": "/",
      "Interval": 5,
```

```
      "Timeout": 2,
      "HealthyThreshold": 1,
      "UnhealthyThreshold": 5
    },
    "AutoScalingConfigurationSummary": {
      "AutoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
      "AutoScalingConfigurationName": "DefaultConfiguration",
      "AutoScalingConfigurationRevision": 1
    },
    "NetworkConfiguration": {
      "IpAddressType": "DUAL_STACK",
      "EgressConfiguration": {
        "EgressType": "DEFAULT"
      },
      "IngressConfiguration": {
        "IsPubliclyAccessible": true
      }
    }
  },
  "OperationId": "24bd100b1e111ae1a1f0e1115c4f11de"
}
```


> **Note**
>
> 目前，App Runner 仅支持公共端点的 IPv6。托管在亚马逊虚拟私有云（亚马逊 VPC）中的 App Runner 服务不支持 IPv6 终端节点。如果您将使用双栈公共终端节点的服务更新为私有终端节点，则您的 App Runner 服务将默认仅支持来自 IPv4 端点的流量，并且无法接收来自 IPv6 端点的流量。

有关 API 参数的更多信息，请参阅NetworkConfiguration。

为出站流量启用 VPC 访问权限

默认情况下，您的 AWS App Runner 应用程序可以向公共终端节点发送消息。这包括您自己的解决方案 AWS 服务以及任何其他公共网站或网络服务。您的应用程序甚至可以从亚马逊虚拟私有云 (Amazon VPC) 向在 VPC 中运行的应用程序的公共终端节点发送消息。如果您在启动环境时未配置 VPC，App Runner 将使用默认 VPC，即公有的 VPC。

您可以选择在自定义 VPC 中启动您的环境，以自定义传出流量的网络和安全设置。您可以允许您的 AWS App Runner 服务从亚马逊虚拟私有云 (Amazon VPC) 访问在私有 VPC 中运行的应用程序。完成此操作后，您的应用程序可以连接托管在 Amazon Virt ual Private Cloud (Amazon VPC) 中的其他应用程序并向其发送消息。例如，Amazon RDS 数据库 ElastiCache、Amazon 以及托管在私有 VPC 中的其他私有服务。

VPC 连接器

您可以通过从 App Runner 控制台创建名为 VPC 连接器的 VPC 终端节点，将您的服务与 VPC 关联起来。要创建 VPC 连接器，请指定 VPC、一个或多个子网以及一个或多个安全组（可选）。配置 VPC 连接器后，您可以将其与一个或多个 App Runner 服务一起使用。

一次性延迟

如果您将 App Runner 服务配置为用于出站流量的自定义 VPC 连接器，则该服务可能会遇到两到五分钟的一次性启动延迟。启动过程将等待 VPC 连接器准备好连接到其他资源，然后才会将服务状态设置为 “正在运行”。您可以在首次创建服务时使用自定义 VPC 连接器对其进行配置，也可以在之后通过更新服务来进行配置。

请注意，如果您为其他服务重复使用相同的 VPC 连接器配置，则不会出现任何延迟。VPC 连接器配置基于安全组和子网组合。对于给定的 VPC 连接器配置，在初始创建 VPC 连接器 Hyperplane ENI（弹性网络接口）期间，延迟仅发生一次。

有关自定义 VPC 连接器和 AWS Hyperplane 的更多信息

App Runner 中的 VPC 连接器基于 AWS Hyperplane，Hyperplane 是亚马逊内部网络系统，位于网络负载均衡器、NAT Gateway 和 A WS PrivateLink 等多种 AWS 资源背后。 AWS Hyperplane 技术提供高吞吐量和低延迟功能，以及更高的共享程度。当您创建 VPC 连接器并将其与您的服务关联时，将在您的子网中创建 Hyperplane ENI。VPC 连接器配置基于安全组和子网组合，您可以跨多个 App Runner 服务引用相同的 VPC 连接器。因此，底层的 Hyperplane ENI 将在你的 App Runner 服务之间共享。即使您扩大了处理请求负载所需的任务数量，这种共享也是可行的，并且可以更有效地利用您的 VPC 中的 IP 空间。有关更多信息，请参阅 AWS 容器博客中的 AWS App Runner VPC 网络深入探讨。

子网

每个子网都位于特定的可用区中。为了获得高可用性，我们建议您在至少三个可用区中选择子网。如果该区域的可用区少于三个，我们建议您在所有支持的可用区中选择您的子网。

为您的 VPC 选择子网时，请确保选择私有子网，而不是公有子网。这是因为，当您创建 VPC 连接器时，App Runner 服务会在每个子网中创建一个 Hyperplane ENI。每个 Hyperplane ENI 仅分配一个私有 IP 地址，并标有AWSAppRunnerManaged密钥标签。如果您选择公有子网，则在运行 App Runner 服务时会出现错误。但是，如果您的服务需要访问互联网或其他公共服务上的某些服务 AWS 服务，请参阅the section called “选择子网时的注意事项 ”。

选择子网时的注意事项

- 当您将服务连接到 VPC 时，出站流量无法访问公共互联网。来自您的应用程序的所有出站流量都将通过您的服务所连接的 VPC 定向。VPC 的所有联网规则都适用于您的应用程序的出站流量。这意味着您的服务无法访问公共互联网和 AWS API。要获得访问权限，请执行以下操作之一：
  - 通过 NAT 网关将子网连接到互联网。
  - 为您要访问的 AWS 服务 设置 VPC 终端节点。通过使用，您的服务将保留在 Amazon VPC 内 AWS PrivateLink。
- 有些可用区中的某些可用区 AWS 区域 不支持可用于 App Runner 服务的子网。如果您在这些可用区中选择子网，则无法创建或更新您的服务。对于这些情况，App Runner 会提供一条详细的错误消息，指向不支持的子网和可用区。出现这种情况时，请通过从请求中删除不支持的子网来排除故障，然后重试。

安全组

您可以选择指定 App Runner 用于在指定子网 AWS 下访问的安全组。如果您未指定安全组，App Runner 将使用 VPC 的默认安全组。默认安全组允许所有出站流量。

添加安全组可为 VCP 连接器提供额外的安全层，使您可以更好地控制网络流量。VPC 连接器仅用于来自您的应用程序的出站通信。您可以使用出站规则来允许与所需的目标终端节点进行通信。您还必须确保与目标资源关联的所有安全组都具有相应的入站规则。否则，这些资源将无法接受来自 VPC 连接器安全组的流量。

> **Note**
>
> 当您将服务与 VPC 关联时，以下流量不会受到影响：
>
> - 入站流量-您的应用程序收到的传入消息不受关联 VPC 的影响。消息通过与您的服务关联的公共域名进行路由，并且不与 VPC 交互。
> - App Runner 流量 — App Runner 代表你管理多项操作，例如提取源代码和图像、推送日志和检索机密。这些操作生成的流量不会通过您的 VPC 路由。

要详细了解如何 AWS App Runner 与 Amazon VPC 集成，请参阅 A AWS pp Runner VPC 网络。

> **Note**
>
> 对于传出流量，App Runner 目前仅支持 IPv4。

管理 VPC 访问权限

> **Note**
>
> 如果您为服务创建出站流量 VPC 连接器，则随后的服务启动过程将出现一次性延迟。您可以在创建新服务时或之后通过服务更新为新服务设置此配置。有关更多信息，请参阅一次性延迟本指南的 “与 App Runner 联网” 一章。

使用以下方法之一管理您的 App Runner 服务的 VPC 访问权限：

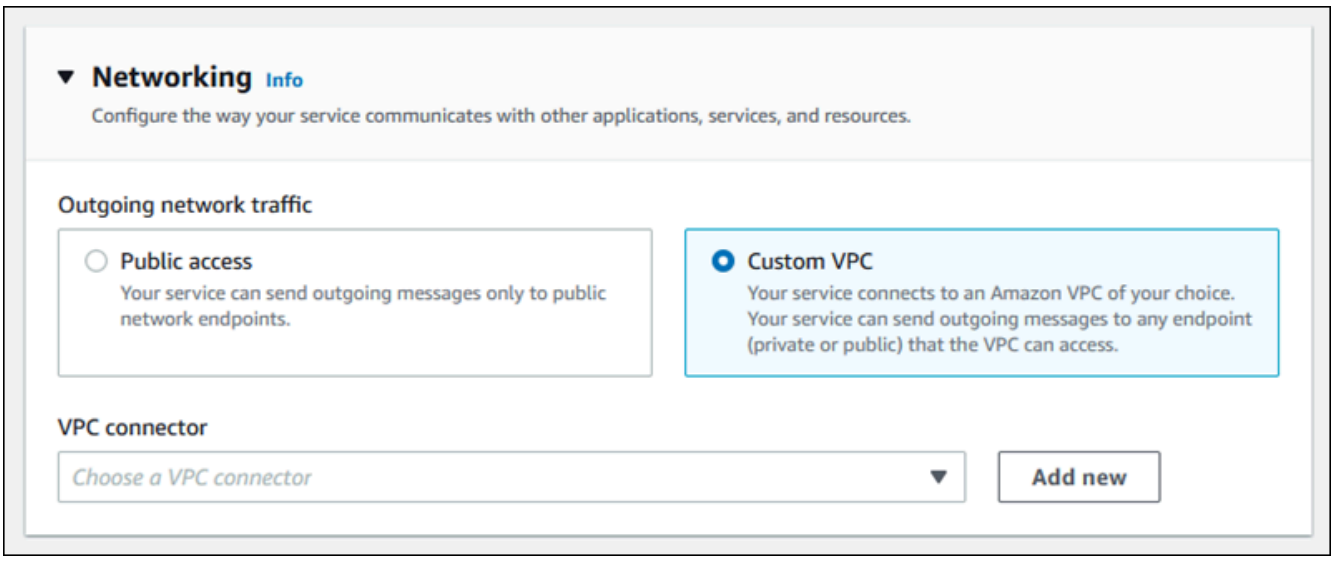
App Runner console

当您使用 App Runner 控制台创建服务时，或者稍后更新其配置时，您可以选择配置您的传出流量。在控制台页面上查找 “网络配置” 部分。对于传出网络流量，请选择以下选项：

- 公共访问：将您的服务与其他人的公共终端节点相关联 AWS 服务。
- 自定义 VPC：将您的服务与亚马逊 VPC 中的 VPC 关联。您的应用程序可以连接托管在 Amazon VPC 中的其他应用程序并向其发送消息。

启用自定义 VPC

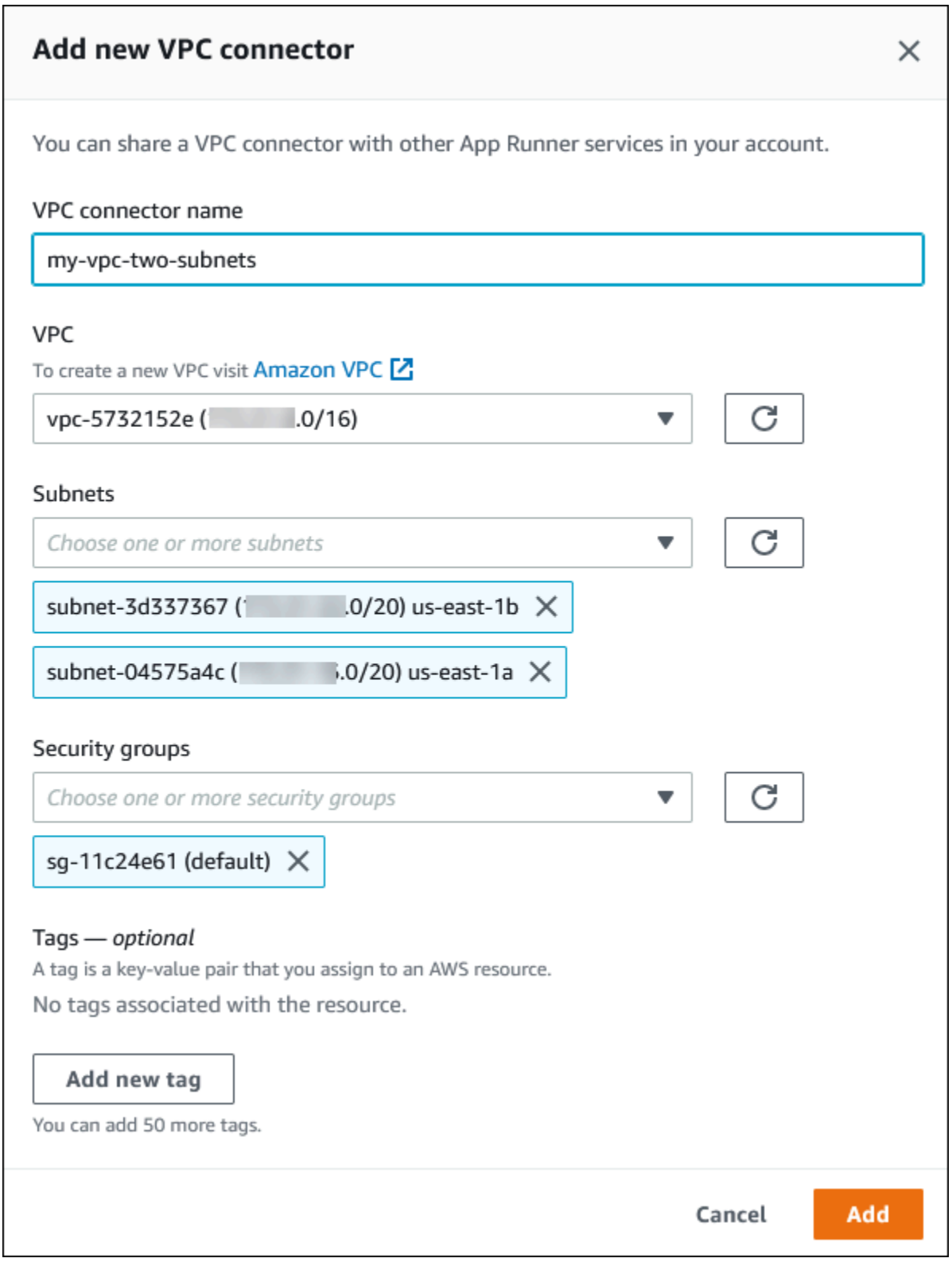
1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 转到 “配置服务” 下的 “网络” 部分。



3. 为传出网络流量选择自定义 VPC。
4. 在导航窗格中，选择 VPC 连接器。

   如果您创建了 VPC 连接器，则控制台会显示您账户中的 VPC 连接器列表。您可以选择现有 VPC 连接器，然后选择 “下一步” 来查看您的配置。然后，移至最后一步。或者，您可以使用以下步骤添加新的 VPC 连接器。
5. 选择新增，为您的服务创建新的 VPC 连接器。

   然后，将打开 “添加新 VPC 连接器” 对话框。



6. 输入 VPC 连接器的名称，然后从可用列表中选择所需的 VPC。
7. 对于子网，为计划从中访问 App Runner 服务的每个可用区选择一个子网。为了获得更好的可用性，请选择三个子网。或者，如果子网少于三个，请选择所有可用的子网。

> **Note**
>
> 确保为 VPC 连接器分配私有子网。如果您将公有子网分配给 VPC 连接器，则您的服务将无法创建或在更新期间自动回滚。

8. （可选）对于安全组，选择要与端点网络接口关联的安全组。
9. （可选）若要添加标签，请选择 Add new tag（添加新标签），然后输入该标签的键和值。
10. 选择添加。

    您创建的 VPC 连接器的详细信息显示在 VPC 连接器下。
11. 选择 “下一步” 查看您的配置，然后选择 “创建并部署”。

    App Runner 会为您创建 VPC 连接器资源，然后将其与您的服务关联。如果成功创建了服务，控制台将显示服务控制面板，其中包含新服务的服务概述。

App Runner API or AWS CLI

当您调用CreateService或 UpdateServiceApp Runner API 操作时，请使用`NetworkConfiguration`参数的`EgressConfiguration`成员为您的服务指定 VPC 连接器资源。

使用以下 App Runner API 操作来管理您的 VPC 连接器资源。

- CreateVpcConnector— 创建新的 VPC 连接器。
- ListVpcConnectors— 返回与您的关联的 VPC 连接器的列表 AWS 账户。该列表包括完整描述。
- DescribeVpcConnector— 返回 VPC 连接器的完整描述。
- DeleteVpcConnector— 删除 VPC 连接器。如果您已达到自己的 VPC 连接器配额 AWS 账户，则可能需要删除不必要的 VPC 连接器。

要在 App Runner 上部署可出站访问 VPC 的应用程序，必须先创建 VPC 连接器。为此，您可以指定一个或多个子网和安全组与应用程序关联。然后，您可以在 “创建” 中或UpdateService通过 CLI 引用 VPC 连接器，如以下示例所示：

```
            cat > vpc-connector.json <<EOF
{
"VpcConnectorName": "my-vpc-connector",
```

```
"Subnets": [
"subnet-a",
"subnet-b",
"subnet-c"
],
"SecurityGroups": [
"sg-1",
"sg-2"
]
}
EOF

aws apprunner create-vpc-connector \
--cli-input-json file:///vpc-connector.json

cat > service.json <<EOF

{
"ServiceName": "my-vpc-connected-service",
"SourceConfiguration": {
"ImageRepository": {
"ImageIdentifier": "<ecr-image-identifier> ",
"ImageConfiguration": {
"Port": "8000"
},
"ImageRepositoryType": "ECR"
}
},
"NetworkConfiguration": {
"EgressConfiguration": {
"EgressType": "VPC",
"VpcConnectorArn": "arn:aws:apprunner:..../my-vpc-connector"
}
}
}
EOF

aws apprunner create-service \
--cli-input-json file:///service.js
```

App Runner 服务的可观察性

AWS App Runner 与多项 AWS 服务集成，为您的 App Runner 服务提供广泛的可观察性工具套件。本章中的主题描述了这些功能。

主题

跟踪 App Runner 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。以下各节介绍如何在 App Runner 控制台中跟踪活动以及如何使用 API。

跟踪 App Runner 服务活动

使用以下方法之一跟踪您的 App Runner 服务活动：

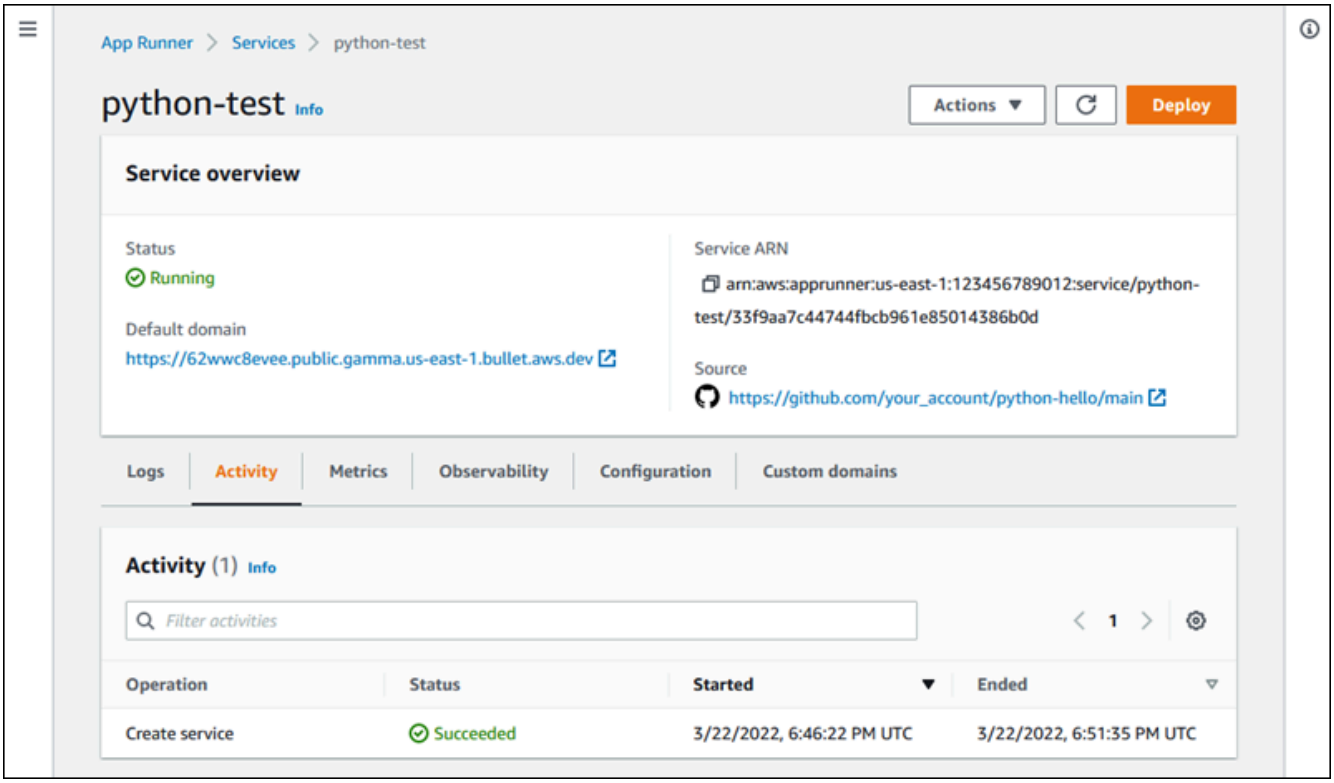
App Runner console

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 App Runner 控制台，然后在“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 App Runner 服务。

   控制台显示带有服务概述的服务仪表板。



3. 在服务仪表板页面上，选择 “活动” 选项卡（如果尚未选择）。

   控制台显示操作列表。

4. 要查找特定的操作，请通过输入搜索词来缩小列表的范围。您可以搜索表格中显示的任何值。
5. 选择任何列出的操作以查看或下载相关日志。

App Runner API or AWS CLI

给定 App Runner 服务的亚马逊资源名称 (ARN)，该ListOperations操作会返回在该服务上发生的操作的列表。每个列表项都包含一个操作 ID 和一些跟踪详情。

查看流向日志的 App Runner CloudWatch 日志

您可以使用 Amazon Lo CloudWatch gs 来监控、存储和访问您在各种 AWS 服务中的资源生成的日志文件。有关更多信息，请参阅 Amazon CloudWatch 日志用户指南。

AWS App Runner 收集您的应用程序部署和活动服务的输出，并将其流式传输到 CloudWatch 日志。以下各节列出了 App Runner 日志流，并向您展示了如何在 App Runner 控制台中查看它们。

App Runner 日志组和直播

CloudWatch 日志将日志数据保存在日志流中，并进一步组织到日志组中。日志流是来自特定来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。

App Runner 为你的每个 App Runner 服务定义了两个 CloudWatch 日志组，每个日志组都有多个日志流 AWS 账户。

服务日志

服务日志组包含 App Runner 在管理您的 App Runner 服务并对其进行操作时生成的日志输出。

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将其暂停。

此外，App Runner 还会为每个与您的服务相关的长时间运行的异步操作创建日志流。日志流名称反映了操作类型和特定的操作 ID。

部署是一种操作。部署日志包含 App Runner 在您创建服务或部署应用程序的新版本时执行的构建和部署步骤的日志输出。部署日志流名称以`deployment/`执行部署的操作的 ID 开头和结尾。此操作要么是CreateService调用初始应用程序部署，要么是StartDeployment调用每一次进一步部署。

在部署日志中，每条日志消息都以前缀开头：

- `[AppRunner]`— App Runner 在部署期间生成的输出。
- `[Build]`— 您自己的生成脚本的输出。

日志流名称	示例
`events`	不适用（固定名称）

日志流名称	示例
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 会为运行您的应用程序的每个实例（缩放单位）创建一个日志流。

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

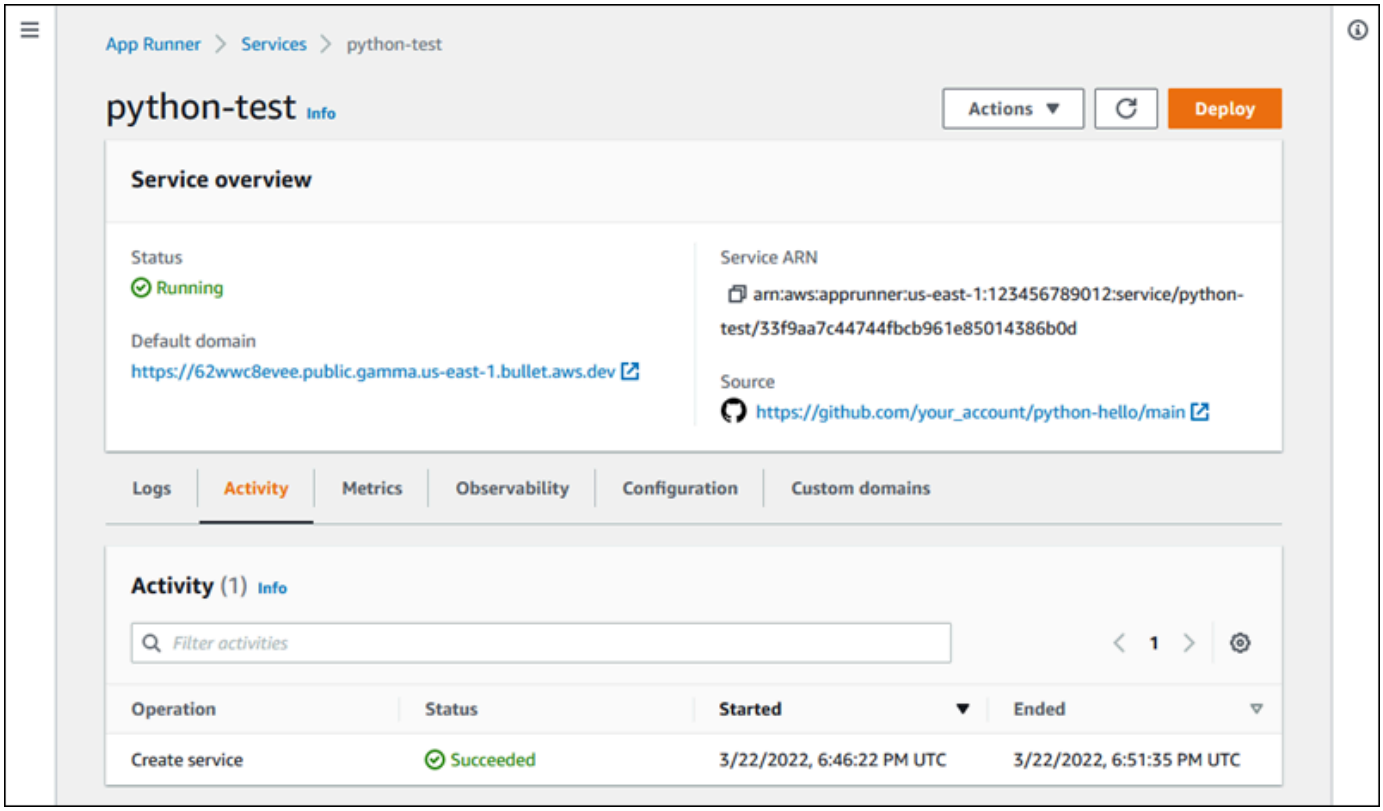
在控制台中查看 App Runner 日志

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

查看服务日志

1. 打开 App Runner 控制台，然后在 “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 App Runner 服务。

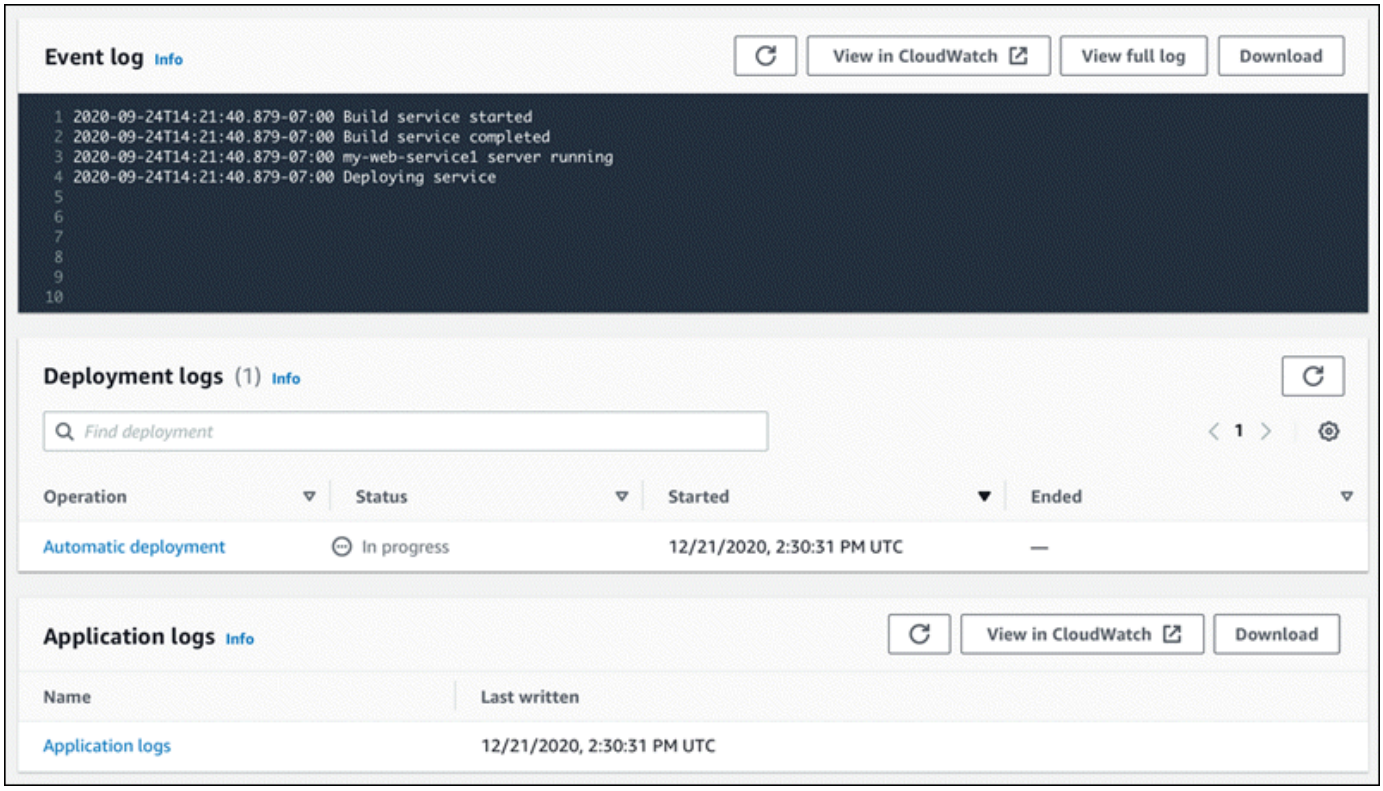
   控制台显示带有服务概述的服务仪表板。



3. 在服务仪表板页面上，选择日志选项卡。

   控制台分几个部分显示几种类型的日志：

   - 事件日志-App Runner 服务生命周期中的活动。控制台显示最新事件。
   - 部署日志-源存储库部署到您的 App Runner 服务。控制台显示每个部署的单独日志流。
   - 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合并到一个日志流中。



4. 要查找特定的部署，请输入搜索词，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. 选择查看 CloudWatch以打开 CloudWatch 控制台，并使用其全部功能浏览您的 App Runner 服务日志。对于部署日志流，请先选择一个日志流。

> **Note**
>
> 如果您想查看特定实例的应用程序日志，而不是合并的应用程序日志，则 CloudWatch 控制台特别有用。

查看报告给的 App Runner 服务指标 CloudWatch

亚马逊会实时 CloudWatch 监控您的亚马逊 Web Services (AWS) 资源和您运行 AWS 的应用程序。您可以使用 CloudWatch 来收集和跟踪指标，这些指标是您可以衡量资源和应用程序的变量。您还可以使用它来创建监视指标的警报。当达到某个阈值时， CloudWatch 会发送通知或自动对受监控的资源进行更改。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。有些指标跟踪运行您的 Web 服务的单个实例，而另一些指标则跟踪整体服务级别。以下各节列出了 App Runner 指标，并向您展示了如何在 App Runner 控制台中查看这些指标。

应用程序运行器指标

App Runner 收集与您的服务相关的以下指标，并将其发布到`AWS/AppRunner`命名空间 CloudWatch 中。

> **Note**
>
> 在 2023 年 8 月 23 日之前，CPU 利用率和内存利用率指标基于 vCPU 单位和内存使用量，而不是今天计算的利用率百分比。如果您的应用程序在此日期之前在 App Runner 上运行，并且您选择返回在 App Runner 或 CloudWatch 主机上查看该日期的指标，则您将看到两个单元中的指标显示，并且还会因此看到一些不规则之处。

> **Important**
>
> 在 2023 年 8 月 23 日之前，您需要更新所有基于 CPU 利用率和内存利用率指标值的 CloudWatch 警报。更新警报，使其根据利用率百分比而非 vCPU 或兆字节触发。有关更多信息，请参阅 Amazon CloudWatch 用户指南。

每个@@ 实例（缩放单位）的实例级别指标是单独收集的。

测量了什么？	指标	描述
CPU utilization	`CPUUtilization`	一分钟内平均的 CPU 使用率占服务配置预留的总 CPU 使用率的百分比。
Memory utilization	`MemoryUtilization`	一分钟内平均内存使用量占服务配置预留的总内存的百分比。

收集整个@@ 服务的服务级别指标。

测量了什么？	指标	描述
CPU utilization	`CPUUtilization`	一分钟内所有实例的聚合 CPU 使用率占服务配置预留的总 CPU 使用率的百分比。
Memory utilization	`MemoryUtilization`	一分钟内所有实例的聚合内存使用量占服务配置预留的总内存的百分比。
Concurrency	`Concurrency`	服务正在处理的并发请求的大致数量。
HTTP request count	`Requests`	服务收到的 HTTP 请求数。

HTTP status counts	`2xxStatusResponses`
	`4xxStatusResponses`
	`5xxStatusResponses`
		返回每种响应状态的 HTTP 请求数，按类别（2XX、4XX、5XX）分组。

HTTP request latency	`RequestLatency`	您的 Web 服务处理 HTTP 请求所用的时间（以毫秒为单位）。
Instance counts	`ActiveInstances`	正在处理针对您的服务的 HTTP 请求的实例数量。 **Note** 如果`ActiveInstances` 指标显示为零，则表示没有对该服务的请求。它并不表示您的服务的实例数为零。

在控制台中查看 App Runner 指标

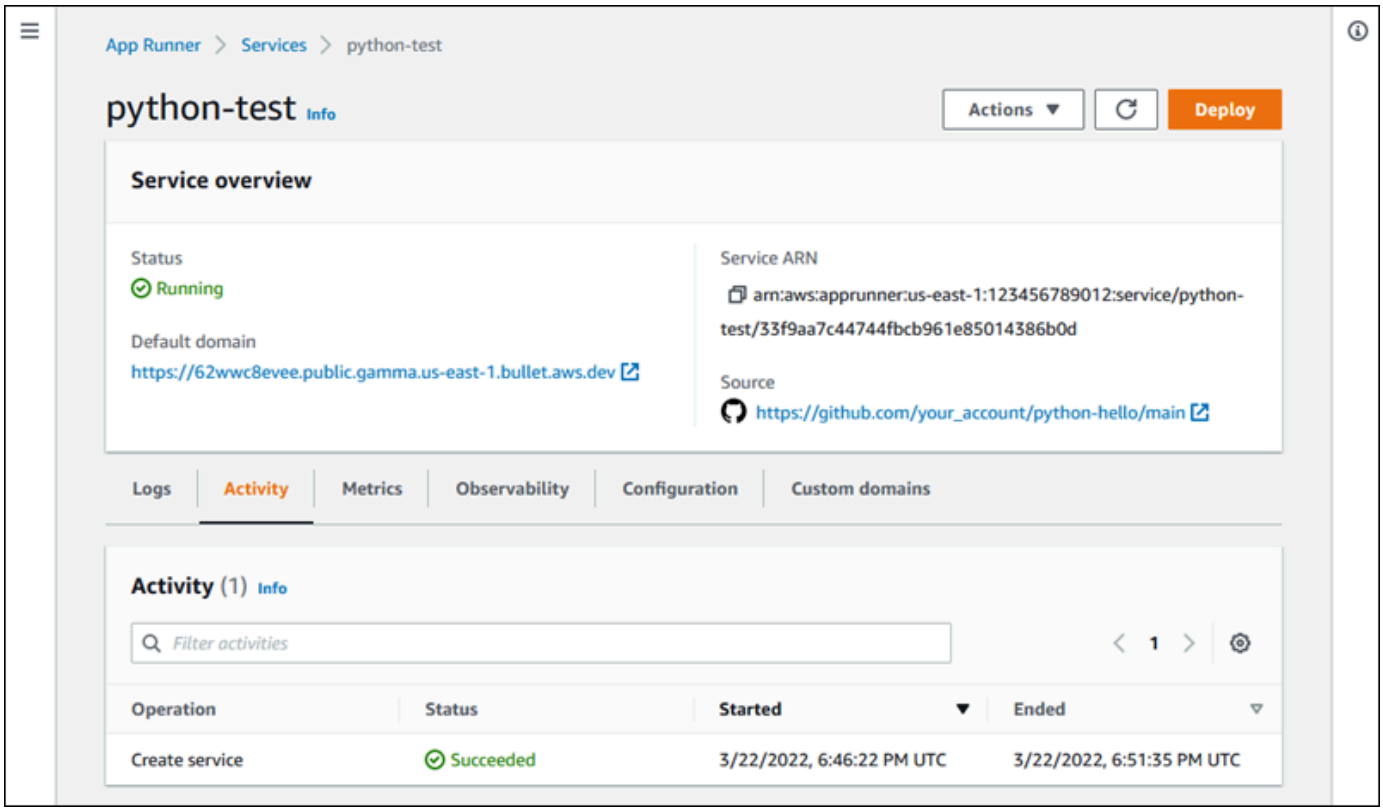
App Runner 控制台以图形方式显示 App Runner 为您的服务收集的指标，并提供更多探索这些指标的方法。

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

查看您的服务日志

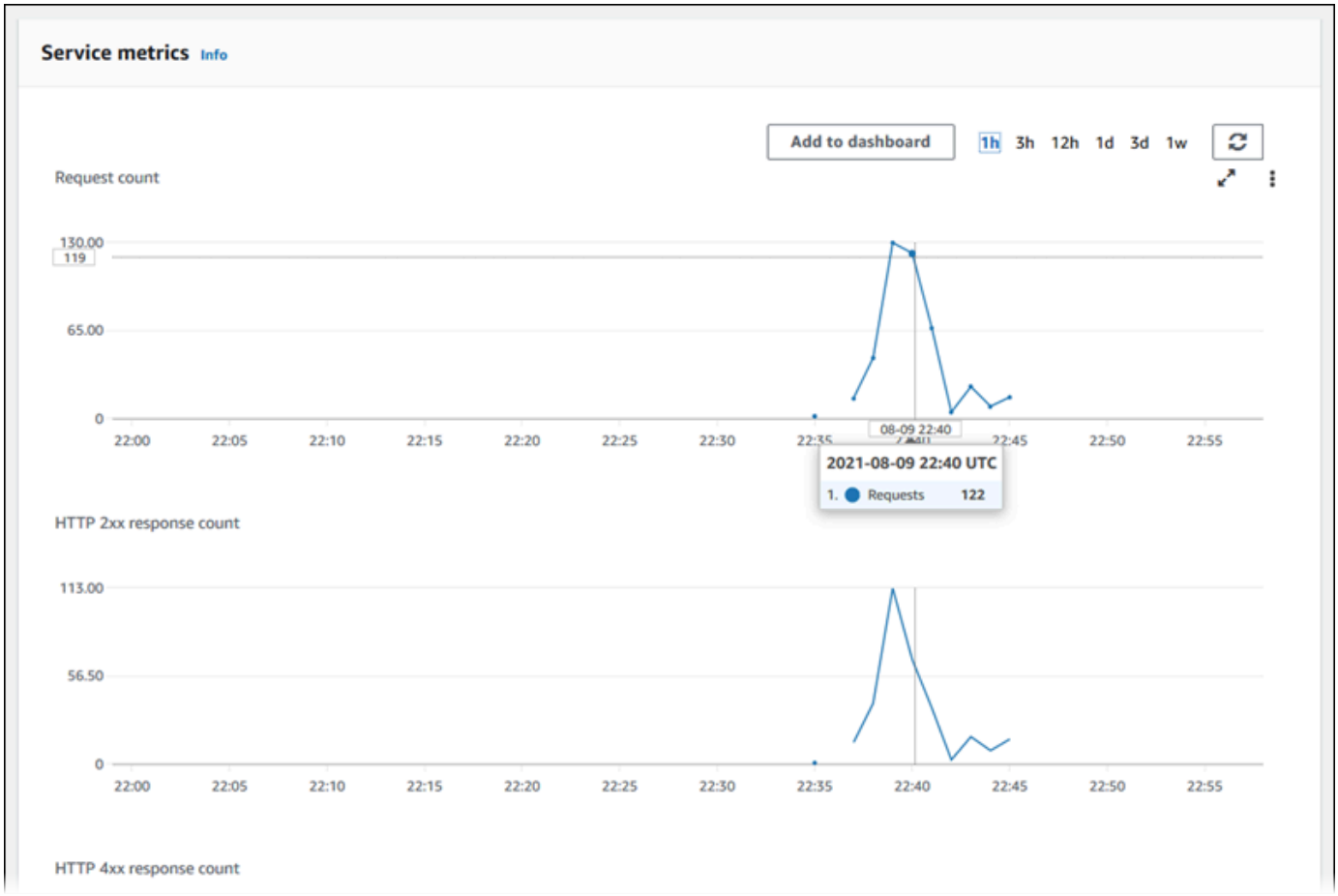
1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 在导航窗格中，选择服务，然后选择您的 App Runner 服务。

   控制台显示带有服务概述的服务仪表板。



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），将指标图表的范围缩小到该持续时间的最近时段。
5. 在其中一个图表部分的顶部选择“添加到仪表板”，或使用任何图表上的菜单，将相关指标添加到CloudWatch控制台的仪表板中以供进一步调查。

在中处理应用程序运行器事件 EventBridge

使用 Amazon EventBridge，您可以设置事件驱动的规则，以监控来自您的 AWS App Runner 服务的实时数据流的某些模式。匹配规则的模式后， EventBridge 会在目标（例如 Amazon ECS 和 Amazon SNS）中启动操作。 AWS Lambda AWS Batch例如，您可以通过在服务部署失败时向 Amazon SNS 主题发送信号，来设置发送电子邮件通知的规则。或者，您可以将 Lambda 函数设置为在服务更新失败时通知 Slack 频道。有关更多信息 EventBridge，请参阅 Amazon EventBridge 用户指南。

App Runner 将以下事件类型发送到 EventBridge

- 服务状态更改-App Runner 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-App Runner 服务上长时间的异步操作状态的变化。例如，服务已开始创建，服务更新成功完成，或者服务部署完成但出现错误。

创建 EventBridge 规则以对 App Runner 事件采取行动

EventBridge 事件是一个对象，它定义了一些标准 EventBridge 字段，例如源 AWS 服务和详细信息（事件）类型，以及一组包含事件详细信息的特定于事件的字段。要创建 EventBridge 规则，您可以使用 EventBridge 控制台定义事件模式（应跟踪哪些赛事）并指定目标动作（在比赛中应该做什么）。事件模式与其匹配的事件类似。您可以指定要匹配的字段子集，然后为每个字段指定可能值的列表。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 Amazon EventBridge 用户指南中的为 AWS 服务创建规则。

> **Note**
>
> 某些服务支持中的预定义模式。 EventBridge这简化了事件模式的创建方式。您可以在表单上选择字段值，然后为您 EventBridge 生成模式。目前，App Runner 不支持预定义的模式。您必须将模式作为 JSON 对象输入。您可以使用本主题中的示例作为起点。

应用程序运行器事件示例

以下是 App Runner 发送到的事件的一些示例 EventBridge。

- 服务状态更改事件。具体而言，是从状态更改`OPERATION_IN_PROGRESS`为`RUNNING`状态的服务。

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "AppRunner 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": {
    "previousServiceStatus": "OPERATION_IN_PROGRESS",
    "currentServiceStatus": "RUNNING",
    "serviceName": "my-app",
```

```
    "serviceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "message": "Service status is set to RUNNING.",
    "severity": "INFO"
  }
}
```

- 操作状态更改事件。具体而言，是成功完成的UpdateService操作。

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "AppRunner 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": {
    "operationStatus": "UpdateServiceCompletedSuccessfully",
    "serviceName": "my-app",
    "serviceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "message": "Service update completed successfully. New application and
 configuration is deployed.",
    "severity": "INFO"
  }
}
```

App Runner 事件模式示例

以下示例演示了可以在 EventBridge 规则中使用的事件模式来匹配一个或多个 App Runner 事件。事件模式类似于事件。仅包含要匹配的字段，并为每个字段提供列表而不是标量。

- 匹配特定账户服务的所有服务状态更改事件，该账户的服务不再处于RUNNING状态。

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

```
    "previousServiceStatus": [ "RUNNING" ]
  }
}
```

- 匹配操作失败的特定账户服务的所有操作状态更改事件。

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

应用程序运行器事件参考

服务状态变更

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

```
"serviceId": "your service ID",
"serviceName": "your service name",
"message": "Service status is set to CurrentStatus.",
"previousServiceStatus": "any valid service status",
"currentServiceStatus": "any valid service status",
"severity": "varies"
```

操作状态变更

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

```
"operationStatus": "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代码和相关消息。

Status	消息
CreateServiceStarted	服务创建已开始。
CreateServiceCompletedSuccessfully	服务创建成功完成。
CreateServiceFailed	服务创建失败。有关详细信息，请参阅服务日志。
DeleteServiceStarted	服务删除已开始。
DeleteServiceCompletedSuccessfully	服务删除已成功完成。
DeleteServiceFailed	服务删除失败。
UpdateServiceStarted	
UpdateServiceCompletedSuccessfully	服务更新成功完成。新的应用程序和配置已部署。
	服务更新成功完成。新配置已部署。
UpdateServiceFailed	服务更新失败。有关详细信息，请参阅服务日志。
DeploymentStarted	部署已开始。
DeploymentCompletedSuccessfully	部署成功完成。
DeploymentFailed	部署失败。有关详细信息，请参阅服务日志。
PauseServiceStarted	服务暂停已开始。

Status	消息
PauseServiceCompletedSuccessfully	服务暂停成功完成。
PauseServiceFailed	服务暂停失败。
ResumeServiceStarted	服务恢复已启动。
ResumeServiceCompletedSuccessfully	服务恢复成功完成。
ResumeServiceFailed	服务恢复失败。

使用记录 App Runner API 调用 AWS CloudTrail

App Runner 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在 App Runner 中执行的操作的记录。 CloudTrail将 App Runner 的所有 API 调用捕获为事件。捕获的调用包括来自 App Runner 控制台的调用和对 App Runner API 操作的代码调用。如果您创建跟踪，则可以允许将 CloudTrail 事件持续传输到 Amazon S3 存储桶，包括应用程序运行器的事件。如果您未配置跟踪，您仍然可以在 CloudTrail 控制台的 “事件历史记录” 中查看最新的事件。使用收集的信息 CloudTrail，您可以确定向 App Runner 发出的请求、发出请求的 IP 地址、谁发出了请求、何时发出请求以及其他详细信息。

要了解更多信息 CloudTrail，请参阅《AWS CloudTrail 用户指南》。

中的 App Runner 信息 CloudTrail

CloudTrail 在您创建账户 AWS 账户 时已在您的账户上启用。当 App Runner 中发生活动时，该活动会与其他 AWS 服务 CloudTrail 事件一起记录在事件历史记录中。您可以在中查看、搜索和下载最近发生的事件 AWS 账户。有关更多信息，请参阅使用事件历史记录查看 CloudTrail 事件。

要持续记录您的 AWS 账户事件（包括 App Runner 的事件），请创建跟踪。跟踪允许 CloudTrail 将日志文件传输到 Amazon S3 存储桶。预设情况下，在控制台中创建跟踪记录时，此跟踪记录应用于所有 AWS 区域。跟踪记录 AWS 分区中所有区域的事件，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，以进一步分析和处理 CloudTrail 日志中收集的事件数据。有关更多信息，请参阅下列内容：

- 创建跟踪概述

- CloudTrail 支持的服务和集成
- 配置 Amazon SNS 通知 CloudTrail
- 接收来自多个区域的 CloudTrail 日志文件和接收来自多个账户的 CloudTrail 日志文件

所有 App Runner 操作均由 API 参考记录 CloudTrail 并记录在 AWS App Runner API 参考中。例如，对`CreateService`、`DeleteConnection`和`StartDeployment`操作的调用会在 CloudTrail 日志文件中生成条目。

每个事件或日记账条目都包含有关生成请求的人员信息。身份信息可帮助您确定以下内容：

- 请求是使用根用户凭证还是 IAM 用户凭证发出的。
- 请求是使用角色还是联合用户的临时安全凭证发出的。
- 请求是否由其他 AWS 服务发出。

有关更多信息，请参阅CloudTrail 用户身份元素。

了解 App Runner 日志文件条目

跟踪是一种配置，允许将事件作为日志文件传输到您指定的 Amazon S3 存储桶。 CloudTrail 日志文件包含一个或多个日志条目。事件代表来自任何来源的单个请求，包括有关请求的操作、操作的日期和时间以及请求参数的信息。 CloudTrail 日志文件不是公共 API 调用的有序堆栈跟踪，因此它们不会按任何特定的顺序出现。

以下示例显示了演示该`CreateService`操作的 CloudTrail 日志条目。

> **Note**
>
> 出于安全考虑，某些属性值在日志中被删掉并替换为文本`HIDDEN_DUE_TO_SECURITY_REASONS`。这样可以防止机密信息意外泄露。但是，您仍然可以看到这些属性是在请求中传递的，或者是在响应中返回的。

`CreateService`App Runner 操作的 CloudTrail 日志条目示例

```
{
  "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"
                },
                "sourceDirectory": "/",
                "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,
```

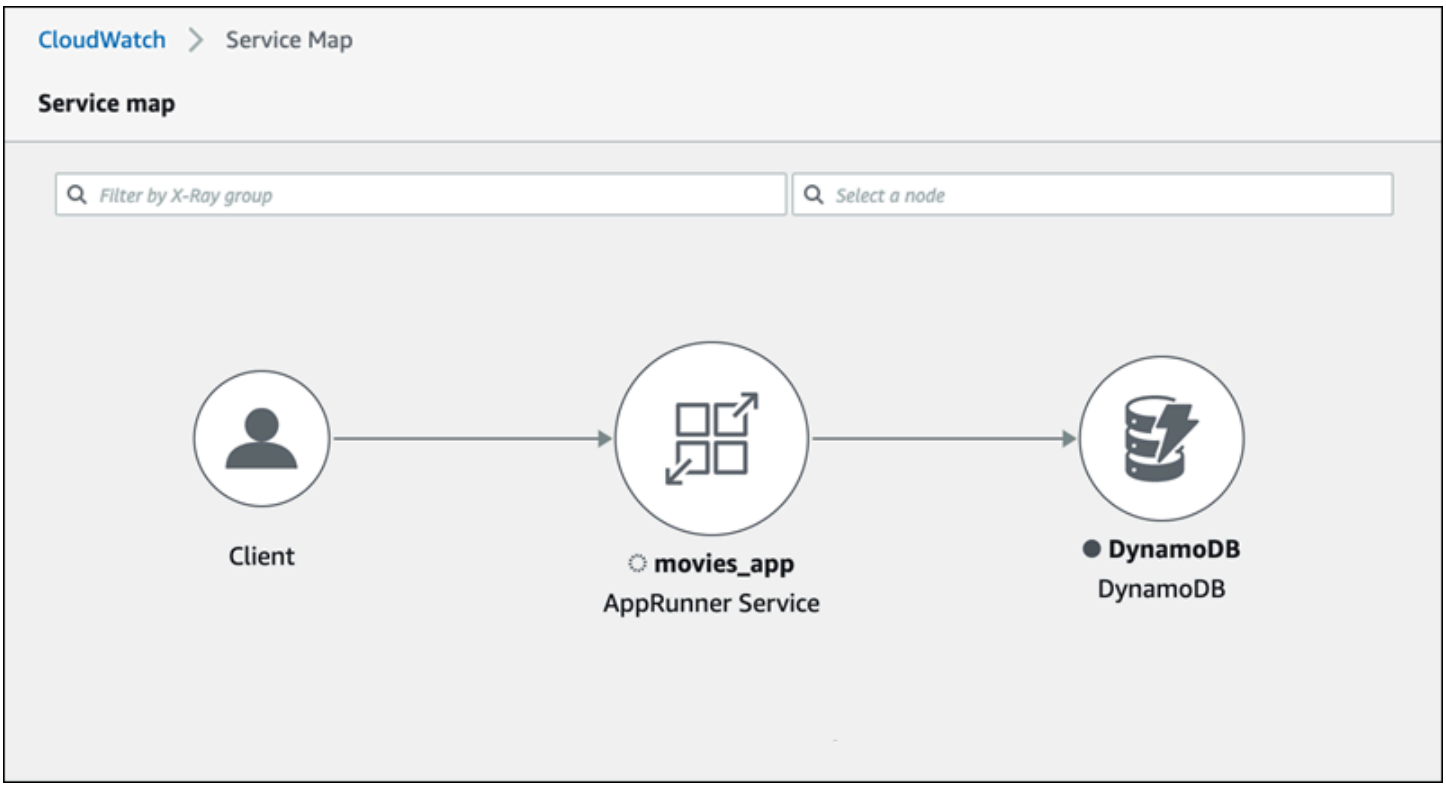
```
            "unhealthyThreshold": 5
        },
        "instanceConfiguration": {
            "cpu": "256",
            "memory": "1024"
        },
        "autoScalingConfigurationSummary": {
            "autoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
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 X-Ray 跟踪您的 App Runner 应用程序

AWS X-Ray 是一项服务，它收集有关您的应用程序所处理的请求的数据，并提供可用于查看、筛选和深入了解这些数据的工具，以识别问题和优化机会。对于对应用程序的任何跟踪请求，您不仅可以查看有关请求和响应的详细信息，还可以查看有关您的应用程序对下游 AWS 资源、微服务、数据库和 HTTP Web API 的调用的详细信息。

X-Ray 使用来自为云应用程序提供支持的 AWS 资源的跟踪数据来生成详细的服务图。该服务图显示客户端、您的前端服务以及前端服务调用来处理请求和保存数据的后端服务。您可以使用服务图来查明瓶颈、延迟峰值和其他需要解决的问题，以提高应用程序性能。

有关 X-Ray 的更多信息，请参阅 AWS X-Ray 开发人员指南。



检测您的应用程序以进行跟踪

使用便携式遥测规范 OpenTelemetry，检测您的 App Runner 服务应用程序以进行跟踪。目前，App Runner 支持 AWS Distro fo OpenTelemetry r (ADOT)，这是一种使用服务收集和呈现遥测信息的 OpenTelemetry实现。 AWS X-Ray 实现了跟踪组件。

根据您在应用程序中使用的特定 ADOT SDK，ADOT 最多支持两种检测方法：自动和手动。有关使用您的 SDK 进行检测的更多信息，请参阅 ADOT 文档，然后在导航窗格中选择您的 SDK。

运行时设置

以下是用于检测您的 App Runner 服务应用程序以进行跟踪的一般运行时设置说明。

为运行时设置跟踪

1. 按照 AWS Distro for OpenTelemetry (ADOT) 中为你的运行时提供的说明来检测你的应用程序。
2. 如果您使用的是源代码存储库，则在`apprunner.yaml`文件`build`部分安装所需的`OTEL`依赖项；如果您使用的是容器镜像，则在 Dockerfile 中安装所需的依赖项。
3. 如果您使用的是源代码存储库，则在`apprunner.yaml`文件中设置环境变量；如果您使用的是容器镜像，则在 Dockerfile 中设置环境变量。

Example 环境变量

> **Note**
>
> 以下示例列出了要添加到`apprunner.yaml`文件中的重要环境变量。如果您使用的是容器镜像，请将这些环境变量添加到您的 Dockerfile 中。但是，每个运行时可能都有自己的特点，您可能需要在以下列表中添加更多环境变量。有关运行时特定说明的更多信息以及有关如何为运行时设置应用程序的示例，请参阅 AWS Distro for for OpenTelemetry 并转到“入门” 下的 “运行时”。

```
env:
    - name: OTEL_PROPAGATORS
      value: xray
    - name: OTEL_METRICS_EXPORTER
      value: none
    - name: OTEL_EXPORTER_OTLP_ENDPOINT
      value: http://localhost:4317
    - name: OTEL_RESOURCE_ATTRIBUTES
      value: 'service.name=example_app'
```

> **Note**
>
> `OTEL_METRICS_EXPORTER=none`是 App Runner 的重要环境变量，因为 App Runner Otel 收集器不接受指标记录。它只接受指标跟踪。

运行时设置示例

以下示例演示如何使用 AD OT Python SDK 自动检测您的应用程序。SDK 会自动生成包含遥测数据的跨度，这些数据描述了 Python 框架在应用程序中使用的值，而无需添加一行 Python 代码。您只需要在两个源文件中添加或修改几行。

首先，添加一些依赖关系，如以下示例所示。

Example requirements.txt

```
opentelemetry-distro[otlp]>=0.24b0
opentelemetry-sdk-extension-aws~=2.0
```

```
opentelemetry-propagator-aws-xray~=1.0
```

然后，对您的应用程序进行检测。执行此操作的方法取决于您的服务来源，即源图像或源代码。

Source image

当您的服务源是镜像时，您可以直接检测 Dockerfile，该文件控制构建容器镜像和在镜像中运行应用程序。以下示例显示了用于 Python 应用程序的经过检测的 DockerFile。新增的仪器以粗体显示。

Example Dockerfile

```
FROM public.ecr.aws/amazonlinux/amazonlinux:latest
RUN yum install python3.7 -y && curl -O https://bootstrap.pypa.io/get-pip.py &&
 python3 get-pip.py && yum update -y
COPY . /app
WORKDIR /app
RUN pip3 install -r requirements.txt
RUN opentelemetry-bootstrap --action=install
ENV OTEL_PYTHON_DISABLED_INSTRUMENTATIONS=urllib3
ENV OTEL_METRICS_EXPORTER=none
ENV OTEL_RESOURCE_ATTRIBUTES='service.name=example_app'
CMD OTEL_PROPAGATORS=xray OTEL_PYTHON_ID_GENERATOR=xray opentelemetry-instrument
 python3 app.py
EXPOSE 8080
```

Source code repository

当您的服务源是包含应用程序源的存储库时，您可以使用 App Runner 配置文件设置间接检测映像。这些设置控制 App Runner 生成的 Dockerfile，这些文件用于为你的应用程序构建镜像。以下示例显示了 Python 应用程序的经过检测的 App Runner 配置文件。新增的仪器以粗体显示。

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    build:
      - pip install -r requirements.txt
      - opentelemetry-bootstrap --action=install
```

```
run:
  command: opentelemetry-instrument python app.py
  network:
    port: 8080
  env:
    - name: OTEL_PROPAGATORS
      value: xray
    - name: OTEL_METRICS_EXPORTER
      value: none
    - name: OTEL_PYTHON_ID_GENERATOR
      value: xray
    - name: OTEL_PYTHON_DISABLED_INSTRUMENTATIONS
      value: urllib3
    - name: OTEL_RESOURCE_ATTRIBUTES
      value: 'service.name=example_app'
```

为您的 App Runner 服务实例角色添加 X-Ray 权限

要将 X-Ray 跟踪与 App Runner 服务一起使用，您必须为该服务的实例提供与 X-Ray 服务交互的权限。为此，您可以将实例角色与您的服务关联并添加具有 X-Ray 权限的托管策略。有关 App Runner 实例角色的更多信息，请参阅the section called “实例角色”。将AWSXRayDaemonWriteAccess托管策略添加到您的实例角色并在创建期间将其分配给您的服务。

为您的 App Runner 服务启用 X-Ray 跟踪

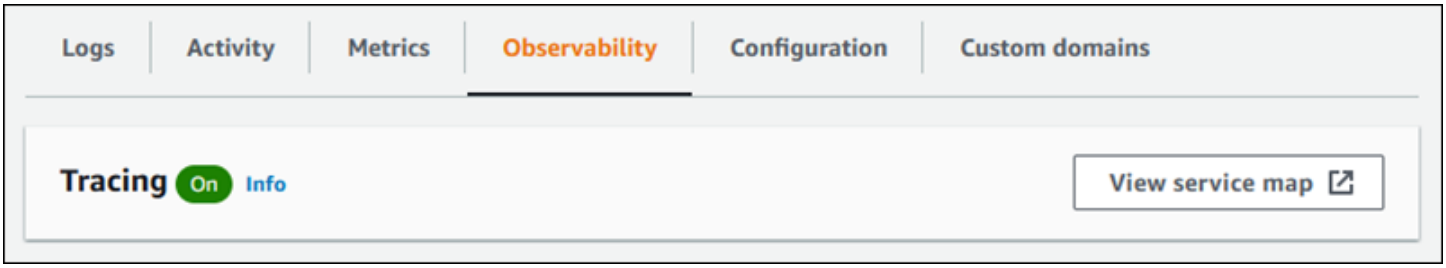
创建服务时，App Runner 默认会禁用跟踪。在配置可观察性时，您可以为服务启用 X-Ray 跟踪。有关更多信息，请参阅 the section called “管理可观察性”。

如果您使用 App Runner API 或 AWS CLI，则ObservabilityConfiguration资源TraceConfiguration对象中的对象包含跟踪设置。要保持跟踪禁用状态，请不要指定TraceConfiguration对象。

在控制台和 API 案例中，请务必将上一节中讨论的实例角色与 App Runner 服务相关联。

查看 App Runner 服务的 X-Ray 跟踪数据

在 App Runner 控制台中服务控制面板页面的可观察性选项卡上，选择查看服务地图以导航到 Amazon CloudWatch 控制台。



使用 Amazon CloudWatch 控制台查看您的应用程序所处理的请求的服务地图和跟踪。服务地图显示请求延迟以及与其他应用程序和 AWS 服务的交互等信息。您向代码中添加的自定义注释允许您轻松搜索跟踪。有关更多信息，请参阅 Amazon CloudWatch 用户指南中的使用 ServiceLens 来监控应用程序的运行状况。

将 AWS WAF Web ACL 与您的服务关联

AWS WAF 是一个 Web 应用程序防火墙，您可以使用它来保护您的 App Runner 服务。借助 AWS WAF Web 访问控制列表 (Web ACL)，您可以保护您的 App Runner 服务端点免受常见的 Web 漏洞和不想要的机器人的侵害。

Web ACL 可让您精细控制所有传入 App Runner 服务的 Web 请求。您可以在 Web ACL 中定义规则，以允许、阻止或监控 Web 流量，确保只有经过授权和合法的请求才能到达您的 Web 应用程序和 API。您可以根据自己的特定业务和安全需求自定义 Web ACL 规则。要详细了解基础设施安全和应用网络 ACL 的最佳实践，请参阅 A mazon VPC 用户指南中的控制网络流量。

> **Important**
>
> 与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WAF Web ACL。

传入的 Web 请求流

当 AWS WAF Web ACL 与 App Runner 服务关联时，传入的 Web 请求将经过以下过程：

1. App Runner 会将原始请求的内容转发给。 AWS WAF
2. AWS WAF 检查请求并将其内容与您在 Web ACL 中指定的规则进行比较。
3. 根据其检查，向 App Runner AWS WAF 返回`allow`或`block`响应。
   - 如果返回`allow`响应，App Runner 会将请求转发给您的应用程序。
   - 如果返回`block`响应，App Runner 会阻止该请求到达您的 Web 应用程序。它会将来自的`block`响应转发 AWS WAF 给您的应用程序。

> **Note**
>
> 默认情况下，如果没有返回任何响应，App Runner 会阻止该请求 AWS WAF。

有关 AWS WAF Web ACL 的更多信息，请参阅《AWS WAF 开发人员指南》中的 Web 访问控制列表 (Web ACL)。

> **Note**
>
> 您需要支付标准 AWS WAF 价格。在 App Runner 服务中使用 AWS WAF 网页 ACL 不会产生任何额外费用。 有关定价的更多信息，请参阅AWS WAF 定价。

将 WAF 网页 ACL 关联到你的 App Runner 服务

以下是将 AWS WAF Web ACL 与 App Runner 服务关联的高级流程：

1. 在 AWS WAF 控制台中创建 Web ACL。有关更多信息，请参阅《AWS WAF 开发者指南》中的创建 Web ACL。
2. 更新您的 AWS Identity and Access Management (IAM) 权限 AWS WAF。有关更多信息，请参阅权限。
3. 使用以下方法之一将 Web ACL 与 App Runner 服务相关联：
   - App Runner 控制台：创建或更新 App Runner 服务时，使用 App Runner 控制台关联现有的 Web ACL。有关说明，请参阅管理 AWS WAF Web ACL。
   - AWS WAF 控制台：使用控制 AWS WAF 台为现有 App Runner 服务关联 Web ACL。有关更多信息，请参阅 AWS WAF 开发人员指南中的将 Web ACL 与 AWS 资源关联或取消关联。
   - AWS CLI：使用 AWS WAF 公共 API 关联 Web ACL。有关 AWS WAF 公共 API 的更多信息，请参阅 AWS WAF API 参考指南中的 AssociateWebACL。

注意事项

- 与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WAF Web ACL。
- 一个 App Runner 服务只能与一个 Web ACL 关联。但是，您可以将一个 Web ACL 与多个 App Runner 服务和多个 AWS 资源相关联。示例包括 Amazon Cognito 用户池和 Application Load Balancer 资源。
- 创建 Web ACL 时，经过一小段时间后，Web ACL 才会完全传播并可用于 App Runner。传播时间可以从几秒钟到几分钟不等。 AWS WAF `WAFUnavailableEntityException`当您尝试在 Web ACL 完全传播之前将其关联时，会返回。

如果您在 Web ACL 完全传播之前刷新浏览器或离开了 App Runner 控制台，则关联将失败。但是，您可以在 App Runner 控制台中导航。

- AWS WAF 当您为处于无效状态的 App Runner 服务调用以下 AWS WAF API 之一时，会返回`WAFNonexistentItemException`错误：
  - `AssociateWebACL`
  - `DisassociateWebACL`
  - `GetWebACLForResource`

  您的 App Runner 服务的无效状态包括：
  - `CREATE_FAILED`
  - `DELETE_FAILED`
  - `DELETED`
  - `OPERATION_IN_PROGRESS`

  > **Note**
  >
  > `OPERATION_IN_PROGRESS`仅当你的 App Runner 服务被删除时，状态才无效。

- 您的请求可能会导致有效载荷大于 AWS WAF 可以检查的限制。有关如何 AWS WAF 处理来自 App Runner 的超大请求的更多信息，请参阅AWS WAF 开发者指南中的超大请求组件处理，了解如何 AWS WAF 处理来自 App Runner 的超大请求。
- 如果您未设置适当的规则或流量模式发生变化，Web ACL 可能无法有效保护您的应用程序。

权限

要在中使用 Web ACL AWS App Runner，请添加以下 IAM 权限 AWS WAF：

- `apprunner:ListAssociatedServicesForWebAcl`
- `apprunner:DescribeWebAclForService`
- `apprunner:AssociateWebAcl`
- `apprunner:DisassociateWebAcl`

有关 IAM 权限的更多信息，请参阅 IAM 用户指南中的 IAM 中的策略和权限。

以下是更新后的 IAM 政策的示例 AWS WAF。此 IAM 策略包括使用 App Runner 服务的必要权限。

Example

```
{
  {
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "wafv2:ListResourcesForWebACL",
            "wafv2:GetWebACLForResource",
            "wafv2:AssociateWebACL",
            "wafv2:DisassociateWebACL",
            "apprunner:ListAssociatedServicesForWebAcl",
            "apprunner:DescribeWebAclForService",
            "apprunner:AssociateWebAcl",
            "apprunner:DisassociateWebAcl"
         ],
         "Resource":"*"
      }
   ]
}
```

> **Note**
>
> 尽管您必须授予 IAM 权限，但列出的操作仅用于说明权限，不对应于 API 操作。

管理 AWS WAF Web ACL

使用以下方法之一管理您的 App Runner 服务的 AWS WAF 网页 ACL：

- the section called “应用程序运行器控制台”
- the section called “AWS CLI”

应用程序运行器控制台

在 App Runner 控制台上创建服务或更新现有服务时，可以关联或取消关联 AWS WAF Web ACL。

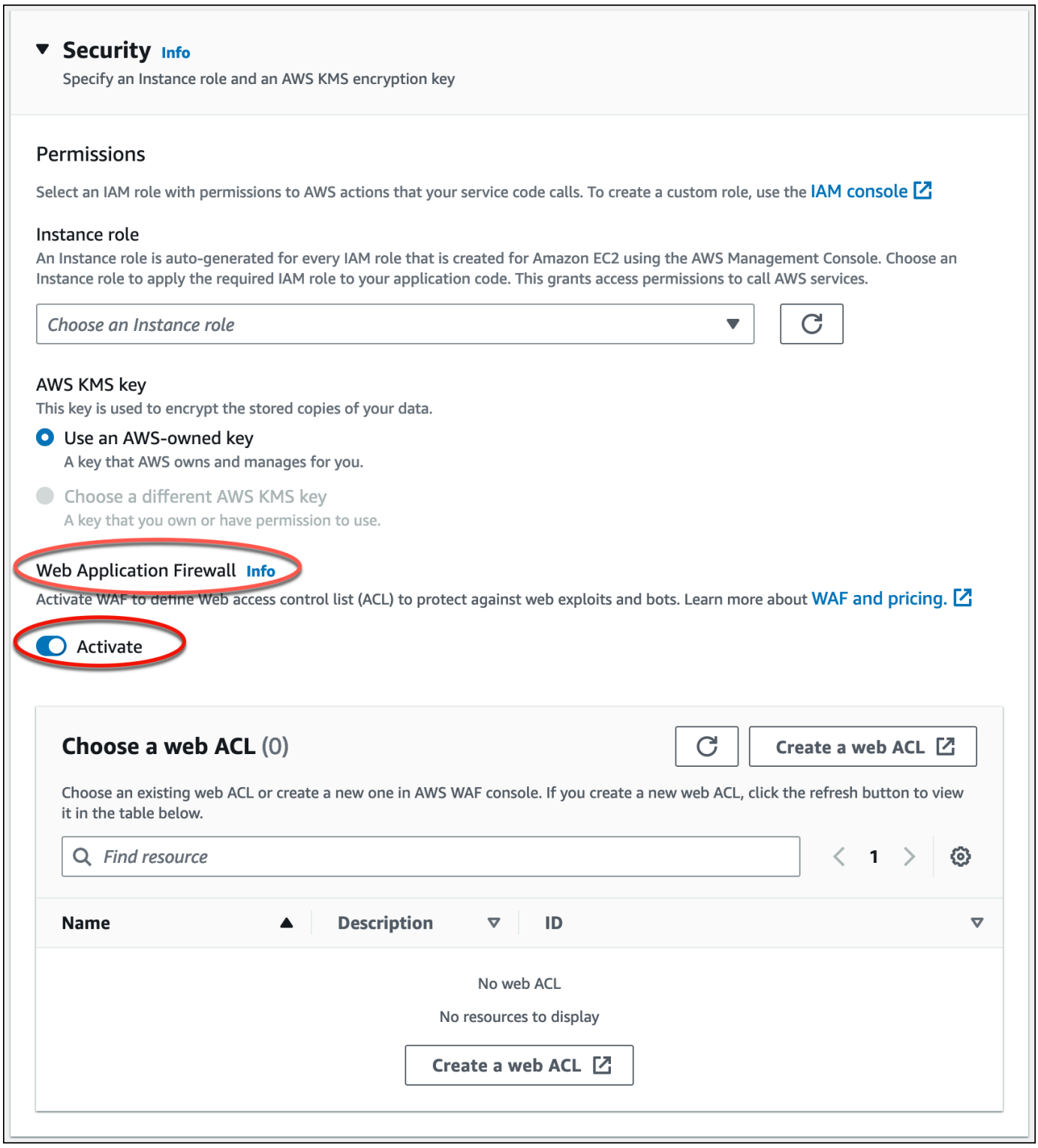
> **Note**
>
> - 一个 App Runner 服务只能与一个 Web ACL 关联。但是，除了其他 AWS 资源之外，您还可以将一个 Web ACL 与多个 App Runner 服务相关联。
> - 在关联 Web ACL 之前，请务必更新您的 IAM 权限 AWS WAF。有关更多信息，请参阅 权限。

关联 AWS WAF Web ACL

> **Important**
>
> 与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WAF Web ACL。

关联 AWS WAF Web ACL

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 根据您是在创建还是更新服务，执行以下步骤之一：
   - 如果您要创建新服务，请选择创建 App Runner 服务，然后转到配置服务。
   - 如果您要更新现有服务，请选择 “配置” 选项卡，然后在 “配置服务” 下选择 “编辑”。
3. 转到 “安全” 下的 Web 应用程序防火墙。
4. 选择 “激活” 切换按钮以查看选项。



5. 执行下列步骤之一：

   - 要关联现有 Web ACL，请执行以下操作：从 “选择要与 App Runner 服务关联的 Web ACL” 表中选择所需的 Web ACL。

- 要创建新的 Web ACL，请执行以下操作：选择创建 Web ACL 以使用 AWS WAF 控制台创建新的 Web ACL。有关更多信息，请参阅《AWS WAF 开发者指南》中的创建 Web ACL。
  1. 选择 “刷新” 按钮，在 “选择 Web ACL” 表中查看新创建的 Web ACL。
  2. 选择所需的 Web ACL。

6. 如果您要创建新服务，请选择 “下一步”；如果要更新现有服务，请选择 “保存更改”。选定的 Web ACL 与您的 App Runner 服务相关联。
7. 要验证 Web ACL 关联，请选择服务的配置选项卡，然后转到配置服务。滚动到 “安全” 下的 Web 应用程序防火墙，查看与您的服务关联的 Web ACL 的详细信息。

> **Note**
>
> 创建 Web ACL 时，经过一小段时间后，Web ACL 才会完全传播并可用于 App Runner。传播时间可以从几秒钟到几分钟不等。 AWS WAF `WAFUnavailableEntityException`当您尝试在 Web ACL 完全传播之前将其关联时，会返回。
> 如果您在 Web ACL 完全传播之前刷新浏览器或离开了 App Runner 控制台，则关联将失败。但是，您可以在 App Runner 控制台中导航。

取消关联 Web ACL AWS WAF

您可以通过更新 App Runner 服务来取消与不再需要的 AWS WAF Web ACL 的关联。

解除与 AWS WAF Web ACL 的关联

1. 打开 App Runner 控制台，然后在 “区域” 列表中，选择您的 AWS 区域。
2. 前往要更新的服务的 “配置” 选项卡，然后在 “配置服务” 下选择 “编辑”。
3. 转到 “安全” 下的 Web 应用程序防火墙。
4. 禁用 “激活” 切换按钮。您会收到一条确认删除的消息。
5. 选择确认。Web ACL 已与您的 App Runner 服务断开关联。

> **Note**
>
> - 如果要将您的服务与其他 Web ACL 关联，请从 “选择 Web ACL” 表中选择一个 Web ACL。App Runner 取消与当前 Web ACL 的关联，并启动与所选网络 ACL 关联的进程。

- 如果没有其他 App Runner 服务或资源使用已取消关联的 Web ACL，请考虑删除 Web ACL。否则，您将继续承担费用。有关定价的更多信息，请参阅AWS WAF 定价。有关如何删除 Web ACL 的说明，请参阅 AWS WAF API 参考中的 DeleteWebACL。
- 您无法删除与其他活跃的 App Runner 服务或其他资源关联的 Web ACL。

AWS CLI

您可以使用 AWS WAF 公共 API 关联或取消关联 AWS WAF Web ACL。要关联或取消关联 Web ACL 的 App Runner 服务必须处于有效状态。

AWS WAF 当您为处于无效状态的 App Runner 服务调用以下 AWS WAF API 之一时，会返回`WAFNonexistentItemException`错误：

- `AssociateWebACL`
- `DisassociateWebACL`
- `GetWebACLForResource`

您的 App Runner 服务的无效状态包括：

- `CREATE_FAILED`
- `DELETE_FAILED`
- `DELETED`
- `OPERATION_IN_PROGRESS`

> **Note**
>
> `OPERATION_IN_PROGRESS`仅当你的 App Runner 服务被删除时，状态才无效。

有关 AWS WAF 公共 API 的更多信息，请参阅 AWS WAF API 参考指南。

> **Note**
>
> 更新您的 IAM 权限 AWS WAF。有关更多信息，请参阅 权限。

使用关联 AWS WAF Web ACL AWS CLI

> **Important**
>
> 与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WAF Web ACL。

关联 AWS WAF Web ACL

1. 使用您对服务的首选规则操作集`Allow`或`Block`对服务的 Web 请求，为您的服务创建 Web ACL。 AWS WAF 有关 AWS WAF API 的更多信息，请参阅 AWS WAF API 参考指南中的 CreateWebACL。

   Example 创建 Web ACL-请求

   ```
   aws wafv2
   create-web-acl
   --region <region>
   --name <web-acl-name>
   --scope REGIONAL
   --default-action Allow={}
   --visibility-config <file-name.json>
   # This is the file containing the WAF web ACL rules.
   ```

2. 使用`associate-web-acl` AWS WAF 公共 API 将您创建的 Web ACL 与 App Runner 服务相关联。有关 AWS WAF API 的更多信息，请参阅 AWS WAF API 参考指南中的 AssociateWebACL。

   > **Note**
   >
   > 创建 Web ACL 时，经过一小段时间后，Web ACL 才会完全传播并可用于 App Runner。传播时间可以从几秒钟到几分钟不等。 AWS WAF `WAFUnavailableEntityException`当您尝试在 Web ACL 完全传播之前将其关联时，会返回。
   >
   > 如果您在 Web ACL 完全传播之前刷新浏览器或离开了 App Runner 控制台，则关联将失败。但是，您可以在 App Runner 控制台中导航。

Example 关联 Web ACL-请求

```
aws wafv2 associate-web-acl
--resource-arn <apprunner_service_arn>
--web-acl-arn <web_acl_arn>
--region <region>
```

3. 使用`get-web-acl-for-resource` AWS WAF 公共 API 验证 Web ACL 是否与您的 App Runner 服务相关联。有关 AWS WAF API 的更多信息，请参阅 AWS WAF API 参考指南 ForResource中的 GetWebACL。

   Example 验证资源的 Web ACL-请求

   ```
   aws wafv2 get-web-acl-for-resource
   --resource-arn <apprunner_service_arn>
   --region <region>
   ```

   如果没有与您的服务关联的 Web ACL，您会收到一条空白响应。

使用删除 AWS WAF Web ACL AWS CLI

如果 AWS WAF Web ACL 与 App Runner 服务相关联，则无法将其删除。

删除 AWS WAF Web ACL

1. 使用`disassociate-web-acl` AWS WAF 公共 API 解除网页 ACL 与您的 App Runner 服务的关联。有关 AWS WAF API 的更多信息，请参阅 AWS WAF API 参考指南中的 DisassociateWebACL。

   Example 取消关联 Web ACL-请求

   ```
   aws wafv2 disassociate-web-acl
   --resource-arn <apprunner_service_arn>
   --region <region>
   ```

2. 使用`get-web-acl-for-resource` AWS WAF 公共 API 验证 Web ACL 是否已与您的 App Runner 服务断开关联。

Example 验证 Web ACL 是否已解除关联-请求

```
aws wafv2 get-web-acl-for-resource
--resource-arn <apprunner_service_arn>
--region <region>
```

未列出您的 App Runner 服务的已取消关联的 Web ACL。如果没有与您的服务关联的 Web ACL，您会收到一条空白响应。

3. 使用`delete-web-acl` AWS WAF 公共 API 删除已取消关联的 Web ACL。有关 AWS WAF API 的更多信息，请参阅 AWS WAF API 参考指南中的 DeleteWebACL。

   Example 删除 Web ACL-请求

   ```
   aws wafv2 delete-web-acl
   --name <web_acl_name>
   --scope REGIONAL
   --id <web_acl_id>
   --lock-token <web_acl_lock_token>
   --region <region>
   ```

4. 使用`list-web-acl` AWS WAF 公共 API 验证网页 ACL 是否已删除。有关 AWS WAF API 的更多信息，请参阅 AP AWS WAF I 参考指南中的 ListWebACL。

   Example 验证 Web ACL 是否已删除-请求

   ```
   aws wafv2 list-web-acls
   --scope REGIONAL
   --region <region>
   ```

   已删除的 Web ACL 已不再列出。

> **Note**
>
> 如果网络 ACL 与其他活跃的 App Runner 服务或其他资源（例如 Amazon Cognito 用户池）相关联，则无法删除该网络 ACL。

列出与 Web ACL 关联的 App Runner 服务

一个 Web ACL 可以与多个 App Runner 服务和其他资源相关联。使用`list-resources-for-web-acl` AWS WAF 公共 API 列出与 Web ACL 关联的 App Runner 服务。有关 AWS WAF API 的更多信息，请参阅 AWS WAF API 参考指南中的 ListResourcesForWebACL。

Example 列出与 Web ACL 关联的 App Runner 服务-请求

```
aws wafv2 list-resources-for-web-acl
--web-acl-arn <WEB_ACL_ARN>
--resource-type APP_RUNNER_SERVICE
--region <REGION>
```

Example 列出与 Web ACL 关联的 App Runner 服务-响应

以下示例说明了没有与 Web ACL 关联的 App Runner 服务时的响应。

```
{
  "ResourceArns": []
}
```

Example 列出与 Web ACL 关联的 App Runner 服务-响应

以下示例说明了存在与 Web ACL 关联的 App Runner 服务时的响应。

```
{
  "ResourceArns": [
    "arn:aws:apprunner:<region>:<aws_account_id>:service/<service_name>/<service_id>"
  ]
}
```

测试和记录 AWS WAF Web ACL

当您在 Web ACL 中将规则操作设置为 C ou nt 时， AWS WAF 会将该请求添加到与该规则匹配的请求计数中。要使用您的 App Runner 服务测试 Web ACL，请将规则操作设置为 C ou nt，然后考虑与每条规则匹配的请求量。例如，您为`Block`操作设置了一条规则，该规则与您确定为普通用户流量的大量请求相匹配。在这种情况下，您可能需要重新配置规则。有关更多信息，请参阅《AWS WAF 开发者指南》中的测试和调整 AWS WAF 保护措施。

您还可以配置 AWS WAF 为将请求标头记录到亚马逊 CloudWatch 日志组、亚马逊简单存储服务 (Amazon S3) 存储桶或亚马逊数据 Firehose。有关更多信息，请参阅 AWS WAF 开发人员指南 中的记录 Web ACL 流量。

要访问与您的 App Runner 服务关联的 Web ACL 相关的日志，请参阅以下日志字段：

- `httpSourceName`: 包含 `APPRUNNER`
- `httpSourceId`: 包含 `customeraccountid-apprunnerserviceid`

有关更多信息，请参阅《AWS WAF 开发人员指南》中的日志示例。

> **Important**
>
> 与 WAF Web ACL 关联的 App Runner 私有服务的源 IP 规则不符合基于 IP 的规则。这是因为我们目前不支持将请求源 IP 数据转发到与 WAF 关联的 App Runner 私有服务。如果您的 App Runner 应用程序需要源 IP/CIDR 传入流量控制规则，则必须对私有端点使用安全组规则，而不是 WAF Web ACL。

使用配置文件设置 App Runner 服务选项

> **Note**
>
> 配置文件仅适用于基于源代码的服务。您不能将配置文件用于基于图像的服务。

使用源代码存储库创建 AWS App Runner 服务时， AWS App Runner 需要有关构建和启动服务的信息。每次使用 App Runner 控制台或 API 创建服务时，您都可以提供这些信息。或者，您可以使用配置文件来设置服务选项。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的跟踪方式与跟踪源代码更改的方式类似。您可以使用 App Runner 配置文件来指定超过 API 支持的选项。如果您只需要API支持的基本选项，则无需提供配置文件。

App Runner 配置文件是一个 YAML 文件，`apprunner.yaml`在应用程序存储库的源目录中命名。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，并提供网络设置和环境变量等运行时上下文。

App Runner 配置文件不包括操作设置，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

- App Runner 配置文件示例
- App Runner 配置文件参考

App Runner 配置文件示例

> **Note**
>
> 配置文件仅适用于基于源代码的服务。您不能将配置文件用于基于图像的服务。

以下示例演示了 AWS App Runner 配置文件。有些是最低限度的，仅包含必需的设置。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App Runner 配置文件。

配置文件示例

最小配置文件

使用最少的配置文件，App Runner 会做出以下假设：

- 在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 apprunner.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

此示例显示了在托管运行时中使用`apprunner.yaml`原始格式的所有配置密钥的情况。

Example apprunner.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"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

完整的配置文件 — (使用修订后的版本)

此示例说明如何使用托管运行时中的`apprunner.yaml`所有配置密钥。

只有修订后的 App Runner 版本支持该`pre-run`参数。如果您的应用程序使用原始 App Runner 版本支持的运行时版本，请不要在配置文件中插入此参数。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

> **Note**
>
> 由于此示例适用于 Python 3.11，因此我们使用`pip3`和`python3`命令。有关更多信息，请参阅 Python 平台主题特定运行时版本的标注中的。

Example apprunner.yaml

```
version: 1.0
runtime: python311
build:
  commands:
    pre-build:
```

```
      - wget -c https://s3.amazonaws.com/DOC-EXAMPLE-BUCKET/test-lib.tar.gz -O - | tar
 -xz
    build:
      - pip3 install pipenv
      - pipenv install
    post-build:
      - python3 manage.py test
  env:
    - name: DJANGO_SETTINGS_MODULE
      value: "django_apprunner.settings"
    - name: MY_VAR_EXAMPLE
      value: "example"
run:
  runtime-version: 3.11
  pre-run:
    - pip3 install pipenv
    - pipenv install
    - python3 copy-global-files.py
  command: pipenv run gunicorn django_apprunner.wsgi --log-file -
  network:
    port: 8000
    env: MY_APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

有关特定托管运行时配置文件的示例，请参阅下基于代码的服务方的特定运行时副主题。

App Runner 配置文件参考

> **Note**
>
> 配置文件仅适用于基于源代码的服务。您不能将配置文件用于基于图像的服务。

本主题是 AWS App Runner 配置文件语法和语义的综合参考指南。有关 App Runner 配置文件的概述，请参阅App Runner 配置文件。

App Runner 配置文件是一个 YAML 文件。为它命名`apprunner.yaml`，然后将其放在应用程序存储库的源目录中。

结构概述

App Runner 配置文件是一个 YAML 文件。为它命名`apprunner.yaml`，然后将其放在应用程序存储库的源目录中。

App Runner 配置文件包含以下主要部分：

- 顶部-包含顶级密钥
- 构建部分-配置构建阶段
- 运行部分-配置运行时阶段

顶部部分

文件顶部的密钥提供了有关文件和服务运行时的一般信息。以下密钥可用：

- `version`— 必填项。App Runner 配置文件版本。理想情况下，使用最新版本。

  语法

  ```
  version: version
  ```

  Example

  ```
  version: 1.0
  ```

- `runtime`— 必填项。您的应用程序使用的运行时名称。要了解 App Runner 提供的不同编程平台的可用运行时，请参阅基于代码的服务。

  > **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。您可以指定生成命令和环境变量。生成命令是必需的。

该部分以`build:`密钥开头，并包含以下子密钥：

- `commands`— 必填项。指定 App Runner 在各个构建阶段运行的命令。包括以下子项：
  - `pre-build`— 可选。App Runner 在构建之前运行的命令。例如，安装npm依赖项或测试库。
  - `build`— 必填项。App Runner 为构建您的应用程序而运行的命令。例如，使用pipenv。
  - `post-build`— 可选。App Runner 在编译后运行的命令。例如，使用 Maven 将构建工件打包到 JAR 或 WAR 文件中，或者运行测试。

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。为构建阶段指定自定义环境变量。定义为名称-值标量映射。你可以在编译命令中按名称引用这些变量。

> **Note**
>
> 在此配置文件中，有两个不同的env条目位于两个不同的位置。一组在 “构建” 部分，另一组在 “运行” 部分。
>
> - 在生成过程中，pre-build、buildpost-build、和pre-run命令可以引用 “构建” 部分中的env集合。
>
> 重要-请注意，这些pre-run命令位于此文件的 “运行” 部分，尽管它们只能访问在 “构建” 部分中定义的环境变量。
> - 运行时环境中的run命令可以引用 “运行” 部分中的env设置。

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"
```

跑步部分

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、预运行命令（仅限修改后的格式）、启动命令、网络端口和环境变量。

该部分以`run:`密钥开头，并包含以下子密钥：

- `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，只有主版本处于锁定状态。App Runner 使用最新的次要版本和补丁版本，这些版本在每次部署或服务更新时都可供运行时使用。如果您指定主要版本和次要版本，则两者都将被锁定，并且 App Runner 仅更新补丁版本。如果您指定了主版本、次要版本和补丁版本，则您的服务将锁定在特定的运行时版本上，App Runner 永远不会对其进行更新。

  语法

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

  > **Note**
  >
  > 某些平台的运行时具有不同的版本组件。有关详细信息，请参阅特定平台主题。

  Example

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

- `pre-run`— 可选。 仅修改版本用法。指定 App Runner 在将应用程序从构建映像复制到运行映像后运行的命令。您可以在此处输入命令来修改`/app`目录外的运行映像。例如，如果您需要安装位于`/`

app目录之外的其他全局依赖项，请在此小节中输入所需的命令来执行此操作。有关 App Runner 构建过程的更多信息，请参阅托管运行时版本和 App Runner 版本。

> **Note**
>
> - 重要-即使`pre-run`命令列在 “运行” 部分，它们也只能引用此配置文件的 “构建” 部分中定义的环境变量。它们不能引用本 “运行” 部分中定义的环境变量。
> - 只有修订后的 App Runner 版本支持该`pre-run`参数。如果您的应用程序使用原始 App Runner 版本支持的运行时版本，请不要在配置文件中插入此参数。有关更多信息，请参阅 托管运行时版本和 App Runner 版本。

语法

```
run:
  pre-run:
      - command
      - …
```

- `command`— 必填项。App Runner 在完成应用程序构建后用来运行应用程序的命令。

  语法

  ```
  run:
    command: command
  ```

- `network`— 可选。指定您的应用程序监听的端口。其中包括以下内容：
  - `port`— 可选。如果指定，则这是您的应用程序监听的端口号。默认值为 `8080`。
  - `env`— 可选。如果指定，App Runner 除了在默认环境变量中传递相同的端口号（而不是代替）外，还会将端口号传递给此环境变量中的容器`PORT`。换句话说，如果您指定`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。为运行阶段定义自定义环境变量。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。

> **Note**
>
> 在此配置文件中，有两个不同的env条目位于两个不同的位置。一组在 “构建” 部分，另一组在 “运行” 部分。
>
> - 在生成过程中，pre-build、buildpost-build、和pre-run命令可以引用 “构建” 部分中的env集合。
>
> 重要-请注意，这些pre-run命令位于此文件的 “运行” 部分，尽管它们只能访问在 “构建” 部分中定义的环境变量。
> - 运行时环境中的run命令可以引用 “运行” 部分中的env设置。

语法

```
run:
  env:
    - name: name1
      value: value1
    - name: name2
      value: value2
  secrets:
    - name: name1
      value-from: arn:aws:secretsmanager:region:aws_account_id:secret:secret-id
    - name: name2
      value-from: arn:aws:ssm:region:aws_account_id:parameter/parameter-name
    - …
```

Example

```
run:
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
  secrets:
    - name: my-secret
      value-from: "arn:aws:secretsmanager:us-
east-1:123456789012:secret:testingstackAppRunnerConstr-kJFXde2ULKbT-
S7t8xR:username::"
    - name: my-parameter
      value-from: "arn:aws:ssm:us-east-1:123456789012:parameter/parameter-name"
    - name: my-parameter-only-name
      value-from: "parameter-name"
```

App Runner API

AWS App Runner 应用程序编程接口 (API) 是一个 RESTful API，用于向 App Runner 服务发出请求。您可以使用 API 来创建、列出、描述、更新和删除您中的 App Runner 资源 AWS 账户。

您可以直接在应用程序代码中调用 API，也可以使用其中一个 AWS SDK。

如需完整的 API 参考信息，请参阅 AWS App Runner API 参考。

有关 AWS 开发者工具的更多信息，请参阅构建工具 AWS。

主题

- 使用与 App Runner 配合使用 AWS CLI
- 使用 AWS CloudShell 来使用 AWS App Runner

使用与 App Runner 配合使用 AWS CLI

对于命令行脚本，AWS CLI请使用调用 App Runner 服务。有关完整的 AWS CLI 参考信息，请参阅《命令参考》中的 apprunner。AWS CLI

AWS CloudShell 允许您跳过在开发环境 AWS CLI 中安装的步骤， AWS Management Console 而是在中使用它。除了避免安装之外，您也不需要配置凭证，也不需要指定区域。您的 AWS Management Console 会话为提供了此上下文 AWS CLI。有关 CloudShell更多信息以及用法示例，请参阅the section called “使用 AWS CloudShell”。

使用 AWS CloudShell 来使用 AWS App Runner

AWS CloudShell 是一个基于浏览器、经过预先验证的 shell，您可以直接从启动。 AWS Management Console您可以使用首选的外壳（Bash PowerShell 或 Z shell AWS App Runner）对 AWS 服务（包括）运行 AWS CLI 命令。您无需下载或安装命令行工具，即可完成此操作。

您可以AWS CloudShell 从启动 AWS Management Console，用于登录控制台的 AWS 凭据将在新的 shell 会话中自动可用。这种对 AWS CloudShell 用户的预身份验证允许您在使用 AWS CLI 版本 2（预先安装在 shell 的计算环境中）与 App Runner 等 AWS 服务进行交互时跳过配置凭据。

主题

- 获取 IAM 权限 AWS CloudShell

- 使用与 App Runner 互动 AWS CloudShell
- 使用验证您的 App Runner 服务 AWS CloudShell

获取 IAM 权限 AWS CloudShell

使用提供的访问管理资源 AWS Identity and Access Management，管理员可以向 IAM 用户授予权限，使他们能够访问 AWS CloudShell 和使用环境的功能。

管理员向用户授予访问权限的最快方法是通过 AWS 托管策略。AWS 托管式策略 是由 AWS创建和管理的独立策略。以下的 AWS 托管策略 CloudShell 可以附加到 IAM 身份：

- `AWSCloudShellFullAccess`：授予使用权限，并 AWS CloudShell 具有对所有功能的完全访问权限。

如果您想限制 IAM 用户可以执行的操作范围 AWS CloudShell，则可以创建使用`AWSCloudShellFullAccess`托管策略作为模板的自定义策略。有关限制中可供用户执行的操作的更多信息 CloudShell，请参阅AWS CloudShell 用户指南中的使用 IAM 策略管理 AWS CloudShell 访问和使用情况。

> **Note**
>
> 您的 IAM 身份还需要一个策略，该策略授予对 App Runner 进行调用的权限。有关更多信息，请参阅 the section called “应用程序运行器和 IAM”。

使用与 App Runner 互动 AWS CloudShell

AWS CloudShell 从启动后 AWS Management Console，您可以立即开始使用命令行界面与 App Runner 进行交互。

在以下示例中，您可以使用中的来检索有关您的某个 App Runner 服务的信息 CloudShell。 AWS CLI

> **Note**
>
> AWS CLI 在中使用时 AWS CloudShell，您无需下载或安装任何其他资源。此外，由于已经在 Shell 中进行了身份验证，因此在进行调用之前无需配置凭证。

Example 使用检索 App Runner 服务信息 AWS CloudShell

1. 从中 AWS Management Console，您可以 CloudShell 通过选择导航栏上的以下可用选项来启动：
   - 选择图 CloudShell 标。
   - 开始**cloudshell**在搜索框中键入内容，然后在搜索结果中看到该CloudShell选项时选择该选项。
2. 要在控制台会话 AWS 区域中列出您 AWS 账户中的所有当前 App Runner 服务，请在命令行中输入以下 CloudShell命令：

   ```
   $ aws apprunner list-services
   ```

   输出列出了您的服务的摘要信息。

   ```
   {
     "ServiceSummaryList": [
       {
         "ServiceName": "my-app-1",
         "ServiceId": "8fe1e10304f84fd2b0df550fe98a71fa",
         "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-
   app-1/8fe1e10304f84fd2b0df550fe98a71fa",
         "ServiceUrl": "psbqam834h.us-east-1.awsapprunner.com",
         "CreatedAt": "2020-11-20T19:05:25Z",
         "UpdatedAt": "2020-11-23T12:41:37Z",
         "Status": "RUNNING"
       },
       {
         "ServiceName": "my-app-2",
         "ServiceId": "ab8f94cfe29a460fb8760afd2ee87555",
         "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-app-2/
   ab8f94cfe29a460fb8760afd2ee87555",
         "ServiceUrl": "e2m8rrrx33.us-east-1.awsapprunner.com",
         "CreatedAt": "2020-11-06T23:15:30Z",
         "UpdatedAt": "2020-11-23T13:21:22Z",
         "Status": "RUNNING"
       }
     ]
   }
   ```

3. 要获取特定 App Runner 服务的详细描述，请使用上一步中检索到的 ARN 之一，在 CloudShell 命令行中输入以下命令：

```
$ aws apprunner describe-service --service-arn arn:aws:apprunner:us-
east-2:123456789012:service/my-app-1/8fe1e10304f84fd2b0df550fe98a71fa
```

输出中列出了您指定的服务的详细描述。

```
{
  "Service": {
    "ServiceName": "my-app-1",
    "ServiceId": "8fe1e10304f84fd2b0df550fe98a71fa",
    "ServiceArn": "arn:aws:apprunner:us-east-2:123456789012:service/my-
app-1/8fe1e10304f84fd2b0df550fe98a71fa",
    "ServiceUrl": "psbqam834h.us-east-1.awsapprunner.com",
    "CreatedAt": "2020-11-20T19:05:25Z",
    "UpdatedAt": "2020-11-23T12:41:37Z",
    "Status": "RUNNING",
    "SourceConfiguration": {
      "CodeRepository": {
        "RepositoryUrl": "https://github.com/my-account/python-hello",
        "SourceCodeVersion": {
          "Type": "BRANCH",
          "Value": "main"
        },
        "CodeConfiguration": {
          "CodeConfigurationValues": {
            "BuildCommand": "[pip install -r requirements.txt]",
            "Port": "8080",
            "Runtime": "PYTHON_3",
            "RuntimeEnvironmentVariables": [
              {
                "NAME": "Jane"
              }
            ],
            "StartCommand": "python server.py"
          },
          "ConfigurationSource": "API"
        }
      },
      "AutoDeploymentsEnabled": true,
      "AuthenticationConfiguration": {
        "ConnectionArn": "arn:aws:apprunner:us-east-2:123456789012:connection/my-
github-connection/e7656250f67242d7819feade6800f59e"
      }
```

```
    },
    "InstanceConfiguration": {
      "CPU": "1 vCPU",
      "Memory": "3 GB"
    },
    "HealthCheckConfiguration": {
      "Protocol": "TCP",
      "Path": "/",
      "Interval": 10,
      "Timeout": 5,
      "HealthyThreshold": 1,
      "UnhealthyThreshold": 5
    },
    "AutoScalingConfigurationSummary": {
      "AutoScalingConfigurationArn": "arn:aws:apprunner:us-
east-2:123456789012:autoscalingconfiguration/
DefaultConfiguration/1/00000000000000000000000000000001",
      "AutoScalingConfigurationName": "DefaultConfiguration",
      "AutoScalingConfigurationRevision": 1
    }
  }
}
```

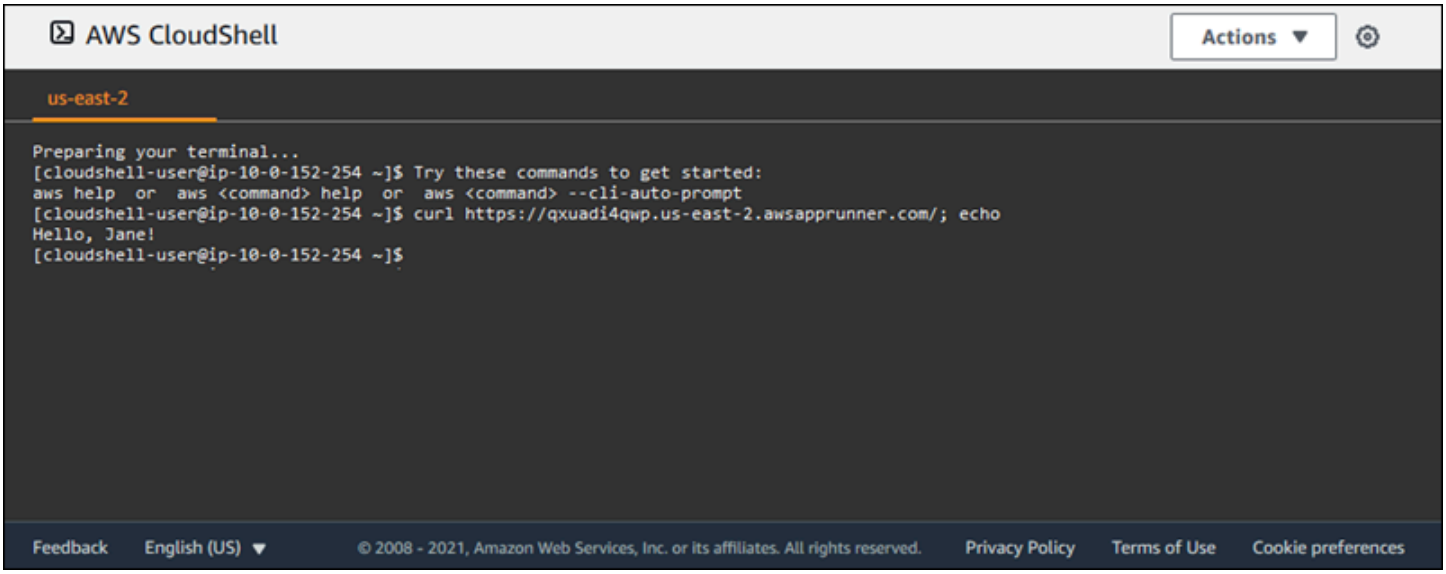
使用验证您的 App Runner 服务 AWS CloudShell

创建 App Runner 服务时，App Runner 会为您的服务的网站创建一个默认域名，并将其显示在控制台中（或在 API 调用结果中返回）。您可以使用拨 CloudShell 打您的网站并验证其是否正常运行。

例如，按照中所述创建 App Runner 服务后开始使用，在中运行以下命令 CloudShell：

```
$ curl https://qxuadi4qwp.us-east-2.awsapprunner.com/; echo
```

输出应显示预期的页面内容。


AWS CloudShell
Actions
us-east-2
Preparing your terminal...
[cloudshell-user@ip-10-0-152-254 ~]$ Try these commands to get started:
aws help or aws <command> help or aws <command> --cli-auto-prompt
[cloudshell-user@ip-10-0-152-254 ~]$ curl https://qxuadi4qwp.us-east-2.awsapprunner.com/; echo
Hello, Jane!
[cloudshell-user@ip-10-0-152-254 ~]$
Feedback
English (US)
© 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Cookie preferences

故障排除

本章提供您在使用 AWS App Runner 服务时可能遇到的常见错误和问题的故障排除步骤。错误消息可能出现在服务页面的控制台、API 或日志选项卡上。

有关更多故障排除建议和常见支持问题的答案，请访问知识中心。

主题

当服务创建失败时

如果您尝试创建 App Runner 服务失败，则该服务将进入CREATE_FAILED状态。此状态在控制台上显示为 “创建失败”。由于与以下一项或多项相关的问题，服务可能无法创建：

- 您的应用程序代码
- 构建过程
- 配置
- 资源配额
- 您的服务使用的底层 AWS 服务 存在临时问题

要对无法创建的服务进行故障排除，我们建议您执行以下操作。

1. 阅读服务事件和日志，找出导致服务创建失败的原因。
2. 对您的代码或配置进行任何必要的更改。
3. 如果您已达到服务配额，请删除一项或多项服务。
4. 如果您已达到其他资源配额，则如果可调整，则可以增加该配额。
5. 完成上述所有步骤后，请尝试再次重建服务。有关如何重建服务的信息，请参阅the section called “重建失败的服务”。

> **Note**
>
> 可能导致问题的可调整资源配额之一是 Fargate 按需 vCPU 资源。vCPU 资源数量决定了 App Runner 可以为您的服务提供的实例数量。这是服务中驻留的 Fargate 按需 vCPU 资源数量的可调整配额值。 AWS Fargate (Fargate) 要查看您账户的 vCPU 配额设置或申请增加配额，请使用中的 Service Quotas 控制台。 AWS Management Console有关更多信息，请参阅AWS Fargate Amazon 弹性容器服务开发人员指南中的服务配额。

> **Important**
>
> 对于失败的服务，除了初始创建尝试之外，您不会产生任何额外费用。即使失败的服务不可用，它仍会计入您的服务配额。App Runner 不会自动删除失败的服务，因此请务必在完成故障分析后将其删除。

自定义域名

本节介绍如何对链接到自定义域名时可能遇到的各种错误进行故障排除和解决。

> **Note**
>
> 为了增强 App Runner 应用程序的安全性，*.awsapprunner.com 域已在公共后缀列表 (PSL) 中注册。为了进一步提高安全性，如果您需要在 App Runner 应用程序的默认域名中设置敏感 Cookie，我们建议您使用带__Host-前缀的 Cookie。这将有助于保护您的域，防范跨站点请求伪造 (CSRF) 攻击。要了解更多信息，请参阅 Mozilla 开发者网络中的 Set-Cookie 页面。

自定义域名出现创建失败错误

- 检查此错误是否由于 CAA 记录有问题所致。如果 DNS 树中的任何地方都没有 CAA 记录，则您会收到一条消息`fail open`，并 AWS Certificate Manager 颁发证书以验证自定义域。这允许 App Runner 接受自定义域。如果您在 DNS 记录中使用 CAA 认证，请确保至少一个域名的 CAA 记录包括。`amazon.com`否则，ACM 将无法颁发证书。因此，无法创建 App Runner 的自定义域。

以下示例使用 DNS 查找工具 diG 来显示缺少必填条目的 CAA 记录。该示例使用example.com作为自定义域。在示例中运行以下命令以检查 CAA 记录。

```
...
;; QUESTION SECTION:
;example.com.            IN  CAA

;; ANSWER SECTION:
example.com.         7200    IN  CAA 0 iodef "mailto:hostmaster@example.com"
example.com.         7200    IN  CAA 0 issue "letsencrypt.org"
...note absence of "amazon.com" in any of the above CAA records...
```

- 更正域名记录，并确保至少有一条 CAA 记录包括amazon.com。
- 重试将自定义域与 App Runner 关联。

有关如何解决 CAA 错误的说明，请参阅以下内容：

- 证书颁发机构授权 (CAA) 问题
- 如何解决在颁发或续订 ACM 证书时出现的 CAA 错误？

自定义域名的 DNS 证书验证待处理错误

- 检查您是否跳过了自定义域名设置中的一个重要步骤。此外，请检查您是否使用 DI G 等 DNS 查询工具错误地配置了 DNS 记录。特别是，请检查是否存在以下错误：
  - 任何错过的步骤。
  - DNS 记录中不支持的字符，例如双引号。
- 纠正错误。
- 重试将自定义域与 App Runner 关联。

有关如何解决 CAA 验证错误的说明，请参阅以下内容。

- 域名系统验证
- the section called “自定义域名”

基本故障排除命令

- 确认可以找到服务。

```
aws apprunner list-services
```

- 描述一项服务并检查其状态。

```
aws apprunner describe-service --service-arn
```

- 检查自定义域的状态。

```
aws apprunner describe-custom-domains --service-arn
```

- 列出所有正在进行的操作。

```
aws apprunner list-operations --service-arn
```

续订自定义域名证书

当你向服务添加自定义域名时，App Runner 会为你提供一组别名记录，供你添加到 DNS 服务器中。这些 CNAME 记录包括证书记录。App Runner 使用 AWS Certificate Manager (ACM) 来验证域。App Runner 会验证这些 DNS 记录，以确保该域名的持续所有权。如果您从 DNS 区域中移除 CNAME 记录，App Runner 将无法再验证 DNS 记录，并且自定义域证书也无法自动续订。

本节介绍如何解决以下自定义域名证书续订问题：

- the section called “CNAME 已从 DNS 服务器中删除”.
- the section called “证书已过期”.

CNAME 已从 DNS 服务器中删除

- 使用 DescribeCustomDomainsAPI 或 App Runner 控制台中的自定义域名设置检索您的别名记录。有关存储的 CNAME 的信息，请参阅CertificateValidationRecords。
- 将证书验证 CNAME 记录添加到您的 DNS 服务器。然后，App Runner 可以验证您是否拥有该域名。添加 CNAME 记录后，最长可能需要 30 分钟才能传播 DNS 记录。App Runner 和 ACM 也可能需要几个小时才能重试证书续订流程。有关如何添加 CNAME 记录的说明，请参阅the section called “管理自定义域名”。

证书已过期

- 使用 App Runner 控制台或 API 取消关联（取消链接），然后关联（链接）您的 App Runner 服务的自定义域。App Runner 会创建新的证书验证 CNAME 记录。
- 将新的证书验证 CNAME 记录添加到您的 DNS 服务器。

  有关如何取消关联（取消关联）和关联（链接）自定义域名的说明，请参阅。the section called “管理自定义域名”

如何验证证书是否已成功续订

您可以检查证书记录的状态，以验证您的证书是否已成功续订。您可以使用诸如 curl 之类的工具来检查证书的状态。

有关证书续订的更多信息，请参阅以下链接：

- 为什么我的 ACM 证书被标记为不符合续订资格？
- ACM 证书的托管续订
- 域名系统验证

HTTP/HTTPS 请求路由错误

本节介绍如何排除和解决将 HTTP/HTTPS 流量路由到 App Runner 服务端点时可能遇到的错误。

404 向 App Runner 服务端点发送 HTTP/HTTPS 流量时出现未找到错误

- 当 App Runner 使用主机标头信息来路由请求时，请验证是否指向 HTTP 请求中的服务 URL。Host Header大多数客户端（例如cURL）和 Web 浏览器会自动将主机标头指向服务 URL。如果您的客户端未将服务 URL 设置为Host Header，则会收到404 Not Found错误消息。

  Example 主机标头不正确

  ```
  $ ~ curl -I -H "host: foobar.com" https://testservice.awsapprunner.com/
  HTTP/1.1 404 Not Found
  transfer-encoding: chunked
  ```

  Example 正确的主机标头

  ```
  $ ~ curl -I -H "host: testservice.awsapprunner.com" https://
  testservice.awsapprunner.com/
  HTTP/1.1 200 OK
  content-length: 11772
  content-type: text/html; charset=utf-8
  ```

- 验证您的客户端是否正确设置了路由到公共或私有服务的请求的服务器名称指示器 (SNI)。对于 TLS 终止和请求路由，App Runner 使用在 HTTPS 连接中设置的 SNI。

当服务无法连接到 Amazon RDS 或下游服务时

如果您的服务无法连接到 Amazon RDS 数据库或其他下游应用程序或服务，则可能存在网络配置问题。本主题将引导您完成一些步骤，以确定您的网络配置是否存在问题，以及纠正这些问题的选项。要了解有关 App Runner 出站流量配置的更多信息，请参阅为出站流量启用 VPC 访问权限。

> **Note**
>
> 要查看您的 VPC 连接器配置，请从 App Runner 控制台的左侧导航窗格中选择网络配置。然后选择“传出流量”选项卡。选择一个 VPC 连接器。下一页显示有关 VPC 连接器的详细信息。在此页面中，您可以查看和深入了解以下内容：使用 VPC 的子网、安全组和 App Runner 服务。

缩小应用程序无法连接到其他下游服务的原因

1. 确保 VPC 连接器中使用的子网是私有子网。如果连接器配置了公有子网，则您的服务将遇到错误，因为每个子网的底层 Hyperplane ENI（弹性网络接口）没有公有 IP 空间。

   如果您的 VPC 连接器使用公有子网，则可以使用以下选项来更正此配置：

   a. 创建一个新的私有子网，并使用它来代替 VPC 连接器的公有子网。有关更多信息，请参阅 Amazon VPC 用户指南中的您的 VPC 的子网。

   b. 通过 NAT 网关路由现有的公有子网。有关更多信息，请参阅 Amazon VPC 用户指南中的 NAT 网关。

2. 验证 VPC 连接器的安全组入口和出口规则是否正确。在 App Runner 控制台的左侧导航窗格中，选择网络配置 > 传出流量。从列表中选择 VPC 连接器。下一页列出了您可以选择检查的安全组。

3. 验证您尝试连接的 RDS 实例或其他下游服务的安全组入站和出站规则是否正确。有关更多信息，请参阅 App Runner 应用程序尝试连接的下游服务的服务指南。

4. 要确认您的 App Runner 配置之外没有其他类型的网络设置问题，请尝试连接到 App Runner 之外的 RDS 或下游服务：

   a. 尝试从同一 VPC 中的 Amazon EC2 实例连接到 RDS 实例或服务。

   b. 如果您正在尝试连接到服务 VPC 终端节点，请通过从同一 VPC 中的 EC2 实例访问相同的终端节点来验证连接。

5. 如果第 4 步中的任何一项连接测试失败，则很可能您的 AWS账户中存在其他资源的 App Runner 配置之外存在问题。请联系 Su AWS pport 寻求帮助，以进一步隔离和修复其他网络配置的问题。

6. 如果您按照步骤 4 中的说明成功连接到 RDS 实例或下游服务，请按照此步骤中的说明继续操作。我们将通过启用和检查 Hyperplane ENI 流量日志来检查流量是否正在进入 ENI。

   > **Note**
   >
   > 为了能够完成这些步骤并获取所需的 ENI 流日志信息，必须在 App Runner 服务成功启动后尝试连接 RDS 或下游服务。当 RDS 或下游服务处于 “运行” 状态时，您的应用程序必须执行与 RDS 或下游服务的连接操作。否则，可以在 App Runner 的回滚工作流程中清理 ENI。这种方法可确保ENI可以继续进行进一步调查。

   a. 从 AWS 控制台启动 EC2 控制台。

   b. 在左侧导航窗格的 “网络和安全” 分组中，选择 “网络接口”。

c. 滚动到接口类型和描述列，在与 VPC 连接器关联的子网中找到 ENI。它们将具有以下命名模式。

   - 接口类型：far gate
   - 描述：以 AWSAppRunner ENI（示例：AWSAppRunner ENI - abcde123-abcd-1234-1234-abcde1233456）开头

d. 使用各行开头的复选框选择适用的 ENI。

e. 从 “操作” 菜单中选择 “创建流日志”。

f. 在提示中输入信息，然后选择页面底部的创建流日志。

g. 检查生成的流日志。

   - 如果您在测试连接时流量正在进入 ENI，则问题与 ENI 设置无关。除了 App Runner 服务之外，您的 AWS 账户中的其他资源可能存在网络配置问题。如需进一步帮助，请联系 AWS Support。
   - 如果您在测试连接时流量没有进入 ENI，我们建议您联系 Support 以查看 Fargate AWS 服务是否存在任何已知问题。

h. 使用网络 Reach ability Analyzer 工具。当无法访问虚拟网络路径中的源时，此工具可识别阻塞组件，从而帮助确定网络配置错误。有关更多信息，请参阅什么是 Reach ability Analyzer？ 在《亚马逊 VPC Reach ability Analyzer 指南》中。

   输入 App Runner ENI 作为源，输入 RDS ENI 作为目的地。

7. 如果您无法进一步缩小问题范围，或者在完成前面的步骤后仍然无法连接到 RDS 或下游服务，我们建议您联系 Su AWS pport 寻求进一步帮助。

应用程序运行器中的安全性

云安全 AWS 是重中之重。作为 AWS 客户，您可以受益于专为满足大多数安全敏感型组织的要求而构建的数据中心和网络架构。

安全是双方共同承担 AWS 的责任。责任共担模式将其描述为云的安全性和云中的安全性：

- 云安全 — AWS 负责保护在云中运行 AWS 服务的基础架构 AWS Cloud。 AWS 还为您提供可以安全使用的服务。作为AWS 合规计划合规计划合规计划合的一部分，第三方审计师定期测试和验证我们安全的有效性。要了解适用的合规计划 AWS App Runner，请参阅按合规计划划分的范围内的AWSAWS 服务按合规计划。
- 云端安全-您的责任由您使用的 AWS 服务决定。您还需要对其他因素负责，包括您的数据的敏感性、您公司的要求以及适用的法律法规。

本文档可帮助您了解在使用 App Runner 时如何应用责任共担模型。以下主题向您介绍如何配置 App Runner 以满足您的安全和合规性目标。您还将学习如何使用其他 AWS 服务来帮助您监控和保护您的 App Runner 资源。

主题

- App Runner 中的数据保护
- App Runner 的身份和访问管理
- 在 App Runner 中记录和监控
- App Runner 合规性验证
- 应用运行器中的弹性
- 中的基础设施安全 AWS App Runner
- 在 VPC 终端节点上使用应用程序运行器
- App Runner 中的配置和漏洞分析
- App Runner 的安全最佳实践

App Runner 中的数据保护

分 AWS 担责任模型适用于中的数据保护 AWS App Runner。如本模型所述 AWS ，负责保护运行所有内容的全球基础架构 AWS Cloud。您负责维护对托管在此基础设施上的内容的控制。您还负责您所使

用的 AWS 服务 的安全配置和管理任务。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 安全性博客 上的 AWS 责任共担模式和 GDPR 博客文章。

出于数据保护目的，我们建议您保护 AWS 账户 凭证并使用 AWS IAM Identity Center 或 AWS Identity and Access Management (IAM) 设置个人用户。这样，每个用户只获得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数据：

- 对每个账户使用多重身份验证 (MFA)。
- 使用 SSL/TLS 与资源通信。 AWS 我们要求使用 TLS 1.2，建议使用 TLS 1.3。
- 使用设置 API 和用户活动日志 AWS CloudTrail。
- 使用 AWS 加密解决方案以及其中的所有默认安全控件 AWS 服务。
- 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 中的敏感数据。
- 如果您在 AWS 通过命令行界面或 API 进行访问时需要经过 FIPS 140-2 验证的加密模块，请使用 FIPS 端点。有关可用的 FIPS 端点的更多信息，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机密信息或敏感信息（如您客户的电子邮件地址）放入标签或自由格式文本字段（如名称字段）。这包括当你 AWS 服务 使用控制台、API 或 AWS SDK 与 App Runner 或其他人合作时。 AWS CLI在用于名称的标签或自由格式文本字段中输入的任何数据都可能会用于计费或诊断日志。如果您向外部服务器提供网址，强烈建议您不要在网址中包含凭证信息来验证对该服务器的请求。

有关其他 App Runner 安全主题，请参阅安全性。

主题

使用加密保护数据

AWS App Runner 从您指定的存储库中读取您的应用程序源（源图像或源代码），并将其存储以部署到您的服务中。有关更多信息，请参阅 架构和概念。

数据保护是指保护传输过程中（往返于 App Runner 时）和静态数据（存储在 AWS 数据中心时）的数据。

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 App 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实现传输中的数据保护：使用传输层安全 (TLS) 对连接进行加密，或者使用客户端加密（在发送对象之前对其进行加密）。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，每当您的应用程序、其开发人员和管理员以及其最终用户发送或接收任何对象时，都要使用 TLS 对其进行加密。App Runner 将您的应用程序设置为通过 TLS 接收流量。

客户端加密不是保护您提供给 App Runner 进行部署的源图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，请务必保护您的开发或部署环境与 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 会对应用程序源映像或源包的所有存储副本进行加密。创建 App Runner 服务时，可以提供 AWS KMS key。如果您提供了一个密钥，App Runner 将使用您提供的密钥来加密您的源代码。如果你不提供一个，App Runner 会 AWS 托管式密钥 改用。

有关 App Runner 服务创建参数的详细信息，请参阅CreateService。有关 AWS Key Management Service (AWS KMS) 的信息，请参阅《AWS Key Management Service 开发者指南》。

互联网络流量隐私保护

App Runner 使用 Amazon Virtual Private Cloud（亚马逊 VPC）在 App Runner 应用程序中的资源之间创建边界，并控制这些资源、您的本地网络和互联网之间的流量。有关亚马逊 VPC 安全的更多信息，请参阅亚马逊 VPC 用户指南中的亚马逊 VPC 中的互联网流量隐私。

有关将您的 App Runner 应用程序与自定义 Amazon VPC 关联的信息，请参阅the section called “传出流量”。

有关使用 VPC 终端节点保护向 App Runner 发出的请求的信息，请参阅the section called “VPC 端点”。

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 App Runner 安全主题，请参阅安全性。

App Runner 的身份和访问管理

AWS Identity and Access Management (IAM) AWS 服务 可帮助管理员安全地控制对 AWS 资源的访问权限。IAM 管理员控制谁可以进行身份验证（登录）和授权（有权限）使用 App Runner 资源。您可以使用 IAM AWS 服务 ，无需支付额外费用。

有关其他 App Runner 安全主题，请参阅安全性。

主题

受众

您的使用方式 AWS Identity and Access Management (IAM) 会有所不同，具体取决于您在 App Runner 中所做的工作。

服务用户-如果您使用 App Runner 服务完成工作，则您的管理员会为您提供所需的凭据和权限。当你使用更多 App Runner 功能来完成工作时，你可能需要额外的权限。了解如何管理访问权限有助于您向管理员请求适合的权限。如果您无法访问 App Runner 中的某项功能，请参阅对 App Runner 的身份和访问权限。

服务管理员-如果您负责公司的 App Runner 资源，则可能拥有对 App Runner 的完全访问权限。您的工作是确定您的服务用户应访问哪些 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。要详细了解贵公司如何在 App Runner 中使用 IAM，请参阅应用程序运行器如何与 IAM 配合使用。

IAM 管理员 — 如果您是 IAM 管理员，则可能需要详细了解如何编写策略来管理 App Runner 的访问权限。要查看您可以在 IAM 中使用的基于身份的 App Runner 策略示例，请参阅。App Runner 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 AWS 使用身份凭证登录的方式。您必须以 IAM 用户身份或通过担 AWS 账户根用户任 IAM 角色进行身份验证（登录 AWS）。

您可以使用通过身份源提供的凭据以 AWS 联合身份登录。 AWS IAM Identity Center （IAM Identity Center）用户、贵公司的单点登录身份验证以及您的 Google 或 Facebook 凭据就是联合身份的示例。当您以联合身份登录时，您的管理员以前使用 IAM 角色设置了身份联合验证。当你使用联合访问 AWS 时，你就是在间接扮演一个角色。

根据您的用户类型，您可以登录 AWS Management Console 或 AWS 访问门户。有关登录的更多信息 AWS，请参阅《AWS 登录 用户指南》中的如何登录到您 AWS 账户的。

如果您 AWS 以编程方式访问，则会 AWS 提供软件开发套件 (SDK) 和命令行接口 (CLI)，以便使用您的凭据对请求进行加密签名。如果您不使用 AWS 工具，则必须自己签署请求。有关使用推荐的方法自行签署请求的更多信息，请参阅 IAM 用户指南中的签署 AWS API 请求。

无论使用何种身份验证方法，您可能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《AWS IAM Identity Center 用户指南》中的 多重身份验证和《IAM 用户指南》中的 在 AWS中使用多重身份验证（MFA）。

AWS 账户 root 用户

创建时 AWS 账户，首先要有一个登录身份，该身份可以完全访问账户中的所有资源 AWS 服务 和资源。此身份被称为 AWS 账户 root 用户，使用您创建账户时使用的电子邮件地址和密码登录即可访问该身份。强烈建议您不要使用根用户执行日常任务。保护好根用户凭证，并使用这些凭证来执行仅根用户可以执行的任务。有关需要您以根用户身份登录的任务的完整列表，请参阅《IAM 用户指南》中的需要根用户凭证的任务。

IAM 用户和群组

I AM 用户是您 AWS 账户 内部对个人或应用程序具有特定权限的身份。在可能的情况下，我们建议使用临时凭证，而不是创建具有长期凭证（如密码和访问密钥）的 IAM 用户。但是，如果您有一些特定的使用场景需要长期凭证以及 IAM 用户，建议您轮换访问密钥。有关更多信息，请参阅《IAM 用户指南》中的 对于需要长期凭证的使用场景定期轮换访问密钥。

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《IAM 用户指南》中的何时创建 IAM 用户（而不是角色）。

IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的身份。它类似于 IAM 用户，但与特定人员不关联。您可以 AWS Management Console 通过切换角色在中临时担任 IAM 角色。您可以通过调用 AWS CLI 或 AWS API 操作或使用自定义 URL 来代入角色。有关使用角色的方法的更多信息，请参阅《IAM 用户指南》中的使用 IAM 角色。

具有临时凭证的 IAM 角色在以下情况下很有用：

- 联合用户访问 – 要向联合身份分配权限，请创建角色并为角色定义权限。当联合身份进行身份验证时，该身份将与角色相关联并被授予由此角色定义的权限。有关联合身份验证的角色的信息，请参阅《IAM 用户指南》中的为第三方身份提供商创建角色。如果您使用 IAM Identity Center，则需要配置权限集。为控制您的身份在进行身份验证后可以访问的内容，IAM Identity Center 将权限集与 IAM 中的角色相关联。有关权限集的信息，请参阅《AWS IAM Identity Center 用户指南》中的 权限集。
- 临时 IAM 用户权限 – IAM 用户可代入 IAM 用户或角色，以暂时获得针对特定任务的不同权限。
- 跨账户存取 – 您可以使用 IAM 角色以允许不同账户中的某个人（可信主体）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，对于某些资源 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 用户指南》中的 IAM 中的跨账户资源访问。
- 跨服务访问 — 有些 AWS 服务 使用其他 AWS 服务服务中的功能。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 Simple Storage Service (Amazon S3) 中存储对象。服务可能会使用发出调用的主体的权限、使用服务角色或使用服务相关角色来执行此操作。
  - 转发访问会话 (FAS) — 当您使用 IAM 用户或角色在中执行操作时 AWS，您被视为委托人。使用某些服务时，您可能会执行一个操作，然后此操作在其他服务中启动另一个操作。FAS 使用调用委托人的权限以及 AWS 服务 向下游服务发出请求的请求。 AWS 服务只有当服务收到需要与其他 AWS 服务 或资源交互才能完成的请求时，才会发出 FAS 请求。在这种情况下，您必须具有执行这两个操作的权限。有关发出 FAS 请求时的策略详情，请参阅转发访问会话。
  - 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分派的 IAM 角色。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《IAM 用户指南》中的创建向 AWS 服务委派权限的角色。

- 服务相关角色-服务相关角色是一种链接到的服务角色。 AWS 服务服务可以代入代表您执行操作的角色。服务相关角色出现在您的中 AWS 账户 ，并且归服务所有。IAM 管理员可以查看但不能编辑服务相关角色的权限。
- 在 A@@ mazon EC2 上运行的应用程序 — 您可以使用 IAM 角色管理在 EC2 实例上运行并发出 AWS CLI 或 AWS API 请求的应用程序的临时证书。这优先于在 EC2 实例中存储访问密钥。要向 EC2 实例分配 AWS 角色并使其可供其所有应用程序使用，您需要创建附加到该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《IAM 用户指南》中的使用 IAM 角色为 Amazon EC2 实例上运行的应用程序授予权限。

要了解是使用 IAM 角色还是 IAM 用户，请参阅IAM 用户指南中的何时创建 IAM 角色（而不是用户）。

使用策略管理访问

您可以 AWS 通过创建策略并将其附加到 AWS 身份或资源来控制中的访问权限。策略是其中的一个对象 AWS ，当与身份或资源关联时，它会定义其权限。 AWS 在委托人（用户、root 用户或角色会话）发出请求时评估这些策略。策略中的权限确定是允许还是拒绝请求。大多数策略都以 JSON 文档的 AWS 形式存储在中。有关 JSON 策略文档的结构和内容的更多信息，请参阅 IAM 用户指南中的 JSON 策略概览。

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

默认情况下，用户和角色没有权限。要授予用户对所需资源执行操作的权限，IAM 管理员可以创建 IAM 策略。管理员随后可以向角色添加 IAM policy，用户可以代入角色。

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。拥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《IAM 用户指南》中的创建 IAM policy。

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。托管策略是独立的策略，您可以将其附加到中的多个用户、群组和角色 AWS 账户。托管策略包括 AWS

托管策略和客户托管策略。要了解如何在托管式策略和内联策略之间进行选择，请参阅 IAM 用户指南中的在托管式策略与内联策略之间进行选择。

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，策略定义指定主体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主体。委托人可以包括账户、用户、角色、联合用户或 AWS 服务。

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

访问控制列表 (ACL)

访问控制列表（ACL）控制哪些主体（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 S3 和 Amazon VPC 就是支持 ACL 的服务示例。 AWS WAF要了解有关 ACL 的更多信息，请参阅《Amazon Simple Storage Service 开发人员指南》中的访问控制列表（ACL）概览。

其他策略类型

AWS 支持其他不太常见的策略类型。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特征，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 用户指南》中的 IAM 实体的权限边界。
- 服务控制策略 (SCP)-SCP 是 JSON 策略，用于指定组织或组织单位 (OU) 的最大权限。 AWS Organizations AWS Organizations 是一项用于对您的企业拥有的多 AWS 账户 项进行分组和集中管理的服务。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 限制成员账户中的实体（包括每个 AWS 账户根用户实体）的权限。有关 Organizations 和 SCP 的更多信息，请参阅《AWS Organizations 用户指南》中的 SCP 的工作原理。
- 会话策略 – 会话策略是当您以编程方式为角色或联合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《IAM 用户指南》中的会话策略。

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。要了解在涉及多种策略类型时如何 AWS 确定是否允许请求，请参阅 IAM 用户指南中的策略评估逻辑。

应用程序运行器如何与 IAM 配合使用

在使用 IAM 管理访问权限之前 AWS App Runner，您应该了解哪些可用于 App Runner 的 IAM 功能。要全面了解 App Runner 和其他 AWS 服务如何与 IAM 配合使用，请参阅 IAM 用户指南中的与 IAM 配合使用的AWS 服务。

有关其他 App Runner 安全主题，请参阅安全性。

主题

- App Runner 基于身份的政策
- App Runner 基于资源的策略
- 基于 App Runner 标签的授权
- App Runner 用户权限
- 应用程序运行器 IAM 角色

App Runner 基于身份的政策

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 用户指南》 中的 IAM JSON 策略元素参考。

操作

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

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。还有一些操作需要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关联操作的权限。

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 `RunInstances` API 操作运行 Amazon EC2 实例的权限，您应将 `ec2:RunInstances` 操作纳入其策略。策略语句必须包含 `Action` 或 `NotAction` 元素。App Runner 定义了自己的一组操作，这些操作描述了您可以使用此服务执行的任务。

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，包括以下操作：

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

要查看 App Runner 操作列表，请参阅《服务授权参考》 AWS App Runner中定义的操作。

资源

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体 可以对什么资源执行操作，以及在什么条件下执行。

`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": "*"
```

App Runner 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，请参阅中的 Amazon 资源名称 (ARN) 和 AWS 服务命名空间。AWS 一般参考

例如，要在语句中指定my-service服务，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/*"
```

某些 App Runner 操作（例如用于创建资源的操作）无法对特定资源执行。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，请参阅服务授权参考 AWS App Runner中定义的资源。要了解可以在哪些操作中指定每个资源的 ARN，请参阅 AWS App Runner定义的操作。

条件键

管理员可以使用 AWS JSON 策略来指定谁有权访问什么。也就是说，哪个主体可以对什么资源执行操作，以及在什么条件下执行。

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，则使用逻辑OR运算来 AWS 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 用户指南》中的 IAM policy 元素：变量和标签。

AWS 支持全局条件密钥和特定于服务的条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

App Runner 支持使用一些全局条件键。要查看所有 AWS 全局条件键，请参阅 IAM 用户指南中的AWS 全局条件上下文密钥。

App Runner 定义了一组特定于服务的条件键。此外，App Runner 支持基于标签的访问控制，该控制是使用条件键实现的。有关更多信息，请参阅 the section called “基于 App Runner 标签的授权”。

要查看 App Runner 条件键列表，请参阅《服务授权参考》 AWS App Runner中的条件密钥。要了解可以使用条件键的操作和资源，请参阅由定义的操作 AWS App Runner。

示例

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

App Runner 基于资源的策略

App Runner 不支持基于资源的策略。

基于 App Runner 标签的授权

您可以将标签附加到 App Runner 资源，也可以在请求中将标签传递给 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。有关为 App Runner 资源添加标签的更多信息，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，请参阅根据标签控制对 App Runner 服务的访问权限。

App Runner 用户权限

要使用 App Runner，IAM 用户需要访问应用程序运行器操作的权限。向用户授予权限的常用方法是向 IAM 用户或群组附加策略。有关管理用户权限的更多信息，请参阅 IAM 用户指南中的更改 IAM 用户的权限。

App Runner 提供了两个托管策略，您可以将其附加到您的用户。

- `AWSAppRunnerReadOnlyAccess`— 授予列出和查看有关 App Runner 资源详细信息的权限。
- `AWSAppRunnerFullAccess`— 授予所有 App Runner 操作的权限。

要更精细地控制用户权限，您可以创建自定义策略并将其附加到您的用户。有关详细信息，请参阅 IAM 用户指南中的创建 IAM 策略。

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",
        "arn:aws:iam::*:role/aws-service-role/networking.apprunner.amazonaws.com/
AWSServiceRoleForAppRunnerNetworking"
      ],
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": [
            "apprunner.amazonaws.com",
            "networking.apprunner.amazonaws.com"
          ]
        }
      }
    },
    {
      "Effect": "Allow",
```

```
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringLike": {
          "iam:PassedToService": "apprunner.amazonaws.com"
        }
      }
    },
    {
      "Sid": "AppRunnerAdminAccess",
      "Effect": "Allow",
      "Action": "apprunner:*",
      "Resource": "*"
    }
  ]
}
```

应用程序运行器 IAM 角色

I AM 角色是您内部具有特定权限 AWS 账户 的实体。

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 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。

App Runner 支持与服务相关的角色。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担任服务角色。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 IAM 账户中，并归该账户所有。这意味着 IAM 用户可以更改此角色的权限。但是，这样做可能会中断服务的功能。

App Runner 支持几个服务角色。

访问角色

访问角色是 App Runner 用于访问您账户中亚马逊弹性容器注册表 (Amazon ECR) Container Registry 中的图像的角色。访问亚马逊 ECR 中的图片是必需的，而亚马逊 ECR Public 则不是必需的。在 Amazon ECR 中基于图像创建服务之前，请使用 IAM 创建服务角色并在其中使

用AWSAppRunnerServicePolicyForECRAccess托管策略。然后，当您在SourceConfiguration参数的AuthenticationConfiguration成员中调用 CreateServiceAPI 或使用 App Runner 控制台创建服务时，您可以将此角色传递给 App Runner。

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了自己的自定义策略，请务必"Resource": "*"为ecr:GetAuthorizationToken操作指定。令牌可用于访问您有权访问的任何 Amazon ECR 注册表。

创建访问角色时，请务必添加一个将 App Runner 服务主体声明build.apprunner.amazonaws.com为可信实体的信任策略。

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。控制台还会列出您账户中的其他角色，您可以根据需要选择其他角色。

实例角色

实例角色是一个可选角色，App Runner 使用该角色为 AWS 服务计算实例所需的服务操作提供权限。如果您的应用程序代码调用 AWS 操作 (API)，则需要向 App Runner 提供实例角色。要么在您的实例角色中嵌入所需的权限，要么创建自己的自定义策略并将其用于实例角色。我们无法预测您的代码使用哪些调用。因此，我们不提供用于此目的的托管政策。

在创建 App Runner 服务之前，请使用 IAM 创建具有所需自定义或嵌入式策略的服务角色。然后，当您在InstanceConfiguration参数的`InstanceRoleArn`成员中调用 CreateServiceAPI 或使用 App Runner 控制台创建服务时，您可以将此角色作为实例角色传递给 App Runner。

创建实例角色时，请务必添加一个将 App Runner 服务委托人声明`tasks.apprunner.amazonaws.com`为可信实体的信任策略。

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，则控制台会列出您账户中的角色，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 “创建”。

App Runner 基于身份的策略示例

默认情况下，IAM 用户和角色无权创建或修改 AWS App Runner 资源。他们也无法使用 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 基于身份的策略，请参阅《IAM 用户指南》中的 在 JSON 选项卡上创建策略。

有关其他 App Runner 安全主题，请参阅安全性。

主题

- 策略最佳实践
- 用户策略
- 根据标签控制对 App Runner 服务的访问权限

策略最佳实践

基于身份的策略决定了某人是否可以在您的账户中创建、访问或删除 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，请遵循以下指南和建议：

- 开始使用 AWS 托管策略并转向最低权限权限 — 要开始向用户和工作负载授予权限，请使用为许多常见用例授予权限的AWS 托管策略。它们在你的版本中可用 AWS 账户。我们建议您通过定义针对您的用例的 AWS 客户托管策略来进一步减少权限。有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管策略 或 工作职能的AWS 托管策略。
- 应用最低权限 – 在使用 IAM policy 设置权限时，请仅授予执行任务所需的权限。为此，您可以定义在特定条件下可以对特定资源执行的操作，也称为最低权限许可。有关使用 IAM 应用权限的更多信息，请参阅《IAM 用户指南》中的 IAM 中的策略和权限。
- 使用 IAM policy 中的条件进一步限制访问权限 – 您可以向策略添加条件来限制对操作和资源的访问。例如，您可以编写策略条件来指定必须使用 SSL 发送所有请求。如果服务操作是通过特定的方式使用的，则也可以使用条件来授予对服务操作的访问权限 AWS 服务，例如 AWS CloudFormation。有关更多信息，请参阅《IAM 用户指南》中的 IAM JSON 策略元素：条件 。
- 使用 IAM Access Analyzer 验证您的 IAM policy，以确保权限的安全性和功能性 – IAM Access Analyzer 会验证新策略和现有策略，以确保策略符合 IAM policy语言 (JSON) 和 IAM 最佳实

践。IAM Access Analyzer 提供 100 多项策略检查和可操作的建议，以帮助您制定安全且功能性强的策略。有关更多信息，请参阅《IAM 用户指南》中的 IAM Acess Analyzer 策略验证。

- 需要多重身份验证 (MFA)-如果 AWS 账户您的场景需要 IAM 用户或根用户，请启用 MFA 以提高安全性。若要在调用 API 操作时需要 MFA，请将 MFA 条件添加到您的策略中。有关更多信息，请参阅《IAM 用户指南》中的 配置受 MFA 保护的 API 访问。

有关 IAM 中的最佳实操的更多信息，请参阅《IAM 用户指南》中的 IAM 中的安全最佳实操。

用户策略

要访问 App Runner 控制台，IAM 用户必须拥有一组最低权限。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。如果您创建的基于身份的策略比所需的最低权限更严格，则控制台将无法按预期运行，供拥有该策略的用户使用。

App Runner 提供了两个托管策略，您可以将其附加到您的用户。

- `AWSAppRunnerReadOnlyAccess`— 授予列出和查看有关 App Runner 资源详细信息的权限。
- `AWSAppRunnerFullAccess`— 授予所有 App Runner 操作的权限。

为确保用户可以使用 App Runner 控制台，请至少向用户附加`AWSAppRunnerReadOnlyAccess`托管策略。您可以改为附加`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《IAM 用户指南》中的为用户添加权限。

对于仅调用 AWS CLI 或 AWS API 的用户，您无需为其设置最低控制台权限。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。

以下示例演示了自定义用户策略。您可以将它们用作定义自己的自定义用户策略的起点。复制示例，和/或移除操作，缩小资源范围，然后添加条件。

示例：控制台和连接管理用户策略

此示例策略允许访问控制台，并允许创建和管理连接。它不允许创建和管理 App Runner 服务。它可以附加到角色为管理 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：使用条件键的用户策略

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，但拒绝使用名为的连接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preprod仅命名为 auto Scaling 配置的 App Runner 服务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/*"
        }
      }
    }
  ]
}
```

根据标签控制对 App Runner 服务的访问权限

您可以使用基于身份的策略中的条件根据标签控制对 App Runner 资源的访问权限。此示例说明如何创建允许删除 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。如果名为的用户`richard-roe`尝试删除 App Runner 服务，则必须对该服务进行标记`Owner=richard-roe`或`owner=richard-roe`。否则，将拒绝其访

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 用户指南 中的 IAM JSON 策略元素：条件。

为 App Runner 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的 IAM 角色，直接关联到 App Runner。服务相关角色由 App Runner 预定义，包括该服务代表您调用其他 AWS 服务所需的所有权限。

主题

- 使用角色进行管理
- 使用角色进行联网

使用角色进行管理

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的 IAM 角色，直接关联到 App Runner。服务相关角色由 App Runner 预定义，包括该服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可以更轻松地设置 App Runner，因为您不必手动添加必要的权限。App Runner 定义了其服务相关角色的权限，除非另有定义，否则只有 App Runner 可以担任其角色。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，您才能删除服务相关角色。这样可以保护您的 App Runner 资源，因为您不会无意中删除访问这些资源的权限。

有关支持服务相关角色的其它服务的信息，请参阅使用 IAM 的AWS 服务并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

App Runner 的服务相关角色权限

App Runner 使用名为AWSServiceRoleForAppRunner的服务相关角色。

该角色允许 App Runner 执行以下任务：

- 将日志推送到 Amazon CloudWatch 日志组。
- 创建亚马逊 CloudWatch 事件规则以订阅亚马逊弹性容器注册表 (Amazon ECR) Container Registry 图片推送。
- 将追踪信息发送到 AWS X-Ray。

AWSServiceRoleForAppRunner 服务相关角色信任以下服务来代入该角色：

- apprunner.amazonaws.com

AWSServiceRoleForAppRunner 服务相关角色的权限策略包含 App Runner 代表您完成操作所需的所有权限。

AppRunnerServiceRolePolicy 托管策略

```
{
  "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:*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:PutRule",
        "events:PutTargets",
        "events:DeleteRule",
        "events:RemoveTargets",
        "events:DescribeRule",
        "events:EnableRule",
        "events:DisableRule"
      ],
      "Resource": "arn:aws:events:*:*:rule/AWSAppRunnerManagedRule*"
```

```
    }
  ]
}
```

X-ray 追踪政策

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "xray:PutTraceSegments",
        "xray:PutTelemetryRecords",
        "xray:GetSamplingRules",
        "xray:GetSamplingTargets",
        "xray:GetSamplingStatisticSummaries"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

您必须配置权限，允许 IAM 实体（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 App Runner 创建服务相关角色

您无需手动创建服务相关角色。当你在 AWS Management Console、或 AWS API 中创建 App Runner 服务时 AWS CLI，App Runner 会为你创建与服务相关的角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。在创建 App Runner 服务时，App Runner 会再次为您创建与服务相关的角色。

编辑 App Runner 的服务相关角色

App Runner 不允许您编辑 AWSServiceRoleForAppRunner 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。但是可以使用 IAM 编辑角色描述。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 App Runner 的服务相关角色

如果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，您必须先清除您的服务相关角色，然后才能手动删除它。

清除服务相关角色

必须先删除服务相关角色使用的所有资源，然后才能使用 IAM 删除该角色。

在 App Runner 中，这意味着要删除你账户中的所有 App Runner 服务。要了解有关删除 App Runner 服务的信息，请参阅the section called “删除”。

> **Note**
>
> 如果您尝试删除资源时 App Runner 服务正在使用该角色，则删除可能会失败。如果发生这种情况，请等待几分钟后重试。

手动删除 服务相关角色

使用 IAM 控制台 AWS CLI、或 AWS API 删除 AWSServiceRoleForAppRunner 服务相关角色。有关更多信息，请参见《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区域

App Runner 支持在提供服务的所有地区使用服务相关角色。有关更多信息，请参阅《AWS 一般参考》中的 AWS App Runner 端点和配额。

使用角色进行联网

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的 IAM 角色，直接关联到 App Runner。服务相关角色由 App Runner 预定义，包括该服务代表您调用其他 AWS 服务所需的所有权限。

服务相关角色可以更轻松地设置 App Runner，因为您不必手动添加必要的权限。App Runner 定义了其服务相关角色的权限，除非另有定义，否则只有 App Runner 可以担任其角色。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，您才能删除服务相关角色。这样可以保护您的 App Runner 资源，因为您不会无意中删除访问这些资源的权限。

有关支持服务相关角色的其它服务的信息，请参阅使用 IAM 的AWS 服务并查找服务相关角色列中显示为是的服务。选择是和链接，查看该服务的服务相关角色文档。

App Runner 的服务相关角色权限

App Runner 使用名为AWSServiceRoleForAppRunnerNetworking的服务相关角色。

该角色允许 App Runner 执行以下任务：

- 将 VPC 附加到您的 App Runner 服务并管理网络接口。

AWSServiceRoleForAppRunnerNetworking 服务相关角色信任以下服务来代入该角色：

- networking.apprunner.amazonaws.com

名为的角色权限策略 AppRunnerNetworkingServiceRolePolicy 包含 App Runner 代表您完成操作所需的所有权限。

AppRunnerNetworkingServiceRolePolicy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ec2:CreateNetworkInterface",
      "Resource": "*",
      "Condition": {
        "ForAllValues:StringEquals": {
          "aws:TagKeys": [
            "AWSAppRunnerManaged"
          ]
        }
      }
```

```
    },
    {
      "Effect": "Allow",
      "Action": "ec2:CreateTags",
      "Resource": "arn:aws:ec2:*:*:network-interface/*",
      "Condition": {
        "StringEquals": {
          "ec2:CreateAction": "CreateNetworkInterface"
        },
        "StringLike": {
          "aws:RequestTag/AWSAppRunnerManaged": "*"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": "ec2:DeleteNetworkInterface",
      "Resource": "*",
      "Condition": {
        "Null": {
          "ec2:ResourceTag/AWSAppRunnerManaged": "false"
        }
      }
    }
  ]
}
```

您必须配置权限，允许 IAM 实体（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 App Runner 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、或 AWS API 中创建 VPC 连接器时 AWS CLI，App Runner 会为您创建服务相关角色。

如果您删除该服务相关角色，然后需要再次创建，您可以使用相同流程在账户中重新创建此角色。当您创建 VPC 连接器时，App Runner 会再次为您创建服务相关角色。

编辑 App Runner 的服务相关角色

App Runner 不允许您编辑 AWSServiceRoleForAppRunnerNetworking 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。但是可以使用 IAM 编辑角色描述。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 App Runner 的服务相关角色

如果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，您必须先清除您的服务相关角色，然后才能手动删除它。

清除服务相关角色

必须先删除服务相关角色使用的所有资源，然后才能使用 IAM 删除该角色。

在 App Runner 中，这意味着解除 VPC 连接器与您账户中所有 App Runner 服务的关联，并删除 VPC 连接器。有关更多信息，请参阅 the section called “传出流量”。

> **Note**
>
> 如果您尝试删除资源时 App Runner 服务正在使用该角色，则删除可能会失败。如果发生这种情况，请等待几分钟后重试。

手动删除服务相关角色

使用 IAM 控制台 AWS CLI、或 AWS API 删除 AWSServiceRoleForAppRunnerNetworking 服务相关角色。有关更多信息，请参见《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区域

App Runner 支持在提供服务的所有地区使用服务相关角色。有关更多信息，请参阅《AWS 一般参考》中的 AWS App Runner 端点和配额。

AWS 的托管策略 AWS App Runner

AWS 托管策略是由创建和管理的独立策略 AWS。 AWS 托管策略旨在为许多常见用例提供权限，以便您可以开始为用户、组和角色分配权限。

请记住， AWS 托管策略可能不会为您的特定用例授予最低权限权限，因为它们可供所有 AWS 客户使用。我们建议通过定义特定于您的使用场景的客户管理型策略来进一步减少权限。

您无法更改 AWS 托管策略中定义的权限。如果 AWS 更新 AWS 托管策略中定义的权限，则更新会影响该策略所关联的所有委托人身份（用户、组和角色）。 AWS 最有可能在启动新的 API 或现有服务可以使用新 AWS 服务 的 API 操作时更新 AWS 托管策略。

有关更多信息，请参阅《IAM 用户指南》中的 AWS 托管式策略。

App Runner 对 AWS 托管政策的更新

查看自该服务开始跟踪这些更改以来 App Runner AWS 托管政策更新的详细信息。要获得有关此页面变更的自动提醒，请在 App Runner 文档历史记录页面上订阅 RSS 提要。

更改	描述	日期
AWSAppRunnerReadOnlyAccess - 新策略	App Runner 添加了一项新政策，允许用户列出和查看有关 App Runner 资源的详细信息。	2022年2月24日
AWSAppRunnerFullAccess – 更新了现有策略	App Runner 更新了该`iam:CreateServiceLinkedRole` 操作的资源列表，以允许创建`AWSServiceRoleForAppRunnerNetworking` 服务相关角色。	2022年2月8日
AppRunnerNetworkingServiceRolePolicy：新策略	App Runner 添加了一项新政策，允许 App Runner 调用 Amazon Virtual Private Cloud，将 VPC 连接到你的 App Runner 服务并代表 App Runner 服务管理网络接口。该策略用于`AWSServiceRoleForAppRunnerNetworking` 服务相关角色。	2022年2月8日
AWSAppRunnerFullAccess：新策略	App Runner 添加了一项新政策，允许用户执行所有 App Runner 操作。	2022年1月10日
AppRunnerServiceRolePolicy：新策略	App Runner 添加了一项新政策，允许 App Runner 代表 App Runner 服	2021年3月1日

更改	描述	日期
	务调用 Amazon CloudWatch Logs 和 Amazon Ev CloudWatch ents。该策略用于`AWSServiceRoleForAppRunner` 服务相关角色。	
AWSAppRunnerServicePolicyForECRAccess：新策略	App Runner 添加了一项新政策，允许 App Runner 访问你账户中的亚马逊 Elastic Container Registry (Amazon ECR) 镜像。	2021年3月1日
App Runner 开始跟踪更改	App Runner 开始跟踪其 AWS 托管策略的更改。	2021年3月1日

对 App Runner 的身份和访问权限

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

有关其他 App Runner 安全主题，请参阅安全性。

主题

- 我无权在 App Runner 中执行操作
- 我想允许我以外的人访问我 AWS 账户 的 App Runner 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项操作，请联系您的管理员寻求帮助。您的管理员是向您提供 AWS 登录凭证的人。

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

```
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 会要求管理员更新她的策略，以允许她使用`apprunner:DescribeService`操作访问*`my-example-service`*资源。

我想允许我以外的人访问我 AWS 账户 的 App Runner 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，可以担任角色。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器如何与 IAM 配合使用。
- 要了解如何提供对您拥有的资源的访问权限 AWS 账户 ，请参阅 IAM 用户指南中的向您拥有 AWS 账户 的另一个 IAM 用户提供访问权限。
- 要了解如何向第三方提供对您的资源的访问权限 AWS 账户，请参阅 IAM 用户指南中的向第三方提供访问权限。 AWS 账户
- 要了解如何通过联合身份验证提供访问权限，请参阅《IAM 用户指南》中的为经过外部身份验证的用户（联合身份验证）提供访问权限。
- 要了解使用角色和基于资源的策略进行跨账户访问的区别，请参阅 IA M 用户指南中的跨账户资源访问。

在 App Runner 中记录和监控

监控是维护 AWS App Runner 服务可靠性、可用性和性能的重要组成部分。从 AWS 解决方案的各个部分收集监控数据可以让您在出现故障时更轻松地进行调试。App Runner 集成了多种 AWS 工具，用于监控您的 App Runner 服务和响应潜在事件。

亚马逊 CloudWatch 警报

借助 Amazon CloudWatch 警报，您可以查看您指定的时间段内的服务指标。如果该指标在给定时间段内超过给定阈值，则您会收到通知。

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

应用程序日志

App Runner 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。此输出中的内容由您决定。例如，您可以包括向 Web 服务发出的请求的详细记录。事实证明，这些日志记录在安全和访问审计中可能很有用。有关更多信息，请参阅 日志（CloudWatch 日志）。

AWS CloudTrail 操作日志

App Runner 与 AWS CloudTrail一项服务集成，该服务提供用户、角色或 AWS 服务在 App Runner 中执行的操作的记录。 CloudTrail 将 App Runner 的所有 API 调用捕获为事件。您可以在 CloudTrail 控制台中查看最新事件，也可以创建跟踪以允许将 CloudTrail 事件持续传送到亚马逊简单存储服务 (Amazon S3) 存储桶。有关更多信息，请参阅 API 操作 (CloudTrail)。

App Runner 合规性验证

AWS App Runner 作为多个合规计划的一部分，第三方审计师对安全性和 AWS 合规性进行评估。其中包括 SOC、PCI、FedRAMP、HIPAA 及其他。

要了解是否属于特定合规计划的范围，请参阅AWS 服务 “按合规计划划分的范围” ”，然后选择您感兴趣的合规计划。 AWS 服务 有关一般信息，请参阅AWS 合规计划AWS。

您可以使用下载第三方审计报告 AWS Artifact。有关更多信息，请参阅中的 “下载报告” 中的 “ AWS Artifact。

您在使用 AWS 服务 时的合规责任取决于您的数据的敏感性、贵公司的合规目标以及适用的法律和法规。 AWS 提供了以下资源来帮助实现合规性：

- 安全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，并提供了部署以安全性和合规性为重点 AWS 的基准环境的步骤。
- 在 A@@ mazon Web Services 上构建 HIPAA 安全与合规架构 — 本白皮书描述了各公司如何使用 AWS 来创建符合 HIPAA 资格的应用程序。

> **Note**
>
> 并非所有 AWS 服务 人都符合 HIPAA 资格。有关更多信息，请参阅符合 HIPAA 要求的服务参考。

- AWS 合规资源AWS — 此工作簿和指南集可能适用于您所在的行业和所在地区。
- AWS 客户合规指南 — 从合规角度了解责任共担模式。这些指南总结了保护的最佳实践， AWS 服务并将指南映射到跨多个框架（包括美国国家标准与技术研究院 (NIST)、支付卡行业安全标准委员会 (PCI) 和国际标准化组织 (ISO)）的安全控制。
- 使用AWS Config 开发人员指南中的规则评估资源 — 该 AWS Config 服务评估您的资源配置在多大程度上符合内部实践、行业准则和法规。

- AWS Security Hub— 这 AWS 服务 提供了您内部安全状态的全面视图 AWS。Security Hub 通过安全控件评估您的 AWS 资源并检查其是否符合安全行业标准和最佳实践。有关受支持服务及控件的列表，请参阅 Security Hub 控件参考。
- Amazon GuardDuty — 它通过监控您的 AWS 账户环境中是否存在可疑和恶意活动，来 AWS 服务检测您的工作负载、容器和数据面临的潜在威胁。 GuardDuty 通过满足某些合规性框架规定的入侵检测要求，可以帮助您满足各种合规性要求，例如 PCI DSS。
- AWS Audit Manager— 这 AWS 服务 可以帮助您持续审计 AWS 使用情况，从而简化风险管理以及对法规和行业标准的合规性。

有关其他 App Runner 安全主题，请参阅安全性。

应用运行器中的弹性

AWS 全球基础设施是围绕 AWS 区域 可用区构建的。 AWS 区域 提供多个物理隔离和隔离的可用区，这些可用区通过低延迟、高吞吐量和高度冗余的网络连接。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 AWS 区域 和可用区的更多信息，请参阅AWS 全球基础设施。

AWS App Runner 代表您管理和自动使用 AWS 全球基础架构。使用 App Runner 时，您可以从 AWS 提供的可用性和容错机制中受益。

有关其他 App Runner 安全主题，请参阅安全性。

中的基础设施安全 AWS App Runner

作为一项托管服务， AWS App Runner 受到《Amazon Web Services：安全流程概述》白皮书中描述的 AWS 全球网络安全程序的保护。

您可以使用 AWS 已发布的 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）1.2 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。

在 VPC 终端节点上使用应用程序运行器

您的 AWS 应用程序可能会将 AWS App Runner 服务与在亚马逊虚拟私 AWS 服务 有云（亚马逊 VPC）的 VPC 中运行的其他服务集成。您的部分应用程序可能会从 VPC 内部向 App Runner 发出请求。例如，您可以使用持续部署 AWS CodePipeline 到您的 App Runner 服务。提高应用程序安全性的一种方法是通过 VPC 终端节点将这些 App Runner 请求（以及向其他用户发送请求 AWS 服务）。

使用 VPC 终端节点，您可以私密地将您的 VPC 连接到支持的 VPC 终端节点服务 AWS 服务 以及由其提供支持的 VPC 终端节点服务 AWS PrivateLink。您不需要互联网网关、NAT 设备、VPN 连接或 AWS Direct Connect 连接。

您的 VPC 中的资源不使用公有 IP 地址与 App Runner 资源进行交互。您的 VPC 和 App Runner 之间的流量不会离开亚马逊网络。有关 VPC 终端节点的更多信息，请参阅AWS PrivateLink 指南中的 VPC 终端节点。

> **Note**
>
> 默认情况下，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 是公有的。这意味着它已连接到互联网。您可以选择将您的应用程序与自定义 VPC 关联。有关更多信息，请参阅 the section called “传出流量”。
>
> 即使您的服务已连接到 VPC，您也可以将服务配置为访问互联网，包括 AWS API。有关如何为 VPC 出站流量启用公共互联网访问的说明，请参阅the section called “选择子网时的注意事项”。
>
> App Runner 不支持为您的应用程序创建 VPC 终端节点。

为 App Runner 设置 VPC 终端节点

要在您的 VPC 中为 App Runner 服务创建接口 VPC 终端节点，请按照AWS PrivateLink 指南中的创建接口终端节点过程进行操作。对于 Service Name (服务名称)，选择 `com.amazonaws.region.apprunner`。

VPC 网络隐私注意事项

> **Important**
>
> 在 App Runner 上使用 VPC 终端节点并不能确保来自你的 VPC 的所有流量都不会进入互联网。VPC 可能是公共的。此外，您的解决方案的某些部分可能不使用 VPC 终端节点发出 AWS

API 调用。例如，AWS 服务 可能会使用其公共终端节点调用其他服务。如果您的 VPC 中的解决方案需要流量隐私，请阅读本节。

为确保您的 VPC 中网络流量的隐私，请考虑以下事项：

- 启用 DNS 名称 — 应用程序的某些部分仍可能使用`apprunner.region.amazonaws.com`公共终端节点通过互联网向 App Runner 发送请求。如果您的 VPC 配置了互联网访问权限，则这些请求会成功，而不会向您发出任何指示。您可以通过确保在创建终端节点时启用 “启用 DNS 名称” 来防止这种情况。默认情况下，它设置为 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向其他人发送请求 AWS 服务。例如，AWS CodePipeline 可能会向发送请求 AWS CodeBuild。为这些服务配置 VPC 终端节点，并在这些终端节点上启用 DNS 名称。
- 配置私有 VPC — 如果可能（如果您的解决方案根本不需要访问互联网），请将您的 VPC 设置为私有，这意味着它没有互联网连接。这样可以确保丢失的 VPC 终端节点会导致明显的错误，这样您就可以添加缺失的终端节点。

使用终端节点策略控制通过 VPC 终端节点进行的访问

App Runner 不支持 VPC 终端节点策略。默认情况下，允许通过接口端点对 App Runner 进行完全访问。或者，您可以将安全组与端点网络接口关联，以控制通过接口终端节点流向 App Runner 的流量。

与接口端点集成

App Runner 支持 AWS PrivateLink，它提供与 App Runner 的私有连接，并消除了互联网流量的暴露。要使您的应用程序能够使用向 App Runner 发送请求 AWS PrivateLink，请配置一种称为接口终端节点的 VPC 终端节点。有关更多信息，请参阅 AWS PrivateLink 指南中的接口 VPC 端点 (AWS PrivateLink)。

App Runner 中的配置和漏洞分析

AWS 而且，我们的客户有责任实现高水平的软件组件安全性和合规性。有关更多信息，请参阅责任 AWS 共担模型。

有关其他 App Runner 安全主题，请参阅安全性。

App Runner 的安全最佳实践

AWS App Runner 提供了多种安全功能，供您在制定和实施自己的安全策略时考虑。以下最佳实践是一般指导原则，并不代表完整安全解决方案。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 App 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

App Runner 为 IAM 用户和访问角色提供 AWS Identity and Access Management (IAM) 托管策略。这些托管策略指定了正确运行 App Runner 服务可能需要的所有权限。

您的应用程序可能不需要我们托管策略中的全部权限。您可以对其进行自定义，并仅授予您的用户和 App Runner 服务执行任务所需的权限。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

监控是维护 App Runner 解决方案的可靠性、安全性、可用性和性能的重要组成部分。 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：

- Amazon App Runner CloudWatch 指标 — 为关键的 App Runner 指标和应用程序的自定义指标设置警报。有关更多信息，请参阅 指标 (CloudWatch)。
- AWS CloudTrail 条目-跟踪可能影响可用性的操作，例如`PauseService`或`DeleteConnection`。有关详细信息，请参阅API 操作 (CloudTrail)。

AWS 词汇表

有关最新 AWS 术语，请参阅《AWS 词汇表 参考资料》中的AWS 词汇表。

本文属于机器翻译版本。若本译文内容与英语原文存在差异，则一律以英文原文为准。