aws

Developer Guide

AWS App Runner

AWS App Runner: Developer Guide

Table of Contents

What is AWS App Runner?

AWS App Runner is an AWS service that provides a fast, simple, and cost-effective way to deploy from source code or a container image directly to a scalable and secure web application in the AWS Cloud. You don't need to learn new technologies, decide which compute service to use, or know how to provision and configure AWS resources.

App Runner connects directly to your code or image repository. It provides an automatic integration and delivery pipeline with fully managed operations, high performance, scalability, and security.
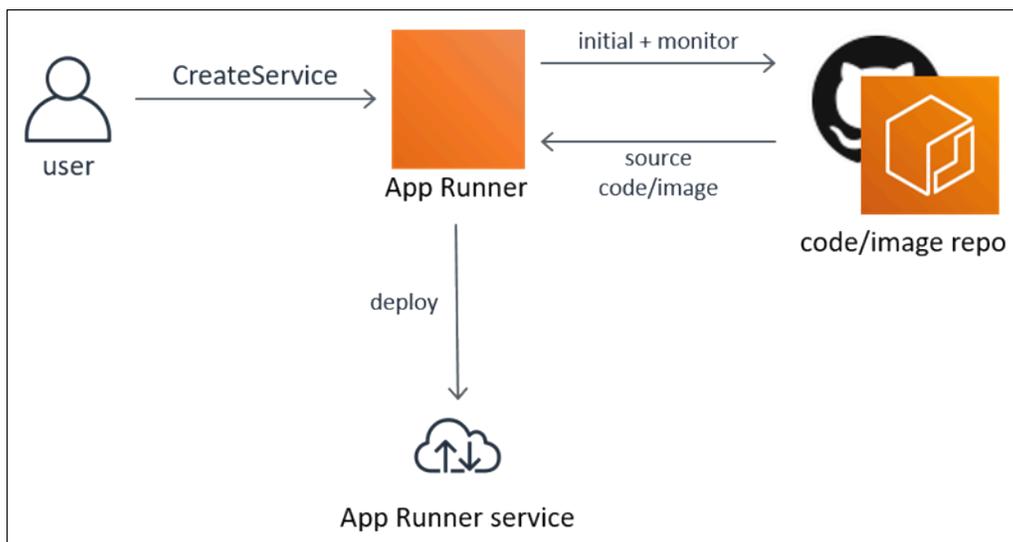
Who is App Runner for?

If you're a *developer*, you can use App Runner to simplify the process of deploying a new version of your code or image repository.

For *operations teams*, App Runner enables automatic deployments each time a commit is pushed to the code repository or a new container image version is pushed to the image repository.

Accessing App Runner

You can define and configure your App Runner service deployments using any one of the following interfaces:

- **App Runner console** – Provides a web interface for managing your App Runner services.

- **App Runner API** – Provides a RESTful API for performing App Runner actions. For more information, see AWS App Runner API Reference.

- **AWS Command Line Interface (AWS CLI)** – Provides commands for a broad set of AWS services, including Amazon VPC, and is supported on Windows, macOS, and Linux. For more information, see AWS Command Line Interface.

- **AWS SDKs** – Provides language-specific APIs and takes care of many of the connection details, such as calculating signatures, handling request retries, and error handling. For more information, see AWS SDKs.

Pricing for App Runner

App Runner provides a cost-effective way to run your application. You only pay for resources that your App Runner service consumes. Your service scales down to fewer compute instances when request traffic is lower. You have control over scalability settings: the lowest and highest number of provisioned instances, and the highest load an instance handles.

For more information about App Runner automatic scaling, see the section called "Auto scaling".

For pricing information, see AWS App Runner pricing.

What's next

Learn how to get started with App Runner in the following topics:

- *Setting up* – Complete the prerequisite steps for using App Runner.

- *Getting started* – Deploy your first application to App Runner.

Setting up for App Runner

If you're a new AWS customer, complete the setup prerequisites that are listed on this page before you start using AWS App Runner.

For these setup procedures, you use the AWS Identity and Access Management (IAM) service. For complete information about IAM, see the following reference materials:

- [AWS Identity and Access Management (IAM)](#)
- [IAM User Guide](#)

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform [tasks that require root user access](#).

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the [AWS Management Console](#) as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see [Signing in as the root user](#) in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see [Enable a virtual MFA device for your AWS account root user (console)](#) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see [Enabling AWS IAM Identity Center](#) in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see [Configure user access with the default IAM Identity Center directory](#) in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see [Signing in to the AWS access portal](#) in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

    For instructions, see [Add groups](#) in the *AWS IAM Identity Center User Guide*.

Grant programmatic access

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see [Login for AWS local development](#) in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see [Login for AWS local development](#) in the *AWS SDKs and Tools Reference Guide*.
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see [Configuring the AWS CLI to use AWS IAM Identity Center](#) in the *AWS Command Line Interface User Guide*.

Which user needs programmatic access?	To	By
		• For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

What's next

You completed the prerequisite steps. To deploy your first application to App Runner, see *Getting started*.

Getting started with App Runner

AWS App Runner is an AWS service that provides a fast, simple, and cost-effective way to turn an existing container image or source code directly into a running web service in the AWS Cloud.

This tutorial covers how you can use AWS App Runner to deploy your application to an App Runner service. It walks through configuring the source code and deployment, the service build, and the service runtime. It also shows how to deploy a code version, make a configuration change, and view logs. Last, the tutorial shows how to clean up the resources that you created while following the tutorial's procedures.

Topics

- [Prerequisites](#)
- [Step 1: Create an App Runner service](#)
- [Step 2: Change your service code](#)
- [Step 3: Make a configuration change](#)
- [Step 4: View logs for your service](#)
- [Step 5: Clean up](#)
- [What's next](#)

Prerequisites

Before you start the tutorial, be sure to take the following actions:

1. Complete the setup steps in *Setting up*.

2. Decide if you'd like to work with either a GitHub repository or a Bitbucket repository.

   - To work with a Bitbucket, first create a [Bitbucket](#) account, if you don't already have one. If you're new to Bitbucket, see [Getting started with Bitbucket](#) in the *Bitbucket Cloud Documentation*.

   - To work with GitHub, create a [GitHub](#) account, if you don't already have one. If you're new to GitHub, see [Getting started with GitHub](#) in the *GitHub Docs*.

> **ⓘ Note**
>
> You can create connections to multiple repository providers from your account. So if you'd like to walk through deploying from both a GitHub and a Bitbucket repository, you can repeat this procedure. The next time through create a new App Runner service and create a new account connection for the other repository provider.

3. Create a repository in your repository provider account. This tutorial uses the repository name `python-hello`. Create files in the root directory of the repository, with the names and content specified in the following examples.

Files for the `python-hello` example repository

Example requirements.txt

```
pyramid==2.0
```

Example server.py

```python
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()
```

Step 1: Create an App Runner service

In this step, you create an App Runner service based on the example source code repository that you created on GitHub or Bitbucket as part of the section called "Prerequisites". The example contains a simple Python website. These are the main steps you take to create a service:

1. Configure your source code.

2. Configure source deployment.

3. Configure application build.

4. Configure your service.

5. Review and confirm.

The following diagram outlines the steps for creating an App Runner service:
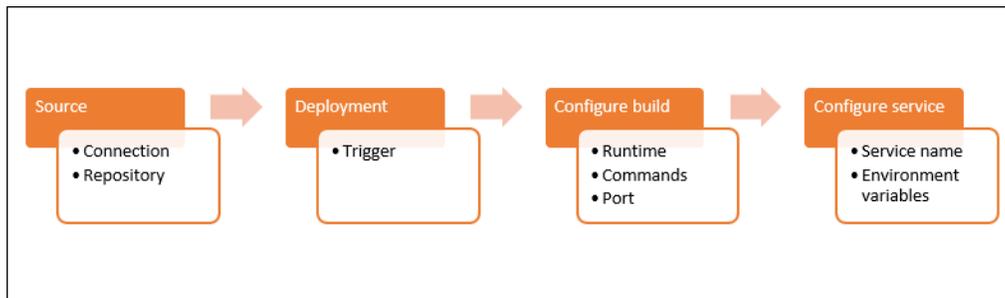
To create an App Runner service based on a source code repository

1.   Configure your source code.

   a.   Open the App Runner console, and in the **Regions** list, select your AWS Region.

   b.   If the AWS account doesn't have any App Runner services yet, the console home page is displayed. Choose **Create an App Runner service**.
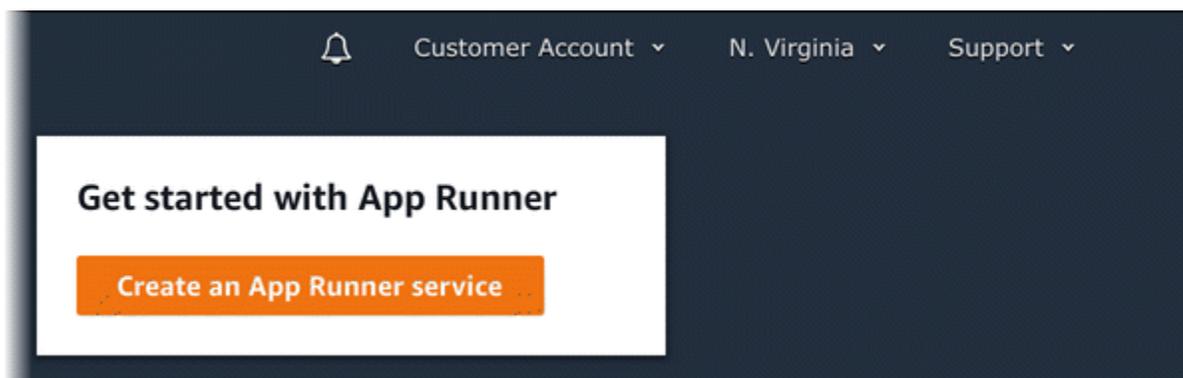
If the AWS account has existing services, the **Services** page with a list of your services is displayed. Choose **Create service**.
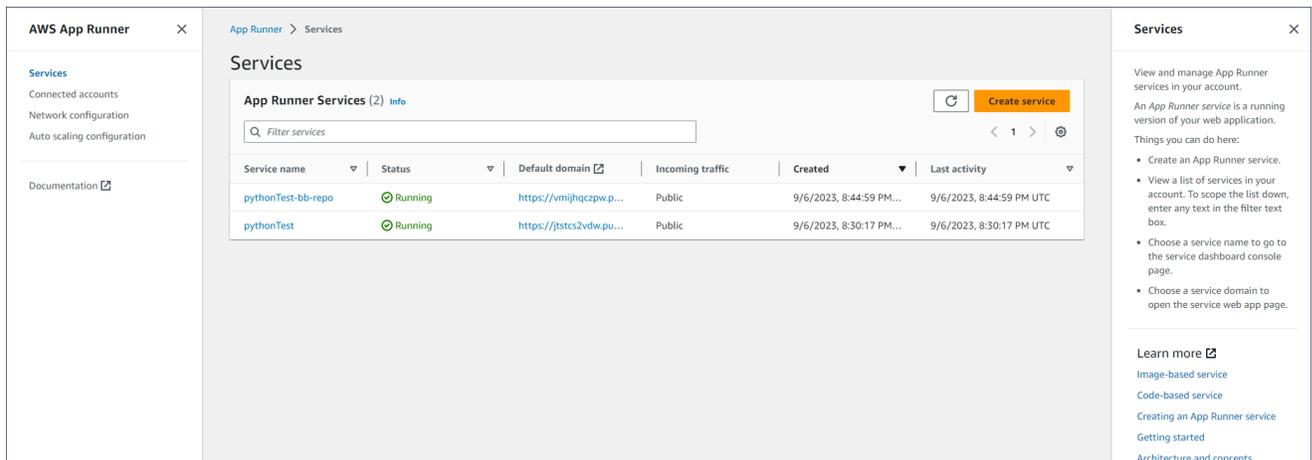
c.   On the **Source and deployment** page, in the **Source** section, for **Repository type**, choose **Source code repository**.

d.   Select a **Provider Type**. Choose either **GitHub** or **Bitbucket**.

e.   Next choose **Add new**. If prompted, provide your GitHub or Bitbucket credentials.

f.   Choose the next set of steps based on the **Provider type** you previously selected.

> **(i) Note**
>
> The following steps to install the AWS connector for GitHub to your GitHub account are one-time steps. You can reuse the connection for creating multiple App Runner services based on repositories in this account. When you have an existing connection, choose it and skip to repository selection.
> The same applies to the AWS connector for your Bitbucket account. If you're using both GitHub and Bitbucket as source code repositories for your App Runner services, you'll need to install one AWS Connector for each provider. You can then reuse each connector for creating more App Runner services.

•   For **GitHub**, follow these steps.

   i.   On the next screen, enter a **Connection Name**.

   ii.   If this your first time using GitHub with App Runner, select **Install another**.

     iii.   In the **AWS Connector for GitHub** dialog box, if prompted, choose your GitHub account name.

     iv.   If prompted to authorize the AWS Connector for GitHub, choose **Authorize AWS Connections**.

     v.   In the **Install AWS Connector for GitHub** dialog box,Choose **Install**.

        Your account name appears as the selected **GitHub account/organization**. You can now choose a repository in your account.

     vi.   For **Repository**, choose the example repository you created, `python-hello`. For **Branch**, choose the default branch name of your repository (for example, **main**).

     vii.   Leave **Source directory** with the default value. The directory defaults to the repository root. You stored your source code in the repository root directory in the previous *Prerequisites* steps.

- For **Bitbucket**, follow these steps.

     i.   On the next screen, enter a **Connection Name**.

     ii.   If this your first time using Bitbucket with App Runner, select **Install another**.

     iii.   In the **AWS CodeStar requests access** dialog box, you can select your workspace and grant access to AWS CodeStar for Bitbucket integration. Select your workspace, then select **Grant access**.

     iv.   Next you'll be redirected to the AWS console. Verify that the Bitbucket application is set to the correct Bitbucket workspace and select **Next**.

     v.   For **Repository**, choose the example repository you created, `python-hello`. For **Branch**, choose the default branch name of your repository (for example, **main**).

     vi.   Leave **Source directory** with the default value. The directory defaults to the repository root. You stored your source code in the repository root directory in the previous *Prerequisites* steps.

2.   Configure your deployments: In the **Deployment settings** section, choose **Automatic**, and then choose **Next**.

> ⓘ **Note**
>
> With automatic deployment, each new commit to your repository source directory automatically deploys a new version of your service.

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.

Step 1: Create an App Runner service   12

Cancel   **Next**

3. Configure application build.

    a. On the **Configure build** page, for **Configuration file**, choose **Configure all settings here**.

    b. Provide the following build settings:

        - **Runtime** – Choose **Python 3**.

        - **Build command** – Enter `pip install -r requirements.txt`.

        - **Start command** – Enter `python server.py`.

        - **Port** – Enter **8080**.

    c. Choose **Next**.

> ⓘ **Note**
>
> The Python 3 runtime builds a Docker image using a base Python 3 image and your example Python code. It then launches a service that runs a container instance of this image.

4.   Configure your service.

   a.   On the **Configure service** page, in the **Service settings** section, enter a service name.

   b.   Under **Environment variables**, select **Add environment variable**. Provide the following values for the environment variable.

      - **Source** – Choose **Plain text**

      - **Environment variable name** – NAME

      - **Environment variable value** – any name (for example, your first name).

> **ⓘ Note**
>
> The example application reads the name you set in this environment variable and displays the name on its webpage.

c. Choose **Next**.

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. On the **Review and create** page, verify all the details you've entered, and then choose **Create and deploy**.

   If the service is successfully created, the console shows the service dashboard, with a **Service overview** of the new service.
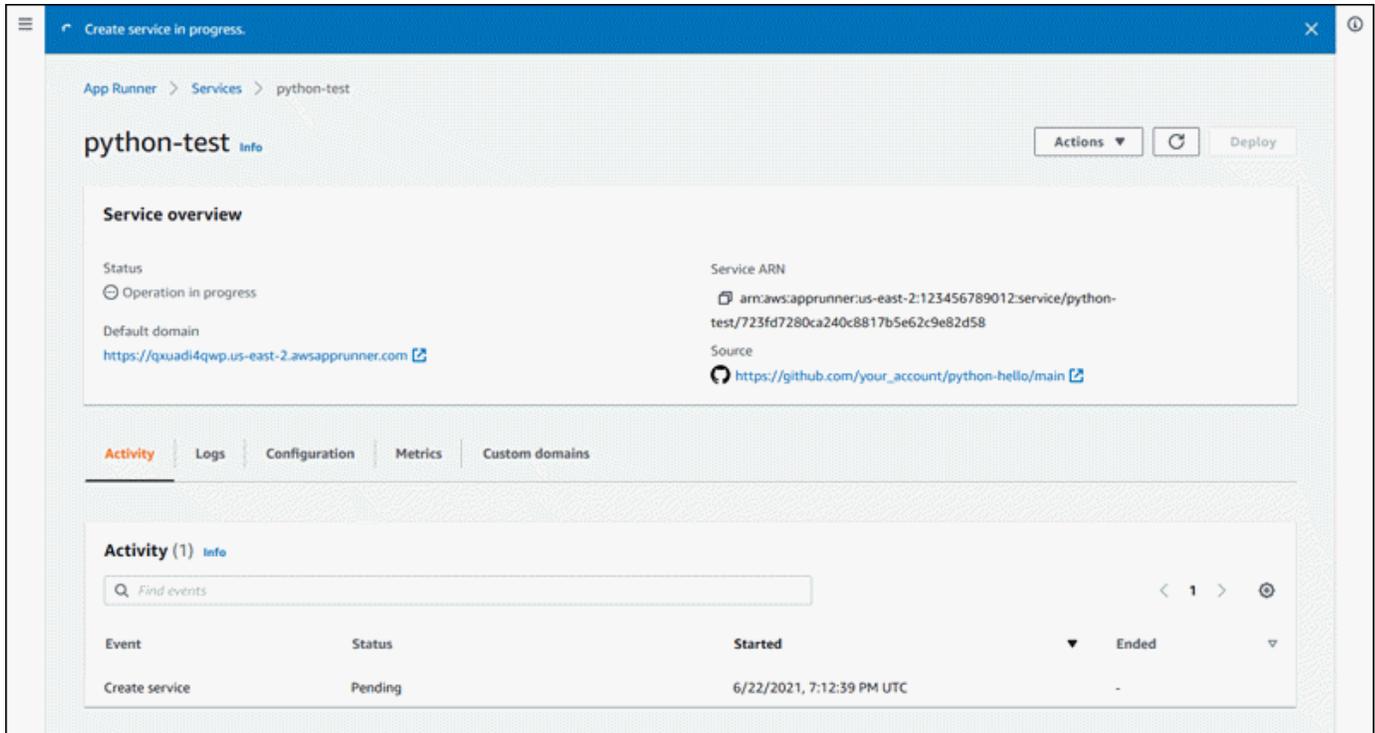
6. Verify that your service is running.

   a. On the service dashboard page, wait until the service **Status** is **Running**.

   b. Choose the **Default domain** value—it's the URL to the website of your service.

   > **ⓘ Note**
   >
   > To augment the security of your App Runner applications, the *.awsapprunner.com* domain is registered in the Public Suffix List (PSL). For further security, we recommend that you use cookies with a __Host- prefix if you ever need to set sensitive cookies in the default domain name for your App Runner applications. This practice will help to defend your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

A webpage displays: **Hello, *your name*!**
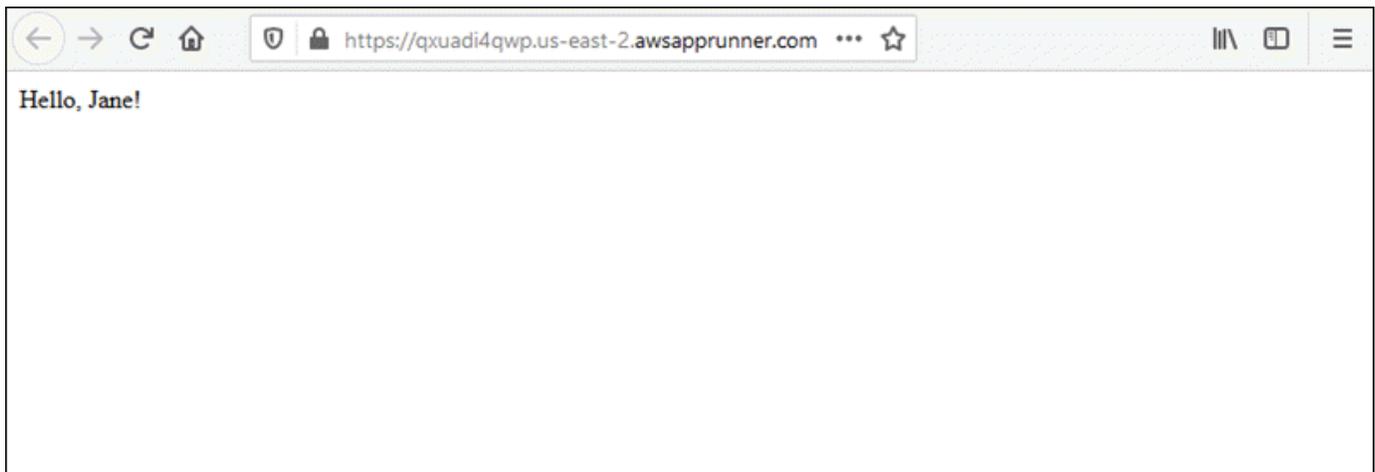
Step 2: Change your service code

In this step, you make a change to your code in the repository source directory. The App Runner CI/CD capability automatically builds and deploys the change to your service.

To make a change to your service code

1. Navigate to your example repository.

2. Edit the file named `server.py`.

3. In the expression assigned to the variable `message`, change the text `Hello` to `Good morning`.

4. Save and commit your changes to the repository.

5. The following steps illustrate changing the service code in a GitHub repository.

   a. Navigate to your example GitHub repository.

   b. Choose the file name `server.py` to navigate to that file.

   c. Choose **Edit this file** (the pencil icon).

   d. In the expression assigned to the variable `message`, change the text `Hello` to `Good morning`.
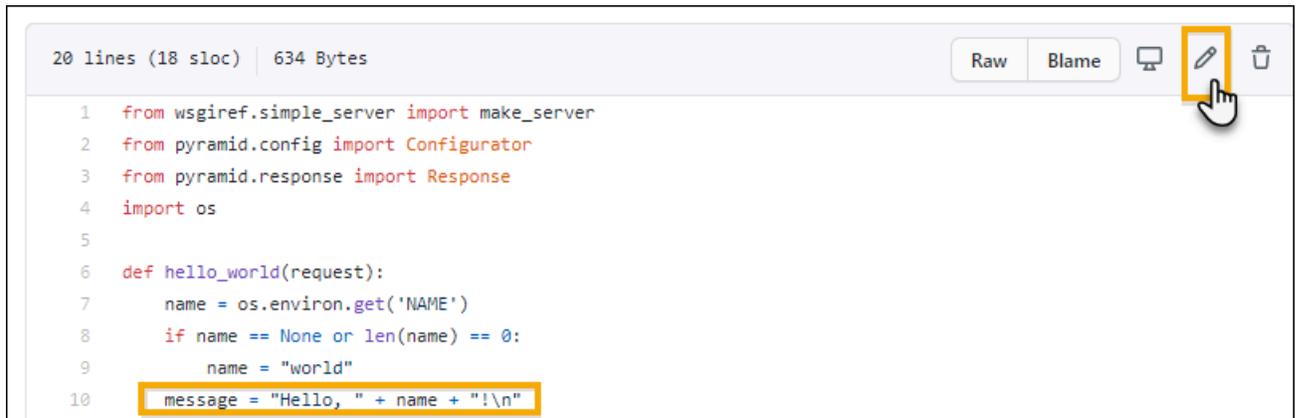
```
20 lines (18 sloc)    634 Bytes                    Raw    Blame

 1    from wsgiref.simple_server import make_server
 2    from pyramid.config import Configurator
 3    from pyramid.response import Response
 4    import os
 5
 6    def hello_world(request):
 7        name = os.environ.get('NAME')
 8        if name == None or len(name) == 0:
 9            name = "world"
10        message = "Hello, " + name + "!\n"
```

    e.   Choose **Commit changes**.

6. The new commit starts to deploy for your App Runner service. On the service dashboard page, the service **Status** changes to **Operation in progress**.

   Wait for the deployment to end. On the service dashboard page, the service **Status** should change back to **Running**.

7. Verify that the deployment is successful: refresh the browser tab where the webpage of your service is displayed.

   The page now displays the modified message: **Good morning, *your name*!**

Step 3: Make a configuration change

In this step, you make a change to the **NAME** environment variable value, to demonstrate a service configuration change.

To change an environment variable value

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

   The console displays the service dashboard with a **Service overview**.

3. On the service dashboard page, choose the **Configuration** tab.

    The console displays your service configuration settings in several sections.

4. In the **Configure service** section, choose **Edit**.

5. For the environment variable with the key **NAME**, change the value to a different name.

6. Choose **Apply changes**.

    App Runner starts the update process. On the service dashboard page, the service **Status** changes to **Operation in progress**.

7. Wait for the update to end. On the service dashboard page, the service **Status** should change back to **Running**.

8. Verify that the update is successful: refresh the browser tab where the webpage of your service is displayed.

    The page now displays the modified name: **Good morning, *new name*!**

Step 4: View logs for your service

In this step, you use the App Runner console to view logs for your App Runner service. App Runner streams logs to Amazon CloudWatch Logs (CloudWatch Logs) and displays them on your service's dashboard. For information about App Runner logs, see [the section called "Logs (CloudWatch Logs)"](#).

To view logs for your service

1. Open the [App Runner console](#), and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

    The console displays the service dashboard with a **Service overview**.

3. On the service dashboard page, choose the **Logs** tab.

    The console displays a few types of logs in several sections:

    - **Event log** – Activity in the lifecycle of your App Runner service. The console displays the latest events.

    - **Deployment logs** – Source repository deployments to your App Runner service. The console displays a separate log stream for each deployment.

    - **Application logs** – The output of the web application that's deployed to your App Runner service. The console combines the output from all running instances into a single log stream.

4. To find specific deployments, scope down the deployment log list by entering a search term. You can search for any value that appears in the table.

5. To view a log's content, choose **View full log** (event log) or the log stream name (deployment and application logs).

6. Choose **Download** to download a log. For a deployment log stream, select a log stream first.

7. Choose **View in CloudWatch** to open the CloudWatch console and use its full capabilities to explore your App Runner service logs. For a deployment log stream, select a log stream first.

> ⓘ **Note**
>
> The CloudWatch console is particularly useful if you want to view application logs of specific instances instead of the combined application log.

Step 5: Clean up

You've now learned how to create an App Runner service, view logs, and make some changes. In this step, you delete the service to remove resources that you don't need anymore.

To delete your service

1. On the service dashboard page, choose **Actions**, and then choose **Delete service**.

2. In the confirmation dialog, enter the requested text, and then choose **Delete**.

    Result: The console navigates to the **Services** page. The service that you just deleted shows a status of **DELETING**. A short time later it disappears from the list.

Consider also deleting the GitHub and Bitbucket connections that you created as part of this tutorial. For more information, see the section called "Connections".

What's next

Now that you've deployed your first App Runner service, learn more in the following topics:

*   *Architecture and concepts* – The architecture, main concepts, and AWS resources related to App Runner.

*   Image-based service and Code-based service – The two types of application source that App Runner can deploy.

*   *Developing for App Runner* – Things you should know when developing or migrating application code for deployment to App Runner.

*   *App Runner console* – Manage and monitor your service using the App Runner console.

*   *Managing your service* – Manage the lifecycle of your App Runner service.

*   *Observability* – Get visibility into your App Runner service operations by monitoring metrics, reading logs, handling events, tracking service action calls, and tracing application events like HTTP calls.

*   App Runner configuration file – A configuration-based way to specify options for the build and runtime behavior of your App Runner service.

*   *App Runner API* – Use the App Runner application programming interface (API) to create, read, update, and delete App Runner resources.

*   *Security* – The different ways that AWS and you ensure cloud security while you use App Runner and other services.

App Runner architecture and concepts

AWS App Runner takes your source code or source image from a repository, and creates and maintains a running web service for you in the AWS Cloud. Typically, you need to call just one App Runner action, CreateService, to create your service.

With a source image repository, you provide a ready-to-use container image that App Runner can deploy to run your web service. With a source code repository, you provide your code and instructions for building and running a web service, and you target a specific runtime environment. App Runner supports several programming platforms, each with one or more managed runtimes for platform major versions.

At this time, App Runner can retrieve your source code from either a Bitbucket or GitHub repository, or it can retrieve your source image from Amazon Elastic Container Registry (Amazon ECR) in your AWS account.

The following diagram shows an overview of the App Runner service architecture. In the diagram, there are two example services: one deploys source code from GitHub, and the other deploys a source image from Amazon ECR. The same flow applies to the Bitbucket repository.
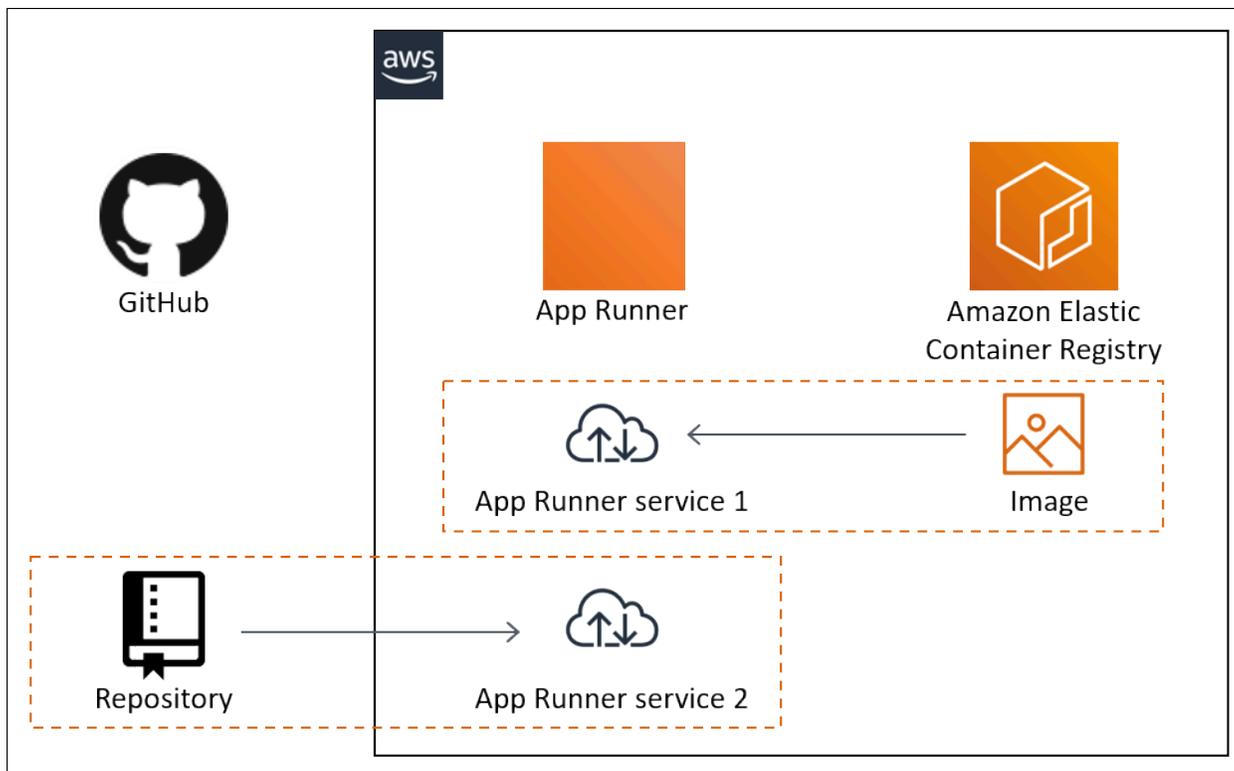
App Runner concepts

The following are key concepts related to your web service that's running in App Runner:

- *App Runner service* – An AWS resource that App Runner uses to deploy and manage your application based on its source code repository or container image. An App Runner service is a running version of your application. For more information about creating a service, see [the section called "Creation"](#).

- *Source type* – The type of source repository that you provide for deploying your App Runner service: [source code](#) or [source image](#).

- *Repository provider* – The repository service that contains your application source (for example, [GitHub](#), [Bitbucket](#), or [Amazon ECR](#)).

- *App Runner connection* – An AWS resource that lets App Runner access a repository provider account (for example, a GitHub account or organization). For more information about connections, see [the section called "Connections"](#).

- *Runtime* – A base image for deploying a source code repository. App Runner provides a variety of *managed runtimes* for different programming platforms and versions. For more information, see [Code-based service](#).

- *Deployment* – An action that applies a version of your source repository (code or image) to an App Runner service. The first deployment to the service occurs as part of service creation. Later deployments can occur in one of two ways:

  - *Automatic deployment* – A CI/CD capability. You can configure an App Runner service to automatically build (for source code) and deploy each version of your application as it appears in the repository. This can be a new commit in a source code repository or a new image version in a source image repository.

  - *Manual deployment* – A deployment to your App Runner service that you explicitly start.

- *Custom domain* – A domain that you associate with your App Runner service. Users of your web application can use this domain to access your web service instead of the default App Runner subdomain. For more information, see [the section called "Custom domain names"](#).

> **ⓘ Note**
>
> To augment the security of your App Runner applications, the **.awsapprunner.com* domain is registered in the [Public Suffix List (PSL)](#). For further security, we recommend that you use cookies with a `__Host-` prefix if you ever need to set sensitive cookies

in the default domain name for your App Runner applications. This practice will help to defend your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

- *Maintenance* – An activity that App Runner occasionally performs on the infrastructure that runs your App Runner service. When maintenance is in progress, service status temporarily changes to OPERATION_IN_PROGRESS (**Operation in progress** in the console) for a few minutes. Actions on your service (for example, deployment, configuration update, pause/resume, or deletion) are blocked during this time. Try the action again a few minutes later, when the service status returns to RUNNING.

> ⓘ **Note**
>
> If your action fails, it doesn't mean that your App Runner service is down. Your application is active and keeps handling requests. It's unlikely for your service to experience any downtime.

In particular, App Runner migrates your service if it detects issues in the underlying hardware hosting the service. To prevent any service downtime, App Runner deploys your service to a new set of instances and shifts traffic to them (a blue-green deployment). You might occasionally see a slight temporary increase in charges.

App Runner supported configurations

When you configure an App Runner service, you specify the virtual CPU and memory configuration to allocate to your service. You pay based on the compute configuration that you select. For more information on pricing, see AWS Resource Groups Pricing.

The following table provides information on the vCPU and memory configurations that App Runner supports:

CPU	Memory
0.25 vCPU	0.5 GB
0.25 vCPU	1 GB

CPU	Memory
0.5 vCPU	1 GB
1 vCPU	2 GB
1 vCPU	3 GB
1 vCPU	4 GB
2 vCPU	4 GB
2 vCPU	6 GB
4 vCPU	8 GB
4 vCPU	10 GB
4 vCPU	12 GB

App Runner resources

When you use App Runner, you create and manage a few types of resources in your AWS account. These resources are used to access your code and manage your services.

The following table provides an overview of these resources:

Resource name	Description
Service	Represents a running version of your application. Much of the rest of this guide describes service types, management, configuration, and monitoring. ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:service/` *`service-name`* `[/`*`service-id`* `]`
Connection	Provides your App Runner services with access to private repositories stored with third-party providers. Exists as a separate resource

Resource name	Description
	for sharing across multiple services. For more information about connections, see the section called "Connections". ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:connecti on/` *`connection-name`* `[/`*`connection-id`* `]`
AutoScalingConfigu ration	Provides your App Runner services with settings that control the automatic scaling of your application. Exists as a separate resource for sharing across multiple services. For more information about automatic scaling, see the section called "Auto scaling". ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:autoscal ingconfiguration/` *`config-name`* `[/`*`config-re vision`* `[/`*`config-id`* `]]`
ObservabilityConfi guration	Configures additional application observability features for your App Runner services. Exists as a separate resource for sharing across multiple services. For more information about observability configura tion, see the section called "Observability configuration". ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:observab ilityconfiguration/` *`config-name`* `[/`*`config-re vision`* `[/`*`config-id`* `]]`
VpcConnector	Configures VPC settings for your App Runner services. Exists as a separate resource for sharing across multiple services. For more information about VPC functionality, see the section called "Outgoing traffic". ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:vpcconne ctor/` *`connector-name`* `[/`*`connector-revision`* `[/`*`connector-id`* `]]`

Resource name	Description
VpcIngressConnection	It's an AWS App Runner resource used to configure incoming traffic. It establishes a connection between a VPC interface endpoint and App Runner service, to make your App Runner service accessible from only within an Amazon VPC. For more information about functionality of VPCIngressConnection, see the section called "Enable Private endpoint". ARN: `arn:aws:apprunner:` *`region`*`:`*`account-id`* `:vpcingressconnection/` *`vpc-ingress-connection-name`* ` [/`*`connector-id`* `]]`

App Runner resource quotas

AWS imposes some quotas (also known as limits) on your account for AWS resource usage in each AWS Region. The following table lists quotas related to App Runner resources. Quotas are also listed in AWS App Runner endpoints and quotas in the *AWS General Reference*.

Resource quota	Description	Default value	Adjustable?
Services	The maximum number of services that you can create in your account for each AWS Region.	30	✓ Yes
Connections	The maximum number of connections that you can create in your account for each AWS Region. You can use a single connection in multiple services.	10	✓ Yes
Auto scaling configurations names	The maximum number of unique names that you can have in auto scaling configurations that you create in your account for each AWS Region. You can use a single auto scaling configuration in multiple services.	10	✓ Yes

Resource quota		Description	Default value	Adjustable?
	revisions per name	The maximum number of auto scaling configuration revisions that you can create in your account for each AWS Region for each unique name. You can use a single auto scaling configuration revision in multiple services.	5	✕ No
Observability configurations	names	The maximum number of unique names that you can have in observability configurations that you create in your account for each AWS Region. You can use a single observability configuration in multiple services.	10	✓ Yes
	revisions per name	The maximum number of observability configuration revisions that you can create in your account for each AWS Region for each unique name. You can use a single observability configuration revision in multiple services.	10	✕ No
VPC connectors		The maximum number of VPC connectors that you can create in your account for each AWS Region. You can use a single VPC connector in multiple services.	10	✓ Yes
VPC Ingress Connection		The maximum number of VPC Ingress Connections that you can create in your account for each AWS Region. You can use a single VPC Ingress Connection to access multiple App Runner services.	1	✕ No

Most quotas are adjustable, and you can request a quota increase for them. For more information, see Requesting a quota increase in the Service Quotas User Guide.

App Runner service based on a source image

You can use AWS App Runner to create and manage services based on two fundamentally different types of service source: *source code* and *source image*. Regardless of the source type, App Runner takes care of starting, running, scaling, and load balancing your service. You can use the CI/CD capability of App Runner to track changes to your source image or code. When App Runner discovers a change, it automatically builds (for source code) and deploys the new version to your App Runner service.

This chapter discusses services based on a source image. For information about services based on source code, see [Code-based service](#).

A *source image* is a public or private container image stored in an image repository. You point App Runner to an image, and it starts a service running a container based on this image. No build stage is necessary. Rather, you provide a ready-to-deploy image.

> ⓘ **Note**
>
> When providing container images, you are responsible for regularly updating and patching these images. While App Runner manages the infrastructure, you should ensure the security and up-to-date status of the provided container images. For more information, see the [AWS App Runner Documentation](#)

Image repository providers

App Runner supports the following image repository providers:

- **Amazon Elastic Container Registry (Amazon ECR)** – Stores images that are private to an AWS account.
- **Amazon Elastic Container Registry Public (Amazon ECR Public)** – Stores images that are publicly readable.

Provider use cases

- [Using an image stored in Amazon ECR in your AWS account](#)
- [Using an image stored in Amazon ECR in a different AWS account](#)

- [Using an image stored in Amazon ECR Public](#)

Using an image stored in Amazon ECR *in your AWS account*

[Amazon ECR](#) stores images in repositories. There are private and public repositories. To deploy your image to an App Runner service from a private repository, App Runner needs permission to read your image from Amazon ECR. To give that permission to App Runner, you need to provide App Runner with an *access role*. This is an AWS Identity and Access Management (IAM) role that has the necessary Amazon ECR action permissions. When you use the App Runner console to create the service, you can choose an existing role in your account. Alternatively, you can use the IAM console to create a new custom role. Or, you can choose for the App Runner console to create a role for you based on managed policies.

When you use the App Runner API or the AWS CLI, you complete a two-step process. First, you use the IAM console to create an access role. You can use a managed policy that App Runner provides or enter your own custom permissions. Then, you provide the access role during service creation using the [CreateService](#) API action.

For information about App Runner service creation, see [the section called "Creation"](#).

Using an image stored in Amazon ECR *in a different AWS account*

When you create an App Runner service, you can use an image stored in an Amazon ECR repository that belongs to an AWS account other than the one that your service is in. There are a few additional considerations to keep in mind when using a cross-account image, in addition to those listed in the previous section about a same-account image.

- The cross-account repository should have a policy attached to it. The repository policy provides your access role with permissions to read images in the repository. Use the following policy for this purpose. Replace the role in the `Princiapal` element with the Amazon Resource Name (ARN) of your access role.

  JSON

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

```
        "Principal": {"AWS": "arn:aws:iam::123456789012:role/MyApplicationRole"},
        "Action": [
          "ecr:BatchGetImage",
          "ecr:DescribeImages",
          "ecr:GetDownloadUrlForLayer"
          ],
          "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/*"
      }
    ]
}
```

For information about attaching a repository policy to an Amazon ECR repository, see Setting a repository policy statement in the *Amazon Elastic Container Registry User Guide*.

- App Runner doesn't support automatic deployment for Amazon ECR images in a different account than the one that your service is in.

Using an image stored in Amazon ECR Public

Amazon ECR Public stores publicly readable images. These are the main differences between Amazon ECR and Amazon ECR Public that you should be aware of in the context of App Runner services:

- Amazon ECR Public images are publicly readable. You don't need to provide an access role when you create a service based on an Amazon ECR Public image. The repository doesn't need any policy attached to it.

- App Runner doesn't support automatic (continuous) deployment for Amazon ECR Public images.

Launch a service directly from Amazon ECR Public

You can directly launch container images of compatible web applications that are hosted on the Amazon ECR Public Gallery as web services running on App Runner. When browsing the gallery, look for **Launch with App Runner** on the gallery page for an image. An image with this option is compatible with App Runner. For more information about the gallery, see Using the Amazon ECR Public Gallery in the *Amazon ECR Public user guide*.

To launch a gallery image as an App Runner service

1. On the gallery page of an image, choose **Launch with App Runner**.

    Result: The App Runner console opens in a new browser tab. The console displays the **Create service** wizard, with most of the required new service details pre-filled.

2. If you want to create your service in an AWS Region other than the one that the console is showing, choose the Region displayed on the console header. Then, select another Region.

3. For **Port**, enter the port number that the image application listens on. You can typically find it on the gallery page for the image.

4. Optionally, change any other configuration details.

5. Choose **Next**, review the settings, and then choose **Create & deploy**.

Image example

The App Runner team maintains the **hello-app-runner** example image in an Amazon ECR Public Gallery. You can use this example to get started with creating an image-based App Runner service. For more information, see hello-app-runner.

App Runner service based on source code

You can use AWS App Runner to create and manage services based on two fundamentally different types of service source: *source code* and *source image*. Regardless of the source type, App Runner takes care of starting, running, scaling, and load balancing your service. You can use the CI/CD capability of App Runner to track changes to your source image or code. When App Runner discovers a change, it automatically builds (for source code) and deploys the new version to your App Runner service.

This chapter discusses services based on source code. For information about services based on a source image, see *Image-based service*.

Source code is application code that App Runner builds and deploys for you. You point App Runner to a source directory in a code repository and choose a suitable *runtime* that corresponds to a programming platform version. App Runner builds an image that's based on the base image of the runtime and your application code. It then starts a service that runs a container based on this image.

App Runner provides convenient platform-specific *managed runtimes*. Each one of these runtimes builds a container image from your source code, and adds language runtime dependencies into your image. You don't need to provide container configuration and build instructions such as a Dockerfile.

Subtopics of this chapter discuss the various platforms that App Runner supports— *managed platforms* that provide managed runtimes for different programming environments and versions.

Topics

- Source code repository providers
- Source directory
- App Runner managed platforms
- End of support for managed runtime versions
- Managed runtime versions and the App Runner build
- Using the Python platform
- Using the Node.js platform
- Using the Java platform
- Using the .NET platform

- [Using the PHP platform](#)
- [Using the Ruby platform](#)
- [Using the Go platform](#)

Source code repository providers

App Runner deploys your source code by reading it from a source code repository. App Runner supports two source code repository providers: [GitHub](#) and [Bitbucket](#).

Deploying from your source code repository provider

To deploy your source code to an App Runner service from a source code repository, App Runner establishes a connection to it. When you use the App Runner console to [create a service](#), you provide connection details and a source directory for App Runner to deploy your source code.

Connections

You provide connection details as part of the service creation procedure. When you use the App Runner API or the AWS CLI, a connection is a separate resource. First, you create the connection using the [CreateConnection](#) API action. Then, you provide the connection's ARN during service creation using the [CreateService](#) API action.

Source directory

When you create a service you also provide a source directory. By default, App Runner uses the root directory of your repository as the source directory. The source directory is the location in your source code repository that stores your application's source code and configuration files. The build and start commands also execute from the source directory. When you use the App Runner API or the AWS CLI to create or update a service you provide the source directory in the [CreateService](#) and [UpdateService](#) API actions. For more information, see the [Source directory](#) section that follows.

For more information about App Runner service creation, see [the section called "Creation"](#). For more information about App Runner connections, see [the section called "Connections"](#).

Source directory

When you create an App Runner service you can provide the source directory, along with the repository and branch. Set the value of the **Source directory** field to the repository directory path

that stores the application's source code and configuration files. App Runner executes the build and start commands from the source directory path that you provide.

Enter the value for source directory path as absolute from the root repository directory. If you don't specify a value, it defaults to the repository top-level directory, also known as the repository root directory.

You also have the option to provide different source directory paths besides the top-level repository directory. This supports a monorepo repository architecture, which means the source code for multiple applications is stored in one repository. To create and support multiple App Runner services from a single monorepo, specify different source directories when you create each service.

> **ⓘ Note**
>
> If you specify the same source directory for multiple App Runner services, both services will deploy and operate individually.

If you opt to use an `apprunner.yaml` configuration file to define your service parameters place it in the source directory folder of the repository.

If the **Deployment trigger** option is set to the **Automatic**, the changes you commit in the source directory will trigger an automatic deployment. *Only the changes in the source directory* path will trigger an automatic deployment. It's important to understand how the location of the source directory affects the scope of an automatic deployment. For more information, see *automated deployments* in [Deployment methods](#).

> **ⓘ Note**
>
> If your App Runner service uses the PHP managed runtimes, and you'd like to designate a source directory other than the default root repository, it's important to use the correct PHP runtime version. For more information, see [Using the PHP platform](#).

App Runner managed platforms

App Runner managed platforms provide managed runtimes for various programming environments. Each managed runtime makes it easy to build and run containers based on a

version of a programming language or runtime environment. When you use a managed runtime, App Runner starts with a managed runtime image. This image is based on the [Amazon Linux Docker image](#) and contains a language runtime package as well as some tools and popular dependency packages. App Runner uses this managed runtime image as a base image, and adds your application code to build a Docker image. It then deploys this image to run your web service in a container.

You specify a runtime for your App Runner service when you [create a service](#) using the App Runner console or the [CreateService](#) API operation. You can also specify a runtime as part of your source code. Use the `runtime` keyword in a [App Runner configuration file](#) that you include in your code repository. The naming convention of a managed runtime is *`<language-name><major-version>`*.

App Runner updates the runtime for your service to the latest version on every deployment or service update. If your application requires a specific version of a managed runtime, you can specify it using the `runtime-version` keyword in the [App Runner configuration file](#). You can lock to any level of version, including a major or minor version. App Runner only makes lower-level updates to the runtime of your service.

End of support for managed runtime versions

When the official provider or community of a managed language runtime officially declares a version to be End of Life (EOL), App Runner follows by declaring the version status to be *End of Support*. If your service is running on a managed language runtime version that has reached End of Support the following policies and recommendations apply.

End of Support for a language runtime version:

- **Existing services** will continue to run and serve traffic even if they use a runtime that has reached End of Support. However, they will run on unsupported runtimes that no longer receive updates, security patches, or technical support.

- **Updates to existing services** that use End of Support runtimes are still allowed, but we do not recommend continued use of End of Support runtimes for a service.

- **New services** cannot be created using the runtimes that have reached the End of Support date.

Actions Required for language runtime versions with End of Support status:

- If your service is **based on a source image**, no further action is required on your part for that service.

- If your service is **based on source code**, update your service configuration to use a supported runtime version. To do so, select a supported runtime version in the App Runner console, update the `runtime` field in the apprunner.yaml configuration file, or use the CreateService/UpdateService API operations or IaC tools to set the `runtime` parameter. For a list of supported runtimes, see the *Release information* page for any specific runtime in this chapter.

- Alternatively, you can switch to App Runner's **container image source** option. For more details, see Image-based service.

> ⓘ **Note**
>
> If you're moving from Node.js 12, 14, or 16 to **Node.js 22**, or from Python 3.7 or 3.8 to **Python 3.11**, be aware that Node.js 22 and Python 3.11 use a revised App Runner build process that offers faster and more efficient builds. To ensure compatibility before upgrading we recommend that you review the build process guidance in the next section.

The following tables lists the App Runner managed runtime versions that have a designated End of Support date.

Runtime Versions	App Runner End of Support date
Python 3.8 Supported runtimes	December 1, 2025
Python 3.7 Supported runtimes	December 1, 2025
Node.js 18 Supported runtimes	December 1, 2025
Node.js 16 Supported runtimes	December 1, 2025
Node.js 14 Supported runtimes	December 1, 2025
Node.js 12 Supported runtimes	December 1, 2025
.NET 6 *	December 1, 2025
PHP 8.1 *	December 31, 2025

Runtime Versions	App Runner End of Support date
Ruby 3.1 *	December 1, 2025
Go 1 *	December 1, 2025

* App Runner will not release any new language versions for the runtimes marked with an asterisk (*). These runtimes are the following: .NET, PHP, Ruby, and Go. If you have a code-based service configured for these runtimes, we recommend one of the following actions:

- If applicable, switch your service configuration to a different supported managed runtime.

- Alternatively, build a custom container image with your preferred runtime version and deploy it using App Runner's Image-based service option. You can host your image in Amazon ECR.

Managed runtime versions and the App Runner build

App Runner offers an updated build process for applications that run on the more recent major version runtimes. This revised build process is faster and more efficient. It also creates a final image with a smaller footprint that only contains your source code, build artifacts, and runtimes needed to run your application.

We refer to the newer build process as the *revised App Runner build* and to the original build process as the *original App Runner build*. To avoid breaking changes to earlier version of runtime platforms, App Runner only applies the revised build to specific runtime versions, typically newly released major releases.

We've introduced a new component to the `apprunner.yaml` configuration file to make the revised build backward compatible for a very specific use case and to also provide more flexibility to configure the build of your application. This is the optional `pre-run` parameter. We explain when to use this parameter along with other useful information about the builds in the sections that follow.

The following table conveys which version of the App Runner build applies to specific managed runtime versions. We'll continue to update this document to keep you informed about our current runtimes.

Platform	Runtime Versions	Build process	App Runner End of Support date
Python – Release information	Python 3.11 (!)	Revised	
	Python 3.8	Original	December 1, 2025
	Python 3.7	Original	December 1, 2025
Node.js – Release information	Node.js 22	Revised	
	Node.js 18	Revised	December 1, 2025
	Node.js 16	Original	December 1, 2025
	Node.js 14	Original	December 1, 2025
	Node.js 12	Original	December 1, 2025
Corretto – Release information	Corretto 11	Original	
	Corretto 8	Original	
.NET – Release information	.NET 6 *	Original	December 1, 2025
PHP – Release information	PHP 8.1 *	Original	December 31, 2025
Ruby – Release information	Ruby 3.1 *	Original	December 1, 2025
Go – Release information	Go 1 *	Original	December 1, 2025

> **ⓘ Note**
>
> Some of the listed runtimes include an **End of Support** date. For more information, see the section called "End of support for managed runtime versions".

> ⚠️ **Important**
>
> **Python 3.11** – We have specific recommendations for the build configuration of services that use the Python 3.11 managed runtime. For more information, see Callouts for specific runtime versions in the *Python platform* topic.

More about the App Runner builds and migration

When you migrate your application to a newer runtime that uses the revised build, you may need to slightly modify your build configuration.

To provide context for migration considerations, we'll first describe the high level processes for both the original App Runner build and the revised build. We'll follow with a section that describes specific attributes about your service that might require some configuration updates.

The original App Runner build

The original App Runner application build process leverages the AWS CodeBuild service. The initial steps are based on images curated by the CodeBuild service. A Docker build process follows that uses the applicable App Runner managed runtime image as the base image.

The general steps are the following:

1. Run `pre-build` commands in a CodeBuild-curated image.

   The `pre-build` commands are optional. They can only be specified in the `apprunner.yaml` configuration file.

2. Run the `build` commands using CodeBuild on the same image from the prior step.

   The `build` commands are required. They can be specified in the App Runner console, the App Runner API, or in the `apprunner.yaml` configuration file.

3. Run a Docker build to generate an image based on the App Runner managed runtime image for your specific platform and runtime version.

4. Copy the `/app` directory from the image that we generated in **Step 2**. The destination is the image based on the App Runner managed runtime image, that we generated in **Step 3**.

5. Run the `build` commands again on the generated App Runner managed runtime image. We run the build commands again to generate build artifacts from the source code in the `/app`

directory that we copied to it in **Step 4**. This image will later be deployed by App Runner to run your web service in a container.

The `build` commands are required. They can be specified in the App Runner console, the App Runner API, or in the `apprunner.yaml` configuration file.

6. Run `post-build` commands in the CodeBuild image from **Step 2**.

The `post-build` commands are optional. They can only be specified in the `apprunner.yaml` configuration file.

After the build completes, App Runner deploys the generated App Runner managed runtime image from **Step 5** to run your web service in a container.

The revised App Runner build

The revised build process is faster and more efficient than the original build process described in the prior section. It eliminates the duplication of the build commands that occurs in the prior version build. It also creates a final image with a smaller footprint that only contains your source code, build artifacts, and runtimes needed to run your application.

This build process uses a Docker multi-stage build. The general process steps are the following:

1. **Build stage** — Start a docker build process that executes `pre-build` and `build` commands on top of the App Runner build images.

   a. Copy the application source code to the `/app` directory.

   > ⓘ **Note**
   >
   > This `/app` directory is designated as the working directory in every stage of the Docker build.

   b. Run `pre-build` commands.

   The`pre-build` commands are optional. They can only be specified in the `apprunner.yaml` configuration file.

   c. Run the `build` commands.

   The `build` commands are required. They can be specified in the App Runner console, the App Runner API, or in the `apprunner.yaml` configuration file.

2. **Packaging stage** — Generates the final customer container image, which is also based on the App Runner run image.

   a. Copy the `/app` directory from the prior **Build stage** to the new Run image. This includes your application source code and the build artifacts from the prior stage.

   b. Run the `pre-run` commands. If you need to modify the runtime image outside of the `/app` directory by using the `build` commands, add the same or required commands to this segment of the `apprunner.yaml` configuration file.

   This is a new parameter that was introduced to support the revised App Runner build.

   The `pre-run` commands are optional. They can only be specified in the `apprunner.yaml` configuration file.

   > **ⓘ Notes**
   >
   > - The `pre-run` commands are only supported by the revised build. Do not add them to the configuration file if your service uses runtime versions that use the original build.
   >
   > - If you don't need to modify anything outside of the `/app` directory with the `build` commands, then you don't need to specify `pre-run` commands.

3. **Post-build stage** — This stage resumes from the *Build stage* and runs `post-build` commands.

   a. Run the `post-build` commands inside the `/app` directory.

   The `post-build` commands are optional. They can only be specified in the `apprunner.yaml` configuration file.

After the build completes, App Runner then deploys the Run image to run your web service in a container.

> **ⓘ Note**
>
> Don't be misled to the `env` entries in the Run section of the `apprunner.yaml` when configuring the build process. Even though the `pre-run` command parameter, referenced in **Step 2(b)**, resides in the Run section, don't use the `env` parameter in the Run section to configure your build. The `pre-run` commands only reference the `env` variables defined in

the Build section of the configuration file. For more information, see [Run section](#) in the *App Runner configuration file chapter*.

Service requirements for migration consideration

If your application environment has either of these two requirements, then you'll need to revise your build configuration, by adding `pre-run` commands.

- If you need to modify anything outside of the `/app` directory with the `build` commands.

- If you need to run the `build` commands twice to create the required environment. This is a very unusual requirement. The vast majority of builds will not do this.

Modifications outside the `/app` directory

- The [revised App Runner build](#) assumes that your application does not have dependencies outside the `/app` directory.

- The commands that you provide either with the `apprunner.yaml` file, the App Runner API, or the App Runner console must generate build artifacts in the `/app` directory.

- You can modify the `pre-build`, `build`, and `post-build` commands to ensure all the build artifacts are in the `/app` directory.

- If your application requires the build to further modify the generated image for your service, outside of the `/app` directory, you can use the new `pre-run` commands in the `apprunner.yaml`. For more information, see [Setting App Runner service options using a configuration file](#).

Running the `build` commands twice

- The [original App Runner build](#) runs the `build` commands twice, first in **Step 2**, then again in **Step 5**. The revised App Runner build remedies this redundancy and only runs the `build` commands one time. If your application should have an unusual requirement for the `build` commands to run twice, the revised App Runner build provides the option to specify and execute the same commands again using the `pre-run` parameter. Doing so retains the same double build behavior.

Using the Python platform

> ⚠️ **Important**
>
> App Runner will end the support for **Python 3.7** and **Python 3.8** on December 1, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

The AWS App Runner Python platform provides managed runtimes. Each runtime makes it easy to build and run containers with web applications based on a Python version. When you use a Python runtime, App Runner starts with a managed Python runtime image. This image is based on the Amazon Linux Docker image and contains the runtime package for a version of Python and some tools and popular dependency packages. App Runner uses this managed runtime image as a base image, and adds your application code to build a Docker image. It then deploys this image to run your web service in a container.

You specify a runtime for your App Runner service when you create a service using the App Runner console or the CreateService API operation. You can also specify a runtime as part of your source code. Use the `runtime` keyword in a App Runner configuration file that you include in your code repository. The naming convention of a managed runtime is *<language-name><major-version>*.

For valid Python runtime names and versions, see the section called "Release information".

App Runner updates the runtime for your service to the latest version on every deployment or service update. If your application requires a specific version of a managed runtime, you can specify it using the `runtime-version` keyword in the App Runner configuration file. You can lock to any level of version, including a major or minor version. App Runner only makes lower-level updates to the runtime of your service.

Version syntax for Python runtimes: *major*[.*minor*[.*patch*]]

For example: `3.8.5`

The following examples demonstrate version locking:

- `3.8` – Lock the major and minor versions. App Runner updates only patch versions.
- `3.8.5` – Lock to a specific patch version. App Runner doesn't update your runtime version.

Topics

Python runtime configuration

When you choose a managed runtime, you must also configure, as a minimum, build and run commands. You configure them while [creating](#) or [updating](#) your App Runner service. You can do this using one of the following methods:

- **Using the App Runner console** – Specify the commands in the **Configure build** section of the creation process or configuration tab.

- **Using the App Runner API** – Call the [CreateService](#) or [UpdateService](#) API operation. Specify the commands using the `BuildCommand` and `StartCommand` members of the [CodeConfigurationValues](#) data type.

- **Using a [configuration file](#)** – Specify one or more build commands in up to three build phases, and a single run command that serves to start your application. There are additional optional configuration settings.

Providing a configuration file is optional. When you create an App Runner service using the console or the API, you specify if App Runner gets your configuration settings directly when it's created or from a configuration file.

Callouts for specific runtime versions

> **ⓘ Note**
>
> App Runner now runs an updated build process for applications based on the following runtime versions: Python 3.11, Node.js 22, and Node.js 18. If your application runs on either one of these runtime versions, see [Managed runtime versions and the App Runner build](#) for more information about the revised build process. Applications that use all other runtime versions are not affected, and they continue to use the original build process.

Python 3.11 (revised App Runner build)

Use the following settings in the *apprunner.yaml* for the managed Python 3.11 runtime.

- Set the `runtime` key in the Top section to `python311`

  Example

  ```
  runtime: python311
  ```

- Use the `pip3` instead of `pip` to install dependencies.

- Use the `python3` interpreter instead of `python`.

- Run the `pip3` installer as a `pre-run`command. Python installs dependencies outside of the /app directory. Since App Runner runs the revised App Runner build for Python 3.11, anything installed outside of the /app directory through commands in the Build section of the `apprunner.yaml` file will be lost. For more information, see The revised App Runner build.

  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 -
  ```

For more information, also see the example of an extended configuration file for Python 3.11 later in this topic.

Python runtime examples

The following examples show App Runner configuration files for building and running a Python service. The last example is the source code for a complete Python application that you can deploy to a Python runtime service.

> **ⓘ Note**
>
> The runtime version that's used in these examples is *3.7.7* and *3.11*. You can replace it with a version you want to use. For latest supported Python runtime version, see the section called "Release information".

Minimal Python configuration file

This example shows a minimal configuration file that you can use with a Python managed runtime. For the assumptions that App Runner makes with a minimal configuration file, see the section called "Configuration file examples".

Python 3.11 uses the `pip3` and `python3` commands. For more information, see the example of an extended configuration file for Python 3.11 later in this topic.

Example apprunner.yaml

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

Extended Python configuration file

This example shows the use of all configuration keys with a Python managed runtime.

> **ⓘ Note**
>
> The runtime version that's used in these examples is *3.7.7*. You can replace it with a version you want to use. For latest supported Python runtime version, see the section called "Release information".
> Python 3.11 uses the `pip3` and `python3` commands. For more information, see the example of an extended configuration file for Python 3.11 later in this topic.

Example apprunner.yaml

```yaml
version: 1.0
runtime: python3
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-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"
```

Extended Python configuration file — Python 3.11 (uses revised build)

This example shows the use of all configuration keys with a Python 3.11 managed runtime in the apprunner.yaml. This example include a pre-run section, since this version of Python uses the revised App Runner build.

The `pre-run` parameter is only supported by the revised App Runner build. Do not insert this parameter in your configuration file if your application uses runtime versions that are supported by the original App Runner build. For more information, see Managed runtime versions and the App Runner build.

> ⓘ **Note**
>
> The runtime version that's used in these examples is *3.11*. You can replace it with a version you want to use. For latest supported Python runtime version, see the section called "Release information".

Example apprunner.yaml

```
version: 1.0
runtime: python311
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-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"
```

Complete Python application source

This example shows the source code for a complete Python application that you can deploy to a
Python runtime service.

Example requirements.txt

```
pyramid==2.0
```

Example server.py

```python
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 runtime release information

> ⚠️ **Important**
>
> App Runner will end the support for **Python 3.7** and **Python 3.8** on December 1, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

This topic lists the full details for the Python runtime versions that App Runner supports.

Supported runtime versions — revised App Runner build

Runtime name	Minor versions	Included packages
Python 3.11 (python311)	3.11.14	SQLite 3.50.2
	3.11.13	SQLite 3.50.2
	3.11.13	SQLite 3.50.1
	3.11.12	SQLite 3.50.0
	3.11.11	SQLite 3.49.1
	3.11.10	SQLite 3.46.1
	3.11.9	SQLite 3.46.1
	3.11.8	SQLite 3.45.2

Runtime name	Minor versions	Included packages
	3.11.7	SQLite 3.44.2

> ⓘ **Notes**
>
> - **Python 3.11** – We have specific recommendations for the build configuration of services that use the Python 3.11 managed runtime. For more information, see Callouts for specific runtime versions in the *Python platform* topic.
>
> - App Runner provides a revised build process for specific major runtimes that have been released more recently. Because of this you'll see references to *revised App Runner build* and *original App Runner build* in certain sections of this document. For more information, see Managed runtime versions and the App Runner build.

Supported runtime versions — original App Runner build

Runtime name	Minor versions	Included packages
Python 3 (python3)	3.8.20	SQLite 3.50.2
	3.8.20	SQLite 3.50.1
	3.8.20	SQLite 3.50.0
	3.8.16	SQLite 3.46.1
	3.8.15	SQLite 3.40.0
	3.7.16	SQLite 3.50.2
	3.7.16	SQLite 3.50.0
	3.7.15	SQLite 3.40.0
	3.7.10	SQLite 3.40.0

> ### ⓘ Note
>
> App Runner provides a revised build process for specific major runtimes that have been released more recently. Because of this you'll see references to *revised App Runner build* and *original App Runner build* in certain sections of this document. For more information, see [Managed runtime versions and the App Runner build](#).

Using the Node.js platform

> ### ⚠ Important
>
> App Runner will end the support for **Node.js 12**, **Node.js 14**, **Node.js 16** and **Node.js 18** on December 1, 2025. For recommendations and more information, see [the section called "End of support for managed runtime versions"](#).

The AWS App Runner Node.js platform provides managed runtimes. Each runtime makes it easy to build and run containers with web applications based on a Node.js version. When you use a Node.js runtime, App Runner starts with a managed Node.js runtime image. This image is based on the [Amazon Linux Docker image](#) and contains the runtime package for a version of Node.js and some tools. App Runner uses this managed runtime image as a base image, and adds your application code to build a Docker image. It then deploys this image to run your web service in a container.

You specify a runtime for your App Runner service when you [create a service](#) using the App Runner console or the [CreateService](#) API operation. You can also specify a runtime as part of your source code. Use the `runtime` keyword in a [App Runner configuration file](#) that you include in your code repository. The naming convention of a managed runtime is *<language-name><major-version>*.

For valid Node.js runtime names and versions, see [the section called "Release information"](#).

App Runner updates the runtime for your service to the latest version on every deployment or service update. If your application requires a specific version of a managed runtime, you can specify it using the `runtime-version` keyword in the [App Runner configuration file](#). You can lock to any level of version, including a major or minor version. App Runner only makes lower-level updates to the runtime of your service.

Version syntax for Node.js runtimes: *major*[.*minor*[.*patch*]]

For example: `22.14.0`

The following examples demonstrate version locking:

- `22.14` – Lock the major and minor versions. App Runner updates only patch versions.
- `22.14.0` – Lock to a specific patch version. App Runner doesn't update your runtime version.

Topics

- [Node.js runtime configuration](#)
- [Callouts for specific runtime versions](#)
- [Node.js runtime examples](#)
- [Node.js runtime release information](#)

Node.js runtime configuration

When you choose a managed runtime, you must also configure, as a minimum, build and run commands. You configure them while [creating](#) or [updating](#) your App Runner service. You can do this using one of the following methods:

- **Using the App Runner console** – Specify the commands in the **Configure build** section of the creation process or configuration tab.
- **Using the App Runner API** – Call the [CreateService](#) or [UpdateService](#) API operation. Specify the commands using the `BuildCommand` and `StartCommand` members of the [CodeConfigurationValues](#) data type.
- **Using a [configuration file](#)** – Specify one or more build commands in up to three build phases, and a single run command that serves to start your application. There are additional optional configuration settings.

Providing a configuration file is optional. When you create an App Runner service using the console or the API, you specify if App Runner gets your configuration settings directly when it's created or from a configuration file.

With Node.js runtimes specifically, you can also configure the build and runtime using a JSON file named `package.json` in the root of your source repository. Using this file, you can configure the Node.js engine version, dependency packages, and various commands (command line applications). Package managers such as npm or yarn interpret this file as input for their commands.

For example:

- **npm install** installs packages defined by the `dependencies` and `devDependencies` node in `package.json`.

- **npm start** or **npm run start** runs the command defined by the `scripts/start` node in `package.json`.

The following is an example `package.json` file.

package.json

```
{
  "name": "node-js-getting-started",
  "version": "0.3.0",
  "description": "A sample Node.js app using Express 4",
  "engines": {
    "node": "22.14.0"
  },
  "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"
  }
}
```

For more information about `package.json`, see Creating a package.json file on the *npm Docs* website.

> ⓘ **Tips**
>
> - If your `package.json` file defines a **start** command, you can use it as a **run** command in your App Runner configuration file, as the following example shows.

Example

package.json

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

apprunner.yaml

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

- When you run **npm install** in your development environment, npm creates the file `package-lock.json`. This file contains a snapshot of the package versions npm just installed. Thereafter, when npm installs dependencies, it uses these exact versions. If you install yarn it creates a `yarn.lock` file. Commit these files to your source code repository to ensure that your application is installed with the versions of dependencies that you developed and tested it with.

- You can also use an App Runner configuration file to configure the Node.js version and start command. When you do this, these definitions override the ones in `package.json`. A conflict between the node version in `package.json` and the `runtime-version` value in the App Runner configuration file causes the App Runner build phase to fail.

Callouts for specific runtime versions

Node.js 22 and Node.js 18 (revised App Runner build)

App Runner now runs an updated build process for applications based on the following runtime versions: Python 3.11, Node.js 22, and Node.js 18. If your application runs on either one of these runtime versions, see Managed runtime versions and the App Runner build for more information about the revised build process. Applications that use all other runtime versions are not affected, and they continue to use the original build process.

Node.js runtime examples

The following examples show App Runner configuration files for building and running a Node.js service.

> **ⓘ Note**
>
> The runtime version that's used in these examples is *22.14.0*. You can replace it with a version you want to use. For latest supported Node.js runtime version, see [the section called "Release information"](#).

Minimal Node.js configuration file

This example shows a minimal configuration file that you can use with a Node.js managed runtime. For the assumptions that App Runner makes with a minimal configuration file, see [the section called "Configuration file examples"](#).

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
build:
  commands:
    build:
      - npm install --production
run:
  command: node app.js
```

Extended Node.js configuration file

This example shows the use of all the configuration keys with a Node.js managed runtime.

> **ⓘ Note**
>
> The runtime version that's used in these examples is *22.14.0*. You can replace it with a version you want to use. For latest supported Node.js runtime version, see [the section called "Release information"](#).

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
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: 22.14.0
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Extended Node.js configuration file – Node.js 22 (uses revised build)

This example shows the use of all the configuration keys with a Node.js managed runtime in the `apprunner.yaml`. This example include a `pre-run` section, since this version of Node.js uses the revised App Runner build.

The `pre-run` parameter is only supported by the revised App Runner build. Do not insert this parameter in your configuration file if your application uses runtime versions that are supported by the original App Runner build. For more information, see Managed runtime versions and the App Runner build.

> ℹ️ **Note**
>
> The runtime version that's used in these examples is *22.14.0*. You can replace it with a version you want to use. For latest supported Node.js runtime version, see [the section called "Release information"](#).

Example apprunner.yaml

```
version: 1.0
runtime: nodejs22
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: 22.14.0
  pre-run:
    - node copy-global-files.js
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
  env:
    - name: MY_VAR_EXAMPLE
      value: "example"
```

Node.js app with Grunt

This example shows how to configure a Node.js application that's developed with Grunt. [Grunt](#) is a command line JavaScript task runner. It runs repetitive tasks and manages process automation to reduce human error. Grunt and Grunt plugins are installed and managed using npm. You configure Grunt by including the `Gruntfile.js` file in the root of your source repository.

Example package.json

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

Example Gruntfile.js

```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**
>
> The runtime version that's used in these examples is *22.14.0*. You can replace it with
> a version you want to use. For latest supported Node.js runtime version, see the section
> called "Release information".

```
version: 1.0
runtime: nodejs22
build:
  commands:
    pre-build:
      - npm install grunt grunt-cli
      - npm install --only=dev
      - npm run build
    build:
      - npm install --production
run:
  runtime-version: 22.14.0
  command: node app.js
  network:
    port: 8000
    env: APP_PORT
```

Node.js runtime release information

> **⚠ Important**
>
> App Runner will end the support for **Node.js 12**, **Node.js 14**, **Node.js 16** and **Node.js 18**
> on December 1, 2025. For recommendations and more information, see the section called
> "End of support for managed runtime versions".

> **ⓘ Note**
>
> App Runner's standard deprecation policy is to deprecate a runtime when any major
> component of the runtime reaches the end of community long-term support (LTS) and

This topic lists the full details for the Node.js runtime versions that App Runner supports.

Supported runtime versions — revised App Runner build

Runtime name	Minor versions	Included packages
Node.js 22 (nodejs22)	22.21.1	npm 10.9.4, yarn 1.22.22
	22.20.0	npm 10.9.3, yarn 1.22.22
	22.17.0	npm 10.9.2, yarn 1.22.22
	22.16.0	npm 10.9.2, yarn 1.22.22
	22.14.0	npm 10.9.2, yarn 1.22.22

> **ⓘ Note**
>
> App Runner provides a revised build process for specific major runtimes that have been released more recently. Because of this you'll see references to *revised App Runner build* and *original App Runner build* in certain sections of this document. For more information, see [Managed runtime versions and the App Runner build](#).

Supported runtime versions — revised App Runner build

Runtime name	Minor versions	Included packages
Node.js 18 (nodejs18)	18.20.8	npm 10.8.2, yarn 1.22.22
	18.20.7	npm 10.8.2, yarn 1.22.22
	18.20.6	npm 10.8.2, yarn 1.22.22

Runtime name	Minor versions	Included packages
	18.20.5	npm 10.8.2, yarn 1.22.22
	18.20.4	npm 10.7.0, yarn 1.22.22
	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 *

Supported runtime versions — original App Runner build

Runtime name	Minor versions	Included packages
Node.js 16 (nodejs16)	16.20.2	npm 8.19.4, yarn 1.22.22
	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 *

Runtime name	Minor versions	Included packages
	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 *

Using the Java platform

The AWS App Runner Java platform provides managed runtimes. Each runtime makes it easy to build and run containers with web applications based on a Java version. When you use a Java runtime, App Runner starts with a managed Java runtime image. This image is based on the Amazon Linux Docker image and contains the runtime package for a version of Java and some tools. App Runner uses this managed runtime image as a base image, and adds your application code to build a Docker image. It then deploys this image to run your web service in a container.

You specify a runtime for your App Runner service when you create a service using the App Runner console or the CreateService API operation. You can also specify a runtime as part of your source code. Use the `runtime` keyword in a App Runner configuration file that you include in your code repository. The naming convention of a managed runtime is *<language-name><major-version>*.

At this time, all the supported Java runtimes are based on Amazon Corretto. For valid Java runtime names and versions, see the section called "Release information".

App Runner updates the runtime for your service to the latest version on every deployment or service update. If your application requires a specific version of a managed runtime, you can specify it using the `runtime-version` keyword in the App Runner configuration file. You can lock to any level of version, including a major or minor version. App Runner only makes lower-level updates to the runtime of your service.

Version syntax for Amazon Corretto runtimes:

Runtime	Syntax	Example
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

The following examples demonstrate version locking:

- `11.0.13` – Lock the Open JDK update version. App Runner updates only Open JDK and Amazon Corretto lower-level builds.

- `11.0.13.08.1` – Lock to a specific version. App Runner doesn't update your runtime version.

Topics

- Java runtime configuration
- Java runtime examples
- Java runtime release information

Java runtime configuration

When you choose a managed runtime, you must also configure, as a minimum, build and run commands. You configure them while creating or updating your App Runner service. You can do this using one of the following methods:

- **Using the App Runner console** – Specify the commands in the **Configure build** section of the creation process or configuration tab.

- **Using the App Runner API** – Call the CreateService or UpdateService API operation. Specify the commands using the BuildCommand and StartCommand members of the CodeConfigurationValues data type.

- **Using a [configuration file](#)** – Specify one or more build commands in up to three build phases, and a single run command that serves to start your application. There are additional optional configuration settings.

Providing a configuration file is optional. When you create an App Runner service using the console or the API, you specify if App Runner gets your configuration settings directly when it's created or from a configuration file.

Java runtime examples

The following examples show App Runner configuration files for building and running a Java service. The last example is the source code for a complete Java application that you can deploy to a Corretto 11 runtime service.

> **ⓘ Note**
>
> The runtime version that's used in these examples is *11.0.13.08.1*. You can replace it with a version you want to use. For latest supported Java runtime version, see [the section called "Release information"](#).

Minimal Corretto 11 configuration file

This example shows a minimal configuration file that you can use with a Corretto 11 managed runtime. For the assumptions that App Runner makes with a minimal configuration file, see .

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 .
```

Extended Corretto 11 configuration file

This example shows how you can use all the configuration keys with a Corretto 11 managed runtime.

> **ⓘ Note**
>
> The runtime version that's used in these examples is *11.0.13.08.1*. You can replace it with a version you want to use. For latest supported Java runtime version, see the section called "Release information".

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"
```

Complete Corretto 11 application source

This example shows the source code for a complete Java application that you can deploy to a Corretto 11 runtime service.

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

```
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/HelloWorld/Main.java

```
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
<?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 runtime release information

This topic lists the full details for the Java runtime versions that App Runner supports.

Supported runtime versions — original App Runner build

Runtime name	Minor versions	Included packages
Corretto 11 (corretto11)	11.0.28.6.1	Maven 3.9.11, Gradle 6.9.4
	11.0.27.6.1	Maven 3.9.10, Gradle 6.9.4
	11.0.27.6.1	Maven 3.9.9, Gradle 6.9.4
	11.0.26.4.1	Maven 3.9.9, Gradle 6.9.4
	11.0.25.9.1	Maven 3.9.9, Gradle 6.9.4

Runtime name	Minor versions	Included packages
	11.0.24.8.1	Maven 3.9.9, Gradle 6.9.4
	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
	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)	8.472.08.1	Maven 3.9.11, Gradle 6.9.4
	8.462.08.1	Maven 3.9.11, Gradle 6.9.4
	8.452.09.2	Maven 3.9.10, Gradle 6.9.4
	8.452.09.2	Maven 3.9.9, Gradle 6.9.4
	8.452.09.1	Maven 3.9.9, Gradle 6.9.4
	8.442.06.1	Maven 3.9.9, Gradle 6.9.4
	8.432.06.1	Maven 3.9.9, Gradle 6.9.4
	8.422.05.1	Maven 3.9.9, Gradle 6.9.4
	8.412.08.1	Maven 3.9.8, Gradle 6.9.4

Runtime name	Minor versions	Included packages
	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 provides a revised build process for specific major runtimes that have been released more recently. Because of this you'll see references to *revised App Runner build* and *original App Runner build* in certain sections of this document. For more information, see Managed runtime versions and the App Runner build.

Using the .NET platform

> ⚠ **Important**
>
> App Runner will end the support for **.NET 6** on December 1, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

The AWS App Runner .NET platform provides managed runtimes. Each runtime makes it easy to build and run containers with web applications based on a .NET version. When you use a .NET runtime, App Runner starts with a managed .NET runtime image. This image is based on the

[Amazon Linux Docker image](#) and contains the runtime package for a version of .NET and some tools and popular dependency packages. App Runner uses this managed runtime image as a base image, and adds your application code to build a Docker image. It then deploys this image to run your web service in a container.

You specify a runtime for your App Runner service when you [create a service](#) using the App Runner console or the [CreateService](#) API operation. You can also specify a runtime as part of your source code. Use the `runtime` keyword in a [App Runner configuration file](#) that you include in your code repository. The naming convention of a managed runtime is *<language-name><major-version>*.

For valid .NET runtime names and versions, see [the section called "Release information"](#).

App Runner updates the runtime for your service to the latest version on every deployment or service update. If your application requires a specific version of a managed runtime, you can specify it using the `runtime-version` keyword in the [App Runner configuration file](#). You can lock to any level of version, including a major or minor version. App Runner only makes lower-level updates to the runtime of your service.

Version syntax for .NET runtimes: *major*[.*minor*[.*patch*]]

For example: `6.0.9`

The following examples demonstrate version locking:

- `6.0` – Lock the major and minor versions. App Runner updates only patch versions.
- `6.0.9` – Lock to a specific patch version. App Runner doesn't update your runtime version.

Topics

- [.NET runtime configuration](#)
- [.NET runtime examples](#)
- [.NET runtime release information](#)

.NET runtime configuration

When you choose a managed runtime, you must also configure, as a minimum, build and run commands. You configure them while [creating](#) or [updating](#) your App Runner service. You can do this using one of the following methods:

- **Using the App Runner console** – Specify the commands in the **Configure build** section of the creation process or configuration tab.

- **Using the App Runner API** – Call the CreateService or UpdateService API operation. Specify the commands using the `BuildCommand` and `StartCommand` members of the CodeConfigurationValues data type.

- **Using a configuration file** – Specify one or more build commands in up to three build phases, and a single run command that serves to start your application. There are additional optional configuration settings.

Providing a configuration file is optional. When you create an App Runner service using the console or the API, you specify if App Runner gets your configuration settings directly when it's created or from a configuration file.

.NET runtime examples

The following examples show App Runner configuration files for building and running a .NET service. The last example is the source code for a complete .NET application that you can deploy to a .NET runtime service.

> **Note**
>
> The runtime version that's used in these examples is *6.0.9*. You can replace it with a version you want to use. For latest supported .NET runtime version, see the section called "Release information".

Minimal .NET configuration file

This example shows a minimal configuration file that you can use with a .NET managed runtime. For the assumptions that App Runner makes with a minimal configuration file, see the section called "Configuration file examples".

Example apprunner.yaml

```
version: 1.0
runtime: dotnet6
build:
```

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

Extended .NET configuration file

This example shows the use of all configuration keys with a .NET managed runtime.

> **ⓘ Note**
>
> The runtime version that's used in these examples is *6.0.9*. You can replace it with a
> version you want to use. For latest supported .NET runtime version, see the section called
> "Release information".

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"
```

Complete .NET application source

This example shows the source code for a complete .NET application that you can deploy to a .NET runtime service.

> **ⓘ Note**
>
> - Run following command to create a simple .NET 6 web app: `dotnet new web --name HelloWorldDotNetApp -f net6.0`
> - Add the `apprunner.yaml` to the created .NET 6 web app.

Example HelloWorldDotNetApp

```yaml
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 runtime release information

> **⚠ Important**
>
> App Runner will end the support for **.NET 6** on December 1, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

This topic lists the full details for the .NET runtime versions that App Runner supports.

Supported runtime versions — original App Runner build

Runtime name	Minor versions	Included packages
.NET 6 (dotnet6)	6.0.36	.NET SDK 6.0.428
	6.0.33	.NET SDK 6.0.425
	6.0.32	.NET SDK 6.0.424
	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
	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

Runtime name	Minor versions	Included packages
	6.0.10	.NET SDK 6.0.402
	6.0.9	.NET SDK 6.0.401

> ⓘ **Note**
>
> App Runner provides a revised build process for specific major runtimes that have been released more recently. Because of this you'll see references to *revised App Runner build* and *original App Runner build* in certain sections of this document. For more information, see Managed runtime versions and the App Runner build.

Using the PHP platform

> ⚠ **Important**
>
> App Runner will end the support for **PHP 8.1** on December 31, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

The AWS App Runner PHP platform provides managed runtimes. You can use each runtime to build and run containers with web applications based on a PHP version. When you use a PHP runtime, App Runner starts with a managed PHP runtime image. This image is based on the Amazon Linux Docker image and contains the runtime package for a version of PHP and some tools. App Runner uses this managed runtime image as a base image, and adds your application code to build a Docker image. It then deploys this image to run your web service in a container.

You specify a runtime for your App Runner service when you create a service using the App Runner console or the CreateService API operation. You can also specify a runtime as part of your source code. Use the `runtime` keyword in a App Runner configuration file that you include in your code repository. The naming convention of a managed runtime is `<language-name><major-version>`.

For valid PHP runtime names and versions, see the section called "Release information".

App Runner updates the runtime for your service to the latest version on every deployment or service update. If your application requires a specific version of a managed runtime, you can specify it using the `runtime-version` keyword in the [App Runner configuration file](#). You can lock to any level of version, including a major or minor version. App Runner only makes lower-level updates to the runtime of your service.

Version syntax for PHP runtimes: *major*[.*minor*[.*patch*]]

For example: `8.1.10`

The following are examples of version locking:

- `8.1` – Lock the major and minor versions. App Runner updates only patch versions.
- `8.1.10` – Lock to a specific patch version. App Runner doesn't update your runtime version.

> ⚠️ **Important**
>
> If you'd like to specify the code repository [source directory](#) for your App Runner service in a location other than the default repository root directory, your PHP managed runtime version must be PHP `8.1.22` or later. PHP runtime versions prior to `8.1.22` may only use the default root source directory.

Topics

- [PHP runtime configuration](#)
- [Compatibility](#)
- [PHP runtime examples](#)
- [PHP runtime release information](#)

PHP runtime configuration

When you choose a managed runtime, you must also configure, as a minimum, build and run commands. You configure them while [creating](#) or [updating](#) your App Runner service. You can do this using one of the following methods:

- **Using the App Runner console** – Specify the commands in the **Configure build** section of the creation process or configuration tab.

- **Using the App Runner API** – Call the [CreateService](#) or [UpdateService](#) API operation. Specify the commands using the `BuildCommand` and `StartCommand` members of the [CodeConfigurationValues](#) data type.

- **Using a [configuration file](#)** – Specify one or more build commands in up to three build phases, and a single run command that serves to start your application. There are additional optional configuration settings.

Providing a configuration file is optional. When you create an App Runner service using the console or the API, you specify if App Runner gets your configuration settings directly when it's created or from a configuration file.

Compatibility

You can run your App Runner services on PHP platform using one of the following web servers:

- Apache HTTP Server
- NGINX

Apache HTTP Server and NGINX are compatible with PHP-FPM. You can start the *Apache HTTP Server* and *NGINX* by using one of the following:

- [Supervisord](#) - For more information about running a *supervisord*, see [Running supervisord](#).
- Startup script

For examples on how to configure your App Runner service with PHP platform using *Apache HTTP Server* or *NGINX*, see [the section called "Complete PHP application source"](#).

File Structure

The `index.php` must be installed in the `public` folder under the `root` directory of the web server.

> **ⓘ Note**
>
> We recommend that the `startup.sh` or `supervisord.conf` files be stored in the root directory of the web server. Make sure that the `start` command points to the location where the `startup.sh` or `supervisord.conf` files are stored.

The following is an example of the file structure if you are using *supervisord*.

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

The following is an example of the file structure if you are using *startup script*.

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

We recommend that you store these file structures in the code repository source directory that's designated for the App Runner service.

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

> ⚠️ **Important**
>
> If you'd like to specify the code repository source directory for your App Runner service in a location other than the default repository root directory, your PHP managed runtime version must be PHP `8.1.22` or later. PHP runtime versions prior to `8.1.22` may only use the default root source directory.
>
> App Runner updates the runtime for your service to the latest version on every deployment or service update. Your service will use the most recent runtimes by default, unless you specified version locking using the `runtime-version` keyword in the App Runner configuration file.

PHP runtime examples

The following are examples of App Runner configuration files that are used for building and running a PHP service.

Minimal PHP configuration file

The following example is a minimal configuration file that you can use with a PHP managed runtime. For more information about a minimal configuration file, see the section called "Configuration file examples".

Example apprunner.yaml

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

Extended PHP configuration file

The following example uses all the configuration keys with a PHP managed runtime.

> **ⓘ Note**
>
> The runtime version that's used in these examples is *8.1.10*. You can replace it with a version you want to use. For latest supported PHP runtime version, see the section called "Release information".

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"
```

Complete PHP application source

The following examples are of PHP application source code that you can use to deploy to a PHP runtime service using *Apache HTTP Server* or *NGINX*. These examples assume that you use the default file structure.

Running PHP platform with Apache HTTP Server using supervisord

Example File structure

> **ⓘ Note**
>
> - The `supervisord.conf` file can be stored anywhere in the repository. Make sure that the `start` command points to where the `supervisord.conf` file is stored.
>
> - The `index.php` must be installed in the `public` folder under the `root` directory.

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

Example supervisord.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>
```

Running PHP platform with Apache HTTP Server using startup script

Example File structure

> ℹ **Note**
>
> - The `startup.sh` file can be stored anywhere in the repository. Make sure that the `start` command points to where the `startup.sh` file is stored.
>
> - The `index.php` must be installed in the `public` folder under the `root` directory.

```
/
## 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**
>
> - Make sure to save the `startup.sh` file as an executable before you commit it to a Git repository. Use `chmod +x startup.sh` to set execute permission on your `startup.sh` file.
>
> - If you don't save the `startup.sh` file as an executable, enter `chmod +x startup.sh` as the `build` command in your `apprunner.yaml` file.

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>
```

Running PHP platform with NGINX using supervisord

Example File structure

> **ⓘ Note**
>
> - The `supervisord.conf` file can be stored anywhere in the repository. Make sure that the `start` command points to where the `supervisord.conf` file is stored.
>
> - The `index.php` must be installed in the `public` folder under the `root` directory.

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

Example supervisord.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>
```

Running PHP platform with NGINX using startup script

Example File structure

> **ⓘ Note**
>
> - The `startup.sh` file can be stored anywhere in the repository. Make sure that the `start` command points to where the `startup.sh` file is stored.
>
> - The `index.php` must be installed in the `public` folder under the `root` directory.

```
/
## public/
```

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

Example startup.sh

```bash
#!/bin/bash

set -o monitor

trap exit SIGCHLD

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

# Start php-fpm
php-fpm -F &

wait
```

> ⓘ **Note**
>
> - Make sure to save the `startup.sh` file as an executable before you commit it to a Git repository. Use `chmod +x startup.sh` to set execute permission on your `startup.sh` file.
>
> - If you don't save the `startup.sh` file as an executable, enter `chmod +x startup.sh` as the `build` command in your `apprunner.yaml` file.

Example apprunner.yaml

```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 runtime release information

> ⚠️ **Important**
>
> App Runner will end the support for **PHP 8.1** on December 31, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

This topic lists the full details for the PHP runtime versions that App Runner supports.

Supported runtime versions — original App Runner build

Runtime name	Minor versions	Included packages
PHP 8.1 (php81)	8.1.33	
	8.1.32	
	8.1.31	
	8.1.29	
	8.1.28	
	8.1.27	

Runtime name	Minor versions	Included packages
	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 provides a revised build process for specific major runtimes that have been released more recently. Because of this you'll see references to *revised App Runner build* and *original App Runner build* in certain sections of this document. For more information, see Managed runtime versions and the App Runner build.

Using the Ruby platform

> **⚠ Important**
>
> App Runner will end the support for **Ruby 3.1** on December 1, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

The AWS App Runner Ruby platform provides managed runtimes. Each runtime makes it easy to build and run containers with web applications based on a Ruby version. When you use a Ruby runtime, App Runner starts with a managed Ruby runtime image. This image is based on the Amazon Linux Docker image and contains the runtime package for a version of Ruby and some tools. App Runner uses this managed runtime image as a base image, and adds your application code to build a Docker image. It then deploys this image to run your web service in a container.

You specify a runtime for your App Runner service when you create a service using the App Runner console or the CreateService API operation. You can also specify a runtime as part of your source code. Use the `runtime` keyword in a App Runner configuration file that you include in your code repository. The naming convention of a managed runtime is `<language-name><major-version>`.

For valid Ruby runtime names and versions, see the section called "Release information".

App Runner updates the runtime for your service to the latest version on every deployment or service update. If your application requires a specific version of a managed runtime, you can specify it using the `runtime-version` keyword in the App Runner configuration file. You can lock to any level of version, including a major or minor version. App Runner only makes lower-level updates to the runtime of your service.

Version syntax for Ruby runtimes: `major[.minor[.patch]]`

For example: `3.1.2`

The following examples demonstrate version locking:

- `3.1` – Lock the major and minor versions. App Runner updates only patch versions.
- `3.1.2` – Lock to a specific patch version. App Runner doesn't update your runtime version.

Topics

- [Ruby runtime configuration](#)
- [Ruby runtime examples](#)
- [Ruby runtime release information](#)

Ruby runtime configuration

When you choose a managed runtime, you must also configure, as a minimum, build and run commands. You configure them while [creating](#) or [updating](#) your App Runner service. You can do this using one of the following methods:

- **Using the App Runner console** – Specify the commands in the **Configure build** section of the creation process or configuration tab.
- **Using the App Runner API** – Call the [CreateService](#) or [UpdateService](#) API operation. Specify the commands using the `BuildCommand` and `StartCommand` members of the [CodeConfigurationValues](#) data type.
- **Using a [configuration file](#)** – Specify one or more build commands in up to three build phases, and a single run command that serves to start your application. There are additional optional configuration settings.

Providing a configuration file is optional. When you create an App Runner service using the console or the API, you specify if App Runner gets your configuration settings directly when it's created or from a configuration file.

Ruby runtime examples

The following examples show App Runner configuration files for building and running a Ruby service.

Minimal Ruby configuration file

This example shows a minimal configuration file that you can use with a Ruby managed runtime. For the assumptions that App Runner makes with a minimal configuration file, see [the section called "Configuration file examples"](#).

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
```

Extended Ruby configuration file

This example shows the use of all the configuration keys with a Ruby managed runtime.

> ⓘ **Note**
>
> The runtime version that's used in these examples is *3.1.2*. You can replace it with a version you want to use. For latest supported Ruby runtime version, see the section called "Release information".

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"
```

Complete Ruby application source

These examples shows the source code for a complete Ruby application that you can deploy to a Ruby runtime service.

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 runtime release information

⚠️ **Important**

App Runner will end the support for **Ruby 3.1** on December 1, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

This topic lists the full details for the Ruby runtime versions that App Runner supports.

Supported runtime versions — original App Runner build

Runtime name	Minor versions	Included packages
Ruby 3.1 (ruby31)	3.1.7	SQLite 3.50.2
	3.1.7	SQLite 3.50.1
	3.1.7	SQLite 3.50.0
	3.1.6	SQLite 3.49.1
	3.1.4	SQLite 3.46.0
	3.1.3	SQLite 3.41.0
	3.1.2	SQLite 3.39.4

ⓘ **Note**

App Runner provides a revised build process for specific major runtimes that have been released more recently. Because of this you'll see references to *revised App Runner build* and *original App Runner build* in certain sections of this document. For more information, see Managed runtime versions and the App Runner build.

Using the Go platform

> ⚠️ **Important**
>
> App Runner will end the support for **Go 1.18** on December 1, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

The AWS App Runner Go platform provides managed runtimes. Each runtime makes it easy to build and run containers with web applications based on a Go version. When you use a Go runtime, App Runner starts with a managed Go runtime image. This image is based on the Amazon Linux Docker image and contains the runtime package for a version of Go and some tools. App Runner uses this managed runtime image as a base image, and adds your application code to build a Docker image. It then deploys this image to run your web service in a container.

You specify a runtime for your App Runner service when you create a service using the App Runner console or the CreateService API operation. You can also specify a runtime as part of your source code. Use the `runtime` keyword in a App Runner configuration file that you include in your code repository. The naming convention of a managed runtime is *<language-name><major-version>*.

For valid Go runtime names and versions, see the section called "Release information".

App Runner updates the runtime for your service to the latest version on every deployment or service update. If your application requires a specific version of a managed runtime, you can specify it using the `runtime-version` keyword in the App Runner configuration file. You can lock to any level of version, including a major or minor version. App Runner only makes lower-level updates to the runtime of your service.

Version syntax for Go runtimes: *major*[.*minor*[.*patch*]]

For example: `1.18.7`

The following examples demonstrate version locking:

- `1.18` – Lock the major and minor versions. App Runner updates only patch versions.
- `1.18.7` – Lock to a specific patch version. App Runner doesn't update your runtime version.

Topics

- [Go runtime configuration](#)
- [Go runtime examples](#)
- [Go runtime release information](#)

Go runtime configuration

When you choose a managed runtime, you must also configure, as a minimum, build and run commands. You configure them while [creating](#) or [updating](#) your App Runner service. You can do this using one of the following methods:

- **Using the App Runner console** – Specify the commands in the **Configure build** section of the creation process or configuration tab.

- **Using the App Runner API** – Call the [CreateService](#) or [UpdateService](#) API operation. Specify the commands using the `BuildCommand` and `StartCommand` members of the [CodeConfigurationValues](#) data type.

- **Using a [configuration file](#)** – Specify one or more build commands in up to three build phases, and a single run command that serves to start your application. There are additional optional configuration settings.

Providing a configuration file is optional. When you create an App Runner service using the console or the API, you specify if App Runner gets your configuration settings directly when it's created or from a configuration file.

Go runtime examples

The following examples show App Runner configuration files for building and running a Go service.

Minimal Go configuration file

This example shows a minimal configuration file that you can use with a Go managed runtime. For the assumptions that App Runner makes with a minimal configuration file, see [the section called "Configuration file examples"](#).

Example apprunner.yaml

```
version: 1.0
```

```
runtime: go1
build:
  commands:
    build:
      - go build main.go
run:
  command: ./main
```

Extended Go configuration file

This example shows the use of all the configuration keys with a Go managed runtime.

> ⓘ **Note**
>
> The runtime version that's used in these examples is *1.18.7*. You can replace it with a version you want to use. For latest supported Go runtime version, see the section called "Release information".

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"
```

Complete Go application source

These examples shows the source code for a complete Go application that you can deploy to a Go runtime service.

Example main.go

```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

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

Go runtime release information

> ⚠️ **Important**
>
> App Runner will end the support for **Go 1.18** on December 1, 2025. For recommendations and more information, see the section called "End of support for managed runtime versions".

This topic lists the full details for the Go runtime versions that App Runner supports.

Supported runtime versions — original App Runner build

Runtime name	Minor versions	Included packages
Go 1 (go1)	1.18.10	
	1.18.9	
	1.18.8	
	1.18.7	

> ℹ️ **Note**
>
> App Runner provides a revised build process for specific major runtimes that have been released more recently. Because of this you'll see references to *revised App Runner build* and *original App Runner build* in certain sections of this document. For more information, see Managed runtime versions and the App Runner build.

Developing application code for App Runner

This chapter discusses runtime information and development guidelines that you should consider when developing or migrating application code for deployment to AWS App Runner.

Runtime information

Whether you provide a container image or App Runner builds one for you, App Runner runs your application code in a container instance. Here are a few key aspects of the container instance runtime environment.

- **Framework support** – App Runner supports any image that implements a web application. It's agnostic to the programming language that you choose and to the web application server or framework that you use, if you use any. For your convenience, we provide platform-specific managed runtimes for various programming platforms, to streamline the application build process and abstract image creation.

- **Web requests** – App Runner provides support for HTTP 1.0 and HTTP 1.1 to the container instances. For more information about configuring your service, see the section called "Configuration". You don't need to implement handling of HTTPS secure traffic. App Runner redirects all incoming HTTP requests to corresponding HTTPS endpoints. You don't need to configure any settings to enable redirecting the HTTP web requests. App Runner terminates the TLS before passing requests to your application container instance.

> **ⓘ Note**
>
> - There is a total of 120 seconds request timeout limit on the HTTP requests. The 120 seconds include the time the application takes to read the request, including the body, and complete writing the HTTP response.
>
> - The request read and response timeout limit is contingent on the applications that you use. These applications may have their own internal timeouts, such as the HTTP server for Python, Gunicorn, has a 30 second default timeout limit. In such cases, the application's timeout limit overrides the App Runner 120 second timeout limit.
>
> - You don't need to configure TLS cipher suites or any other parameters as App Runner being a fully managed service, manages the TLS termination for you.

- **Stateless apps** – Currently App Runner doesn't support a stateful app. Hence, App Runner doesn't guarantee state persistence beyond the duration of processing a single incoming web request.

- **Storage** – App Runner automatically scales the instances up or down for your App Runner application in accordance to incoming traffic volume. You can configure Auto scaling options for your App Runner application. Since the number of currently active instances processing the web requests is based on the incoming traffic volume, App Runner cannot guarantee that the files can persist beyond the processing of a single request. Hence, App Runner implements the file system in your container instance as ephemeral storage, which entails that the files are transient. For example, the files don't persist when you pause and resume your App Runner service.

  App Runner provides you with 3 GB of ephemeral storage and uses a part of the 3 GB of ephemeral storage for its pulled, compressed, and the uncompressed container image on the instance. The remaining ephemeral storage can be used by your App Runner service. However, this is *not a permanent storage* owing to its stateless nature.

  > **ⓘ Note**
  >
  > There could be scenarios when the storage files do persist across requests. For example, if the next request lands on the same instance the storage files will persist. The persistence of storage files across requests can be useful in certain situations. For example, when handling a request, you can cache files that your application downloads if future requests might need them. This might speed up future request handling, but can't guarantee the speed gains. Your code shouldn't assume that a file that has been downloaded in a previous request still exists.
  > For guaranteed caching using a high throughput, low latency in-memory data store, use a service such as Amazon ElastiCache.

- **Environment variables** – By default, App Runner makes the PORT environment variable available in your container instance. You can configure the variable value with port information, and add custom environment variables and values. You can also reference sensitive data stored in *AWS Secrets Manager* or *AWS Systems Manager Parameter Store* as environment variables. For more information about creating environment variables, see *Reference Environment variables*.

- **Instance role** – If your application code makes calls to any AWS services, using the service APIs or one of the AWS SDKs, create an instance role using AWS Identity and Access Management (IAM). Then, attach it to your App Runner service when you create it. Include all AWS service action

permissions that your code requires in your instance role. For more information, see the section called "Instance role".

Code development guidelines

Consider these guidelines when developing code for an App Runner web application.

- **Patching container images** – When providing container images, you are responsible for regularly updating and patching these images. While App Runner manages the infrastructure, you should ensure the security and up-to-date status of the provided container images. For more information, see the AWS App Runner Documentation

- **Design stateless code** – Design the web application you deploy to your App Runner service to be stateless. Your code should assume that no state persists beyond the duration of processing a single incoming web request.

- **Delete temporary files** – When you create files, they're stored on a file system, and take up part of the storage allocation of your service. To avoid out-of-storage errors, don't keep temporary files for extended periods. Balance storage size with request handling speed when making file caching decisions.

- **Instance startup** – App Runner provides five minutes of instance startup time. Your instance must listen for requests on their configured listening ports and be healthy within five minutes of their startup. During the startup time, App Runner instances are allocated virtual CPU (vCPU) based on your vCPU configuration. For more information about available vCPU configuration, see the section called "App Runner supported configurations".

  After the instance successfully starts up, it goes into an idle state and waits for requests. You pay based on the instance startup duration, with the minimum charge of one minute per instance start. For information about pricing, see AWS App Runner pricing.

Using the App Runner console

Use the AWS App Runner console to create, manage, and monitor your App Runner services and related resources, such as connected accounts. You can view existing services, create new ones, and configure a service. You can view the status of an App Runner service as well as view logs, monitor activity, and track metrics. You can also navigate to the website of your service or to your source repository.

The following sections describe the layout and functionality of the console, and point you to related information.

Overall console layout

The App Runner console has three areas. From left to right:

- **Navigation pane** – A side pane that can be collapsed or expanded. Use it to choose the top-level console page you want to use.

- **Content pane** – The main part of the console page. Use it to view information and perform your tasks.

- **Help pane** – A side pane for more information. Expand it to get help about the page you're on. Or choose any **Info** link on a console page to get contextual help.

The Services page

The **Services** page lists App Runner services in your account. You can scope the list down by using the filter text box.

To get to the Services page

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**.

Things you can do here:

- Create an App Runner service. For more information, see the section called "Creation".

- Choose a service name to go to the service dashboard console page.

- Choose a service domain to open the service web app page.

The service dashboard page

You can view information about an App Runner service and manage it from the service dashbaord page. At the top of the page, you can see the service name.

To get to the service dashboard, navigate to the **Services** page (see previous section), and then choose your App Runner service.

The **Service overview** section provides basic details about the App Runner service and your application. Things you can do here:

- View service details such as status, health, and ARN.

- Navigate to the **Default domain**—the domain that App Runner provides for the web application running in your service. This is a subdomain in the `awsapprunner.com` domain owned by App Runner.

- Navigate to the source repository deployed to the service.

- Start a source repository deployment to your service.

- Pause, resume, and delete your service.

The tabs below the service overview are for service [management](#) and [observability](#).

The Connected accounts page

The **Connected accounts** page lists App Runner connections to source code repository providers in your account. You can scope the list down by using the filter text box. For more information about connected accounts, see the section called "Connections".

To get to the Connected accounts page

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Connected accounts**.
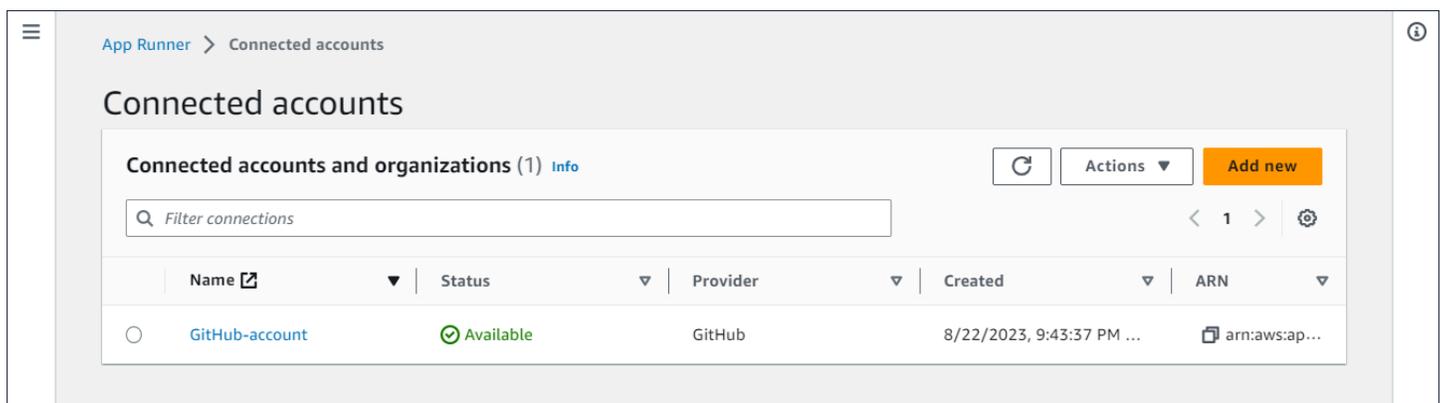
Things you can do here:

- View a list of repository provider connections in your account. To scope the list down, enter any text in the filter text box.

- Choose a connection name to go to the related provider account or organization.

- Select a connection to complete the handshake for a connection that you just established (as part of creating a service), or to delete the connection.

The Auto scaling configurations page

The **Auto scaling configurations** page lists the auto scaling configurations that you have set up in your account. You can configure a few parameters to adjust auto scaling behavior and save them in different configurations that you can later assign to one or more App Runner services. You can scope the list down by using the filter text box. For more information about auto scaling configurations, see Manage auto scaling for a service.

To get to the Auto scaling configuration page

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Auto scaling configuration**.

Things you can do here:

- View the list of existing auto scaling configurations in your account.
- Create a new auto scaling configuration or a revision for an existing one.
- Set an auto scaling configuration as the default for new services you create.
- Delete a configuration.
- Select the name of a configuration to navigate to the **Auto scaling revisions** panel to manage revisions.

Managing your App Runner service

This chapter describes how to manage your AWS App Runner service. In this chapter, you learn how to manage the life cycle of your service: create, configure, and delete a service, deploy new application versions to your service, and control the availability of your web service by pausing and resuming your service. You also learn how to manage other aspects of your service, like connections and auto scaling.

Topics

- [Creating an App Runner service](#)
- [Rebuilding a failed App Runner service](#)
- [Deploying a new application version to App Runner](#)
- [Configuring an App Runner service](#)
- [Managing App Runner connections](#)
- [Managing App Runner automatic scaling](#)
- [Managing custom domain names for an App Runner service](#)
- [Pausing and resuming an App Runner service](#)
- [Deleting an App Runner service](#)

Creating an App Runner service

AWS App Runner automates transitioning from a container image or a source code repository to a running web service that scales automatically. You point App Runner to your source image or code, specifying only a small number of required settings. App Runner builds your application if needed, provisions compute resources, and deploys your application to run on them.

When you create a service, App Runner creates a *service* resource. In some cases, you might need to provide a *connection* resource. If you use the App Runner console, the console implicitly creates the connection resource. For more information about App Runner resource types, see [the section called "App Runner resources"](#). These resource types have quotas that are associated with your account in each AWS Region. For more information, see [the section called "App Runner resource quotas"](#).

There are subtle differences in the procedure for creating a service depending on the source type and provider. This topic covers different procedures for creating these source types so that you can

follow whichever is suitable for your situation. For starting a basic procedure with a code example, see *Getting started*.

Prerequisites

Before you create your App Runner service, make sure to complete the following actions:

* Complete the setup steps in *Setting up*.

* Make sure that your application source ready. You can use either a code repository in GitHub, Bitbucket, or a container image in Amazon Elastic Container Registry (Amazon ECR) to create an App Runner service.

Create a service

This section walks through the creation process for the two App Runner service types: based on source code, and based on a container image.

> ℹ️ **Note**
>
> If you create an outbound traffic VPC connector for a service, the service startup process that follows will experience a one-time latency. You can set this configuration for a new service when you create it, or afterward, with a service update. For more information, see One-time latency in the *Networking with App Runner* chapter of this guide.

Create a service from a code repository

The following sections show how to create an App Runner service when your source is a code repository in GitHub or Bitbucket. When you use a code repository, App Runner must connect to the provider organization or account. Therefore, you need to help establish this connection. For more information about App Runner connections, see the section called "Connections".

When you create the service, App Runner builds a Docker image that contains your application code and dependencies. It then launches a service that runs a container instance of this image.

Creating a service from code using the App Runner console

To create an App Runner service using the console

1. Configure your source code.

a. Open the App Runner console, and in the **Regions** list, select your AWS Region.

b. If the AWS account doesn't have any App Runner services yet, the console home page is displayed. Choose **Create an App Runner service**.
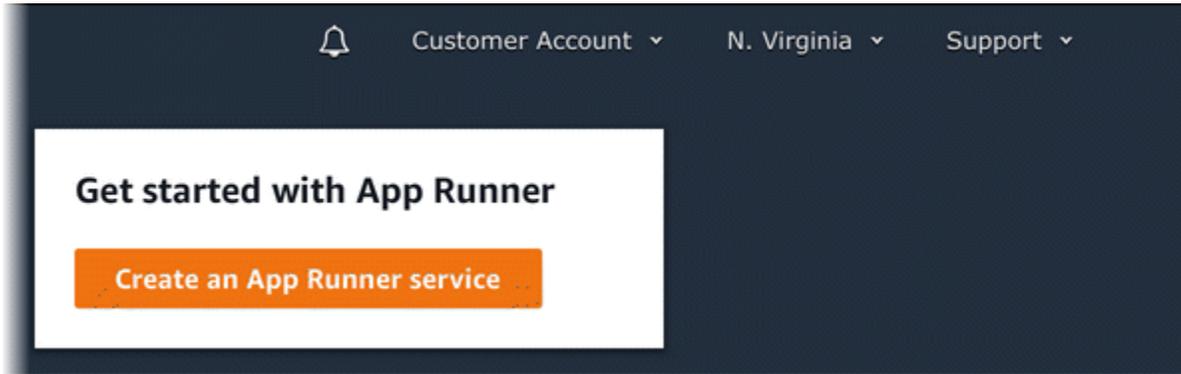
If the AWS account has existing services, the **Services** page with a list of your services is displayed. Choose **Create service**.

c. On the **Source and deployment** page, in the **Source** section, for **Repository type**, choose **Source code repository**.

d. Select a **Provider Type**. Choose either **GitHub** or **Bitbucket**.

e. Next select an account or organization for the Provider that you've used before, or choose **Add new**. Then, go through the process of providing your code repository credentials and choosing an account or organization to connect to.

f. For **Repository**, select the repository that contains your application code.

g. For **Branch**, select the branch that you want to deploy.

h. For **Source directory**, enter the directory in the source repository that stores your application code and configuration files.

> **ⓘ Note**
>
> The build and start commands execute from the source directory that you specify. App Runner handles the path as absolute from root. If you don't specify a value here, the directory defaults to the repository root.

2. Configure your deployments.

   a. In the **Deployment settings** section, choose **Manual** or **Automatic**.

      For more information about deployment methods, see the section called "Deployment methods".

   b. Choose **Next**.

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. Configure the application build.

    a. On the **Configure build** page, for **Configuration file**, choose **Configure all settings here** if your repository doesn't contain an App Runner configuration file, or **Use a configuration file** if it does.

        > ⓘ **Note**
        >
        > An App Runner configuration file is a way to maintain your build configuration as part of your application source. When you provide one, App Runner reads some values from the file and doesn't let you set them in the console.

    b. Provide the following build settings:

        - **Runtime** – Choose a specific managed runtime for your application.

        - **Build command** – Enter a command that builds your application from its source code. This might be a language-specific tool or a script provided with your code.

        - **Start command** – Enter the command that starts your web service.

        - **Port** – Enter the IP port that your web service listens to.

    c. Choose **Next**.

4. Configure your service.

    a. On the **Configure service** page, in the **Service settings** section, enter a service name.

        > **ⓘ Note**
        >
        > All other service settings are either optional or have console-provided defaults.

    b. Optionally change or add other settings to meet your application requirements.

    c. Choose **Next**.

5. On the **Review and create** page, verify all the details you entered, and then choose **Create and deploy**.

Result: If the service is created successfully, the console displays the service dashboard with a **Service overview** of the new service.

6.   Verify that your service is running.

   a.   On the service dashboard page, wait until the service **Status** is **Running**.

   b.   Choose the **Default domain** value. It's the URL to your service's website.

   c.   Use your website and verify that it's running properly.

Creating a service from code using the App Runner API or AWS CLI

To create a service using the App Runner API or AWS CLI, call the `CreateService` API action. For more information and an example, see CreateService. If this is the first time that you're creating a service using a specific organization or account for a source code repository (GitHub or Bitbucket), start by calling CreateConnection. This establishes a connection between App Runner and the repository provider's organization or account. For more information about App Runner connections, see the section called "Connections".

If the call returns a successful response with a Service object showing `"Status": "CREATING"`, your service starts to create.

For an example call, see [Create a source code repository service](#) in the *AWS App Runner API Reference*

Create a service from an Amazon ECR image

The following sections show how to create an App Runner service when your source is a container image stored in [Amazon ECR](#). Amazon ECR is an AWS service. Therefore, to create a service based on an Amazon ECR image, you provide App Runner with an access role containing the necessary Amazon ECR action permissions.

> ⓘ **Note**
>
> Images stored in Amazon ECR Public are publicly available. So, if your image is stored in Amazon ECR Public, an access role isn't required.

When your service is being created, App Runner launches a service that runs a container instance of the image you provide. There's no build phase in this case.

For more information, see *Image-based service*.

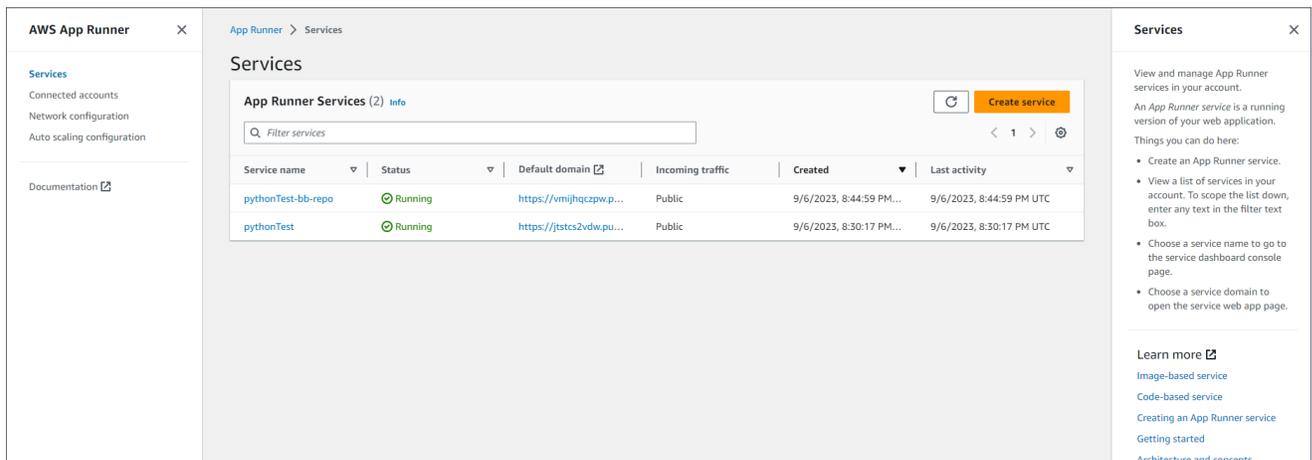**Creating a service from an image using the App Runner console**

To create an App Runner service using the console

1. Configure your source code.

   a. Open the [App Runner console](#), and in the **Regions** list, select your AWS Region.

   b. If the AWS account doesn't have any App Runner services yet, the console home page is displayed. Choose **Create an App Runner service**.
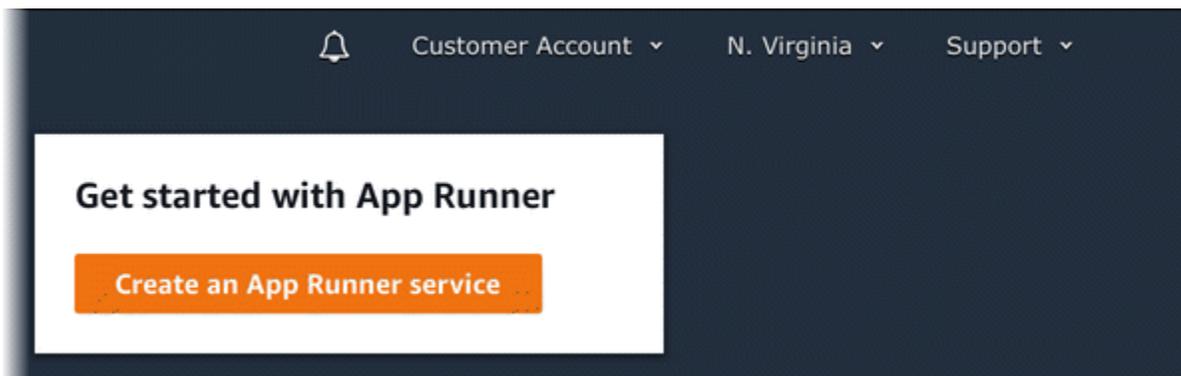
If the AWS account has existing services, the **Services** page with a list of your services is displayed. Choose **Create service**.
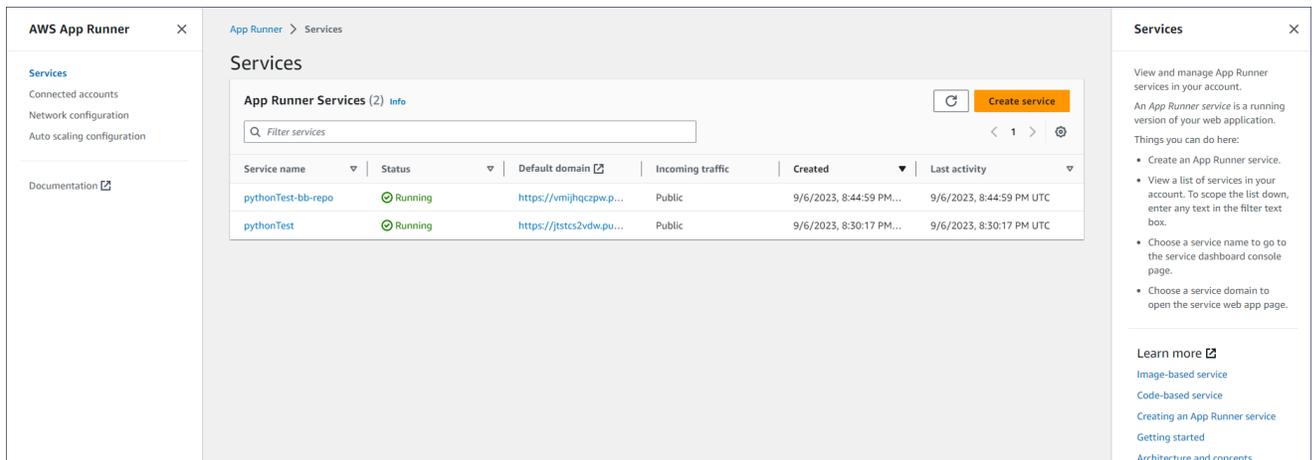
c.   On the **Source and deployment** page, in the **Source** section, for **Repository type**, choose **Container registry**.

d.   For **Provider**, choose the provider where your image is stored:

  • **Amazon ECR** – A private image that's stored in Amazon ECR.

  • **Amazon ECR Public** – A publicly readable image that's stored in Amazon ECR Public.

e.   For **Container image URI**, choose **Browse**.

f.   In the **Select Amazon ECR container image** dialog box, for **Image repository**, select the repository that contains your image.

g.   For **Image tag**, select the specific image tag that you want to deploy (for example, **latest**), and then choose **Continue**.

2. Configure your deployments.

    a. In the **Deployment settings** section, choose **Manual** or **Automatic**.

        > **ⓘ Note**
        >
        > App Runner doesn't support automatic deployment for Amazon ECR Public
        > images, and for images in an Amazon ECR repository that belongs to a different
        > AWS account than the one that your service is in.

        For more information about deployment methods, see the section called "Deployment methods".

    b. [**Amazon ECR** provider] For **ECR access role**, choose an existing service role in your account or choose to create a new role. If you're using manual deployment, you can also choose to use the IAM user role at the time of deployment.

    c. Choose **Next**.

Source and deployment Info

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

Source

Repository type

- ● **Container registry**
  Deploy your service from a container image stored in a container registry.

- ○ **Source code repository**
  Deploy your service from code hosted in a source code repository.

Provider

- ● **Amazon ECR**

- ○ **Amazon ECR Public**

Container image URI

Enter a URI to an image you can access, or browse images in your Amazon ECR account.

123456789012.dkr.ecr.us-east-2.amazonaws.com/test-image:latest

Browse

Deployment settings

Deployment trigger

- ○ **Manual**
  Start each deployment yourself using the App Runner console or AWS CLI.

- ● **Automatic**
  App Runner monitors your registry and deploys a new version of your service for each image push.

ECR access role Info

This role gives App Runner permission to access ECR. To create a custom role, go to the **IAM console** ⬚

- ● Create new service role
- ○ Use existing service role

Service role name

The name of an IAM role that App Runner creates in your account with an attached managed policy for ECR access.

AppRunnerECRAccessRole

Cancel   **Next**

3. Configure your service.

   a. On the **Configure service** page, in the **Service settings** section, enter a service name and the IP port that your service website listens to.

   > ⓘ **Note**
   >
   > All other service settings are either optional or have console-provided defaults.

   b. (Optional) Change or add other settings to suit your application's needs.

   c. Choose **Next**.

Configure service Info

Service settings

Service name

```
image-test
```

Enter a unique name. Use letters, numbers, and dashes. Can't be changed after service creation.

Virtual CPU & memory

1 vCPU ▼	2 GB ▼

Environment variables — *optional*

Key-value pairs that you can use to store custom configuration values.

No environment variables have been configured.

[Add environment variable]

Port

Your service uses this IP port.

```
8080    ⬍
```

▶ **Additional configuration**

▶ **Auto scaling** Info

Configure automatic scaling behavior.

▶ **Health check** Info

Configure load balancer health checks.

▶ **Security** Info

Specify an Instance role and an AWS KMS encryption key

▶ **Tags** Info

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

Cancel   [Previous]   [**Next**]

4. On the **Review and create** page, verify all the details that you entered, and then choose **Create and deploy**.

    Result: If the service is created successfully, the console shows the service dashboard, with a **Service overview** of the new service.
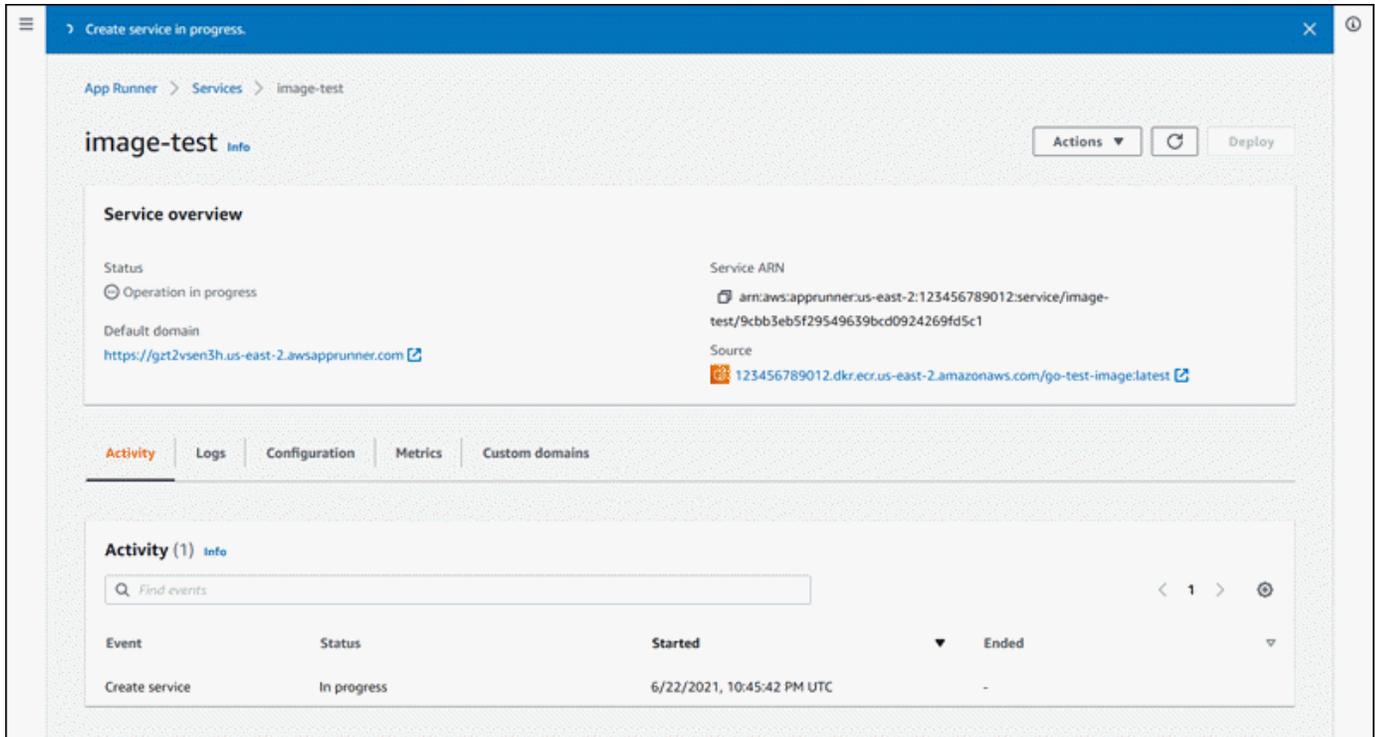
5. Verify that your service is running.

    a. On the service dashboard page, wait until the service **Status** is **Running**.

    b. Choose the **Default domain** value. It's the URL to your service's website.

    c. Use your website and verify that it's running properly.

Creating a service from an image using the App Runner API or AWS CLI

To create a service using the App Runner API or AWS CLI, call the CreateService API action.

Your service creation starts if the call returns a successful response with a Service object showing `"Status": "CREATING"`.

For an example call, see Create a source image repository service in the *AWS App Runner API Reference*

Rebuilding a failed App Runner service

If you receive a **Failed to create** error when creating an App Runner service, you can do one of the following.

- Follow the steps in [the section called "Failed to create service"](#) to identify the cause of the error.

- If you find an error in the source or configuration, make the necessary changes and then rebuild your service.

- If a temporary issue with App Runner caused your service to fail, rebuild your failed service without making any changes to the source or configuration.

You can rebuild your failed service either through the [App Runner console](#) or the [App Runner API or AWS CLI](#).

Rebuilding a failed App Runner service using the App Runner console

Rebuild with updates

Creating a service can fail for a variety of reasons. When this happens, it's important to identify and rectify the root cause of the issue before rebuilding your service. For more information, see [the section called "Failed to create service"](#).

To rebuild a failed service with updates

1. Go to the **Configurations** tab on your service page and choose **Edit**.

   The page opens a summary panel that displays a list of all your updates.

2. Make the required changes and review them in the summary panel.

3. Choose **Save and rebuild**.

   You can monitor progress on the **Logs** tab of your service page.

Rebuild without updates

If a temporary issue causes your service creation to fail, you can rebuild your service without modifying its source or configuration settings.

To rebuild a failed service without updates

- Choose **Rebuild** on the top right corner of your service page.

  You can monitor progress on the **Logs** tab of your service page.

- If your service fails to create again, follow the troubleshooting instructions in the section called "Failed to create service". Make the necessary changes and then rebuild your service.

Rebuilding failed App Runner service using the App Runner API or AWS CLI

Rebuild with updates

To rebuild a failed service:

1. Follow the instructions in the section called "Failed to create service" to find the cause of the error.

2. Make the necessary changes to the branch or the image of the source repository or the configuration that caused the error.

3. Rebuild by calling the UpdateService API action with the new source code repository or source image repository parameters. App Runner retrieves the latest commit from the source code repository.

Example Rebuilding with updates

In the following example the source configuration of an image-based service is being updated. The value of the Port is changed to 80.

Updating the input.json file for image-based App Runner service

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

```
        }
}
```

Calling the `UpdateService` API action.

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

Rebuild without updates

To rebuild your failed service using the App Runner API or AWS CLI, call the [UpdateService](#) API action without making any changes to source or configuration of your service. Choose to rebuild without making updates only if your service creation failed due a temporary issue with App Runner.

Deploying a new application version to App Runner

When you [create a service](#) in AWS App Runner, you configure an application source—a container image or a source repository. App Runner provisions resources to run your service and deploys your application to them.

This topic describes ways to redeploy your application source to your App Runner service when a new version becomes available. This can be a new image version in the image repository or a new commit in the code repository. App Runner provides two methods to deploy to a service: *automatic* and *manual*.

Deployment methods

App Runner provides the following methods for you to control how application deployments are initiated.

Automatic deployment

Use automatic deployment when you want continuous integration and deployment (CI/CD) behavior for your service. App Runner monitors your image or code repository for changes.

Image repository – Whenever you push a new image version to your image repository, or a new commit to your code repository, App Runner automatically deploys it to your service without further action on your side.

Code repository – Whenever you push a new commit to your code repository that makes changes in the [source directory](#), App Runner deploys your entire repository. Because *only changes in the source directory* trigger an automatic deployment, it's important to understand how the source directory location affects the scope of an automated deployment.

- *Top-level directory (repository root)* – This is the default value that's set for the source directory when you create a service. If your source directory is set to this value, this means the entire repository is inside the source directory. So *all commits* that you push to the source repository will trigger a deployment in this case.

- *Any directory path that's not the repository root (non-default)* – Because *only changes that are pushed within the source directory* will trigger an automatic deployment, any changes pushed to your repository that are *not* in the source directory will not trigger an automatic deployment. Therefore, you must use a manual deployment to deploy changes that you push outside of the source directory.

> **ⓘ Note**
>
> App Runner doesn't support automatic deployment for Amazon ECR Public images, and for images in an Amazon ECR repository that belongs to a different AWS account than the one that your service is in.

Manual deployment

Use manual deployment when you want to explicitly initiate each deployment to your service. You initiate a deployment if the repository that you configured for your service has a new version that you want to deploy. For more information, see [the section called "Manual deployment"](#).

> **ⓘ Note**
>
> When you run a manual deployment, App Runner deploys source from the full repository.

You can configure the deployment method for your service in the following ways:

- *Console* – For a new service you're creating or for an existing service, in the **Deployment settings** section of the **Source and deployment** configuration page, choose **Manual** or **Automatic**.
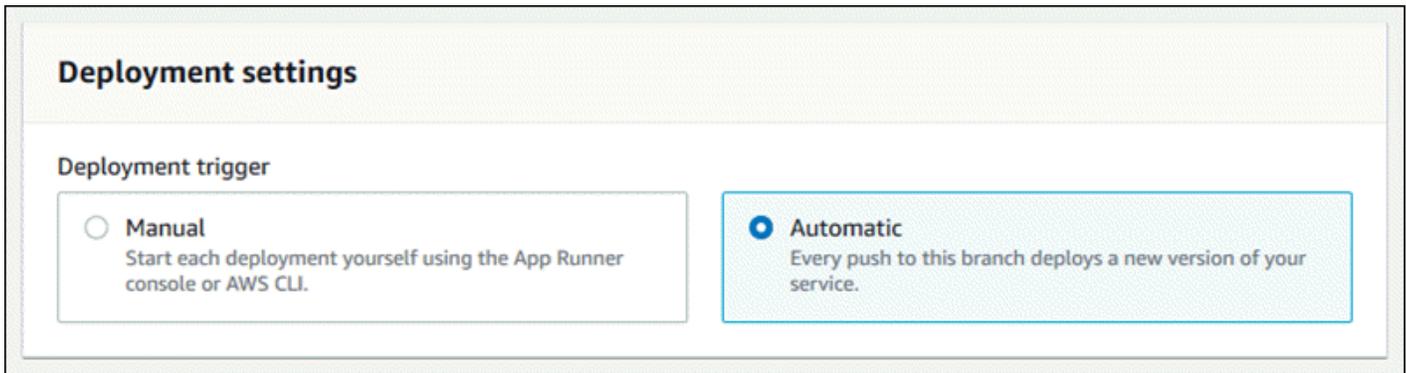
- *API or AWS CLI* – In a call to either the CreateService or UpdateService action, set the AutoDeploymentsEnabled member of the SourceConfiguration parameter to `False` for manual deployment or `True` for automatic deployment.

> ⓘ **Comparing automatic and manual deployments**
>
> Both automatic and manual deployments yield the same result: both methods deploy the full repository.
> The difference between the two methods is the triggering mechanism:
>
> - Manual deployments are triggered by a deploy from the console, a call to the AWS CLI, or a call to the App Runner API. The Manual deployment section that follows provides the procedures for these.
>
> - Automatic deployments are triggered by a change within the contents of the source directory.

Manual deployment

With manual deployment, you need to explicitly initiate each deployment to your service. When you have a new version of your application image or code ready to deploy, you can refer to the following sections to learn how to perform a deployment using the console and the API.

> ⓘ **Note**
>
> When you run a manual deployment, App Runner deploys source from the full repository.

Deploy a version of your application using one of the following methods:

App Runner console

To deploy using the App Runner console

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

   The console displays the service dashboard with a **Service overview**.
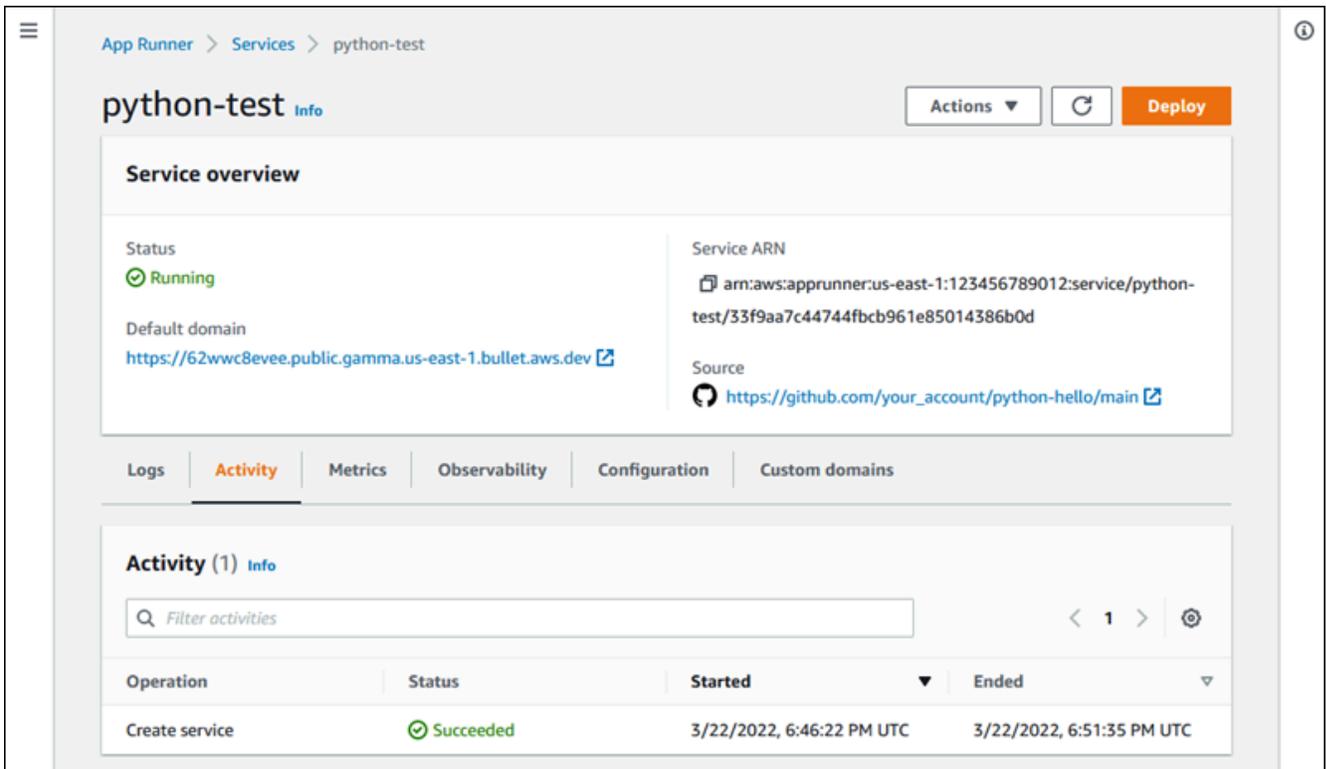
3. Choose **Deploy**.

   Result: Deployment of the new version starts. On the service dashboard page, the service
   Status changes to **Operation in progress**.

4. Wait for the deployment to end. On the service dashboard page, the service **Status** should
   change back to **Running**.

5. To verify that the deployment is successful, on the service dashboard page, choose the
   Default domain value—it's the URL to your service's website. Inspect or interact with your
   web application and verify your version change.

> **ⓘ Note**
>
> To augment the security of your App Runner applications, the **.awsapprunner.com*
> domain is registered in the [Public Suffix List (PSL)](). For further security, we
> recommend that you use cookies with a `__Host-` prefix if you ever need to set
> sensitive cookies in the default domain name for your App Runner applications.
> This practice will help to defend your domain against cross-site request forgery
> attempts (CSRF). For more information see the [Set-Cookie]() page in the Mozilla
> Developer Network.

App Runner API or AWS CLI

To deploy using the App Runner API or AWS CLI, call the [StartDeployment]() API action. The only
parameter to pass is your service ARN. You already configured your application source location
when you created the service, and App Runner can find the new version. Your deployment starts
if the call returns a successful response.

Configuring an App Runner service

When you [create an AWS App Runner service](), you set various configuration values. You can
change some of these configuration settings after you create the service. Other settings can be
applied only while creating the service and cannot be changed thereafter. This topic discusses
the configuration of your service using the App Runner API, the App Runner console, and an App
Runner configuration file.

Topics

- [Configure your service using the App Runner API or AWS CLI]()
- [Configure your service using the App Runner console]()
- [Configure your service using an App Runner configuration file]()
- [Configuring observability for your service]()
- [Configuring service settings using sharable resources]()
- [Configuring health checks for your service]()

Configure your service using the App Runner API or AWS CLI

The API defines which settings can be changed after service creation. The following list discusses the relevant actions, types, and limitations.

- UpdateService action – Can be called after creation to update some configuration settings.

  - *Can be updated* – You can update settings in the `SourceConfiguration`, `InstanceConfiguration`, and `HealthCheckConfiguration` parameters. However, in `SourceConfiguration`, you can't switch your source type from code to image or the other way around. You must provide the same repositoryparameter as you provided when you created the service. It's either `CodeRepository` or `ImageRepository`.

    You can also update the following ARNs of separate configuration resources associated with the service:

    - `AutoScalingConfigurationArn`

    - `VpcConnectorArn`

  - *Cannot be updated* – You can't change the `ServiceName` and `EncryptionConfiguration` parameters that are available in the CreateService action. They can't be changed after they're created. The UpdateService action doesn't include these parameters.

  - *API vs. file* – You can set the `ConfigurationSource` parameter of the CodeConfiguration type (used for source code repositories as part of `SourceConfiguration`) to `Repository`. In this case, App Runner ignores the configuration settings in `CodeConfigurationValues`, and reads these settings from a configuration file in your repository. If you set `ConfigurationSource` to API, App Runner gets all configuration settings from the API call and ignores the configuration file, even if one exists.

- TagResource action – Can be called after your service is created to add tags to the service or update values of existing tags.

- UntagResource action – Can be called after your service is created to remove tags from the service.

> **ⓘ Note**
>
> If you create an outbound traffic VPC connector for a service, the service startup process that follows will experience a one-time latency. You can set this configuration for a new

service when you create it, or afterward, with a service update. For more information, see [One-time latency](#) in the *Networking with App Runner* chapter of this guide.

Configure your service using the App Runner console

The console uses the App Runner API to apply configuration updates. The update rules that the API imposes, as defined in the previous section, determine what you can configure using the console. Some settings that were available during service creation aren't available for modification later on. In addition, if you decide to use a [configuration file](#), additional settings are hidden in the console, and App Runner reads them from the file.

To configure your service

1. Open the [App Runner console](#), and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

   The console displays the service dashboard with a **Service overview**.

3. On the service dashboard page, choose the **Configuration** tab.

Result: The console displays the current configuration settings of your service in several sections: **Source and deployment**, **Configure build**, and **Configure service**.

4. To update settings in any category, choose **Edit**.

5. On the configuration edit page, make any desired changes, and then choose **Save changes**.

> ⓘ **Note**
>
> If you create an outbound traffic VPC connector for a service, the service startup process that follows will experience a one-time latency. You can set this configuration for a new service when you create it, or afterward, with a service update. For more information, see [One-time latency](#) in the *Networking with App Runner* chapter of this guide.

Configure your service using an App Runner configuration file

When you create or update an App Runner service, you can instruct App Runner to read some configuration settings from a configuration file that you provide as part of your source repository. By doing this, you can manage the settings that are related to your source code under source control, together with the code itself. The configuration file also provides certain advanced settings that you can't set using the console or the API. For more information, see *[App Runner configuration file](#)*.

> ⓘ **Note**
>
> If you create an outbound traffic VPC connector for a service, the service startup process that follows will experience a one-time latency. You can set this configuration for a new service when you create it, or afterward, with a service update. For more information, see [One-time latency](#) in the *Networking with App Runner* chapter of this guide.

Configuring observability for your service

AWS App Runner integrates with several AWS services to provide you with an extensive observability suite of tools for your App Runner service. For more information, see *[Observability](#)*.

App Runner supports enabling some observability features and configuring their behavior by using a sharable resource called *ObservabilityConfiguration*. You can provide an observability

configuration resource when you create or update a service. The App Runner console creates one for you when you create a new App Runner service. Providing an observability configuration is optional. If you don't provide one, App Runner provides a default observability configuration.

You can share a single observability configuration across multiple App Runner services to ensure they have the same observability behavior. For more information, see the section called "Configuration resources".

You can configure the following observability features using observability configurations:

- *Trace configuration* – Settings for tracing requests that your application serves and downstream calls that it makes. For more information about tracing, see the section called "Tracing (X-Ray)".

Manage observability

Manage observability for your App Runner services using one of the following methods:

App Runner console

When you create a service using the App Runner console, or when you update its configuration later, you can configure observability features for your service. Look for the **Observability** configuration section on the console page.
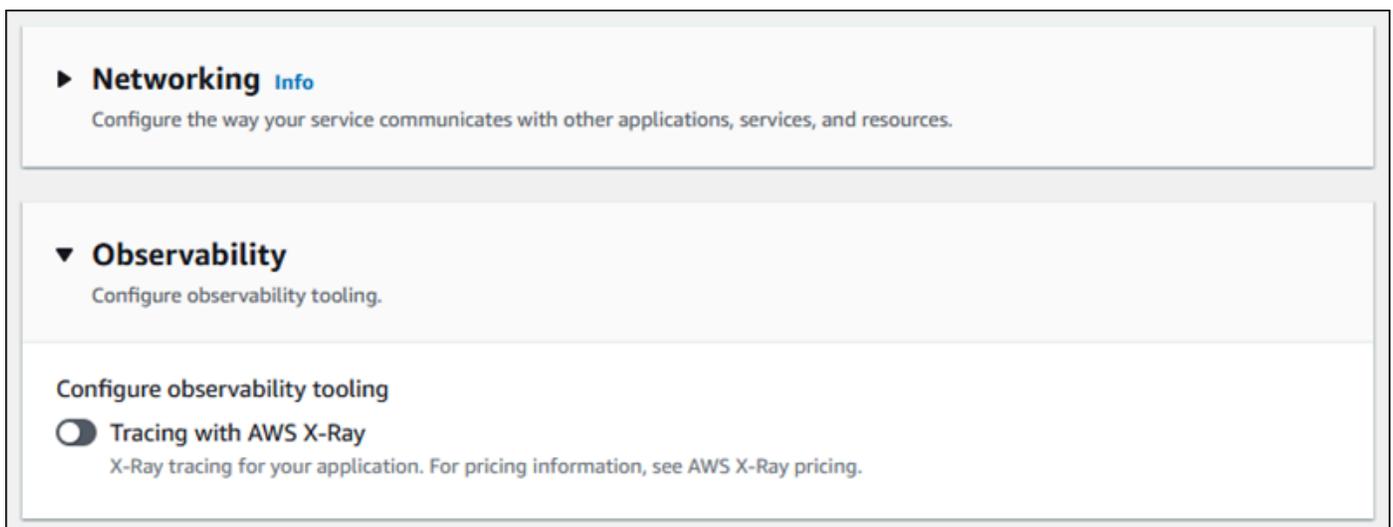
App Runner API or AWS CLI

When you call the CreateService or UpdateService App Runner API actions, you can use the ObservabilityConfiguration parameter object to enable observability features and specify an observability configuration resource for your service.

Use the following App Runner API actions to manage your observability configuration resources.

- CreateObservabilityConfiguration – Creates a new observability configuration or a revision to an existing one.

- ListObservabilityConfigurations – Returns a list of the observability configurations that are associated with your AWS account, with summary information.

- DescribeObservabilityConfiguration – Returns a full description of an observability configuration.

- DeleteObservabilityConfiguration – Deletes an observability configuration. You can delete a specific revision or the latest active revision. You might need to delete unnecessary observability configurations if you reach the observability configuration quota for your AWS account.

Configuring service settings using sharable resources

For some features, it makes sense to share configuration across AWS App Runner services. For example, you might want a set of services to have the same auto scaling behavior. Or you might want identical observability settings for all of your services. App Runner lets you share settings by using separate sharable resources. You create a resource that defines a set of configuration settings for a feature, and then you provide the Amazon Resource Name (ARN) of this configuration resource to one or more App Runner services.

App Runner implements sharable configuration resources for the following features:

- Auto scaling
- Observability
- VPC access

The document page for each of these features provides information about the available settings and the management procedures.

Features using separate configuration resources share some design traits and considerations.

- **Revisions** – Some configuration resources can have revisions. Auto scaling and observability are examples of two configuration resources that use revisions. In these cases, each configuration has a *name* and a numeric *revision*. Multiple revisions of a configuration have the same name and

different revision numbers. You can use different configuration names for different scenarios. For each name, you can add multiple revisions to fine-tune the settings for a specific scenario.

The first configuration that you create with a name gets the revision number 1. Subsequent configurations with the same name get consecutive revision numbers (starting with 2). You can associate your App Runner service with a specific configuration revision or with the latest revision of configuration.

- **Shared** – You can share a single configuration resource across multiple App Runner services. This is useful if you want to maintain identical configurations across these services. In particular, if your resources support revisions, you can configure multiple services to use the latest revision of a configuration. You can do so by specifying only the configuration name, but not a revision. Any of the services that you configured this way receives configuration updates when you update the service. For more information about configuration changes, see the section called "Configuration".

- **Resource management** – You can use App Runner to create and delete configurations. You can't directly update a configuration. Instead, for resources that support revisions, you can create a new revision to an existing configuration name to effectively update the configuration.

> **ⓘ Note**
>
> For auto scaling, you can create configurations and *multiple* revisions with both the App Runner console and the App Runner API. Both the App Runner console and the App Runner API can also delete configurations and revisions. For more details, see Managing App Runner automatic scaling.
> For other configuration types, like observability configurations, you can only create a configuration with a *single* revision with the App Runner console. To create more revisions, and to delete configurations, you must use the App Runner API.

- **Resource quota** – There are set quotas for the number of unique configuration names and revisions that you can have for your configuration resources in each AWS Region. If you reach these quotas, you must either delete a configuration name or at least some of its revisions before you can create more. For auto scaling configurations revisions, you can use the App Runner console or the App Runner API to delete them. For more details, see Managing App Runner automatic scaling. You must use the App Runner API to delete other resources. For more information about quotas, see the section called "App Runner resource quotas".

- **No resource cost** – You don't incur additional cost for creating a configuration resource. You might incur cost for the feature itself (for example, you are charged for normal AWS X-Ray

cost when you turn on X-Ray tracing), but not for the App Runner configuration resource that configures the feature for your App Runner service.

Configuring health checks for your service

AWS App Runner monitors the health of your service by performing health checks. The default health check protocol is TCP. App Runner pings the domain assigned to your service. You can alternatively set the health check protocol to HTTP. App Runner sends health check HTTP requests to your web application.

You can configure a few settings related to health checks. The following table describes the health check settings and their default values.

Setting	Description	Default
Protocol	The IP protocol that App Runner uses to perform health checks for your service. If you set the protocol to TCP, App Runner pings the default domain assigned to your service at the port that your application is listening to. If you set the protocol to HTTP, App Runner sends health check requests to the configured path.	TCP
Path	The URL that App Runner sends HTTP health check requests to. Applicable only to HTTP checks.	/
Interval	The time interval, in seconds, between health checks.	5
Timeout	The time, in seconds, to wait for a health check response before deciding it failed.	2
Healthy threshold	The number of consecutive checks that must succeed before App Runner decides that the service is healthy.	1
Unhealthy threshold	The number of consecutive checks that must fail before App Runner decides that the service is unhealthy.	5

Configure health checks

Configure health checks for your App Runner service using one of the following methods:

App Runner console

> When you create your App Runner service using the App Runner console, or when you update
> its configuration later, you can configure health check settings. For full console procedures, see
> [the section called "Creation"](#) and [the section called "Configuration"](#). In both cases, look for the
> **Health check** configuration section on the console page.
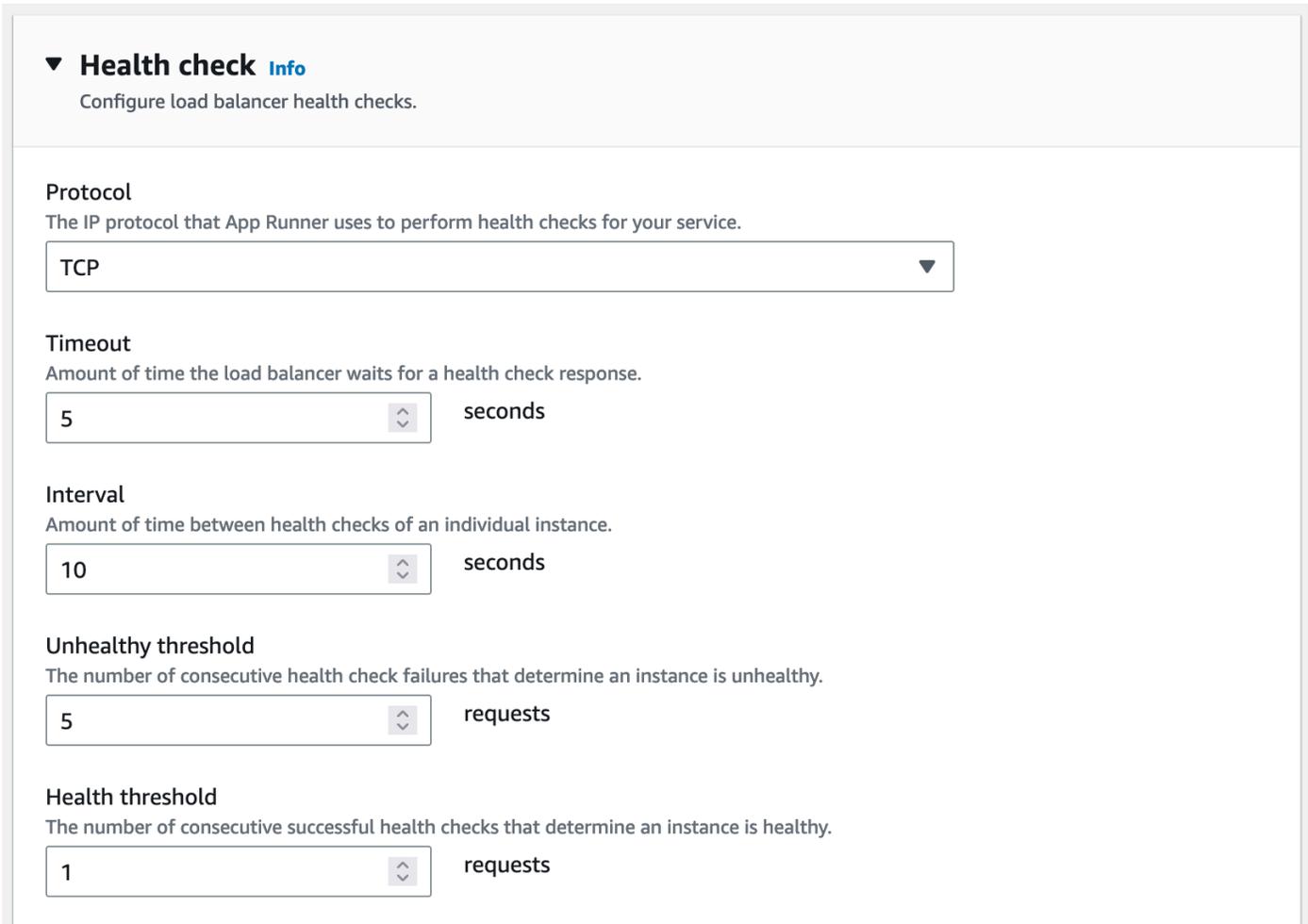
> **▼ Health check** Info
> Configure load balancer health checks.
>
> **Protocol**
> The IP protocol that App Runner uses to perform health checks for your service.
>
> | TCP | ▼ |
>
> **Timeout**
> Amount of time the load balancer waits for a health check response.
>
> | 5 | ◊ | seconds
>
> **Interval**
> Amount of time between health checks of an individual instance.
>
> | 10 | ◊ | seconds
>
> **Unhealthy threshold**
> The number of consecutive health check failures that determine an instance is unhealthy.
>
> | 5 | ◊ | requests
>
> **Health threshold**
> The number of consecutive successful health checks that determine an instance is healthy.
>
> | 1 | ◊ | requests

App Runner API or AWS CLI

> When you call the [CreateService](#) or [UpdateService](#) API actions, you can use the
> `HealthCheckConfiguration` parameter to specify health check settings.
>
> For information about the parameter's structure, see [HealthCheckConfiguration](#) in the *AWS App
> Runner API Reference*.

Managing App Runner connections

When you [create a service](#) in AWS App Runner, you configure an application source—a container image or a source repository that's stored with a provider. App Runner has to establish an authenticated and authorized connection with the provider. Then, App Runner can read your repository and deploy it to your service. App Runner doesn't require connection establishment when you create a service that accesses code stored in your AWS account.

App Runner maintains connection information in a resource called a *connection*. The App Runner console and this guide also refer to connections as *connected accounts*. App Runner requires a connection resource when you create a service that needs third-party connection information. The following is some important information about connections:

- **Providers** – App Runner currently requires connection resources with [GitHub](#) or [Bitbucket](#).

- **Shared** – You can use a connection resource to create multiple App Runner services that use the same repository provider account.

- **Resource management** – In App Runner, you can create and delete connections. However, you can't modify an existing connection.

- **Resource quota** – Connection resources have a set quota that's associated with your AWS account in each AWS Region. If you reach this quota, you might need to delete a connection before you can connect to a new provider account. You can delete a connection using the App Runner console or API as described in the following section, [the section called "Manage connections"](#). For more information, see [the section called "App Runner resource quotas"](#).

Manage connections

Manage your App Runner connections using one of the following methods:

App Runner console

> When you use the App Runner console to [create a service](#), you provide connection details. You don't have to explicitly create a connection resource. In the console, you can choose to connect to a GitHub or Bitbucket account that you've connected to before, or connect to a new account. When necessary, App Runner creates a connection resource for you. For a new connection, some providers require you to complete an authentication handshake before you can use the connection. The console takes you through this process.

The console also has a page for managing your existing connections. You can complete the authentication handshake for a connection if you didn't do it when you created your service. You can also delete connections that you're no longer using. The following procedure shows how you can manage repository provider connections.

To manage connections in your account

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Connected accounts**.

   The console then displays a list of repository provider connections in your account.
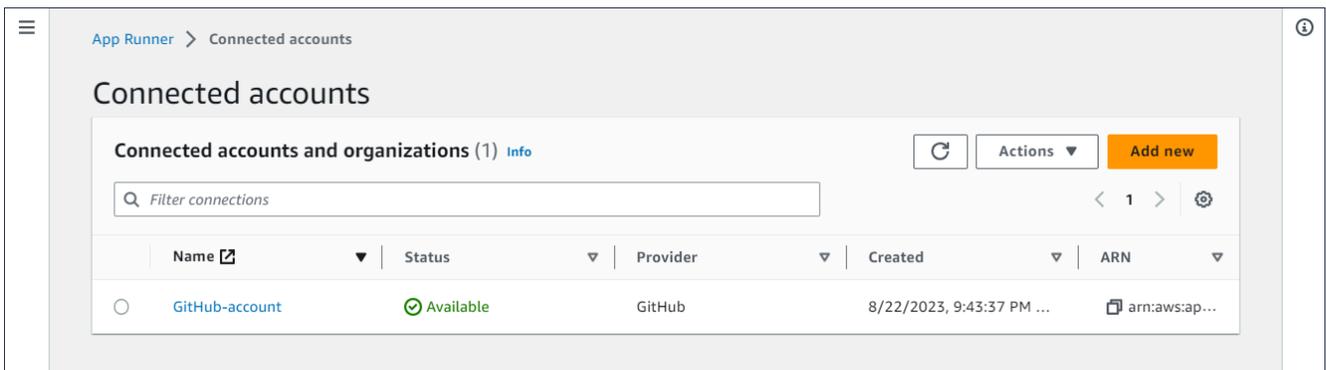
3. You can now do one of the following actions with any connection on the list:

   - *Open GitHub/Bitbucket account or organization* – Choose the name of the connection.

   - *Complete authentication handshake* – Select the connection, and then from the **Actions** menu choose **Complete handshake**. The console takes you through the authentication handshake process.

   - *Delete connection* – Select the connection, and then from the **Actions** menu choose **Delete**. Follow the instructions on the deletion prompt.

App Runner API or AWS CLI

You can use the following App Runner API actions to manage your connections.

- CreateConnection – Creates a connection to a repository provider account. After the connection is created, you must manually complete the authentication handshake using the App Runner console. This process is explained in the previous section.

- ListConnections – Returns a list of App Runner connections associated with your AWS account.

- **DeleteConnection** – Deletes a connection. You might need to delete unnecessary connections if you reach the connection quota for your AWS account.

Managing App Runner automatic scaling

AWS App Runner automatically scales compute resources, specifically instances, up or down for your App Runner application. Automatic scaling provides adequate request handling when traffic is heavy, and reduces your cost when traffic slows down.

Auto scaling configuration

You can configure a few parameters to adjust auto scaling behavior for your service. App Runner maintains auto scaling settings in a sharable resource that's called *AutoScalingConfiguration*. You can create and maintain stand-alone auto scaling configurations, before you assign them to services. After they've been associated to a service, you can continue to maintain the configurations. You can also choose to create a new auto scaling configuration while you're in the process of creating a new service or configuring an existing one. Once the new auto scaling configuration is created, you can associate it to the service and continue on with the process of creating or configuring your service.

Naming and revisions

An auto scaling configuration has a *name* and a numeric *revision*. Multiple revisions of a configuration have the same name and different revision numbers. You can use different configuration names for different auto scaling scenarios, such as *high availability* or *low cost*. For each name, you can add multiple revisions to fine-tune the settings for a specific scenario. You can have up to 10 unique auto scaling configuration names and up to 5 revisions for each configuration. If you reach the limit and need to create more, you can delete one and then create another one. App Runner will not allow you to delete a configuration that's set as the default or in use by an active service. For more information about quotas, see the section called "App Runner resource quotas".

Setting a default configuration

When you create or update an App Runner service, you can provide an auto scaling configuration resource. Providing an auto scaling configuration is optional. If you don't provide one, App Runner provides a default auto scaling configuration with recommended values. The auto scaling configuration feature provides you the option to set your own default auto scaling configuration

instead of using the default that App Runner provides. Once you specify another auto scaling configuration as a default, that configuration is automatically assigned as the default to the new services you create in the future. The new default designation doesn't affect the associations that were previously set for existing services.

Configuring services with auto scaling

You can share a single auto scaling configuration across multiple App Runner services to ensure the services have the same auto scaling behavior. For more information about configuring auto scaling configurations with the App Runner console or the App Runner API, see the sections that follow in this topic. For more general information about shareable resources, see the section called "Configuration resources".

Configurable settings

You can configure the following auto scaling settings:

- *Max concurrency* – The maximum number of concurrent *requests* that an instance processes. When the number of concurrent requests exceeds this quota, App Runner scales up the service.

- *Max size* – The maximum number of *instances* that your service can scale up to. This is the highest number of instances that can concurrently handle your service's traffic.

- *Min size* – The minimum number of *instances* that App Runner can provision for your service. The service always has at least this number of provisioned instances. Some of these instances actively handle traffic. The remainder of them are part of the cost-effective compute capacity reserve, which is ready to be quickly activated. You pay for the memory usage of all the provisioned instances. You pay for the CPU usage of only the active subset.

> **ⓘ Note**
>
> The vCPU resource count determines the number of instances that App Runner can provide to your service. This is an adjustable quota value for the *Fargate On-Demand* vCPU resource count that resides in the AWS Fargate service. To view the vCPU quota settings for your account or to request a quota increase, use the Service Quotas console in the AWS Management Console. For more information, see AWS Fargate service quotas in the *Amazon Elastic Container Service Developer Guide*.

Manage auto scaling for a service

Manage auto scaling for your App Runner services using one of the following methods:

App Runner console

When you [create a service](#) using the App Runner console or [update a service configuration](#), you can specify an auto scaling configuration.

> ⓘ **Note**
>
> When you change the auto scaling configuration or revision that's associated to a service, your service is re-deployed.

The **Auto scaling** configuration page offers several options to configure auto scaling for your service.

- **To assign an existing configuration and revision** – Choose a value from the **Existing configurations** drop-down. The latest revision version will default in the adjacent drop-down. If a different revision exists that you would prefer to select, do so from the revision drop-down. The configuration values for the revision version display.

- **To create and assign a new auto scaling configuration** – Select **Create new ASC** from the **Create** menu. This launches the **Add custom auto scaling configuration** page. Enter a **Configuration name** and values for the auto scaling parameters. Then select **Add**. App Runner creates the new auto scaling configuration resource for you and returns you to **Auto scaling** section with the new configuration selected and displayed.

- **To create and assign a new revision** – First select the configuration name from the **Existing configurations** drop-down. Then select **Create ASC revision** from the **Create** menu. This launches the **Add custom auto scaling configuration** page. Enter values for the auto scaling parameters. Then select **Add**. App Runner creates a new auto scaling configuration revision for you and returns you to **Auto scaling** section with the new revision selected and displayed.

App Runner API or AWS CLI

When you call the [CreateService](#) or [UpdateService](#) App Runner API actions, you can use the `AutoScalingConfigurationArn` parameter to specify an auto scaling configuration resource for your service.

The next section provides guidance to manage your auto scaling configuration resources.

Manage auto scaling configurations resources

Manage the App Runner auto scaling configurations and revisions for your account using one of the following methods:

App Runner console

Manage auto scaling configurations

The **Auto scaling configurations** page lists the auto scaling configurations that you have set up in your account. You can create and manage your auto scaling configurations on this page and then later assign them to one or more App Runner services.

You can do any of the following from this page:

- Create a new auto scaling configuration.

- Create a new revision for an existing auto scaling configuration.

- Delete an auto scaling configuration.

- Set an auto scaling configuration as the default.

To manage auto scaling configurations in your account

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Auto scaling configurations**. The console displays a list of auto scaling configurations in your account.

    You can now do any of the following.

    - **To create a new auto scaling configuration**, follow these steps.

        a. On the **Auto scaling configurations** page, select **Create**.

            The **Create auto scaling configuration** page displays.

        b. Enter values for **Configuration name**, **Concurrency**, **Minimum size**, and **Maximum size**.

        c. (Optional) If you'd like to add tags, select **Auto new tag**. Then on the fields that appear enter a **Name** and a **Value** (optional).

        d. Select **Create**.

    - **To create a new revision for an existing auto scaling configuration**, follow these steps.

a. On the **Auto scaling configurations** page, select the radio button next to the configuration that needs the new revision. Then select **Create revision** from the **Actions** menu.

The **Create revision** page displays.

b. On , enter values for **Concurrency**, **Minimum size**, and **Maximum size**.

c. (Optional) If you'd like to add tags, select **Auto new tag**. Then on the fields that appear enter a **Name** and a **Value** (optional).

d. Select **Create**.

- **To delete an auto scaling configuration**, follow these steps.

    a. On the **Auto scaling configurations** page, select the radio button next to the configuration that you need to delete.

    b. Select **Delete** from the **Actions** menu.

    c. To proceed with the deletion, select **Delete** on the confirmation dialogue. Otherwise, select **Cancel**.

    > ⓘ **Note**
    >
    > App Runner validates that your deletion choice is not set as a default or is currently in use by any active services.

- **To set an auto scaling configuration as the default**, follow these steps.

    a. On the **Auto scaling configurations** page, select the radio button next to the configuration that you need to set as the default.

    b. Select **Set as default** from the **Actions** menu.

    c. A dialogue displays informing you that App Runner will use the latest revision as the default configuration for all the new services you create. Select **Confirm** to proceed. Otherwise select **Cancel**.

    > ⓘ **Note**
    >
    > - When you set an auto scaling configuration as default, it automatically gets assigned as the default configuration to the new services you create in future.

- The new default designation doesn't affect the associations that were previously set for existing services.
- If the designated default auto scaling configuration has revisions, App Runner assigns its latest revision as the default.

Manage revisions

The console also has a page for creating and managing your existing auto scaling revisions called **Auto scaling revisions**. Access this page by selecting the name of a configuration on the **Auto scaling configurations** page.

You can do any of the following from the **Auto scaling revisions** page:

- Create a new auto scaling revision.
- Set an auto scaling configuration revision as the default.
- Delete a revision.
- Delete the whole auto scaling configuration, including all of the associated revisions.
- View the configuration details for a revision.
- View a list of the services associated to a revision.
- Change the revision for a listed service.

To manage auto scaling revisions in your account

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Auto scaling configurations**. The console displays a list of auto scaling configurations in your account. The prior set of procedures in the [Manage auto scaling configurations](#) section includes a screen image of this page.

3. Now you can drill down into a specific auto scaling configuration to view and manage all of its revisions. In the **Auto scaling configurations** pane, under the **Configuration name** column, choose an auto scaling configurations name. Select the actual name, rather than the radio button. This will navigate you to a list of all the revisions for that configuration on the **Auto scaling revisions** page.

4. You can now do any of the following.

   - **To create a new revision for an existing auto scaling configuration**, follow these steps.

     a. On the **Auto scaling revisions** page, select **Create revision**.

        The **Create revision** page displays.

     b. Enter values for **Concurrency**, **Minimum size**, and **Maximum size**.

     c. (Optional) If you'd like to add tags, select **Auto new tag**. Then on the fields that appear enter a **Name** and a **Value** (optional).

     d. Select **Create**.

   - **To delete the whole auto scaling configuration, including all of the associated revisions**, follow these steps.

     a. Select **Delete configuration** on the top right of the page.

     b. To proceed with the deletion, select **Delete** on the confirmation dialogue. Otherwise, select **Cancel**.

        > **ⓘ Note**
        >
        > App Runner validates that your deletion choice is not set as a default or is currently in use by any active services.

   - **To set an auto scaling revision as the default**, follow these steps.

     a. Select the radio button next to the revision that you need to set as the default.

     b. Select **Set as default** from the **Actions** menu.

> **ⓘ Note**
>
> - When you set an auto scaling configuration as default, it automatically gets assigned as the default configuration to the new services you create in future.
>
> - The new default designation doesn't affect the associations that were previously set for existing services.

- **To view the configuration details for a revision**, follow these steps.

    - Select the radio button next to the revision.

      The configuration details for the revision, including the ARN, displays in the lower split panel. Refer to the screen image at the end of this procedure.

- **To view a list of the services associated to a revision**, follow these steps.

    - Select the radio button next to the revision.

      The **Services** panel, displays in the lower split panel, beneath the revision configuration details. The panel lists all of the services that use this auto scaling configuration revision. Refer to the screen image at the end of this procedure.

- **To change the revision for a listed service**, follow these steps.

    a. Select the radio button next to the revision, if you haven't done so already.

      The **Services** panel, displays in the lower split panel, beneath the revision configuration details. The panel lists all of the services that use this auto scaling configuration revision. Refer to the screen image at the end of this procedure.

    b. On the **Services** panel, select the radio button next to the service that you want to modify. Then select **Change revisions.**

    c. The **Change ASC revision** panel displays. Choose from the available revisions in the drop-down. Only the revisions of the auto scaling configuration you chose earlier are available. If you need to change to a different auto scaling configuration, follow the procedures under the prior section the section called "Manage auto scaling for a service".

      Select **Update** to proceed with the change. Otherwise select **Cancel**.

> ⓘ **Note**
>
> When you change a revision that's associated to a service, your service is
> re-deployed.
> You must select refresh on this panel to see the updated associations.
> To see the ongoing activity and the status for the service redeployment,
> use the panel breadcrumbs to navigate to **App Runner > Services**, select
> the service, then view the **Logs** tab from the **Service overview** panel.

App Runner API or AWS CLI

Use the following App Runner API actions to manage your auto scaling configuration resources.

- CreateAutoScalingConfiguration – Creates a new auto scaling configuration or a revision to an existing one.

- UpdateDefaultAutoScalingConfiguration –Sets an auto scaling configuration to be the default. The existing default auto scaling configuration will be set to non-default automatically.

- ListAutoScalingConfigurations – Returns a list of the auto scaling configurations that are associated with your AWS account, with summary information.

- ListServicesForAutoScalingConfiguration – Returns a list of the associated App Runner services using an auto scaling configuration.

- DescribeAutoScalingConfiguration – Returns a full description of an auto scaling configuration.

- DeleteAutoScalingConfiguration – Deletes an auto scaling configuration. You can delete a top level auto scaling configuration, a specific revision of one, or all revisions associated with the top level configuration. Use the optional `DeleteAllRevisions` parameter to delete all of the revisions. If you reach the auto scaling configuration resource quota for your AWS account, you might need to delete unnecessary auto scaling configurations.

Managing custom domain names for an App Runner service

When you create an AWS App Runner service, App Runner allocates a domain name for it. This is a subdomain in the `awsapprunner.com` domain that's owned by App Runner. You can use the domain name to access the web application that's running in your service.

> **ⓘ Note**
>
> To augment the security of your App Runner applications, the *.awsapprunner.com* domain is registered in the Public Suffix List (PSL). For further security, we recommend that you use cookies with a `__Host-` prefix if you ever need to set sensitive cookies in the default domain name for your App Runner applications. This practice will help to defend your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

If you own a domain name, you can associate it to your App Runner service. After App Runner validates your new domain, you can use your domain to access your application in addition to the App Runner domain. You can associate up to five custom domains.

> ⓘ **Note**
>
> You can optionally include the www subdomain of your domain. However, this is currently only supported by the API. The App Runner console doesn't support including www subdomain of your domain.

> ⓘ **Note**
>
> AWS App Runner doesn't support using Route 53 private hosted zones. Private hosted zones customize domain name resolution for Amazon VPC traffic. For more information about private hosted zones, see Working with private hosted zones in the Route 53 documentation.

Associate (link) a custom domain to your service

When you associate a custom domain to your service, you must add the CNAME records and DNS target records to your DNS server. The following sections provide information on CNAME records and DNS target records and how to use them.

> ⓘ **Note**
>
> If you're using Amazon Route 53 as your DNS provider, App Runner automatically configures your custom domain with the required certificate validation and DNS records to link to your App Runner web application. This happens when you use the App Runner console to link your custom domain to your service. The Manage custom domains topic that follows provides more information.

CNAME records

When you associate a custom domain with your service, App Runner provides you with a set of certificate validation records for certificate validation. You must add these certificate validation

records to your Domain Name System (DNS) server. Add the certificate validation records, provided by App Runner, to your DNS server. This way, App Runner can validate that you own or control the domain.

> **ⓘ Note**
>
> To auto-renew your custom domain certificates, ensure that you don't delete the certificate validation records from your DNS server. For information about how to resolve issues that are related to the renewal of the certificate, see the section called "Custom domain certificate renewal".

App Runner uses ACM to verify the domain. If you're using CAA records in your DNS records, make sure that at least one CAA record references `amazon.com`. Otherwise, ACM can't verify the domain and successfully create your domain.

If you receive errors related to CAA, see the following links to learn how to resolve them:

- Certification Authority Authorization (CAA) problems
- How do I resolve CAA errors for issuing or renewing an ACM certificate?
- Custom domain names

> **ⓘ Note**
>
> If you're using Amazon Route 53 as your DNS provider, App Runner automatically configures your custom domain with the required certificate validation and DNS records to link to your App Runner web application. This happens when you use the App Runner console to link your custom domain to your service. The Manage custom domains topic that follows provides more information.

DNS target records

Add the DNS target records to your DNS server to target the App Runner domain. Add one record for the custom domain, and another for the www subdomain, if you chose this option. Then, wait for the custom domain status to become **Active** in the App Runner console. This typically takes several minutes, but might take up to 24—48 hours (1—2 days). When your custom domain is validated, App Runner starts routing traffic from this domain to your web application.

> ⓘ **Note**
>
> For better compatibility with App Runner services, we recommend that you use Amazon Route 53 as your DNS provider. If you don't use Amazon Route 53 to manage your public DNS records, contact your DNS provider to find out how to add records.
>
> If you're using Amazon Route 53 as your DNS provider, you can add either CNAME or alias record for *subdomain*. For *root domain*, ensure that you use the alias record.

You can purchase a domain name from Amazon Route 53 or another provider. To purchase a domain name with Amazon Route 53, see [Registering a new domain](), in the *Amazon Route 53 Developer Guide*.

For instructions on how to configure a DNS target in Route 53, see [Routing traffic to your resources](), in the *Amazon Route 53 Developer Guide*.

For instructions on how to configure a DNS target on other registrars, such as GoDaddy, Shopify, Hover and so on, refer to their specific documentation on adding DNS Target records.

Specify a domain to associate with your App Runner service

You can specify a domain to associate with your App Runner service in the following ways:

- *A root domain* – DNS has some inherent limitations which might block you from creating CNAME records for the root domain name. For example, if your domain name is `example.com`, you can create a CNAME record that routes traffic for `acme.example.com` to your App Runner service. However, you can't create a CNAME record that routes traffic for `example.com` to your App Runner service. To create a root domain, ensure that you add an alias record.

  An alias record is specific to Route 53 and has the following advantages over CNAME records:

  - Route 53 provides you with more flexibility as alias records can be created for root domain or subdomain. For example, if your domain name is `example.com`, you can create a record that routes requests for `example.com` or `acme.example.com` to your App Runner service.

  - It is more cost efficient. This is because Route 53 doesn't charge for requests that use an alias record to route traffic.

- *A subdomain* – For example, `login.example.com` or `admin.login.example.com`. You can optionally also associate the www subdomain as part of the same operation. You can add either CNAME or alias record for subdomain.

- *A wildcard* – For example, `*.example.com`. You can't use the www option in this case. You can specify a wildcard only as the immediate subdomain of a root domain and only on its own. These aren't valid specifications: `login*.example.com`, `*.login.example.com`. This wildcard specification associates all immediate subdomains, and doesn't associate the root domain itself. The root domain must be associated in a separate operation.

A more specific domain association overrides a less specific one. For example, `login.example.com` overrides `*.example.com`. The certificate and CNAME of the more specific association are used.

The following example shows how you can use multiple custom domain associations:

1. Associate `example.com` with the home page of your service. Enable the www to associate `www.example.com`.

2. Associate `login.example.com` with the login page of your service.

3. Associate `*.example.com` with a custom "not found" page.

Disassociate (unlink) a custom domain

You can disassociate (unlink) a custom domain from your App Runner service. When you unlink a domain, App Runner stops routing traffic from this domain to your web application.

> ℹ️ **Note**
>
> You must delete the records for the domain you disassociated from your DNS server.

App Runner internally creates certificates that track domain validity. These certificates are stored in AWS Certificate Manager (ACM). App Runner doesn't delete these certificates for 7 days after a domain is disassociated from your service or after the service is deleted.

Manage custom domains

Manage custom domains for your App Runner service using one of the following methods:

> ⓘ **Note**
>
> For better compatibility with App Runner services, we recommend that you use Amazon Route 53 as your DNS provider. If you don't use Amazon Route 53 to manage your public DNS records, contact your DNS provider to find out how to add records.
>
> If you're using Amazon Route 53 as your DNS provider, you can add either CNAME or alias record for *subdomain*. For *root domain*, ensure that you use alias record.

App Runner console

To associate (link) a custom domain using the App Runner console

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

    The console displays the service dashboard with a **Service overview**.
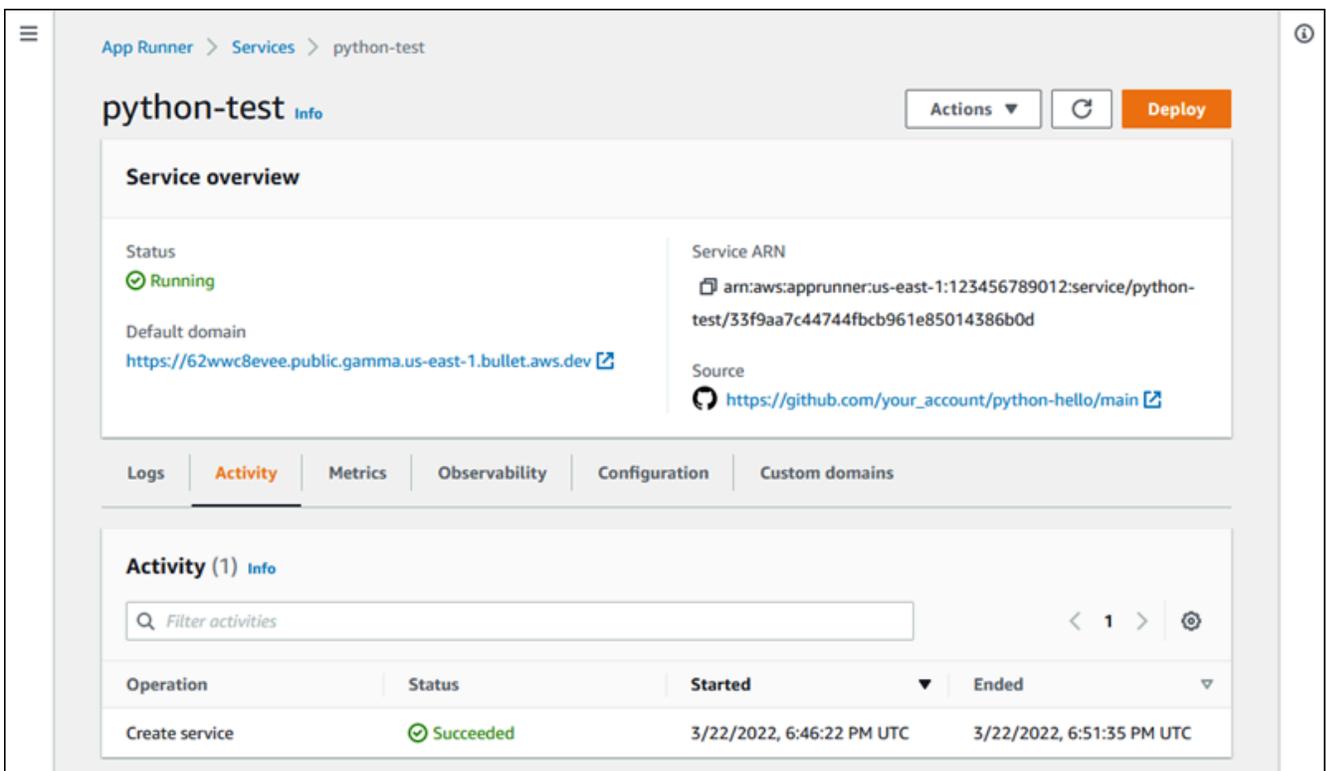
3. On the service dashboard page, choose the **Custom domains** tab.

    The console shows the custom domains that are associated with your service, or **No custom domains**.
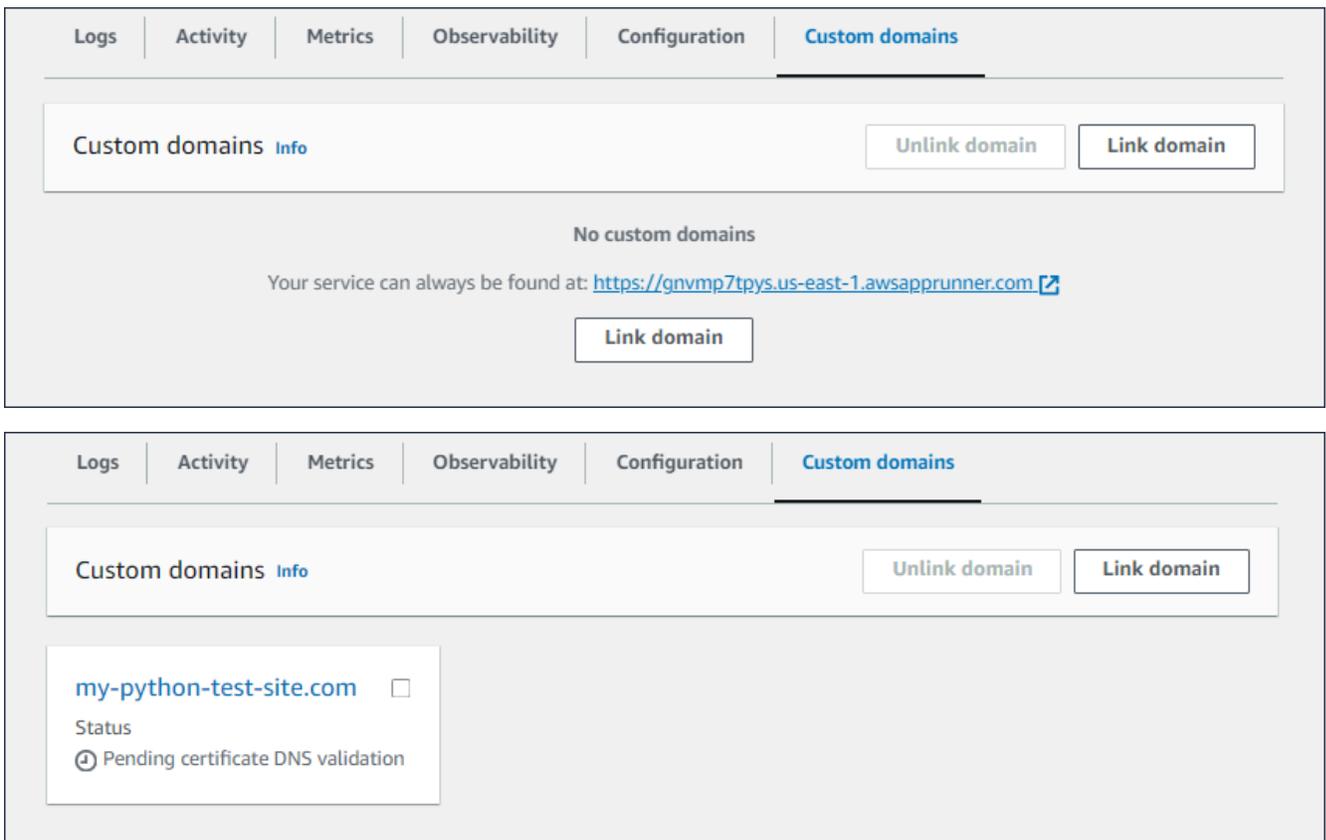
4. On the **Custom domains** tab, choose **Link domain**.

5. The **Link custom domain** page displays.

   - If your custom domain is registered with Amazon Route 53, select **Amazon Route 53** for **Domain registrar**.

     a. Select the **Domain name** from the drop-down list. This list displays the name of your Route 53 domain names and the hosted zone id.

     > ⓘ **Note**
     >
     > You must first create a Route 53 domain using the Amazon Route 53 service from the same AWS account that you use to manage your other App Runner resources.

     b. Select the **DNS record type**.

     c. Choose **Link domain**.

> **ⓘ Note**
>
> If App Runner displays an error message stating that the automatic configuration attempt failed, you can proceed by configuring the DNS records manually. This issue can arise if the same domain name was previously unlinked from a service, without the DNS provider records that point to the service being deleted afterward. In this case App Runner is blocked from automatically overwriting these records. To finish the DNS configuration, skip the remainder of the steps in this procedure and then follow the instructions in [Configure an Amazon Route 53 alias record](#).

- If your custom domain is registered with another domain registrar, select **Non–Amazon** for **Domain registrar**.

  a. Enter the **Domain name**.

  b. Choose **Link domain**.

6.   The **Configure DNS** page displays.

   • If Amazon Route 53 is your DNS provider, then this step is optional.

     At this point App Runner has automatically configured your Route 53 domain with the
     required certificate validation and DNS records.

     > ⓘ **Note**
     >
     > If this same domain name was previously unlinked from a service, without the
     > DNS provider records that point to the service being deleted afterward, the
     > automatic configuration that App Runner attempted could have failed. To work
     > around this issue and complete the DNS association, proceed with steps **(1)**
     > and **(2)** on the **Configure DNS** page to copy the current target and certificate
     > records to the DNS provider.

   • Copy the certificate validation records and DNS target records, and add them
     to your DNS server. App Runner can then validate that you own or control the
     domain.

> **ⓘ Note**
>
> To auto-renew your custom domain certificates, make sure not to delete the certificate validation records from your DNS server.

- For more information about **Configure certificate validation**, see DNS Validation in the *AWS Certificate Manager User Guide*.

- For information about how to **Configure DNS target** with Amazon Route 53 alias record, see the section called "Configure an Amazon Route 53 alias record".

- If you're using a DNS provider other than Amazon Route 53, follow these steps.

  - Copy the certificate validation records and DNS target records, and add them to your DNS server. App Runner can then validate that you own or control the domain.

    > **ⓘ Note**
    >
    > To auto-renew your custom domain certificates, make sure not to delete the certificate validation records from your DNS server.

    - For more information about **Configure certificate validation**, see DNS Validation in the *AWS Certificate Manager User Guide*.

    - For instructions on how to configure a DNS target on other registrars, such as GoDaddy, Shopify, Hover and so on, refer to their specific documentation on adding DNS Target.

7.   Choose **Close**

The console shows the dashboard again. The **Custom domains** tab has a new tile showing
the domain that you just linked in the **Pending certificate DNS validation** status.
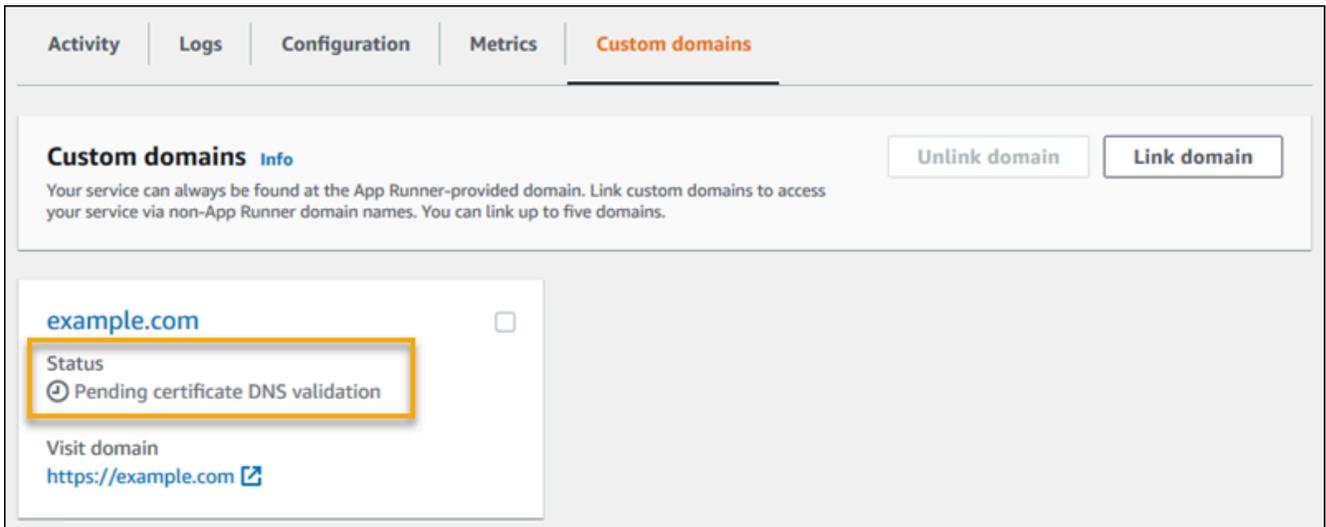
8. When the domain status changes to **Active**, verify that the domain works for routing traffic by browsing to it.
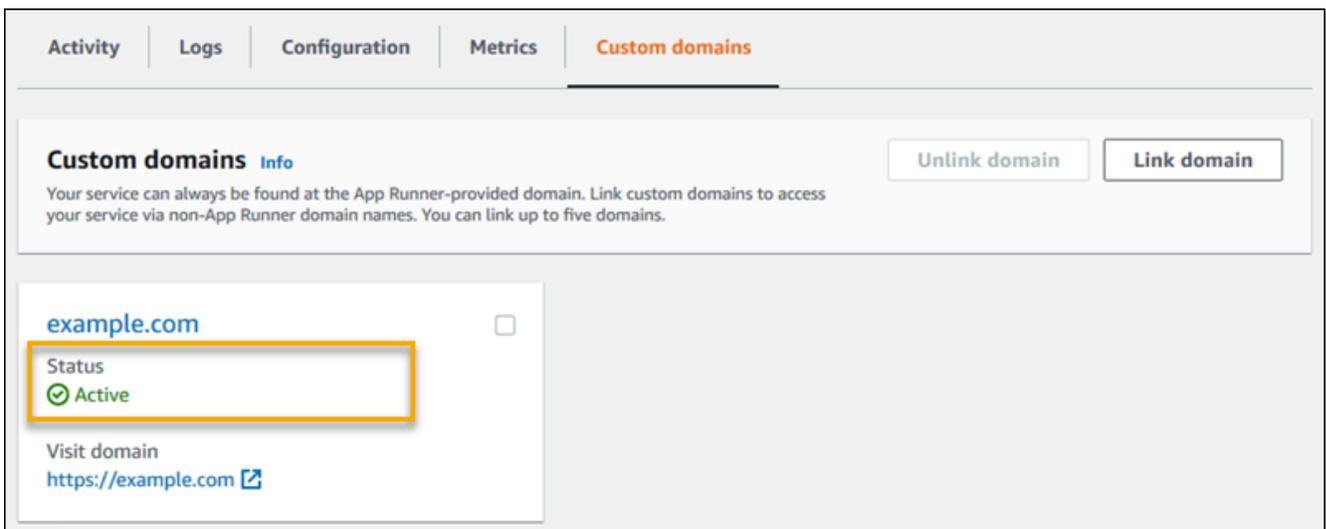
> **① Note**
>
> For instructions on how to troubleshoot errors related to custom domain, see the section called "Custom domain names".

To disassociate (unlink) a custom domain using the App Runner console

1. On the **Custom domains** tab, select the tile for the domain you want to disassociate, and then choose **Unlink domain**.

2.   In the **Unlink domain** dialog, verify the action by choosing **Unlink domain**.

> ⓘ **Note**
>
> You must delete the records for the domain that you disassociated from your DNS
> server.

App Runner API or AWS CLI

To associate a custom domain with your service using the App Runner API or AWS CLI, call
the AssociateCustomDomain API action. When the call succeeds, a CustomDomain object is
returned that describes the custom domain that's being associated with your service. The object
shows a CREATING status and contains a list of CertificateValidationRecord objects. The call
also returns the target alias that you can use to configure the DNS target. These are records that
you can add to your DNS.

To disassociate a custom domain from your service using the App Runner API or AWS CLI, call
the DisassociateCustomDomain API action. When the call succeeds, a CustomDomain object is
returned that describes the custom domain that's being disassociated from your service. The
object shows a DELETING status.

Topics

- Configure Amazon Route 53 alias record for your target DNS

Configure Amazon Route 53 alias record for your target DNS

> ⓘ **Note**
>
> You don't need to follow this procedure if Amazon Route 53 is your DNS provider. In
> this case App Runner automatically configures your Route 53 domain with the required
> certificate validation and DNS records to link to your App Runner web application.
> If App Runner's automatic configuration attempt failed, follow this procedure to complete
> the DNS configuration. If the same domain name was previously unlinked from a service,
> without the DNS provider records that point to the service being deleted afterward, App

Runner is blocked from automatically overwriting these records. This procedure explains how to manually copy them to your Route 53 DNS.

You can use Amazon Route 53 as your DNS provider to route traffic to your App Runner service. It's a highly available and scalable Domain Name System (DNS) web service. The Amazon Route 53 record contains the settings that control how traffic is routed to your App Runner service. You create either a CNAME record or an ALIAS record. For a comparison on CNAME and alias records, see Choosing between alias and non-alias records , in the *Amazon Route 53 Developer Guide*.

> ⓘ **Note**
>
> Amazon Route 53 currently supports alias record for services that are created after August 1, 2022.

Amazon Route 53 console

To configure Amazon Route 53 alias record

1. Sign in to the AWS Management Console and open the Route 53 console.
2. In the navigation pane, choose **Hosted zones**.
3. Choose the name of the hosted zone that you want to use to route traffic to your App Runner service.
4. Choose **Create record**.
5. Specify the following values:

   - **Routing policy**: Choose the applicable routing policy. For more information, see Choosing a routing policy.
   - **Record name**: Enter the domain name that you want to use to route traffic to your App Runner service. The default value is the name of the hosted zone. For example, if the name of the hosted zone is `example.com` and you want to use `acme.example.com` to route traffic to your environment, enter `acme`.
   - **Value/Route traffic to**: Choose **Alias to App Runner Application**, then choose the **Region** that the endpoint is from. Choose the domain name of the application that you want to route traffic to.
   - **Record type**: Accept the default, **A – IPv4 address**.

- **Evaluate target health**: Accept the default value, **Yes**.

6. Choose **Create records**.

The Route 53 alias record that you created gets propagated on all Route 53 servers within 60 seconds. When the Route 53 servers are propagated with your alias record, you can route traffic to your App Runner service by using the name of the alias record that you created.

For information about how to troubleshoot if the DNS changes are taking too long to propagate, see [Why is it taking so long for my DNS changes to propagate in Route 53 and public resolvers?](#).

Amazon Route 53 API or AWS CLI

To configure Amazon Route 53 alias record using the Amazon Route 53 API or AWS CLI call the [ChangeResourceRecordSets](#) API action. To learn about the target hosted zone id of Route 53, see [Service endpoints](#).

Pausing and resuming an App Runner service

If you need to disable your web application temporarily and stop the code from running, you can pause your AWS App Runner service. App Runner reduces the compute capacity for the service to zero.

When you're ready to run your application again, you can resume your App Runner service. App Runner provisions new compute capacity, deploys your application to it, and runs the application. Your application source isn't redeployed, and no build is necessary. Rather, App Runner resumes with your currently deployed version. Your application retains its App Runner domain.

> ⚠️ **Important**
>
> - When you pause your service, your application loses its state. For example, any ephemeral storage that your code used is lost. For your code, pausing and resuming your service is the equivalent of deploying to a new service.
>
> - If you pause a service due to a flaw in your code (for example, a discovered bug or security issue), you can't deploy a new version before resuming the service.
>
>   Therefore, we recommend that you keep the service running and roll back to your last stable application version instead.

- When you resume your service, App Runner deploys the last application version that was used before you paused the service. If you added any new source versions since pausing your service, App Runner doesn't automatically deploy them even if automatic deployment is selected. For example, assume you have new image versions in the image repository or new commits in the code repository. These versions aren't automatically deployed .

  To deploy a newer version, perform a manual deployment or add another version to your source repository after resuming your App Runner service.

Pausing and deleting compared

Pause your App Runner service to *temporarily* disable it. Only compute resources are terminated, and your stored data (for example, the container image with your application version) remains intact. Resuming your service is quick—your application is ready to be deployed to new compute resources. Your App Runner domain remains the same.

Delete your App Runner service to *permanently* remove it. Your stored data is deleted. If you need to recreate the service, App Runner needs to fetch your source again, and also to build it if it's a code repository. Your web application gets a new App Runner domain.

When your service is paused

When you pause your service and it's in the **Paused** status, it responds differently to action requests, including API calls or console operations. When a service is paused, you can still perform App Runner actions that don't modify the definition or configuration of the service in a way that affects its runtime. In other words, if an action changes the behavior, scale, or other characteristics of a running service, you cannot perform that action on a paused service.

The following lists provide information about API actions that you can and cannot perform on a paused service. The equivalent console operations are similarly allowed or denied.

Actions you *can* perform on a paused service

- `List* and Describe* actions` – Actions that only read information.
- `DeleteService` – You can always delete a service.
- `TagResource, UntagResource` – Tags are associated with a service, but aren't part of its definition and don't affect its runtime behavior.

Actions you *cannot* perform on a paused service

- *StartDeployment actions* (or a [manual deployment](#) using the console)

- *UpdateService* (or a configuration change using the console, except for tagging changes)

- *CreateCustomDomainAssociations, DeleteCustomDomainAssociations*

- *CreateConnection, DeleteConnection*

Pause and resume your service

Pause and resume your App Runner service using one of the following methods:

App Runner console

To pause your service using the App Runner console

1. Open the [App Runner console](#), and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

   The console displays the service dashboard with a **Service overview**.
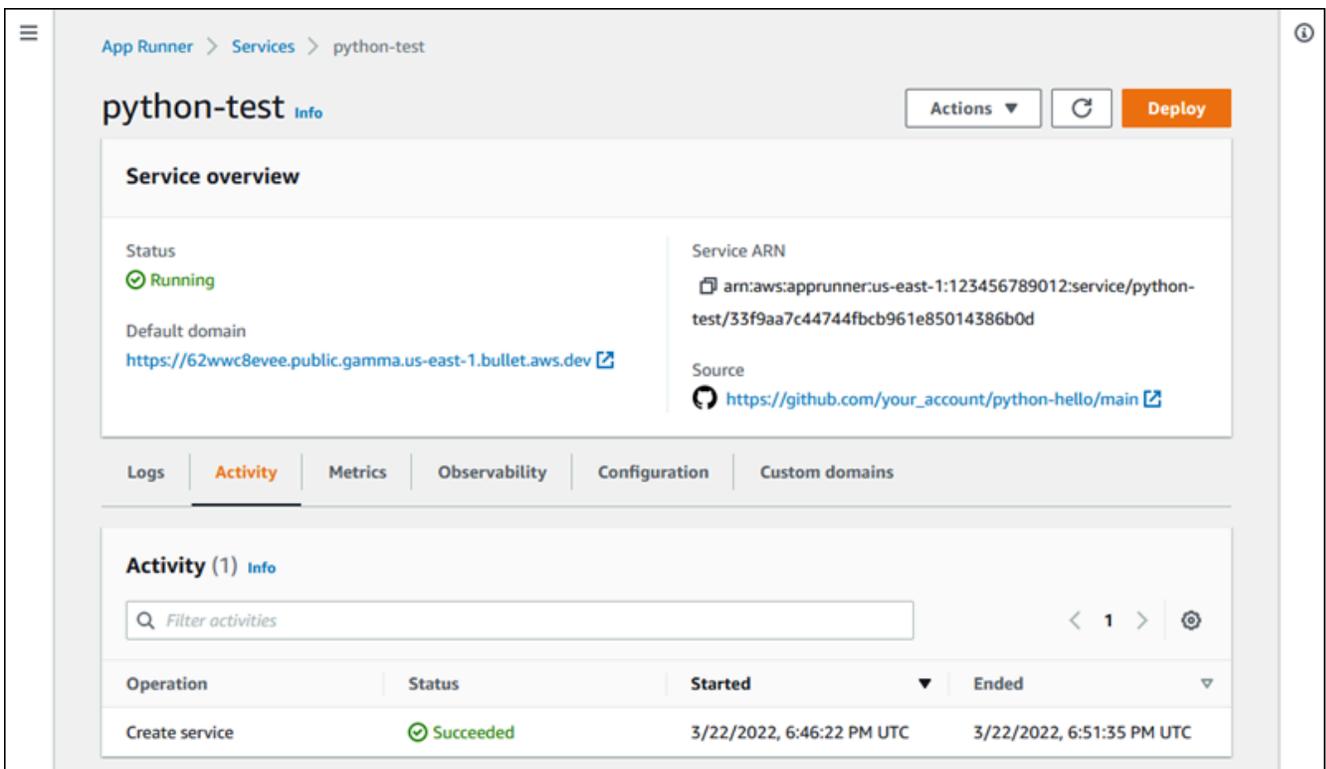
3. Choose **Actions**, and then choose **Pause**.

On the service dashboard page, the service **Status** changes to **Operation in progress**, and then changes to **Paused**. Your service is now paused.

To resume your service using the App Runner console

1. Choose **Actions**, and then choose **Resume**.

   On the service dashboard page, the service **Status** changes to **Operation in progress**.

2. Wait for the service to resume. On the service dashboard page, the service **Status** changes back to **Running**.

3. To verify that resuming the service is successful, on the service dashboard page, choose the **App Runner domain** value. It's the URL for your service's website. Verify that your web application is running correctly.

App Runner API or AWS CLI

To pause your service using the App Runner API or AWS CLI, call the [PauseService](#) API action. If the call returns a successful response with a [Service](#) object showing `"Status":` `"OPERATION_IN_PROGRESS"`, App Runner starts pausing your service.

To resume your service using the App Runner API or AWS CLI, call the [ResumeService](#) API action. If the call returns a successful response with a [Service](#) object showing `"Status":` `"OPERATION_IN_PROGRESS"`, App Runner starts resuming your service.

Deleting an App Runner service

When you want to terminate the web application that's running in your AWS App Runner service, you can delete the service. Deleting a service stops the running web service, removes the underlying resources, and deletes your associated data.

You might want to delete an App Runner service for one or more of the following reasons:

- *You don't need the web application anymore* – For example, it's retired, or it's a development version that you're done using.

- *You've reached the App Runner service quota* – You want to create a new service in the same AWS Region and you've reached the quota associated with your account. For more information, see [the section called "App Runner resource quotas"](#).

- *Security or privacy considerations* – You want App Runner to delete the data that it stores for your service.

Pausing and deleting compared

Pause your App Runner service to *temporarily* disable it. Only compute resources are terminated, and your stored data (for example, the container image with your application version) remains intact. Resuming your service is quick—your application is ready to be deployed to new compute resources. Your App Runner domain remains the same.

Delete your App Runner service to *permanently* remove it. Your stored data is deleted. If you need to recreate the service, App Runner needs to fetch your source again, and also to build it if it's a code repository. Your web application gets a new App Runner domain.

What does App Runner delete?

When you delete your service, App Runner deletes some associated items, and doesn't delete others. The following lists provide the details.

Items that App Runner deletes:

- *Container image* – A copy of the image that you deployed or the image that App Runner built from your source code. It's stored in Amazon Elastic Container Registry (Amazon ECR) using internal AWS accounts that are owned by App Runner.

- *Service configuration* – The configuration settings that are associated with your App Runner service. They're stored in Amazon DynamoDB using internal AWS accounts that are owned by App Runner.

Items that App Runner doesn't delete:

- *Connection* – You might have a connection that's associated with your service. An App Runner connection is a separate resource that might be shared among several App Runner services. If you don't need the connection anymore, you can explicitly delete it. For more information, see the section called "Connections".

- *Custom domain certificates* – If you link custom domains to an App Runner service, App Runner internally creates certificates that track domain validity. They're stored in AWS Certificate Manager (ACM). App Runner doesn't delete the certificate for seven days after a domain is

unlinked from your service or after the service is deleted. For more information, see the section called "Custom domain names".

Delete your service

Delete your App Runner service using one of the following methods:

App Runner console

To delete your service using the App Runner console

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

   The console displays the service dashboard with a **Service overview**.
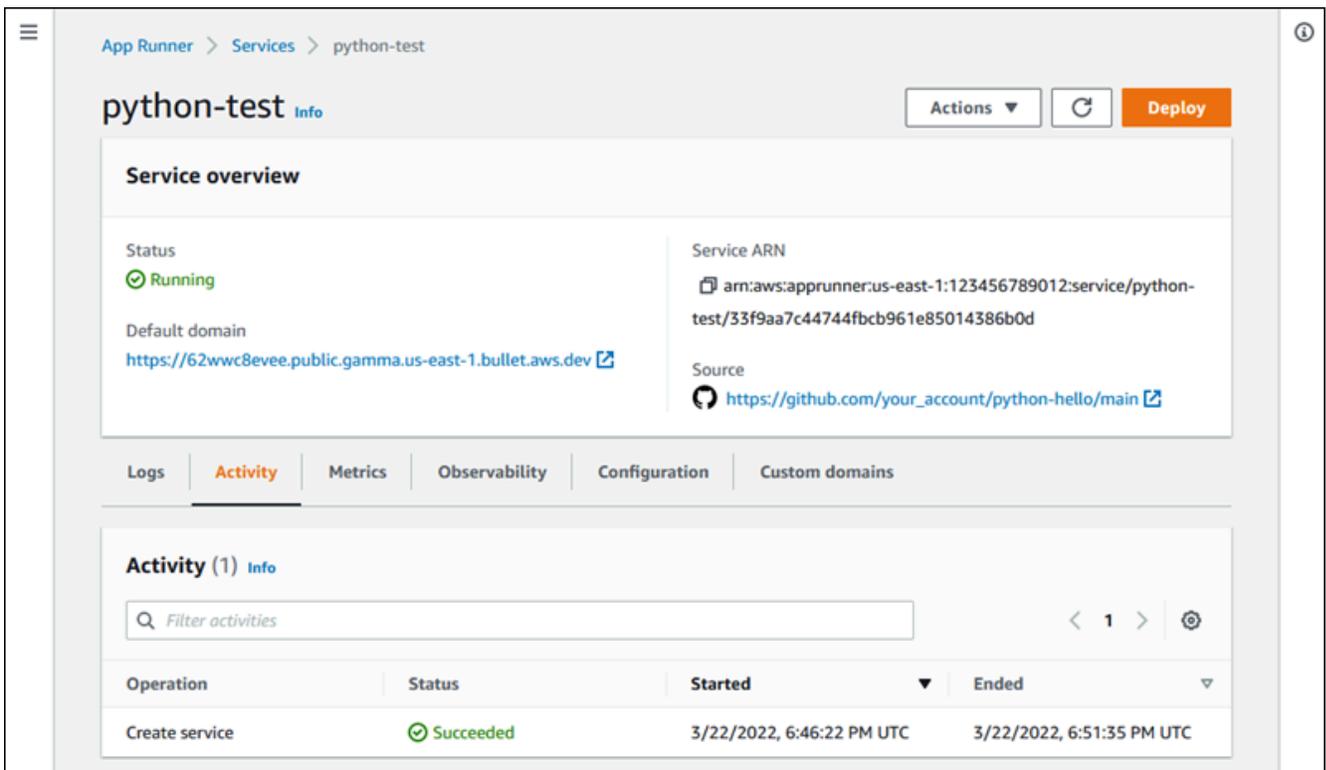
3. Choose **Actions**, and then choose **Delete**.

   The console takes you to the **Services** page. The deleted service displays the **Operation in progress** status, and then the service disappears from the list. Your service is now deleted.

App Runner API or AWS CLI

To delete your service using the App Runner API or AWS CLI, call the DeleteService API action. If the call returns a successful response with a Service object showing "Status": "OPERATION_IN_PROGRESS", App Runner starts deleting your service.

Referencing environment variables

With App Runner, you can reference secrets and configurations as environment variables in your service when you [create a service](#) or [update a service](#).

You can reference non-sensitive configuration data such as timeouts and retry counts in **Plain Text** as key-value pairs. The configuration data that you reference in **Plain Text** isn't encrypted and is visible to others in App Runner service configuration and application logs.

> ⓘ **Note**
>
> For security reasons, don't reference any sensitive data in **Plain Text** in your App Runner service.

Referencing sensitive data as environment variables

App Runner supports securely referencing sensitive data as environment variables in your service. Consider storing the sensitive data that you want to reference in *AWS Secrets Manager* or *AWS Systems Manager Parameter Store*. Then, you can securely reference them in your service as environment variables from App Runner console or by calling the API. This effectively separates secret and parameter management from your application code and service configuration, improving the overall security of your applications running on App Runner.

> ⓘ **Note**
>
> App Runner doesn't charge you for referencing Secrets Manager and SSM Parameter Store as environment variables. However, you pay standard pricing for using Secrets Manager and SSM Parameter Store.
> For more information about pricing, see the following:
>
> - [AWS Secrets Manager Pricing](#)
> - [AWS SSM Parameter Store Pricing](#)

The following is the process to reference sensitive data as environment variables:

1. Store sensitive data, such as API keys, database credentials, database connection parameters, or application versions as secrets or parameters in either AWS Secrets Manager or AWS Systems Manager Parameter Store.

2. Update the IAM policy of your instance role so App Runner can access the secrets and parameters stored in Secrets Manager and SSM Parameter Store. For more information, see [Permissions](#).

3. Securely reference the secrets and parameters as environment variables by assigning a name and providing their Amazon Resource Name (ARN). You can add environment variables when you [create a service](#) or [update a service's configuration](#). You can use one of the following options to add environment variables:

   - App Runner console

   - App Runner API

   - `apprunner.yaml` configuration file

   > **ⓘ Note**
   >
   > You cannot assign PORT as a name for an environment variable when creating or updating your App Runner service. It's a reserved environment variable for App Runner service.

   For more information on how to reference secrets and parameters, see [Managing environment variables](#).

   > **ⓘ Note**
   >
   > Since App Runner only stores the reference to secret and parameter ARNs, the sensitive data isn't visible to others in the App Runner service configuration and application logs.

Considerations

- Make sure that you update your instance role with appropriate permissions to access the secrets and parameters in AWS Secrets Manager or in AWS Systems Manager Parameter Store. For more information, see [Permissions](#).

- Make sure that AWS Systems Manager Parameter Store is in the same AWS account as the service that you want to launch or update. Currently, you can't reference SSM Parameter Store parameters across accounts.

- When the secrets and parameter values are rotated or changed they are not automatically updated in your App Runner service. Redeploy your App Runner service as App Runner only pulls secrets and parameters during deployment.

- You also have the option to directly call AWS Secrets Manager and AWS Systems Manager Parameter Store through the SDK in your App Runner service.

- To avoid errors make sure of the following when referencing them as the environment variables:

  - You specify the right ARN of the secret.

  - You specify the right name or ARN of the parameter.

Permissions

To enable referencing secrets and parameters stored in the AWS Secrets Manager or SSM Parameter Store, add appropriate permissions to the IAM policy of your *instance role* to access Secrets Manager and SSM Parameter Store.

> **ⓘ Note**
>
> App Runner can't access resources in your account without your permission. You provide the permission through updating your IAM policy.

You can use the following policy templates to update your instance role in the IAM console. You can modify these policy templates to meet your specific requirement. For more information about updating an instance role, see Modifying a role in the *IAM User Guide*.

> **ⓘ Note**
>
> You can also copy the following templates from the App Runner console when creating the environment variables.

Copy, the following template to your instance role to add permission to reference *secrets* from *AWS Secrets Manager*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetSecretValue",
            "kms:Decrypt*"
        ],
        "Resource": [
            "arn:aws:secretsmanager:us-east-1:111122223333:secret:my-secret",
            "arn:aws:kms:us-east-1:111122223333:key/my-key"
         ]
     }
   ]
}
```

Copy the following template to your instance role to add permission to reference *parameters* from *AWS Systems Manager* Parameter Store.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
            "arn:aws:ssm:us-east-1:111122223333:parameter/my-parameter"
            ]
        }
    ]
}
```

Managing your environment variables

Manage the environment variables for your App Runner service by using one of the following methods:

- the section called "App Runner console"
- the section called "App Runner API or AWS CLI"

App Runner console

When you create a service or update a service on the App Runner console, you can add environment variables.

Adding environment variable

To add environment variable

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.
2. Based on whether you're creating or updating a service, perform one of the following steps:

   - If you're creating a new service, choose **Create an App Runner service** and go to **Configure Service**.
   - If you're updating an existing service, select the service that you want to update and go to the **Configuration** tab of the service.

3. Go to **Environment variables - optional** under **Service settings**.
4. Choose any of the following options based on your requirement:

   - Choose **Plain Text** from the **Environment variable source** and enter its key-value pairs under **Environment variable name** and **Environment variable value**, respectively.

     > **ⓘ Note**
     >
     > Choose **Plain Text** if you want to reference non-sensitive data. This data isn't encrypted and is visible to others in the App Runner service configuration and application logs.

   - Choose **Secrets Manager** from the **Environment variable source** to reference the secret that's stored in AWS Secrets Manager as environment variable in your service. Provide

the environment variable name and Amazon Resource Name (ARN) of the secret that you're referencing under **Environment variable name** and **Environment variable value** respectively.

- Choose **SSM Parameter Store** from the **Environment variable source** to reference the parameter stored in SSM Parameter Store as environment variable in your service. Provide the environment variable name and ARN of the parameter that you're referencing under **Environment variable name** and **Environment variable value** respectively.

> **ⓘ Note**
>
> - You cannot assign PORT as a name for an environment variable when creating or updating your App Runner service. It's a reserved environment variable for App Runner service.
>
> - If the SSM Parameter Store parameter is in the same AWS Region as the service that you want to launch, you can specify the full Amazon Resource Name (ARN) or the name of the parameter. If the parameter is in a different Region, you need to specify the full ARN.
>
> - Make sure that parameter that you're referencing to is in the same account as the service that you're launching or updating. Currently, you can't reference SSM Parameter Store parameter across accounts.

5. Choose **Add environment variable** to reference to another environment variable.

6. Expand **IAM policy templates** to view and copy the IAM policy templates provided for the AWS Secrets Manager and SSM Parameter Store. You only need to do this if you didn't yet update the IAM policy of your instance role with the required permissions. For more information, see [Permissions](#).

Removing environment variable

Before you delete an environment variable make sure that your application code is updated to reflect the same. If the application code is not updated, your App Runner service might fail.

To remove environment variables

1. Open the [App Runner console](#), and in the **Regions** list, select your AWS Region.

2. Go to **Configuration** tab of the service you want to update.

3. Go to **Environment variables - optional** under **Service settings**.

4. Choose **Remove** next to the environment variable that you want to remove. You receive a message to confirm the deletion.

5. Choose **Delete**.

App Runner API or AWS CLI

You can reference sensitive data stored in Secrets Manager and SSM Parameter Store by adding them as environment variables in your service.

> **ⓘ Note**
>
> Update the IAM policy of your instance role so App Runner can access secrets and parameters stored in Secrets Manager and SSM Parameter Store. For more information, see [Permissions](#).

To reference secrets and configurations as environment variables

1. Create a secret or configuration in the Secrets Manager or SSM Parameter Store.

   The following examples show how to create a secret and a parameter using the SSM Parameter Store.

   Example Creating a secret - Request

   The following example shows how to create a secret that represents the database credential.

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

   Example Creating a secret - Response

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

Example Creating a configuration - Request

The following example shows how to create a parameter that represents the RDS connection string.

```
aws systemsmanager put-parameter \
—name DevRdsConnectionString \
—value "mysql2://dev-mysqlcluster-rds.com:3306/diegor" \
—type "String" \
—description "Rds connection string for development account."
```

Example Creating a configuration - Response

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

2. Reference the secrets and configurations that are stored in Secrets Manager and SSM Parameter Store by adding them as environment variables. You can add environment variables when you create or update your App Runner service.

The following examples shows how to reference secrets and configurations as environment variables on a code-based and an image-based App Runner service.

Example Input.json file for image-based App Runner service

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

 "Credential1":"arn:aws:secretsmanager:<region>:<aws_account_id>:secret:XXXXXXXXXXX",
          "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 Image-based App Runner service – Request

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

Example Image-based App Runner service – Response

```
{
...
     "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 Input.json file for code-based App Runner service

```
{
  "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 Code-based App Runner service – Request

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

Example Code-based App Runner service – Response

```
{
...
    "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. The `apprunner.yaml` model is updated to reflect the added secrets.

   The following is an example of the updated `apprunner.yaml`model.

   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"
```

Networking with App Runner

This chapter describes networking configurations for your AWS App Runner services.

From this chapter you will learn the following:

- How to configure your incoming traffic for private and public endpoints. For more information, see [Setting up networking configurations for incoming traffic](#).
- How to configure your outgoing traffic to access to other applications running in an Amazon VPC. For more information, see [Enabling VPC access for outgoing traffic](#).

Topics

- [Terminology](#)
- [Setting up networking configurations for incoming traffic](#)
- [Enabling VPC access for outgoing traffic](#)

Terminology

In order to know how to customize your network traffic to suit your needs, let's understand the following terms that are used in this chapter.

General Terms

To know what is needed to associate with an Amazon Virtual Private Cloud (VPC), let's understand the following terms:

- *VPC*: An *Amazon VPC* is a logically isolated virtual network that gives you complete control over your virtual networking environment, including resource placement, connectivity, and security. It is a virtual network that closely resembles a traditional network that you'd operate in your own data center.
- *VPC interface endpoint*: *VPC interface endpoint*, an AWS PrivateLink resource, connects a VPC to an endpoint service. Create a VPC interface endpoint to send traffic to endpoint services that use a Network Load Balancer to distribute traffic. Traffic destined for the endpoint service is resolved using DNS.
- *Regions*: Each *Region* is a separate geographic area where you can host an App Runner service.

- *Availability Zones*: An *Availability Zone* is an isolated location within an AWS Region. It is one or more discrete data centers with redundant power, networking, and connectivity. Availability Zones help you to make production applications highly available, fault tolerant, and scalable.

- *Subnets*: A *subnet* is a range of IP addresses in your VPC. A subnet must reside in a single Availability Zone. You can launch an AWS resource into a specified subnet. Use a public subnet for resources that must be connected to the internet, and a private subnet for resources that won't be connected to the internet.

- *Security groups*: A *security group* controls the traffic that is allowed to reach and leave the resources that it is associated with. Security groups provide an additional layer of security to protect the AWS resources in each subnet, giving you more control over your network traffic. When you create a VPC, it comes with a default security group. You can create additional security groups for each VPC. You can associate a security group only with resources within the VPC for which it is created.

- *Dual-stack*: A *dual-stack* is an address type that supports network traffic from both IPv4 and IPv6 endpoints.

Term specific to configuring outgoing traffic

VPC Connector

A *VPC Connector* is an App Runner resource that enables App Runner service to access applications that run in a private Amazon VPC.

Terms specific to configuring incoming traffic

To know how you can make your services privately accessible only from within an Amazon VPC, let's understand the following terms:

- *VPC Ingress Connection*: *VPC Ingress Connection* is an App Runner resource that provides an App Runner endpoint for incoming traffic. App Runner assigns the VPC Ingress Connection resource behind the scenes when you choose **Private endpoint** on the App Runner console for your incoming traffic. The VPC Ingress Connection resource connects your App Runner service to the VPC interface endpoint of the Amazon VPC.

> ### ⓘ Note
>
> If you are using App Runner API, the VPC Ingress Connection resource is not automatically created.

- *Private endpoint*: *Private endpoint* is an App Runner console option that you select to configure the incoming network traffic to be accessible from only within an Amazon VPC.

Setting up networking configurations for incoming traffic

You can configure your service to receive incoming traffic from private or public endpoint.

A **Public Endpoint** is the default configuration. It opens your service to any incoming traffic from the public internet. It also provides you with the flexibility to choose between IPv4 or dual-stack (IPv4 and IPv6) address types for your service.

A **Private endpoint** only allows traffic from an Amazon VPC to access your App Runner service. This is achieved by setting up a VPC interface endpoint, an AWS PrivateLink resource, for your App Runner service. Thereby, creating a private connection between the Amazon VPC and your App Runner service. It also provides you with the flexibility to choose between IPv4 or dual-stack (IPv4 and IPv6) address types for your service.

The following are the topics that are covered as part of setting up your network configurations for incoming traffic:

- How to configure your incoming traffic to make your service privately available only from within an Amazon VPC. For more information, see Enabling Private endpoint for incoming traffic.
- How to configure your service to receive internet traffic from the dual-stack address type. For more information, see Enabling dual stack for public incoming traffic.

Headers

With App Runner you can access the original source IPv4 and IPv6 addresses of the traffic entering your application. The original source IP addresses are preserved by assigning the `X-Forwarded-For` request header to them. This enables your applications to fetch the original source IP addresses when needed.

> ⓘ **Note**
>
> If your service is configured to use private endpoint, then `X-Forwarded-For` request header cannot be used to access original source IP addresses. If used, it retrieves false values.

Enabling Private endpoint for incoming traffic

By default when you create an AWS App Runner service, the service is accessible over the internet. However, you can also make your App Runner service private and only accessible from within an Amazon Virtual Private Cloud (Amazon VPC).

With your App Runner service private, you have complete control over incoming traffic, adding an additional layer of security. This is helpful in a variety of use cases, including running internal APIs, corporate web applications, or applications that are still in development that require a greater level of privacy and security, or have the need to meet specific compliance requirements.

> ⓘ **Note**
>
> If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. As a result, source IP rules for App Runner private services that are associated with WAF web ACLs do not adhere to IP based rules.
> To learn more about infrastructure security and security groups, including best practices, see the following topics in the *Amazon VPC User Guide*: Control network traffic and Control traffic to your AWS resources using security groups.

When your App Runner service is private, you can access your service from within an Amazon VPC. An internet gateway, NAT device, or VPN connection isn't required.

> ⓘ **Note**
>
> App Runner supports IPv4 and dual-stack, (both IPv4 and IPv6), for both incoming traffic and outgoing traffic.

Considerations

- Before you set up a VPC interface endpoint for App Runner, review Considerations in the *AWS PrivateLink Guide*.

- VPC endpoint policies are not supported for App Runner. By default, full access to App Runner is allowed through the VPC interface endpoint. Alternatively, you can associate a security group with the endpoint network interfaces to control traffic to App Runner through the VPC interface endpoint.

- If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. As a result, source IP rules for App Runner private services that are associated with WAF web ACLs do not adhere to IP based rules.

- After you enable a Private endpoint, your service is only accessible from your VPC, and can't be accessed from the internet.

- For higher availability, it's recommended that you select at least two subnets across Availability Zone different for the VPC interface endpoint. We don't recommend using only one subnet.

- If you're choosing the dual stack option for the IP address type, ensure your subnets can support dual-stack traffic.

- You can use the same VPC interface endpoint to access multiple App Runner services in a VPC.

For information on the terms used in this section, see Terminology.

Permissions

The following is the list of permissions required to enable **Private endpoint**:

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

VPC interface endpoint

A VPC interface endpoint is an *AWS PrivateLink* resource that connects an Amazon VPC to an endpoint service. You can specify which Amazon VPC you would like your App Runner service to be accessible in by passing a VPC interface endpoint. To create a VPC interface endpoint specify the following:

- The Amazon VPC to enable the connectivity.

- Add Security groups. By default, a security group is assigned to VPC interface endpoint. You can choose to associate a custom security group to bring further control to incoming network traffic.

- Add subnets. To ensure higher availability, it is recommended to select at least two subnets for each Availability Zone from which you'll access the App Runner service. A network interface endpoint is created in each subnet that you enable for the VPC interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for App Runner. A requester-managed network interface is a network interface that an AWS service creates in your VPC on your behalf.

- If you are using the API, add the App Runner VPC interface endpoint `Servicename`. For example,

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

You can create a VPC interface endpoint using one of the following AWS services:

- App Runner console. For more information, see Manage Private endpoint.

- Amazon VPC console or API, and AWS Command Line Interface (AWS CLI). For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

> ⓘ **Note**
>
> You're charged for each VPC interface endpoint that you use based on AWS PrivateLink Pricing. Therefore, for better cost efficiency, you can use the same VPC interface endpoint to access multiple App Runner services within a VPC. However, for better isolation, consider associating a different VPC interface endpoint for each of your App Runner services.

VPC Ingress Connection

A *VPC Ingress Connection* is an App Runner resource that specifies an App Runner endpoint for incoming traffic. App Runner assigns the VPC Ingress Connection resource behind the scenes when you choose **Private endpoint** on the App Runner console for your incoming traffic. Choose this option to only allow traffic from an Amazon VPC to access your App Runner service. The VPC Ingress Connection resource connects your App Runner service to the VPC interface endpoint of the Amazon VPC. You can create a VPC Ingress Connection resource only if you are using the API operations to configure the network settings for incoming traffic. For more information how to create VPC Ingress Connection resource, see [CreateVpcIngressConnection](#) in the *AWS App Runner API Reference*.

> ⓘ **Note**
>
> One VPC Ingress Connection resource of the App Runner can connect to one VPC interface endpoint of the Amazon VPC. Also, you can only create one VPC Ingress Connection resource for each App Runner service.

Private endpoint

Private endpoint is an App Runner console option that you can choose if you only want to receive incoming traffic from an Amazon VPC. Choosing the **Private endpoint** option on the App Runner console provides you with the option to connect your service to a VPC by configuring its *VPC interface endpoint*. Behind the scenes, App Runner assigns a VPC Ingress Connection resource to the VPC interface endpoint that you configure.

Summary

Make your service private by only allowing traffic from an Amazon VPC to access your App Runner service. To achieve this, you create a VPC interface endpoint for the selected Amazon VPC using either App Runner or Amazon VPC. On the App Runner console, you create a VPC interface endpoint when you enable the **Private endpoint** for the **Incoming traffic**. App Runner then automatically creates a *VPC Ingress Connection* resource and connects to the VPC interface endpoint and your App Runner service. This creates a private service connection that ensures that only traffic from the selected VPC can access your App Runner service.

Managing Private endpoints

Manage the Private endpoint for the incoming traffic using one of the following methods:

- the section called "App Runner console"
- the section called "App Runner API or AWS CLI"

> **ⓘ Note**
>
> If your App Runner application requires source IP/CIDR incoming traffic control rules,
> you must use security group rules for private endpoints instead of WAF web ACLs. This
> is because we currently don't support forwarding request source IP data to App Runner
> private services associated with WAF. As a result, source IP rules for App Runner private
> services that are associated with WAF web ACLs do not adhere to IP based rules.
> To learn more about infrastructure security and security groups, including best practices,
> see the following topics in the *Amazon VPC User Guide*: Control network traffic and Control
> traffic to your AWS resources using security groups.

App Runner console

When you create a service using the App Runner console, or when you update its configuration later, you can choose to configure the incoming traffic.

To configure your incoming traffic, choose one of the following.

- **Public endpoint**: To make your service accessible to all services over the internet. By default, **Public endpoint** is selected.
- **Private endpoint**: To make your App Runner service accessible from only within an Amazon VPC.

Enable Private endpoint

Enable a **Private endpoint** by associating it with VPC interface endpoint of the Amazon VPC you want to access. You can either create a new VPC interface endpoint or choose an existing one.

To create a VPC interface endpoint

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. Go to **Networking** section under **Configure service**.

3. Choose **Private endpoint**, for **Incoming network traffic**. Options to connect to a VCP using VPC interface endpoint opens.

4. Choose **Create new endpoint**. The **Create new VPC interface endpoint** dialog-box opens.

5. Enter a name for VPC interface endpoint.

6. Choose the required VPC interface endpoint from the available drop-down list.

7. Choose security group from the drop-down list. Adding security groups provides an additional layer of security to the VPC interface endpoint. It's recommended to choose two or more security groups. If you don't choose a security group, App Runner assigns a default security group to the VPC interface endpoint. Ensure that the security group rules don't block the resources that want to communicate with your App Runner service. The security group rules must allow resources that will interact with your App Runner service.

    > ### ⓘ Note
    >
    > If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. As a result, source IP rules for App Runner private services that are associated with WAF web ACLs do not adhere to IP based rules. To learn more about infrastructure security and security groups, including best practices, see the following topics in the *Amazon VPC User Guide*: Control network traffic and Control traffic to your AWS resources using security groups.

8. Choose the required subnets from the drop-down list. It is recommended to select at least two subnets for each Availability Zone from which you'll access the App Runner service.

    > ### ⓘ Note
    >
    > If you're configuring the endpoint for dual stack make sure that your infrastructure and VPC endpoint supports dual-stack traffic.

9. (Optional) Choose **Add new tag** and enter the tag key and the tag value.

10. Choose **Create**. The **Configure service** page opens showing the message of successful creation of VPC interface endpoint on the top bar.

To choose an existing VPC interface endpoint

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. Go to **Networking** section under **Configure service**.

3. Choose **Private endpoint**, for **Incoming network traffic**. Options to connect to a VPC using VPC interface endpoint opens. A list of available VPC interface endpoints is shown.

4. Choose the required VPC interface endpoint listed under **VPC interface endpoints**.

5. Choose **Next** to create your service. App Runner enables the Private endpoint.

> ⓘ **Note**
>
> After your service is created you can choose to edit the Security groups and Subnets associated with the VPC interface endpoint, if required.

To check the details of the **Private endpoint**, go to your service and expand the **Networking** section under **Configuration** tab. It shows details of the VPC and the VPC interface endpoint associated with the **Private endpoint**.

Update VPC interface endpoint

After your App Runner service is created, you can edit the VPC interface endpoint associated with the Private endpoint.

> ⓘ **Note**
>
> You cannot update the **Endpoint name** and the **VPC** fields.

To update VPC interface endpoint

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. Go to your service and choose **Networking configurations** on the left panel.

3. Choose **Incoming traffic** to view the VPC interface endpoints associated with the respective services.

4. Choose the VPC interface endpoint you want to edit.

5. Choose **Edit**. The dialog-box to edit the VPC interface endpoint opens.

6. Choose the required **Security groups** and **Subnets** and click **Update**. The page showing the VPC interface endpoint details opens with the message of successful update of the VPC interface endpoint on the top bar.

> ⓘ **Note**
>
> If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. As a result, source IP rules for App Runner private services that are associated with WAF web ACLs do not adhere to IP based rules. To learn more about infrastructure security and security groups, including best practices, see the following topics in the *Amazon VPC User Guide*: Control network traffic and Control traffic to your AWS resources using security groups.

Delete VPC interface endpoint

If you don't want your App Runner service to be privately accessible, you can set your incoming traffic to **Public**. Changing to **Public** removes the Private endpoint, but it doesn't delete the VPC interface endpoint

To delete VPC interface endpoint

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. Go to your service and choose **Networking configurations** on the left panel.

3. Choose **Incoming traffic** to view the VPC interface endpoints associated with the respective services.

> ⓘ **Note**
>
> Before deleting a VPC interface endpoint, remove it from all the services its connected to by updating your service.

4. Choose **Delete**.

If there are services connected to VPC interface endpoint, then you receive a **Cannot delete VPC interface endpoint** message. If there are no services connected to the VPC interface endpoint, you receive a message to confirm the deletion.

5. Choose **Delete**. The **Network configurations** page opens for the **Incoming traffic** with the message of successful deletion of the VPC interface endpoint on the top bar.

App Runner API or AWS CLI

You can deploy an application on App Runner that is only accessible from within an Amazon VPC.

For information on permissions required to make your service private, see the section called "Permissions".

To create a private service connection to Amazon VPC

1. Create a VPC interface endpoint, an AWS PrivateLink resource, to connect to App Runner. To do this, specify subnets and security groups to associate with the application. The following is an example of creating a VPC interface endpoint.

> **ⓘ Note**
>
> If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. As a result, source IP rules for App Runner private services that are associated with WAF web ACLs do not adhere to IP based rules. To learn more about infrastructure security and security groups, including best practices, see the following topics in the *Amazon VPC User Guide*: Control network traffic and Control traffic to your AWS resources using security groups.

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. Reference the VPC interface endpoint by using the [CreateService](#) or [UpdateService](#) App
    Runner API actions through the CLI. Configure your service to not be publicly accessible.
    Set IsPubliclyAccessible to False in the IngressConfiguration member of the
    NetworkConfiguration parameter. Optionally you can set the IpAddressType field to
    IPV4 or DUAL_STACK. If not set, this value defaults to IPV4. The following example references
    the VPC interface endpoint.

    Example

    ```
    aws apprunner create-service
      --network-configuration:
        {
        "IngressConfiguration":
          {
          "IsPubliclyAccessible": False
          },
          "IpAddressType": "IPV4"
        }
      --service-name: com.amazonaws.us-east-1.apprunner.requests
      --source-configuration: <source_configuration>
    ```

3. Call the create-vpc-ingress-connection API action to create the VPC Ingress
    Connection resource for App Runner and associate it with the VPC interface endpoint you
    created in the previous step. It returns a domain name that is used to access your service in the
    specified VPC. The following is an example of creating a VPC Ingress Connection resource.

    Example Request

    ```
    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 Response

    ```
    {
        "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>
    }
```

Update VPC Ingress Connection

You can update the VPC Ingress Connection resource. The VPC Ingress Connection must be in one of the following states to be updated:

- AVAILABLE

- FAILED_CREATION

- FAILED_UPDATE

The following is an example of updating a VPC Ingress Connection resource.

Example Request

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

Example Response

```
{
    "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>
}
```

Delete VPC Ingress Connection

You can delete the VPC Ingress Connection resource if you no longer need the private connection to the Amazon VPC.

The VPC Ingress Connection must be in one of the following states to be deleted:

- AVAILABLE

- FAILED CREATION

- FAILED UPDATE

- FAILED DELETION

The following is an example of deleting a VPC Ingress Connection

Example Request

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

Example Response

```
{
    "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>
}
```

Use the following App Runner API actions to manage the private inbound traffic for your service.

- CreateVpcIngressConnection – Create a new VPC Ingress Connection resource. App Runner requires this resource when you want to associate your App Runner service to an Amazon VPC endpoint.

- ListVpcIngressConnections – Return a list of AWS App Runner VPC Ingress Connection endpoints that are associated with your AWS account.

- DescribeVpcIngressConnection – Return a full description of AWS App Runner VPC Ingress Connection resource.

- UpdateVpcIngressConnection – Update the AWS App Runner VPC Ingress Connection resource.

- DeleteVpcIngressConnection – Delete an App Runner VPC Ingress Connection resource that's associated with the App Runner service.

For more information on using App Runner API, see [App Runner API Reference guide](#).

Enabling IPv6 for incoming traffic

If you want your service to receive incoming network traffic from IPv6 addresses, or from both IPv4 and IPv6 addresses, choose the **Dual-stack** address type for the endpoint. When you're creating a new application, you can find this setting under **Configure service** > **Networking** section. The following procedures explain how to enable IPv4 or dual-stack (IPv6 and IPv4) using the App Runner console or the App Runner API.

Managing dual stack for incoming traffic

Manage the dual-stack address type for incoming traffic using one of the following methods:

- [the section called "App Runner console"](#)
- [the section called "App Runner API or AWS CLI"](#)

> ⓘ **Note**
>
> The following procedures explain how to manage the network address type for **public** incoming traffic. For information about managing dual-stack or IPv4 address types for **private** endpoints, see [the section called "Manage Private endpoints"](#).

App Runner console

You can choose dual-stack address type for the incoming internet traffic, when you create a service using the App Runner console, or when you update its configuration later.

To enable dual-stack address type

1. When [creating](#) or [updating](#) a service, expand the **Networking** section under **Configure service**.

2. Choose **Public endpoint** for **Incoming network traffic**. If you select **Public endpoint**, the **Endpoint IP address type** option opens.

   See [the section called "Manage Private endpoints"](#) for a procedure to manage dual-stack or IPv4 address types for private endpoints.

3. Expand **Endpoint IP address type** to view the following IP address types.

- **IPv4**

- **Dual-stack (IPv4 and IPv6)**

> **ⓘ Note**
>
> If you do not expand **Endpoint IP address type** to make a selection, then App Runner assigns IPv4 as the default configuration.

4. Choose **Dual-stack (IPv4 and IPv6)**.

5. Choose **Next** and then **Create & Deploy** if you are creating a service. Else, choose **Save changes** if you are updating a service.

   When the service is deployed, your application starts receiving network traffic from both IPv4 and IPv6 endpoints.

To change the address type

1. Follow the steps to [update](#) a service and navigate to Networking.

2. Navigate to **Endpoint IP address type** under **Incoming network traffic** and select the required address type.

3. Choose **Save changes.** Your service is updated with your selection.

App Runner API or AWS CLI

When you call the [CreateService](#) or [UpdateService](#) App Runner API actions, use the `IpAddressType` member of the `NetworkConfiguration` parameter to specify the address type. The supported values that you can specify are `IPv4` and `DUAL_STACK`. Specify `DUAL_STACK` if you want your service to receive internet traffic from IPv4 and IPv6 endpoints. If you do not specify any value for `IpAddressType`, by default IPv4 is applied.

> **ⓘ Note**
>
> For private endpoint examples, see [the section called "App Runner API or AWS CLI"](#).

The following is the example to create a service with the dual stack as IP address. This example calls an `input.json` file.

Example Request to create a service with dual stack support

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

Example Contents of `input.json`

```
{
  "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 Response

```
{
  "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"
}
```

For more information on the API parameter, see [NetworkConfiguration](#).

Enabling VPC access for outgoing traffic

By default, your AWS App Runner application can send messages to public endpoints. This includes your own solutions, AWS services, and any other public website or web service. Your application

can even send messages to public endpoints of applications that run in a VPC from [Amazon Virtual Private Cloud](#) (Amazon VPC). If you don't configure a VPC when you launch your environment, App Runner uses the default VPC, which is public.

You can choose to launch your environment in a custom VPC to customize networking and security settings for outgoing traffic. You can enable your AWS App Runner service to access applications that run in a private VPC from Amazon Virtual Private Cloud (Amazon VPC). After you do this, your application can connect with and send messages to other applications that are hosted in an [Amazon Virtual Private Cloud](#) (Amazon VPC). Examples are an Amazon RDS database, Amazon ElastiCache, and other private services that are hosted in a private VPC.

VPC Connector

You can associate your service with a VPC by creating a VPC endpoint from the App Runner console, called VPC Connector. To create a VPC Connector, specify the VPC, one or more subnets, and optionally one or more security groups. After you configure a VPC Connector, you can use it with one or more App Runner services.

One-time latency

If you configure your App Runner service with a custom VPC connector for outbound traffic, it may experience a one-time startup latency of two to five minutes. The startup process waits until the VPC Connector is ready to connect to other resources before it sets the service status to *Running*. You can configure a service with a custom VPC connector when you first create it, or you can do so afterward by doing a service update.

Note that if you reuse the *same* VPC connector configuration for another service there wont be any latency. The VPC connector configuration is based on the security group and subnet combination. For a given VPC connector configuration, the latency only happens once, during the initial creation of the VPC Connector Hyperplane ENIs (elastic network interfaces).

More about Custom VPC connectors and AWS Hyperplane

The VPC connectors in App Runner are based on AWS Hyperplane, the internal Amazon network system that's behind several AWS resources, such as [Network Load Balancer](#), [NAT Gateway](#), and [AWS PrivateLink](#). The AWS Hyperplane technology provides high throughput and low latency capabilities, along with a higher degree of sharing. A Hyperplane ENI is created in your subnets when you create a VPC connector and associate it with your service. A VPC connector configuration

is based on a security group and subnet combination, and you can reference the same VPC Connector across multiple App Runner services. As a result, the underlying Hyperplane ENIs are shared across your App Runner services. This sharing is feasible, even as you scale up the number of tasks required to handle the request load, and results in more efficient utilization of the IP space in your VPC. For more information, see [Deep Dive on AWS App Runner VPC Networking](#) in the *AWS Container Blog*.

Subnet

Each subnet is in a specific Availability Zone. For high availability, we recommend that you select subnets across at least three Availability Zones. If the Region has less than three Availability Zones, we recommend you select your subnets across all the supported Availability Zones.

When selecting a subnet for your VPC, ensure that you choose a private subnet, not a public subnet. This is because, when you create a VPC Connector, the App Runner service creates a Hyperplane ENI in each of the subnets. Each Hyperplane ENI is assigned a private IP address only and is tagged with a tag of the *AWSAppRunnerManaged* key. If you choose a public subnet, errors will occur when running your App Runner service. However, if your service needs to access some services that are on the internet or other public AWS services, see [the section called "Considerations when selecting a subnet"](#).

Considerations when selecting a subnet

- When you connect your service to a VPC, the outbound traffic doesn't have access to the public internet. All outbound traffic from your application is directed through the VPC that your service is connected to. All networking rules for the VPC apply to the outbound traffic of your application. This means that your services can't access the public internet and AWS APIs. To gain access, do one of the following:

  - Connect the subnets to the internet through a [NAT Gateway.](#)

  - Set up [VPC endpoints](#) for the AWS services that you want to access. Your service stays within the Amazon VPC by using AWS PrivateLink.

- Some Availability Zones in some AWS Regions don't support the subnets that can be used with App Runner services. If you choose subnets in these Availability Zones, your service fails to be created or updated. For these situations, App Runner provides a detailed error message pointing to the unsupported subnets and Availability Zones. When this occurs, troubleshoot by removing the unsupported subnets from your request, and then try again.

- The subnets you select must all have the same IP address type, either IPv4 or dual-stack.

Security group

You can optionally specify the security groups that App Runner uses to access AWS under the specified subnets. If you don't specify security groups, App Runner uses the default security group of the VPC. The default security group allows all outbound traffic.

Adding a security group provides an additional layer of security to the VPC Connectors, giving you more control over the network traffic. The VPC Connector is only used for outbound communication from your application. You use outbound rules to allow communication to the desired destination endpoints. You must also ensure that any security groups that are associated with the destination resource have the appropriate inbound rules. Otherwise, these resources can't accept traffic that comes from the VPC Connector security groups.

> **ⓘ Note**
>
> When you associate your service with a VPC, the following traffic isn't affected:
>
> - **Inbound traffic** – Incoming messages that your application receives are unaffected by an associated VPC. The messages are routed through the public domain name that's associated with your service and don't interact with the VPC.
> - **App Runner traffic** – App Runner manages several actions on your behalf, such as pulling source code and images, pushing logs, and retrieving secrets. The traffic that these actions generate isn't routed through your VPC.

To know more about how AWS App Runner integrates with Amazon VPC, see AWS App Runner VPC Networking.

Manage VPC access

> **ⓘ Note**
>
> If you create an outbound traffic VPC connector for a service, the service startup process that follows will experience a one-time latency. You can set this configuration for a new service when you create it, or afterward, with a service update. For more information, see One-time latency in the *Networking with App Runner* chapter of this guide.

Manage VPC access for your App Runner services using one of the following methods:

App Runner console

When you create a service using the App Runner console, or when you update its configuration later, you can choose to configure your outgoing traffic. Look for the **Networking** configuration section on the console page. For **Outgoing network traffic**, choose in the following:

- **Public access**: To associate your service with public endpoints of other AWS services.

- **Custom VPC**: To associate your service with a VPC from Amazon VPC. Your application can connect with and send messages to other applications that are hosted in an Amazon VPC.

To enable Custom VPC

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. Go to **Networking** section under **Configure service**.
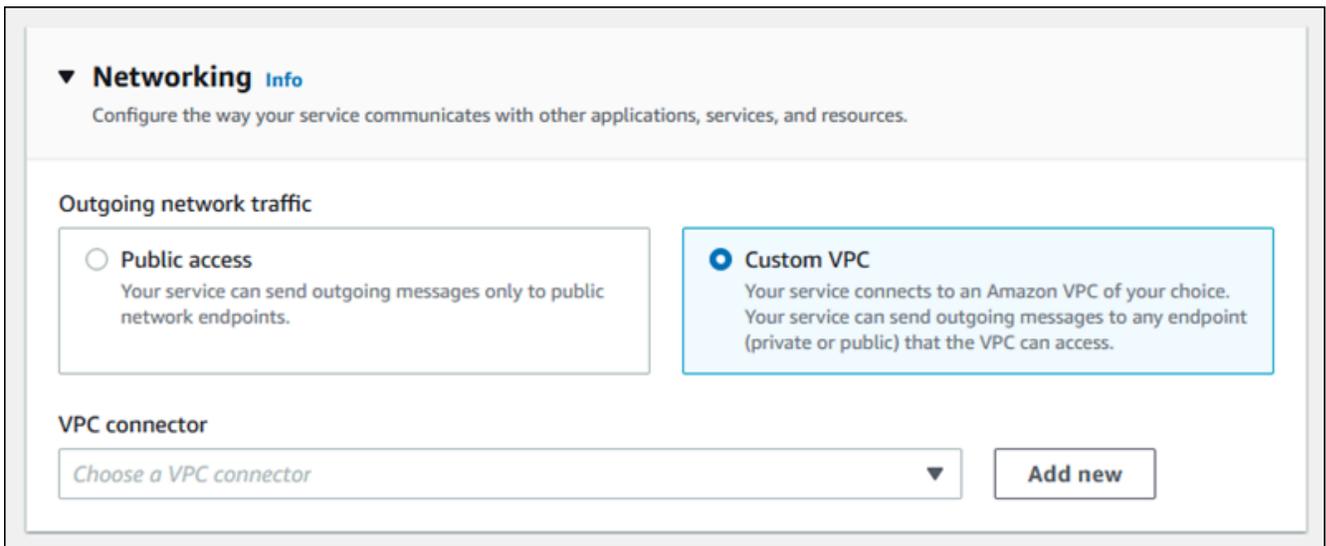
3. Choose **Custom VPC**, for **Outgoing network traffic**.

4. In the navigation pane, choose **VPC connector**.

   If you created the VPC connectors, the console displays a list of VPC connectors in your account. You can choose an existing VPC connector and choose **Next** to review your configuration. Then, move to the last step. Alternatively, you can add a new VPC connector using the following steps.

5. Choose **Add new** to create a new VPC connector for your service.

   Then, the **Add new VPC connector** dialog box opens.

6. Enter a name for your VPC connector and select the required VPC from the available list.

7. For **Subnets** select one subnet for each Availability Zone that you plan to access the App Runner service from. For better availability, choose three subnets. Or, if there are less than three subnets, choose all available subnets.

    > ⓘ **Note**
    >
    > - Make sure you assign *private* subnets to the VPC connector. If you assign public subnets to VPC connector, your service fails to create or rolls back automatically during an update.
    >
    > - If your outgoing traffic is dual-stack, make sure that all of the subnets that you select are configured for dual-stack in the VPC Console.

8. (Optional) For **Security group**, select the security groups to associate with the endpoint network interfaces.

9. (Optional) To add a tag, choose **Add new tag** and enter the tag key and the tag value.

10. Choose **Add**.

    The details of the VPC connector you created appear under **VPC connector**.

11. Choose **Next** to review your configuration, and then choose **Create and deploy**.

    App Runner creates a VPC connector resource for you, and then associates it with your service. If the service is successfully created, the console shows the service dashboard, with a **Service overview** of the new service.

App Runner API or AWS CLI

When you call the [CreateService](#) or [UpdateService](#) App Runner API actions, use the `EgressConfiguration` member of the `NetworkConfiguration` parameter to specify a VPC connector resource for your service.

Use the following App Runner API actions to manage your VPC Connector resources.

- [CreateVpcConnector](#) – Creates a new VPC connector.

- [ListVpcConnectors](#) – Returns a list of the VPC connectors that are associated with your AWS account. The list includes full descriptions.

- [DescribeVpcConnector](#) – Returns a full description of a VPC connector.

- **DeleteVpcConnector** – Deletes a VPC connector. If you reach the VPC connector quota for your AWS account, you might need to delete unnecessary VPC connectors.

To deploy an application on App Runner that has outbound access to a VPC, you must first create a VPC Connector. You can do this by specifying one or more subnets and security groups to associate with the application. You can then reference the VPC Connector in the **Create** or **UpdateService** through the CLI, as illustrated in the following example:

```
            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
```

Observability for your App Runner service

AWS App Runner integrates with several AWS services to provide you with an extensive observability suite of tools for your App Runner service. Topics in this chapter describe these capabilities.

Topics

- [Tracking App Runner service activity](#)
- [Viewing App Runner logs streamed to CloudWatch Logs](#)
- [Viewing App Runner service metrics reported to CloudWatch](#)
- [Handling App Runner events in EventBridge](#)
- [Logging App Runner API calls with AWS CloudTrail](#)
- [Tracing for your App Runner application with X-Ray](#)

Tracking App Runner service activity

AWS App Runner uses a list of operations to keep track of activity in your App Runner service. An operation represents an asynchronous call to an API action, such as creating a service, updating a configuration, and deploying a service. The following sections show you how to track activity in the App Runner console and using the API.

Track App Runner service activity

Track your App Runner service activity using one of the following methods:

App Runner console

The App Runner console displays your App Runner service activity and provides more ways to explore operations.

To view activity of your service

1. Open the [App Runner console](#), and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

   The console displays the service dashboard with a **Service overview**.
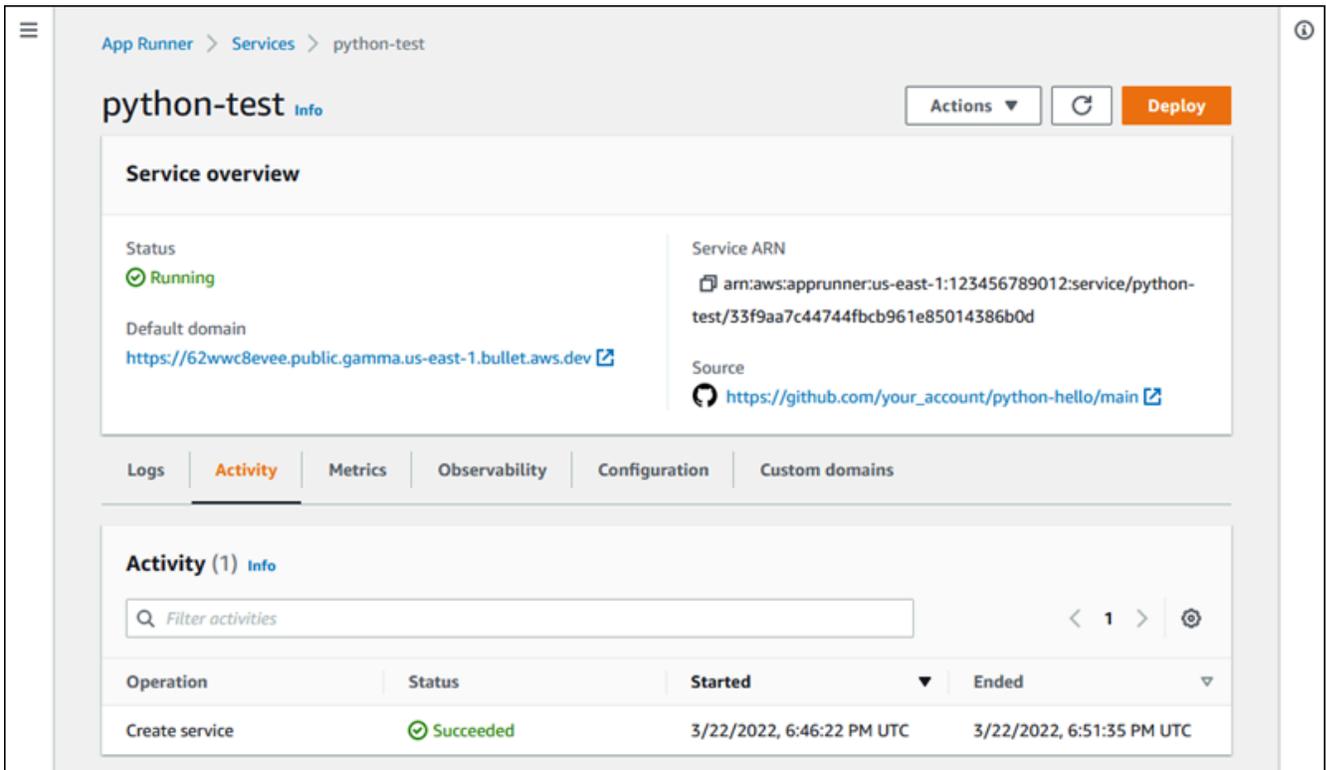
3. On the service dashboard page, choose the **Activity** tab, if it isn't already chosen.

   The console displays a list of operations.

4. To find specific operations, scope down the list by entering a search term. You can search for any value that appears in the table.

5. Choose any listed operation to see or download the related log.

App Runner API or AWS CLI

The [ListOperations](#) action, given the Amazon Resource Name (ARN) of an App Runner service, returns a list of operations that occurred on this service. Each list item contains an operation ID and some tracking details.

Viewing App Runner logs streamed to CloudWatch Logs

You can use Amazon CloudWatch Logs to monitor, store, and access log files that your resources in various AWS services generate. For more information, see [Amazon CloudWatch Logs User Guide](#).

AWS App Runner collects the output of your application deployments and of your active service and streams it to CloudWatch Logs. The following sections list App Runner log streams and show you how to view them in the App Runner console.

App Runner log groups and streams

CloudWatch Logs keeps log data in log streams that it further organizes in log groups. A *log stream* is a sequence of log events from a specific source. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

App Runner defines two CloudWatch Logs log groups, each with multiple log streams, for each App Runner service in your AWS account.

Service logs

The service log group contains logging output generated by App Runner as it manages your App Runner service and acts on it.

Log group name	Example
`/aws/apprunner/ `*`service-n`*`ame `*`/service-id`*` /service`	`/aws/apprunner/python-test/` `ac7ec8b51ff34746bcb6654e0bc` `b23da/service`

Within the service log group, App Runner creates an events log stream to capture activity in the lifecycle of your App Runner service. For example, this might be launching your application or pausing it.

In addition, App Runner creates a log stream for each long-running asynchronous operation that's related to your service. The log stream name reflects the operation type and specific operation ID.

A *deployment* is a type of operation. Deployment logs contain the logging output of the build and deployment steps that App Runner performs when you create a service or deploy a new version of your application. Deployment log stream names start with `deployment/`, and end with the ID of the operation that performs the deployment. This operation is either a CreateService call for the initial application deployment or a StartDeployment call for each further deployment.

Within a deployment log, each log message starts with a prefix:

- [AppRunner] – Output that App Runner generates during the deployment.

- [Build] – Output of your own build scripts.

Log stream name	Example
events	*N/A (fixed name)*
operation-type /*operation-id*	deployment/c2c8eeedea164f45 9cf78f12a8953390

Application logs

The application log group contains the output of your running application code.

Log group name	Example
/aws/apprunner/ *service-n ame* /*service-id* /application	/aws/apprunner/python-test/ ac7ec8b51ff34746bcb6654e0bcb23da/ application

Within the application log group, App Runner creates a log stream for each instance (scaling unit) that's running your application.

Log stream name	Example
instance/ *instance-id*	instance/1a80bc9134a84699b7 b3432ebeebb591

Viewing App Runner logs in the console

The App Runner console displays a summary of all logs for your service and allows you to view, explore, and download them.

To view logs for your service

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

   The console displays the service dashboard with a **Service overview**.

3. On the service dashboard page, choose the **Logs** tab.

   The console displays a few types of logs in several sections:

   - **Event log** – Activity in the lifecycle of your App Runner service. The console displays the latest events.

   - **Deployment logs** – Source repository deployments to your App Runner service. The console displays a separate log stream for each deployment.

   - **Application logs** – The output of the web application that's deployed to your App Runner service. The console combines the output from all running instances into a single log stream.

4. To find specific deployments, scope down the deployment log list by entering a search term. You can search for any value that appears in the table.

5. To view a log's content, choose **View full log** (event log) or the log stream name (deployment and application logs).

6. Choose **Download** to download a log. For a deployment log stream, select a log stream first.

7. Choose **View in CloudWatch** to open the CloudWatch console and use its full capabilities to explore your App Runner service logs. For a deployment log stream, select a log stream first.

> (i) **Note**
>
> The CloudWatch console is particularly useful if you want to view application logs of specific instances instead of the combined application log.

Viewing App Runner service metrics reported to CloudWatch

Amazon CloudWatch monitors your Amazon Web Services (AWS) resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are

variables you can measure for your resources and applications. You can also use it to create alarms that watch metrics. When a certain threshold is reached, CloudWatch sends notifications, or automatically makes changes to the monitored resources. For more information, see the Amazon CloudWatch User Guide.

AWS App Runner collects a variety of metrics that provide you with greater visibility into the usage, performance, and availability of your App Runner services. Some metrics track individual instances that run your web service, whereas others are at the overall service level. The following sections list App Runner metrics and show you how to view them in the App Runner console.

App Runner metrics

App Runner collects the following metrics relating to your service and publishes them to CloudWatch in the AWS/AppRunner namespace.

> ⓘ **Note**
>
> Prior to August 23, 2023, the **CPU utilization** and **Memory utilization** metrics were based on vCPU units and megabytes of memory utilized, instead of *percent utilization*, as calculated today. If your application ran on App Runner before this date, and you choose to go back to view metrics for this date on either the App Runner or the CloudWatch console, you'll see a display of the metrics in both units and will also see some irregularities as a result.

> ⚠ **Important**
>
> You'll need to update any CloudWatch alarms that are based on the *CPU utilization* and *Memory utilization* metric values prior to August 23, 2023. Update the alarms to trigger based on percentage utilization rather than vCPU or megabytes. For more information, see the Amazon CloudWatch User Guide.

Instance level metrics are collected for each instance (scaling unit) individually.

What's measured?	Metric	Description
CPU utilization	`CPUUtilization`	The percentage of average CPU usage during one-minute periods out of the total CPU usage reserved by the service configuration.
Memory utilization	`MemoryUtilization`	The percentage of average memory usage during one-minute periods out of the total memory reserved by the service configuration.

Service level metrics are collected for the entire service.

What's measured?	Metric	Description
CPU utilization	`CPUUtilization`	The percentage of aggregated CPU usage across all instances during one minute periods out of the total CPU usage reserved by the service configuration.
Memory utilization	`MemoryUtilization`	The percentage of aggregated memory usage across all instances during one minute periods out of the total memory reserved by the service configuration.
Concurrency	`Concurrency`	The approximate number of concurrent requests being handled by the service.
HTTP request count	`Requests`	The number of HTTP requests that the service received.
HTTP status counts	`2xxStatus Responses` `4xxStatus Responses`	The number of HTTP requests that returned each response status, grouped by category (2XX, 4XX, 5XX).

What's measured?	Metric	Description
	`5xxStatus Responses`	
HTTP request latency	`RequestLatency`	The time, in milliseconds, that it took your web service to process HTTP requests.
Instance counts	`ActiveInstances`	The number of instances that are processing HTTP requests for your service. ⓘ **Note** If the `ActiveInstances` metric displays zero, it means that there are no requests for the service. It does not indicate that the number of instances for your service is zero.

Viewing App Runner metrics in the console

The App Runner console graphically displays the metrics that App Runner collects for your service and provides more ways to explore them.

> ⓘ **Note**
>
> At this time, the console displays only service metrics. To view instance metrics, use the CloudWatch console.

To view logs for your service

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. In the navigation pane, choose **Services**, and then choose your App Runner service.

   The console displays the service dashboard with a **Service overview**.

3. On the service dashboard page, choose the **Metrics** tab.

The console displays a set of metrics graphs.
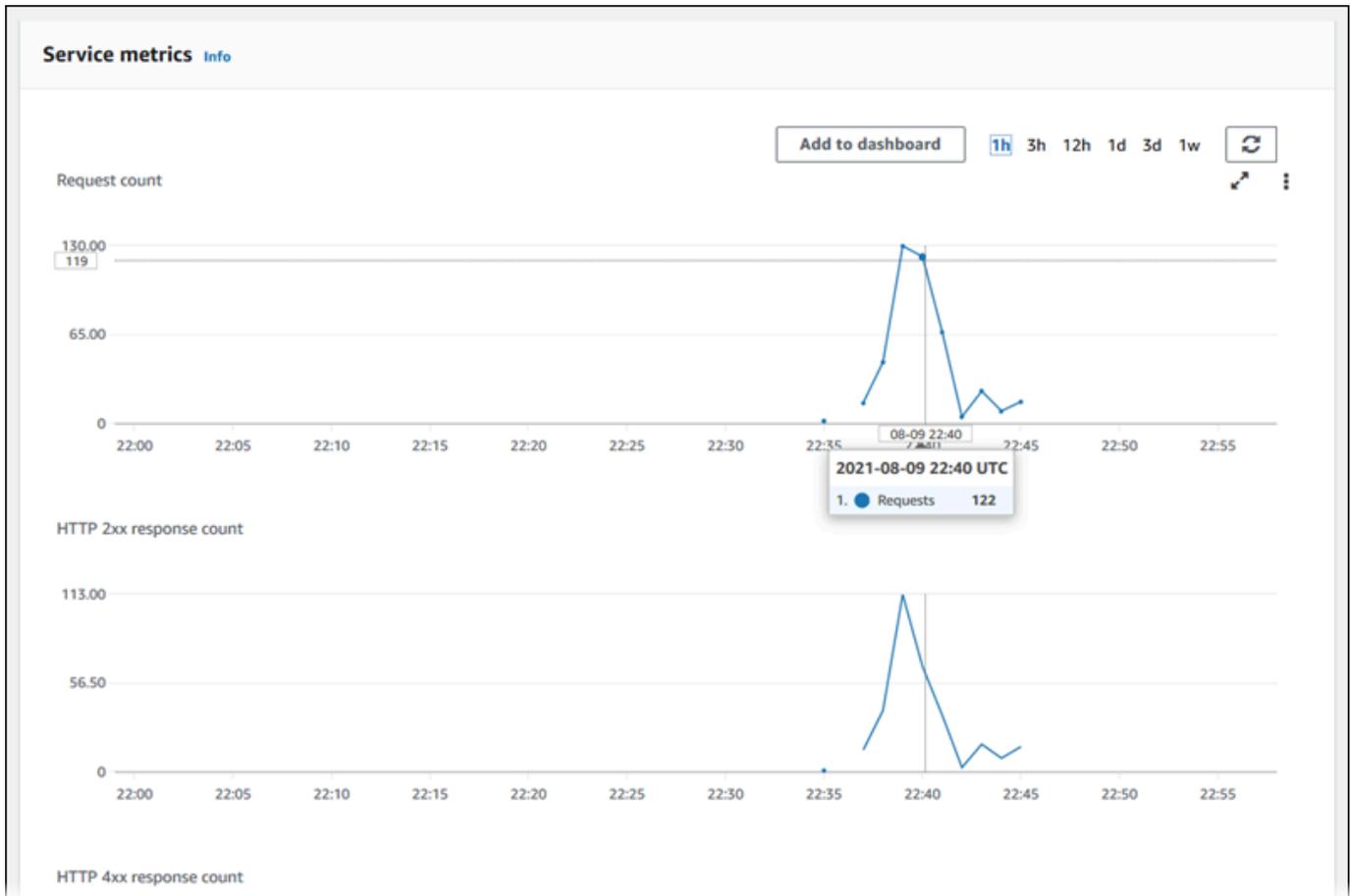
4. Choose a duration (for example, **12h**) to scope metrics graphs to the recent period of that duration.

5. Choose **Add to dashboard** at the top of one of the graph sections, or use the menu on any graph, to add the relevant metrics to a dashboard in the CloudWatch console for further investigation.

Handling App Runner events in EventBridge

Using Amazon EventBridge, you can set up event-driven rules that monitor a stream of real-time data from your AWS App Runner service for certain patterns. When a pattern for a rule is matched, EventBridge initiates an action in a target such as AWS Lambda, Amazon ECS, AWS Batch, and Amazon SNS. For example, you can set a rule for sending out email notifications by signaling an Amazon SNS topic whenever a deployment to your service fails. Or, you can set a Lambda function to notify a Slack channel whenever a service update fails. For more information about EventBridge, see Amazon EventBridge User Guide.

App Runner sends the following event types to EventBridge

- *Service status change* – A change in the status of an App Runner service. For example, a service status changed to `DELETE_FAILED`.
- *Service operation status change* – A change in the status of a long, asynchronous operation on an App Runner service. For example, a service started to create, a service update successfully completed, or a service deployment completed with errors.

Creating an EventBridge rule to act on App Runner events

An EventBridge *event* is an object that defines some standard EventBridge fields, such as the source AWS service and the detail (event) type, and an event-specific set of fields with the event details. To create an EventBridge rule, you use the EventBridge console to define an *event pattern* (which events should bet tracked) and specify a *target action* (what should be done on a match). An event pattern is similar to the events that it matches. You specify a subset of fields to match, and for each field, you specify a list of possible values. This topic provides examples of App Runner events and event patterns.

For more information about creating EventBridge rules, see [Creating a rule for an AWS service](#) in the *Amazon EventBridge User Guide*.

> ℹ️ **Note**
>
> Some services support *pre-defined patterns* in EventBridge. This simplifies how an event pattern is created. You select field values on a form, and EventBridge generates the pattern for you. At this time, App Runner doesn't support pre-defined patterns. You have to enter the pattern as a JSON object. You can use the examples in this topic as a starting point.

App Runner event examples

These are some examples to events that App Runner sends to EventBridge.

- A service status change event. Specifically, a service that changed from the `OPERATION_IN_PROGRESS` to the `RUNNING` status.

```
{
  "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"
    }
}
```

- An operation status change event. Specifically, an `UpdateService` operation that completed successfully.

```
{
  "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 event pattern examples

The following examples demonstrate event patterns that you can use in EventBridge rules to match one or more App Runner events. An event pattern is similar to an event. Include only the fields that you want to match, and provide a list instead of a scalar to each one.

- Match all service status change events for services of a specific account, where the service is no longer in RUNNING status.

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

- Match all operation status change events for services of a specific account, where the operation failed.

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

App Runner event reference

Service status change

A service status change event has `detail-type` set to `AppRunner Service Status Change`. It has the following detail fields and values:

```
"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"
```

Operation status change

An operation status change event has `detail-type` set to `AppRunner Service Operation Status Change`. It has the following detail fields and values:

```
"operationStatus": "see following table",
"serviceName": "your service name",
"serviceId": "your service ID",
"message": "see following table",
"severity": "varies"
```

The following table lists all possible status codes and related messages.

Status	Message
`CreateServiceStarted`	Service creation started.
`CreateServiceCompl etedSuccessfully`	Service creation completed successfully.
`CreateServiceFailed`	Service creation failed. For details, see service logs.
`DeleteServiceStarted`	Service deletion started.
`DeleteServiceCompl etedSuccessfully`	Service deletion completed successfully.

Status	Message
`DeleteServiceFailed`	Service deletion failed.
`UpdateServiceStarted`	
`UpdateServiceCompletedSuccessfully`	Service update completed successfully. New application and configuration is deployed.
	Service update completed successfully. New configuration is deployed.
`UpdateServiceFailed`	Service update failed. For details, see service logs.
`DeploymentStarted`	Deployment started.
`DeploymentCompletedSuccessfully`	Deployment completed successfully.
`DeploymentFailed`	Deployment failed. For details, see service logs.
`PauseServiceStarted`	Service pause started.
`PauseServiceCompletedSuccessfully`	Service pause completed successfully.
`PauseServiceFailed`	Service pause failed.
`ResumeServiceStarted`	Service resume started.
`ResumeServiceCompletedSuccessfully`	Service resume completed successfully.
`ResumeServiceFailed`	Service resume failed.

Logging App Runner API calls with AWS CloudTrail

App Runner is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in App Runner. CloudTrail captures all API calls for App Runner as events. The calls captured include calls from the App Runner console and code calls to the

App Runner API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for App Runner. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to App Runner, the IP address from where the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

App Runner information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in App Runner, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for App Runner, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

All App Runner actions are logged by CloudTrail and are documented in the AWS App Runner API Reference. For example, calls to the `CreateService`, `DeleteConnection`, and `StartDeployment` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.

- Whether the request was made by another AWS service.

For more information, see the [CloudTrail userIdentity Element](#).

Understanding App Runner log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, and request parameters. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateService` action.

> **ⓘ Note**
>
> For security reasons, some property values are redacted in the logs and replaced with the text `HIDDEN_DUE_TO_SECURITY_REASONS`. This prevents unintended exposure of secret information. However, you can still see that these properties were passed in the request or returned in the response.

Example CloudTrail log entry for the `CreateService` App Runner action

```
{
  "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"
  }
```

Tracing for your App Runner application with X-Ray

AWS X-Ray is a service that collects data about requests that your application serves, and provides tools you can use to view, filter, and gain insights into that data to identify issues and opportunities for optimization. For any traced request to your application, you can see detailed information not only about the request and response, but also about calls that your application makes to downstream AWS resources, microservices, databases and HTTP web APIs.

X-Ray uses trace data from the AWS resources that power your cloud applications to generate a detailed service graph. The service graph shows the client, your front-end service, and backend services that your front-end service calls to process requests and persist data. Use the service graph to identify bottlenecks, latency spikes, and other issues to solve to improve the performance of your applications.

For more information about X-Ray, see the AWS X-Ray Developer Guide.
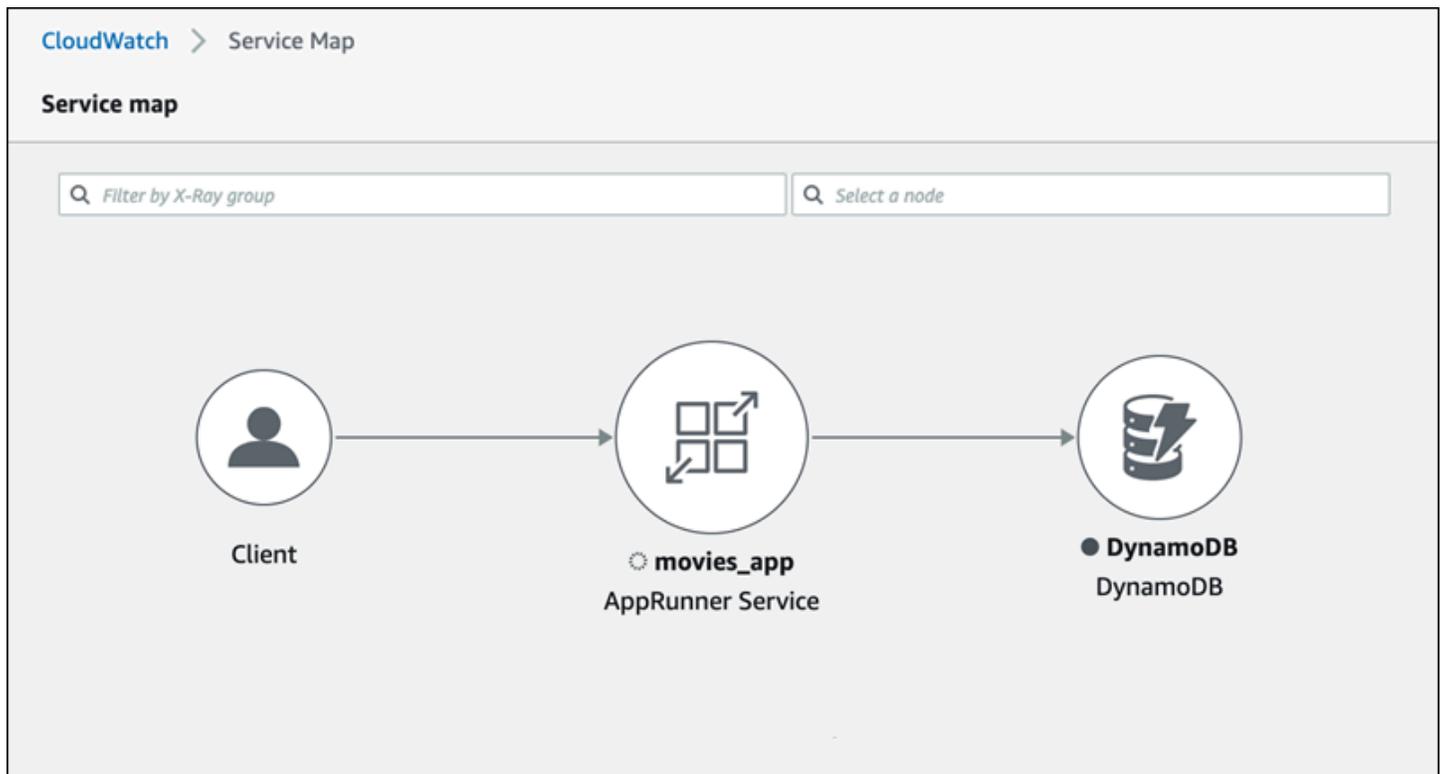
Instrument your application for tracing

Instrument your App Runner service application for tracing using OpenTelemetry, a portable telemetry specification. At this time, App Runner supports the AWS Distro for OpenTelemetry (ADOT), an OpenTelemetry implementation that collects and presents telemetry information using AWS services. X-Ray implements the tracing component.

Depending on the specific ADOT SDK that you use in your application, ADOT supports up to two instrumentation approaches: *automatic* and *manual*. For more information about instrumentation with your SDK, see the ADOT documentation, and choose your SDK on the navigation pane.

Runtime setup

The following are the general runtime setup instructions to instrument your App Runner service application for tracing.

To setup tracing for your runtime

1. Follow the instructions provided for your runtime in AWS Distro for OpenTelemetry (ADOT), to instrument your application.

2. Install the required OTEL dependencies in the `build` section of the `apprunner.yaml` file
    if you are using the source code repository or in the Dockerfile if you are using a container
    image.

3. Setup your environment variables in the `apprunner.yaml` file if you are using the source
    code repository or in the Dockerfile if you are using a container image.

Example Environment variables

> ⓘ **Note**
>
> The following example lists the important environment variables to add to the
> `apprunner.yaml` file. Add these environment variables to your Dockerfile if you are
> using a container image. However, each runtime can have their own idiosyncrasies
> and you may need to add more environment variables to the following list. For more
> information on your runtime specific instructions and examples on how to setup
> your application for your runtime, see [AWS Distro for OpenTelemetry](#) and go to your
> runtime, under *Getting Started*.

```
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 is an important environment variable for App
> Runner since the App Runner Otel collector doesn't accept metrics logging. It only
> accepts metrics tracing.

Runtime setup example

The following example demonstrates auto-instrumenting your application with the [ADOT Python SDK](#). The SDK automatically produces spans with telemetry data describing the values used by the Python frameworks in your application without adding a single line of Python code. You need to add or modify just a few lines in two source files.

First, add some dependencies, as shown in the following example.

Example requirements.txt

```
opentelemetry-distro[otlp]>=0.24b0
opentelemetry-sdk-extension-aws~=2.0
opentelemetry-propagator-aws-xray~=1.0
```

Then, instrument your application. The way to do it depends on your service source—source image or source code.

Source image

> When your service source is an image, you can directly instrument the Dockerfile that controls building your container image and running the application in the image. The following example shows an instrumented Dockerfile for a Python application. Instrumentation additions are emphasized in bold.
>
> **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

When your service source is a repository containing your application source, you indirectly instrument your image using App Runner configuration file settings. These settings control the Dockerfile that App Runner generates and uses to build the image for your application. The following example shows an instrumented App Runner configuration file for a Python application. Instrumentation additions are emphasized in bold.

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'
```

Add X-Ray permissions to your App Runner service instance role

To use X-Ray tracing with your App Runner service, you have to provide the service's instances with permissions to interact with the X-Ray service. You do this by associating an instance role with your service and adding a managed policy with X-Ray permissions. For more information about an App Runner instance role, see the section called "Instance role". Add the

`AWSXRayDaemonWriteAccess` managed policy to your instance role and assign it to your service during creation.

Enable X-Ray tracing for your App Runner service

When you [create a service](#), App Runner disables tracing by default. You can enable X-Ray tracing for your service as part of configuring observability. For more information, see [the section called "Manage observability"](#).

If you use the App Runner API or the AWS CLI, the [TraceConfiguration](#) object within the [ObservabilityConfiguration](#) resource object contains tracing settings. To keep tracing disabled, don't specify a `TraceConfiguration` object.

In both the console and API cases, be sure to associate your instance role discussed in the previous section with your App Runner service.

View X-Ray tracing data for your App Runner service

On the **Observability** tab of the [service dashboard page](#) in the App Runner console, choose **View service map** to navigate to the Amazon CloudWatch console.
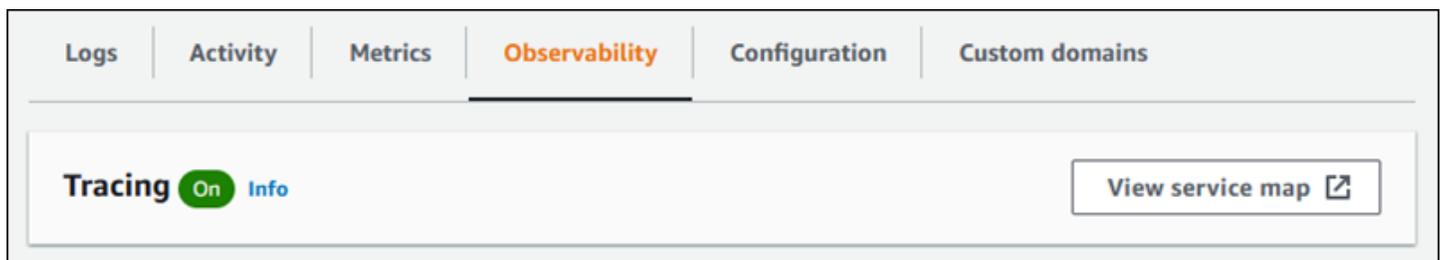
Use the Amazon CloudWatch console to view service maps and traces for requests that your application serves. Service maps show information like request latency and interactions with other applications and AWS services. The custom annotations that you add to your code allow you to easily search for traces. For more information, see [Using ServiceLens to monitor the health of your applications](#) in the *Amazon CloudWatch User Guide*.

Associating an AWS WAF web ACL with your service

AWS WAF is a web application firewall that you can use to secure your App Runner service. With AWS WAF web access control lists (web ACLs), you can guard your App Runner service endpoints against common web exploits and unwanted bots.

A web ACL provides you with fine-grained control over all incoming web requests to your App Runner service. You can define rules in a web ACL to allow, block, or monitor web traffic, to ensure that only authorized and legitimate requests reach your web applications and APIs. You can customize the web ACL rules based on your specific business and security needs. To learn more about infrastructure security and best practices for applying network ACLs, see Control network traffic in the *Amazon VPC User Guide*.

> ⚠️ **Important**
>
> Source IP rules for App Runner private services that are associated with WAF web ACLs *do not adhere to IP based rules*. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs.

Incoming web request flow

When an AWS WAF web ACL is associated with an App Runner service, incoming web requests go through the following process:

1. App Runner forwards the contents of the origin request to AWS WAF.

2. AWS WAF inspects the request and compares its contents to the rules that you specified in your web ACL.

3. Based on its inspection, AWS WAF returns an `allow` or `block` response to App Runner.

   - If an `allow` response is returned, App Runner forwards the request to your application.

   - If a `block` response is returned, App Runner blocks the request from reaching your web application. It forwards the `block` response from AWS WAF to your application.

> **ⓘ Note**
>
> By default App Runner blocks the request if no response is returned from AWS WAF.

For more information about AWS WAF web ACLs, see [Web access control lists (web ACLs)](#) in the *AWS WAF Developer Guide*.

> **ⓘ Note**
>
> You pay standard AWS WAF pricing. You don't incur any additional costs for using AWS WAF web ACLs for your App Runner services.
> For more information about pricing, see [AWS WAF Pricing](#).

Associating WAF web ACLs to your App Runner service

The following is the high-level process to associate an AWS WAF web ACL with your App Runner service:

1. Create a web ACL in the AWS WAF console. For more information, see [Creating a web ACL](#) in the *AWS WAF Developer Guide*.

2. Update your AWS Identity and Access Management (IAM) permissions for AWS WAF. For more information, see [Permissions](#).

3. Associate the web ACL with the App Runner service using one of the following methods:

   - **App Runner console**: Associate an existing web ACL using App Runner console when you [create](#) or [update](#) an App Runner service. For instructions, see [Managing AWS WAF web ACLs](#).

   - **AWS WAF console**: Associate the web ACL using the AWS WAF console for an existing App Runner service. For more information, see [Associating or disassociating a web ACL with an AWS resource](#) in the *AWS WAF Developer Guide*.

   - **AWS CLI**: Associate the web ACL using the AWS WAF public APIs. For more information about AWS WAF public APIs, see [AssociateWebACL](#) in the *AWS WAF API Reference Guide*.

Considerations

- Source IP rules for App Runner private services that are associated with WAF web ACLs *do not adhere to IP based rules*. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs.

- An App Runner service can be associated with only one web ACL. However, you can associate one web ACL with multiple App Runner services and with multiple AWS resources. Examples include Amazon Cognito user pools and Application Load Balancer resources.

- When you create a web ACL, a small amount of time passes before the web ACL fully propagates and is available to App Runner. The propagation time can be from a few seconds to a number of minutes. AWS WAF returns a `WAFUnavailableEntityException` when you try to associate a web ACL before it has fully propagated.

  If you refresh the browser or navigate away from the App Runner console before the web ACL is fully propagated, the association fails to occur. However, you can navigate within the App Runner console.

- AWS WAF returns a `WAFNonexistentItemException` error when you call one of the following AWS WAF APIs for an App Runner service which is in an invalid state:

  - `AssociateWebACL`

  - `DisassociateWebACL`

  - `GetWebACLForResource`

  The invalid states for your App Runner service include:

  - `CREATE_FAILED`

  - `DELETE_FAILED`

  - `DELETED`

  - `OPERATION_IN_PROGRESS`

  > ⓘ **Note**
  >
  > `OPERATION_IN_PROGRESS` state is invalid only if your App Runner service is being deleted.

- Your request might result in a payload that is larger than the limits of what AWS WAF can inspect. For more information about how AWS WAF handles oversize requests from App Runner, see [Oversize request component handling](#) in the *AWS WAF Developer Guide* to learn how AWS WAF handles oversize requests from App Runner.

- If you don't set appropriate rules or your traffic patterns change, a web ACL might not be as effective at securing your application.

Permissions

To work with a web ACL in AWS App Runner, add the following IAM permissions for AWS WAF:

- `apprunner:ListAssociatedServicesForWebAcl`

- `apprunner:DescribeWebAclForService`

- `apprunner:AssociateWebAcl`

- `apprunner:DisassociateWebAcl`

For more information about IAM permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.

The following is an example of the updated IAM policy for AWS WAF. This IAM policy includes the necessary permissions to work with an App Runner service.

Example

JSON

```
{
    "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**
>
> Though you must grant IAM permissions, the listed actions are permission-only and don't correspond to an API operation.

Managing AWS WAF web ACLs

Manage the AWS WAF web ACLs for your App Runner service by using one of the following methods:

- the section called "App Runner console"
- the section called "AWS CLI"

App Runner console

When you create a service or update an existing one on the App Runner console, you can associate or disassociate an AWS WAF web ACL.

> **ⓘ Note**
>
> - An App Runner service can be associated with only one web ACL. However, you can associate one web ACL with more than one App Runner service in addition to other AWS resources.
>
> - Before you associate a web ACL, make sure to update your IAM permissions for AWS WAF. For more information, see Permissions.

Associating AWS WAF web ACL

> ⚠️ **Important**
>
> Source IP rules for App Runner private services that are associated with WAF web ACLs *do not adhere to IP based rules*. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs.

To associate an AWS WAF web ACL

1. Open the App Runner console, and in the **Regions** list, select your AWS Region.

2. Based on whether you're creating or updating a service, perform one of the following steps:

   - If you're creating a new service, choose **Create an App Runner service** and go to **Configure Service**.

   - If you're updating an existing service, choose the **Configuration** tab, and then choose **Edit** under **Configure service**.

3. Go to **Web application firewall** under **Security**.

4. Choose the **Activate** toggle button to view the options.

5.   Perform one of the following steps:

   - **To associate an existing web ACL**: Choose the required web ACL from the **Choose a web ACL** table to associate with your App Runner service.

- **To create a new web ACL**: Choose **Create web ACL** to create a new web ACL using the AWS WAF console. For more information, see [Creating a web ACL](#) in the *AWS WAF Developer Guide*.

   1. Choose the refresh button to view the newly created web ACL in the **Choose a web ACL** table.

   2. Select the required web ACL.

6. Choose **Next** if you're creating a new service or **Save changes** if you're updating an existing service. The selected web ACL is associated with your App Runner service.

7. To verify the web ACL association, choose the **Configuration** tab of your service and go to **Configure service**. Scroll to **Web application firewall** under **Security** to view the details of the web ACL associated with your service.

> ⓘ **Note**
>
> When you create a web ACL, a small amount of time passes before the web ACL fully propagates and is available to App Runner. The propagation time can be from a few seconds to a number of minutes. AWS WAF returns a `WAFUnavailableEntityException` when you try to associate a web ACL before it has fully propagated.
> If you refresh the browser or navigate away from the App Runner console before the web ACL is fully propagated, the association fails to occur. However, you can navigate within the App Runner console.

Disassociating an AWS WAF web ACL

You can disassociate AWS WAF web ACl that you no longer need by [updating](#) your App Runner service.

To disassociate an AWS WAF web ACl

1. Open the [App Runner console](#), and in the **Regions** list, select your AWS Region.

2. Go to **Configuration** tab of the service you want to update and choose **Edit** under **Configure service**.

3. Go to **Web application firewall** under **Security**.

4. Disable the **Activate** toggle button. You receive a message to confirm the deletion.

5.   Choose **Confirm**. The web ACL is disassociated from your App Runner service.

> ⓘ **Note**
>
> - If you want to associate your service with another web ACL, select a web ACL from the **Choose a web ACL** table. App Runner disassociates the current web ACL and starts the process to associate with the selected web ACL.
>
> - If no other App Runner services or resources use a disassociated web ACL, consider deleting the web ACL. Otherwise, you will continue to incur costs. For more information about pricing, see [AWS WAF Pricing](). For instruction on how to delete a web ACL, see [DeleteWebACL]() in the *AWS WAF API Reference*.
>
> - You can't delete a web ACL that's associated with other active App Runner services or other resources.

AWS CLI

You can associate or disassociate an AWS WAF web ACL by using the AWS WAF public APIs. The App Runner service, with which you want to associate or disassociate a web ACL, must be in a valid state.

AWS WAF returns a `WAFNonexistentItemException` error when you call one of the following AWS WAF APIs for an App Runner service which is in an invalid state:

- `AssociateWebACL`

- `DisassociateWebACL`

- `GetWebACLForResource`

The invalid states for your App Runner service include:

- `CREATE_FAILED`

- `DELETE_FAILED`

- `DELETED`

- `OPERATION_IN_PROGRESS`

> **ⓘ Note**
>
> OPERATION_IN_PROGRESS state is invalid only if your App Runner service is being deleted.

For more information about AWS WAF public APIs, see *AWS WAF API Reference Guide*.

> **ⓘ Note**
>
> Update your IAM permissions for AWS WAF. For more information, see Permissions.

Associating AWS WAF web ACL using AWS CLI

> **⚠ Important**
>
> Source IP rules for App Runner private services that are associated with WAF web ACLs *do not adhere to IP based rules*. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs.

To associate an AWS WAF web ACL

1. Create an AWS WAF web ACL for your service with your preferred set of rule actions to `Allow` or `Block` the web requests to your service. For more information about AWS WAF APIs, see CreateWebACL in the *AWS WAF API Reference Guide*.

   Example Create a web ACL - Request

   ```
   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. Associate the web ACL that you created with the App Runner service using the `associate-web-acl` AWS WAF public API. For more information about AWS WAF APIs, see [AssociateWebACL](#) in the *AWS WAF API Reference Guide*.

    > **ⓘ Note**
    >
    > When you create a web ACL, a small amount of time passes before the web ACL fully propagates and is available to App Runner. The propagation time can be from a few seconds to a number of minutes. AWS WAF returns a `WAFUnavailableEntityException` when you try to associate a web ACL before it has fully propagated.
    > If you refresh the browser or navigate away from the App Runner console before the web ACL is fully propagated, the association fails to occur. However, you can navigate within the App Runner console.

    Example Associating a web ACL - Request

    ```
    aws wafv2 associate-web-acl
    --resource-arn <apprunner_service_arn>
    --web-acl-arn <web_acl_arn>
    --region <region>
    ```

3. Verify that the web ACL is associated with your App Runner service using the `get-web-acl-for-resource` AWS WAF public API. For more information about AWS WAF APIs, see [GetWebACLForResource](#) in the *AWS WAF API Reference Guide*.

    Example Verify web ACL for resource - Request

    ```
    aws wafv2 get-web-acl-for-resource
    --resource-arn <apprunner_service_arn>
    --region <region>
    ```

    If there are no web ACLs associated with your service, you receive a blank response.

Deleting an AWS WAF web ACL using AWS CLI

You can't delete an AWS WAF web ACL if it's associated with an App Runner service.

To delete an AWS WAF web ACL

1. Disassociate the web ACL from your App Runner service by using the `disassociate-web-acl` AWS WAF public API. For more information about AWS WAF APIs, see [DisassociateWebACL](#) in the *AWS WAF API Reference Guide*.

   Example Disassociating a web ACL - Request

   ```
   aws wafv2 disassociate-web-acl
   --resource-arn <apprunner_service_arn>
   --region <region>
   ```

2. Verify that the web ACL is disassociated from your App Runner service using the `get-web-acl-for-resource` AWS WAF public API.

   Example Verify that the web ACL is disassociated - Request

   ```
   aws wafv2 get-web-acl-for-resource
   --resource-arn <apprunner_service_arn>
   --region <region>
   ```

   The disassociated web ACL isn't listed for your App Runner service. If there are no web ACLs associated with your service, you receive a blank response.

3. Delete the disassociated web ACL using the `delete-web-acl` AWS WAF public API. For more information about AWS WAF APIs, see [DeleteWebACL](#) in the *AWS WAF API Reference Guide*.

   Example Delete a web ACL - Request

   ```
   aws wafv2 delete-web-acl
   --name <web_acl_name>
   --scope REGIONAL
   --id <web_acl_id>
   --lock-token <web_acl_lock_token>
   --region <region>
   ```

4. Verify that the web ACL is deleted using the `list-web-acl` AWS WAF public API. For more information about AWS WAF APIs, see [ListWebACLs](#) in the *AWS WAF API Reference Guide*.

Example Verify that the web ACL is deleted - Request

```
aws wafv2 list-web-acls
--scope REGIONAL
--region <region>
```

The deleted web ACL is no longer be listed.

> ### ⓘ Note
>
> If a web ACL is associated with other active App Runner services or other resources, such as Amazon Cognito user pools, the web ACL can't be deleted.

Listing App Runner services that are associated with a web ACL

A web ACL can be associated with multiple App Runner services and other resources. List the App Runner services associated with a web ACL using the `list-resources-for-web-acl` AWS WAF public API. For more information about AWS WAF APIs, see [ListResourcesForWebACL](#) in the *AWS WAF API Reference Guide*.

Example List App Runner services associated with a web ACL - Request

```
aws wafv2 list-resources-for-web-acl
--web-acl-arn <WEB_ACL_ARN>
--resource-type APP_RUNNER_SERVICE
--region <REGION>
```

Example List App Runner services associated with a web ACL - Response

The following example illustrates the response when there are no App Runner services that are associated with a web ACL.

```
{
  "ResourceArns": []
}
```

Example List App Runner services associated with a web ACL - Response

The following example illustrates the response when there are App Runner services that are associated with a web ACL.

```
{
  "ResourceArns": [
    "arn:aws:apprunner:<region>:<aws_account_id>:service/<service_name>/<service_id>"
  ]
}
```

Testing and logging AWS WAF web ACLs

When you set a rule action to **Count** in your web ACL, AWS WAF adds the request to a count of requests that match the rule. To test a web ACL with your App Runner service, set rule actions to **Count** and consider the volume of requests that match each rule. For example, you set a rule for the `Block` action that matches a large number of requests that you determine to be normal user traffic. In that case, you might need to reconfigure your rule. For more information, see Testing and tuning your AWS WAF protections in the *AWS WAF Developer Guide*.

You can also configure AWS WAF to log request headers to an Amazon CloudWatch Logs log group, an Amazon Simple Storage Service (Amazon S3) bucket, or an Amazon Data Firehose. For more information, see Logging web ACL traffic in the *AWS WAF Developer Guide*.

To access logs related to the web ACL that's associated with your App Runner service, refer to the following log fields:

- `httpSourceName: Contains APPRUNNER`
- `httpSourceId: Contains customeraccountid-apprunnerserviceid`

For more information, see Log Examples in the *AWS WAF Developer Guide*.

> ⚠️ **Important**
>
> Source IP rules for App Runner private services that are associated with WAF web ACLs *do not adhere to IP based rules*. This is because we currently don't support forwarding request source IP data to App Runner private services associated with WAF. If your App Runner application requires source IP/CIDR incoming traffic control rules, you must use security group rules for private endpoints instead of WAF web ACLs.

Setting App Runner service options using a configuration file

> **ⓘ Note**
>
> Configuration files are applicable only to services that are based on source code. You can't use configuration files with image-based services.

When you create an AWS App Runner service using a source code repository, AWS App Runner requires information about building and starting your service. You can provide this information each time you create a service using the App Runner console or API. Alternatively, you can set service options by using a *configuration file.* The options that you specify in a file become part of your source repository, and any changes to these options are tracked similarly to how changes to the source code are tracked. You can use the App Runner configuration file to specify more options than the API supports. You don't need to provide a configuration file if you only need the basic options that the API supports.

The App Runner configuration file is a YAML file that's named `apprunner.yaml` in the source directory of your application's repository. It provides build and runtime options for your service. Values in this file instruct App Runner how to build and start your service, and provide runtime context such as network settings and environment variables.

The App Runner configuration file doesn't include operational settings, such as CPU and memory.

For examples of App Runner configuration files, see the section called "Examples". For a complete reference guide, see the section called "Reference".

Topics

- App Runner configuration file examples
- App Runner configuration file reference

App Runner configuration file examples

> ℹ️ **Note**
>
> Configuration files are applicable only to [services that are based on source code](#). You can't use configuration files with [image-based services](#).

The following examples demonstrate AWS App Runner configuration files. Some are minimal and contain only required settings. Others are complete, including all configuration file sections. For an overview of App Runner configuration files, see *App Runner configuration file*.

Configuration file examples

Minimal configuration file

With a minimal configuration file, App Runner makes the following assumptions:

- No custom environment variables are necessary during build or run.

- The latest runtime version is used.

- The default port number and port environment variable are used.

Example apprunner.yaml

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

Complete configuration file

This example shows the use of all configuration keys in the `apprunner.yaml` original format with a managed runtime.

Example apprunner.yaml

```
version: 1.0
runtime: python3
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-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"
```

Complete configuration file — (uses revised build)

This example shows the use of all configuration keys in the `apprunner.yaml` with a managed runtime.

The `pre-run` parameter is only supported by the revised App Runner build. Do not insert this parameter in your configuration file if your application uses runtime versions that are supported by the original App Runner build. For more information, see Managed runtime versions and the App Runner build.

> **ⓘ Note**
>
> Since this examples is for Python 3.11, we use the `pip3` and `python3` commands. For more information, see Callouts for specific runtime versions in the Python platform topic.

Example apprunner.yaml

```yaml
version: 1.0
runtime: python311
build:
  commands:
    pre-build:
      - wget -c https://s3.amazonaws.com/amzn-s3-demo-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"
```

For examples of specific managed runtime configuration files, see the specific runtime subtopic under Code-based service.

App Runner configuration file reference

> **ⓘ Note**
>
> Configuration files are applicable only to services that are based on source code. You can't use configuration files with image-based services.

This topic is a comprehensive reference guide to the syntax and semantics of an AWS App Runner configuration file. For an overview of App Runner configuration files, see *App Runner configuration file*.

The App Runner configuration file is a YAML file. Name it `apprunner.yaml`, and place it in the source directory of your application's repository.

Structure overview

The App Runner configuration file is a YAML file. Name it `apprunner.yaml`, and place it in the source directory of your application's repository.

The App Runner configuration file contains these main parts:

- *Top section* – Contains top-level keys
- *Build section* – Configures the build stage
- *Run section* – Configures the runtime stage

Top section

The keys at the top of the file provide general information about the file and your service runtime. The following keys are available:

- `version` – *Required.* The App Runner configuration file version. Ideally, use the latest version.

  Syntax

  ```
  version: version
  ```

  Example

  ```
  version: 1.0
  ```

- `runtime` – *Required.* The name of the runtime that your application uses. To learn about available runtimes for the different programming platforms that App Runner offers, see *Code-based service*.

  > **ⓘ Note**
  >
  > The naming convention of a managed runtime is `<language-name><major-version>`.

  Syntax

  ```
  runtime: runtime-name
  ```

  Example

  ```
  runtime: python3
  ```

Build section

The build section configures the build stage of the App Runner service deployment. You can specify build commands and environment variables. Build commands are required.

The section starts with the `build:` key, and has the following subkeys:

- commands – *Required.* Specifies the commands that App Runner runs during various build phases. Includes the following subkeys:

  - `pre-build` – *Optional.* The commands that App Runner runs before the build. For example, install **npm** dependencies or test libraries.

  - `build` – *Required.* The commands that App Runner runs to build your application. For example, use **pipenv**.

  - `post-build` – *Optional.* The commands that App Runner runs after the build. For example, use Maven to package build artifacts into a JAR or WAR file, or run a test.

Syntax

```
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 – *Optional.* Specifies custom environment variables for the build stage. Defined as name-value scalar mappings. You can refer to these variables by name in your build commands.

> **ⓘ Note**
>
> There are two distinct `env` entries in two different locations in this configuration file. One set is in the **Build** section and the other in the **Run** section.
>
> - The `env` set in the Build section can be referenced by the `pre-build`, `build`, `post-build`, and `pre-run` commands during the *build process*.
>
>   *Important* - Note that the `pre-run` commands are located in the Run section of this file, even though they can only access the environment variables that are defined in the Build section.
>
> - The `env` set in the Run section can be referenced by the `run` command in the runtime environment.

Syntax

```
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"
```

Run section

The run section configures the container running stage of the App Runner application deployment. You can specify runtime version, pre-run commands (revised format only), start command, network port, and environment variables.

The section starts with the `run:` key, and has the following subkeys:

- `runtime-version` – *Optional.* Specifies a runtime version that you want to lock for your App Runner service.

  By default, only the major version is locked. App Runner uses the latest minor and patch versions that are available for the runtime on every deployment or service update. If you specify major and minor versions, both become locked, and App Runner updates only patch versions. If you specify major, minor, and patch versions, your service is locked on a specific runtime version and App Runner never updates it.

  Syntax

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

  > ⓘ **Note**
  >
  > The runtimes of some platforms have different version components. See specific platform topics for details.

  Example

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

- `pre-run` – *Optional.* ***Revised build*** *usage only*. Specifies the commands that App Runner runs after copying your application from the build image to the run image. You can enter commands here to the modify the run image outside the `/app` directory. For example, if you need to install additional global dependencies that reside outside of the `/app` directory, enter the required

commands in this sub-section to do so. For more information about the App Runner build process, see Managed runtime versions and the App Runner build.

> **ⓘ Note**
>
> - *Important* – Even though the `pre-run` commands are listed in the Run section, they can only reference the environment variables defined in the Build section of this configuration file. They cannot reference the environment variables defined in this Run section.
>
> - The `pre-run` parameter is only supported by the revised App Runner build. Do not insert this parameter in your configuration file if your application uses runtime versions that are supported by the original App Runner build. For more information, see Managed runtime versions and the App Runner build.

Syntax

```
run:
  pre-run:
      - command
      - …
```

- command – *Required.* The command that App Runner uses to run your application after it completes the application build.

Syntax

```
run:
  command: command
```

- `network` – *Optional.* Specifies the port that your application listens to. It includes the following:

  - `port` – *Optional.* If specified, this is the port number that your application listens to. The default is `8080`.

  - `env` – *Optional.* If specified, App Runner passes the port number to the container in this environment variable, in addition to (not instead of) passing the same port number in the default environment variable, `PORT`. In other words, if you specify `env`, App Runner passes the port number in two environment variables.

Syntax

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

Example

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

- env – *Optional.* Definition of custom environment variables for the run stage. Defined as name-value scalar mappings. You can refer to these variables by name in your runtime environment.

> ⓘ **Note**
>
> There are two distinct env entries in two different locations in this configuration file. One set is in the **Build** section and the other in the **Run** section.
>
> - The env set in the Build section can be referenced by the pre-build, build, post-build, and pre-run commands during the *build process*.
>
>   *Important* - Note that the pre-run commands are located in the Run section of this file, even though they can only access the environment variables that are defined in the Build section.
>
> - The env set in the Run section can be referenced by the run command in the runtime environment.

Syntax

```
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"
```

The App Runner API

The AWS App Runner application programming interface (API) is a RESTful API for making requests to the App Runner service. You can use the API to create, list, describe, update, and delete App Runner resources in your AWS account.

You can call the API directly in your application code, or you can use one of the AWS SDKs.

For complete API reference information, see the AWS App Runner API Reference.

For more information about AWS developer tools, see Tools to Build on AWS.

Topics

- Using the AWS CLI to work with App Runner
- Using AWS CloudShell to work with AWS App Runner

Using the AWS CLI to work with App Runner

For command line scripts, use the AWS CLI to make calls to the App Runner service. For complete AWS CLI reference information, see the apprunner in the *AWS CLI Command Reference*.

AWS CloudShell allows you to skip installing the AWS CLI in your development environment, and use it in the AWS Management Console instead. In addition to avoiding installation, you also don't need to configure credentials, and you don't need to specify region. Your AWS Management Console session provides this context to the AWS CLI. For more information about CloudShell, and for a usage example, see the section called "Using AWS CloudShell".

Using AWS CloudShell to work with AWS App Runner

AWS CloudShell is a browser-based, pre-authenticated shell that you can launch directly from the AWS Management Console. You can run AWS CLI commands against AWS services (including AWS App Runner) using your preferred shell (Bash, PowerShell or Z shell). And you can do this without needing to download or install command line tools.

You launch AWS CloudShell from the AWS Management Console, and the AWS credentials you used to sign in to the console are automatically available in a new shell session. This pre-authentication of AWS CloudShell users allows you to skip configuring credentials when interacting

with AWS services such as App Runner using AWS CLI version 2 (pre-installed on the shell's compute environment).

Topics

- Obtaining IAM permissions for AWS CloudShell
- Interacting with App Runner using AWS CloudShell
- Verifying your App Runner service using AWS CloudShell

Obtaining IAM permissions for AWS CloudShell

Using the access management resources provided by AWS Identity and Access Management, administrators can grant permissions to IAM users so they can access AWS CloudShell and use the environment's features.

The quickest way for an administrator to grant access to users is through an AWS managed policy. An AWS managed policy is a standalone policy that's created and administered by AWS. The following AWS managed policy for CloudShell can be attached to IAM identities:

- `AWSCloudShellFullAccess`: Grants permission to use AWS CloudShell with full access to all features.

If you want to limit the scope of actions that an IAM user can perform with AWS CloudShell, you can create a custom policy that uses the `AWSCloudShellFullAccess` managed policy as a template. For more information about limiting the actions that are available to users in CloudShell, see Managing AWS CloudShell access and usage with IAM policies in the *AWS CloudShell User Guide*.

> ⓘ **Note**
>
> Your IAM identity also requires a policy that grants permission to make calls to App Runner. For more information, see the section called "App Runner and IAM".

Interacting with App Runner using AWS CloudShell

After you launch AWS CloudShell from the AWS Management Console, you can immediately start to interact with App Runner using the command line interface.

In the following example, you retrieve information about one of your App Runner services using the AWS CLI in CloudShell.

> ⓘ **Note**
>
> When using AWS CLI in AWS CloudShell, you don't need to download or install any additional resources. Moreover, because you're already authenticated within the shell, you don't need to configure credentials before making calls.

Example Retrieving App Runner service information using AWS CloudShell

1. From the AWS Management Console, you can launch CloudShell by choosing the following options available on the navigation bar:

   - Choose the CloudShell icon.

   - Start typing **cloudshell** in the search box, and then choose the **CloudShell** option when you see it in the search results.

2. To list all current App Runner services in your AWS account in the console session's AWS Region, enter the following command in the CloudShell command line:

```
$ aws apprunner list-services
```

The output lists summary information for your services.

```
{
  "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. To get a detailed description of a particular App Runner service, enter the following command in the CloudShell command line, using one of the ARNs retrieved in the previous step:

```
$ aws apprunner describe-service --service-arn arn:aws:apprunner:us-
east-2:123456789012:service/my-app-1/8fe1e10304f84fd2b0df550fe98a71fa
```

The output lists a detailed description of the service you specified.

```
{
  "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
  }
 }
}
```

Verifying your App Runner service using AWS CloudShell

When you create an App Runner service, App Runner creates a default domain for your service's website, and shows it in the console (or returns it in the API call result). You can use CloudShell to make calls to your website and verify that it's working correctly.

For example, after you create an App Runner service as described in *Getting started*, run the following command in CloudShell:

```
$ curl https://qxuadi4qwp.us-east-2.awsapprunner.com/; echo
```

The output should show the expected page content.

Troubleshooting

This chapter provides troubleshooting steps for common errors and issues that you might encounter while using your AWS App Runner service. The error messages can appear on the console, API, or the Logs tab of your service page.

For more troubleshooting advice and answers to common support questions, visit the [Knowledge Center](#).

Topics

- [When the service fails to create](#)
- [Custom domain names](#)
- [HTTP/HTTPS request routing error](#)
- [When the service fails to connect to Amazon RDS or downstream service](#)
- [When there are not enough IP addresses for launching instances or scaling](#)

When the service fails to create

If your attempt to create an App Runner service fails, the service enters a `CREATE_FAILED` status. This status appears as **Create failed** on the console. A service might fail to create because of issues that are related to one or more of the following:

- Your application code
- The build process
- Configuration
- Resource quotas
- Temporary issues with the underlying AWS services that your service uses

To troubleshoot a service failing to create, we recommend that you do the following.

1. Read the service events and logs to find out what caused the service to fail to create.

2. Make any necessary changes to your code or configuration.

3. If you reached your service quota, delete one or more services.

4. If you reached another resource quota, you might be able to increase it if it's adjustable.

5. Try rebuilding the service again after completing all of the above steps. For information on how to rebuild your service, see the section called "Rebuild failed service".

> ⓘ **Note**
>
> One of the adjustable resource quotas that might be causing an issue is the *Fargate On-Demand* vCPU resource.
>
> The vCPU resource count determines the number of instances that App Runner can provide to your service. This is an adjustable quota value for the *Fargate On-Demand* vCPU resource count that resides in the AWS Fargate service. To view the vCPU quota settings for your account or to request a quota increase, use the Service Quotas console in the AWS Management Console. For more information, see AWS Fargate service quotas in the *Amazon Elastic Container Service Developer Guide*.

> ⚠ **Important**
>
> You don't incur any additional charges beyond the initial creation attempt for a failed service. Even though the failed service isn't usable, it still counts towards your service quota. App Runner doesn't automatically delete the failed service, so make sure that you delete it when you're done analyzing the failure.

Custom domain names

This section covers how you can troubleshoot and resolve various errors that you might run into while linking to a custom domain.

> ⓘ **Note**
>
> To augment the security of your App Runner applications, the **.awsapprunner.com* domain is registered in the Public Suffix List (PSL). For further security, we recommend that you use cookies with a `__Host-` prefix if you ever need to set sensitive cookies in the default domain name for your App Runner applications. This practice will help to defend your domain against cross-site request forgery attempts (CSRF). For more information see the Set-Cookie page in the Mozilla Developer Network.

Getting Create Fail error for custom domain

- Check if this error is because of an issue with the CAA records. If there are no CAA records anywhere in the DNS tree, you receive a message `fail open`, and AWS Certificate Manager issues a certificate to verify the custom domain. This allows App Runner to accept the custom domain. If you're using CAA certifications in the DNS records, make sure that at least one domain's CAA records include `amazon.com`. Otherwise, ACM fails to issue a certificate. As a result, the custom domain for App Runner fails to be created.

  The following example uses the DNS lookup tool *DiG* to show CAA records missing a required entry. The example uses `example.com` as the custom domain. Run the following commands in the example to check the CAA records.

  ```
  ...
  ;; 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...
  ```

- Correct the domain records and ensure that at least one CAA record includes `amazon.com`.
- Retry to link the custom domain with App Runner.

For instructions on how to resolve CAA errors, see the following:

- [Certification Authority Authorization (CAA) problems](#)
- [How do I resolve CAA errors for issuing or renewing an ACM certificate?](#)

Getting DNS certificate validation pending error for custom domain

- Check if you skipped an important step in the custom domain setup. Additionally check if you incorrectly configured a DNS record using a DNS lookup tool such as *DiG*. In particular, check for the following mistakes:
  - Any missed steps.

- Unsupported characters such as double quotations in the DNS records.

- Correct the mistakes.

- Retry to link the custom domain with App Runner.

For instructions on how to resolve CAA validation errors, see the following.

- [DNS Validation](#)

- [the section called "Custom domain names"](#)

Basic troubleshooting commands

- Confirm that a service can be found.

```
aws apprunner list-services
```

- Describe a service and check its status.

```
aws apprunner describe-service --service-arn
```

- Check status of custom domain.

```
aws apprunner describe-custom-domains --service-arn
```

- List all operations in progress.

```
aws apprunner list-operations --service-arn
```

Custom domain certificate renewal

When you add a custom domain to your service, App Runner provides you with a set of CNAME records that you add to your DNS server. These CNAME records include certificate records. App Runner uses AWS Certificate Manager (ACM) to verify the domain. App Runner validates these DNS records to ensure continued ownership of this domain. If you remove the CNAME records from your DNS zone, App Runner can no longer validate the DNS records, and the custom domain certificate fails to renew automatically.

This section covers how to resolve the following custom domain certificate renewal issues:

- the section called "The CNAME is removed from the DNS server".

- the section called "The certificate has expired".

The CNAME is removed from the DNS server

- Retrieve your CNAME records using the DescribeCustomDomains API or from the **Custom Domain** settings in the App Runner console. For information about stored CNAMEs, see CertificateValidationRecords.

- Add the certificate validation CNAME records to your DNS server. App Runner can then validate that you own the domain. After you add the CNAME records, it can take up to 30 minutes for the DNS records to be propagated. It can also take several hours for App Runner and ACM to retry the certificate renewal process. For instructions on how to add CNAME records, see the section called "Manage custom domains".

The certificate has expired

- Disassociate (unlink) and then associate (link) the custom domain for your App Runner service using the App Runner console or API. App Runner creates a new certificate validation CNAME records.

- Add the new certificate validation CNAME records to your DNS server.

  For instructions on how to disassociate (unlink) and associate (link) the custom domain, see the section called "Manage custom domains".

How do I verify that the certificate was successfully renewed

You can check the status of your certificate records to verify your certificate was successfully renewed. You can check the status of the certificates by using tools like curl.

For more information about certificate renewal, see the following links:

- [Why is my ACM certificate marked as ineligible for renewal?](#)

- [Managed renewal for ACM certificates](#)

- [DNS validation](#)

HTTP/HTTPS request routing error

This section covers how you can troubleshoot and resolve errors that you might run into when routing HTTP/HTTPS traffic to your App Runner service endpoints.

404 Not found error when sending HTTP/HTTPS traffic to App Runner service endpoints

- Verify that the `Host Header` is pointing to the service URL in the HTTP request as App Runner uses the host header information to route requests. Most clients, like cURL, and web browsers automatically point the host header to the service URL. If your client doesn't set the service URL as the `Host Header`, you receive a `404 Not Found` error.

  Example Incorrect host header

  ```
  $ ~ curl -I -H "host: foobar.com" https://testservice.awsapprunner.com/
  HTTP/1.1 404 Not Found
  transfer-encoding: chunked
  ```

  Example Correct host header

  ```
  $ ~ 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
  ```

- Verify that your client is correctly setting the server name indicator (SNI) for requests routing to public or private services. For TLS termination and request routing, App Runner uses the SNI set in HTTPS connection.

When the service fails to connect to Amazon RDS or downstream service

There may be a network configuration issue with your service if it fails to connect to an Amazon RDS database or other downstream application or service. This topic walks you through some steps to determine if there are any issues with your network configuration and the options to correct them. To learn more about outbound traffic configuration for App Runner, see Enabling VPC access for outgoing traffic.

> **ⓘ Note**
>
> To view your VPC Connector configuration, from the App Runner console left navigation pane, select **Network configuration**. Then select the **Outgoing traffic** tab. Select a VPC Connector. The next page displays details about the VPC Connector. From this page you can view and drill down into the following: **Subnets**, **Security groups**, and **App Runner services** that use the VPC.

To narrow down the cause of your application's inability to connect to another downstream service

1. Ensure that the subnets used in the VPC Connectors are private subnets. If a connector is configured with a public subnet your service will encounter errors, because the underlying Hyperplane ENIs (elastic network interfaces) for each subnet don't have a public IP space.

   If your VPC connectors are using public subnets, you have the following options to correct this configuration:

   a. Create a new private subnet, and use it instead of the public subnet for the VPC Connector. For more information, see Subnets for your VPC in the Amazon VPC User Guide.

   b. Route the existing public subnet via NAT gateways. For more information see NAT gateways in the Amazon VPC User Guide.

2. Verify that the security group ingress and egress rules for the VPC Connector are correct. From the App Runner console left navigation pane, select **Network configuration** > **Outgoing traffic**. Select the VPC Connector from the list. The next page lists the **Security groups** that you can select to inspect.

3. Verify that the security group inbound and outbound rules are correct for the RDS instance or other downstream service that you're attempting connection to. For more information, see the service guide for the downstream service to which your App Runner application is trying to connect.

4. To confirm that there isn't some other type of network setup issue outside of your App Runner configurations, try connecting to the RDS or downstream service outside of App Runner:

    a. From an Amazon EC2 instance in the same VPC, try connecting to the RDS instance or service.

    b. If you're trying to connect to a service VPC endpoint, verify connectivity by accessing the same endpoint from an EC2 instance in the same VPC.

5. If either of the connection tests in *Step 4* fail, more than likely there's an issue outside of your App Runner configurations with another resource in your AWSaccount. Contact AWS Support for assistance to further isolate and fix the issue with your other network configurations.

6. If you successfully connect to the RDS instance or downstream service by doing the instructions in *Step 4*, then proceed with the instructions in this step. We'll check if traffic is entering the ENI by enabling and inspecting the Hyperplane ENI flow logs.

> ⓘ **Note**
>
> To be able to complete these steps and obtain the required ENI flow log information, the connection attempt to the RDS or downstream service must occur after your App Runner service has started up successfully. Your application must perform the connect operation to the RDS or downstream service when it's in a Running state. Otherwise, the ENIs could be cleaned up as part of App Runner's rollback workflows. This approach ensures that the ENIs remain available for further investigation.

    a. From the AWS console, launch the EC2 console.

    b. From the left navigation pane, in the **Network & Security** grouping, select **Network Interfaces**.

c. Scroll over to the **Interface Type** and **Description** columns to locate the ENIs in the subnets associated with the VPC Connector. They will have the following naming patterns.

- **Interface Type:** *fargate*

- **Description:** begins with *AWSAppRunner ENI* (example: *AWSAppRunner ENI - abcde123-abcd-1234-1234-abcde1233456*)

d. Use the check boxes at the beginning of the rows to select the ENIs that apply.

e. From the **Actions** menu select **Create flow log**.

f. Enter the information in the prompts and select **Create flow flog** at the bottom of the page.

g. Inspect the generated flow log.

- If traffic was entering the ENI when you were testing the connection, then the issue is not related to the ENI setup. There may be network configuration issues with another resource in your AWS Account besides App Runner services. Contact AWS Support for further assistance.

- If traffic was not entering the ENI when you were testing the connection, we advise that you contact AWS Support to see if there are any known issues with the Fargate service.

h. Use the network *Reachability Analyzer* tool. This tool helps determine network misconfigurations by identifying blocking components when a source in the virtual network path isn't reachable. For more information, see [What is Reachability Analyzer?](#) in the *Amazon VPC Reachability Analyzer Guide*.

Enter the App Runner ENI as the source, and the RDS ENI as the destination.

7. If you're unable to narrow down the issue further, or if you're still unable to connect to the RDS or downstream service after completing the prior steps, we advise that you contact AWS Support for further assistance.

When there are not enough IP addresses for launching instances or scaling

> 🛈 **Note**
>
> For public services, App Runner does not create an Elastic Network Interface (ENI) in your VPCs, so your public services are not affected by this change.

This guide helps you resolve IP exhaustion errors you may encounter on App Runner services with VPC access for outgoing traffic enabled.

App Runner will launch instances in the subnets associated with your VPC connector. App Runner creates 1 ENI per instance in the subnet where your instance is launched. Each ENI uses a private IP in that subnet. Subnets have fixed number of IPs available, depending on the CIDR block associated with that subnet. If App Runner is unable to find subnet(s) with sufficient IPs to create an ENI, it will fail to launch new instances for your App Runner service. This may lead to issues with scaling up your services. In such cases you will see App Runner event logs indicating that App Runner is unable to find subnets with available IPs. You can update your services with instructions below to resolve such errors.

How to update your services to have more available IPs

Number of IP addresses available in a subnet is based on the CIDR block associated with that subnet. CIDR blocks associated with a subnet cannot be updated after creation. App Runner VPC connectors can also not be updated once they are created. To provide more IPs to your App Runner services with VPC access for outgoing traffic enabled :

1. Create new subnet(s) with a larger CIDR block.
2. Create a new VPC connector with the new subnet(s).
3. Update your App Runner service to use the new VPC connector.

Calculating IPs needed for your services

Before attempting to create new subnet(s) with larger CIDR blocks, determine the number of IPs you will need across your App Runner services. We recommend calculating number of IPs needed in your connector as follows :

1. For each services with VPC access for outgoing traffic enabled, note the max size (maximum instances) in the auto scaling configuration.

2. Sum the values across all services.

3. Double this sum to account for the new instances launched during blue-green deployments.

Example

Consider two services A and B using the same VPC connector.

1. Service A has the max size configured as 25.

2. Service B has max size configured as 15.

Required IPs = 2 × (25 + 15) = 80

Ensure your subnets have at least 80 available IPs combined.

Create new subnet(s)

1. Determine the CIDR block size needed for IPv4 using this formula (Note that 5 IPs are reserved by AWS: Subnet Sizing)

```
Number of available IP addresses = 2^(32 - prefix length) - 5
```

```
Example :
For 192.168.1.0/24:
Prefix length is 24
Number of available IP addresses = 2^(32 - 24) - 5 = 2^8-5 = 251 IP addresses

For 10.0.0.0/16:
Prefix length is 16
Number of available IP addresses = 2^(32 - 16) - 5 = 2^16-5 = 65,531 IP addresses

Quick reference:
/24 = 251 IP addresses
/16 = 65,531 IP addresses
```

2. Create a new subnet by using the AWS EC2 CLI.

```
aws ec2 create-subnet --vpc-id <my-vpc-id> --cidr-block <cidr-block>
```

Example (creates a subnet with 4,096 IPs) :

```
aws ec2 create-subnet --vpc-id my-vpc-id --cidr-block 10.0.0.0/20
```

3. Create a new VPC connector. See : [Manage VPC Access](#)

4. Update your services with outgoing traffic to VPC enabled to use this new VPC connector.App Runner will start using the new subnets once your service is updated.

> **ⓘ Note**
>
> VPCs are also limited with number of available IPs that can be allocated to the subnets by CIDR blocks. If you are unable to create subnets with larger CIDR blocks, you might need to update your VPC with secondary CIDR blocks before creating the new subnet(s).

Attaching Secondary CIDR blocks to your VPC

Associate secondary CIDR block to this VPC.

```
aws ec2 associate-vpc-cidr-block --vpc-id <my-vpc-id> --cidr-block <cidr-block>
```

Example :

```
aws ec2 associate-vpc-cidr-block --vpc-id my-vpc-id --cidr-block 10.1.0.0/16
```

Verification

Once you have updated your service. You can use the following to perform verification of your fix

1. **Monitor event logs :** Monitor your App Runner service event [logs](#) to validate no new IP or ENI unavailability errors show up

2. **Check Service Scaling:**

    1. Fully scale up service by changing the min instance count in your autoscaling configuration

2. Verify that all new instances are launched without any IP-related errors

3. Monitor through several scaling events to ensure consistent performance

3. **Console Banner:** If you're using the AWS Management Console, confirm that App Runner no longer displays a banner warning about insufficient IPs.

4. **VPC and Subnet IP Utilization:**

   1. Use the VPC Dashboard or CLI commands to check IP address utilization in your new subnets.

   2. Confirm that there's still a healthy margin of available IPs after your service has scaled up

Common Pitfalls

When addressing IP exhaustion in App Runner services, be aware of these potential issues:

1. **Inadequate IP Address Planning:** Underestimating future IP needs can lead to recurring exhaustion issues. Conduct thorough capacity planning, considering potential service growth and peak usage scenarios.

2. **Overlooking VPC-wide IP Usage:** Remember that other AWS services within the same VPC also consume IP addresses. Consider the IP requirements of all services when planning your VPC and subnet configurations.

3. **Neglecting to Update Services:** After creating new subnets or VPC connectors, ensure you update your App Runner services to use the new configurations. Failure to do so will result in continued use of the exhausted IP range.

4. **Misunderstanding CIDR Block Overlaps:** When adding secondary CIDR blocks to a VPC, ensure they don't overlap with existing blocks. Overlapping CIDR blocks can cause routing conflicts and IP address ambiguity.

5. **Exceeding VPC Limits:** Be aware that a VPC can have a maximum of 5 CIDR blocks (1 primary and 4 secondary). Plan your IP address space expansion within these constraints.

6. **Ignoring Subnet AZ Distribution:** When creating new subnets, ensure they are distributed across multiple Availability Zones for high availability and fault tolerance.

7. **Overlooking ENI Limits:** Remember that there are limits to the number of ENIs that can be attached to instances. Verify that your AWS account limits align with your planned network interface usage.

By being aware of these pitfalls, you can more effectively manage your VPC resources and avoid IP exhaustion issues in your App Runner services.

Additional Resources

1. [AWS VPC Documentation](#)

2. [Understanding CIDR Blocks](#)

3. [App Runner VPC Connectors](#)

Glossary

1. **ENI:** Elastic Network Interface, a virtual network interface in AWS.

2. **CIDR:** Classless Inter-Domain Routing, a method for allocating IP addresses.

3. **VPC Connector:** A resource that enables App Runner to connect to your VPC.

Security in App Runner

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to AWS App Runner, see [AWS Services in Scope by Compliance Program](#).

- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using App Runner. The following topics show you how to configure App Runner to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your App Runner resources.

Topics

- [Data protection in App Runner](#)

- [Identity and access management for App Runner](#)

- [Logging and monitoring in App Runner](#)

- [Compliance validation for App Runner](#)

- [Resilience in App Runner](#)

- [Infrastructure security in AWS App Runner](#)

- [Using App Runner with VPC endpoints](#)

- [Configuration and vulnerability analysis in App Runner](#)

- [Security best practices for App Runner](#)

Data protection in App Runner

The AWS [shared responsibility model](#) applies to data protection in AWS App Runner. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the [Data Privacy FAQ](#). For information about data protection in Europe, see the [AWS Shared Responsibility Model and GDPR](#) blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.

- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.

- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.

- Use AWS encryption solutions, along with all default security controls within AWS services.

- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.

- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard (FIPS) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with App Runner or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

For other App Runner security topics, see *[Security](#)*.

Topics

- [Protecting data using encryption](#)

- [Internetwork traffic privacy](#)

Protecting data using encryption

AWS App Runner reads your application source (source image or source code) from a repository that you specify and stores it for deployment to your service. For more information, see *Architecture and concepts*.

Data protection refers to protecting data while *in transit* (as it travels to and from App Runner) and *at rest* (while it is stored in AWS data centers).

For more information about data protection, see the section called "Data protection".

For other App Runner security topics, see *Security*.

Encryption in transit

You can achieve data protection in transit in two ways: encrypt the connection using Transport Layer Security (TLS), or use client-side encryption (where the object is encrypted before it is sent). Both methods are valid for protecting your application data. To secure the connection, encrypt it using TLS whenever your application, its developers and administrators, and its end users send or receive any objects. App Runner sets up your application to receive traffic over TLS.

Client-side encryption isn't a valid method for protecting the source image or code that you provide to App Runner for deployment. App Runner needs access to your application source, so it can't be encrypted. Therefore, be sure to secure the connection between your development or deployment environment and App Runner.

Encryption at rest and key management

To protect your application's data at rest, App Runner encrypts all stored copies of your application source image or source bundle. When you create an App Runner service, you can provide an AWS KMS key. If you provide one, App Runner uses your provided key to encrypt your source. If you don't provide one, App Runner uses an AWS managed key instead.

For details about App Runner service creation parameters, see CreateService. For information about AWS Key Management Service (AWS KMS), see the AWS Key Management Service Developer Guide.

Internetwork traffic privacy

App Runner uses Amazon Virtual Private Cloud (Amazon VPC) to create boundaries between resources in your App Runner application and control traffic between them, your on-premises network, and the internet. For more information about Amazon VPC security, see Internetwork traffic privacy in Amazon VPC in the *Amazon VPC User Guide*.

For information about associating your App Runner application with a custom Amazon VPC, see the section called "Outgoing traffic".

For information about securing requests to App Runner using a VPC endpoint, see the section called "VPC endpoints".

For more information about data protection, see the section called "Data protection".

For other App Runner security topics, see *Security*.

Identity and access management for App Runner

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use App Runner resources. IAM is an AWS service that you can use with no additional charge.

For other App Runner security topics, see *Security*.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How App Runner works with IAM
- App Runner identity-based policy examples
- Using service-linked roles for App Runner
- AWS managed policies for AWS App Runner
- Troubleshooting App Runner identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see [Troubleshooting App Runner identity and access](#))

- **Service administrator** - determine user access and submit permission requests (see [How App Runner works with IAM](#))

- **IAM administrator** - write policies to manage access (see [App Runner identity-based policy examples](#))

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

IAM users and groups

An [*IAM user*](#) is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see [Require human users to use federation with an identity provider to access AWS using temporary credentials](#) in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must [specify a principal](#) in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list (ACL) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.

- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see [Service control policies](#) in the *AWS Organizations User Guide*.

- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see [Resource control policies (RCPs)](#) in the *AWS Organizations User Guide*.

- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How App Runner works with IAM

Before you use IAM to manage access to AWS App Runner, you should understand what IAM features are available to use with App Runner. To get a high-level view of how App Runner and other AWS services work with IAM, see AWS Services That Work with IAM in the *IAM User Guide*.

For other App Runner security topics, see *Security*.

Topics

- App Runner identity-based policies
- App Runner resource-based policies
- Authorization based on App Runner tags
- App Runner user permissions
- App Runner IAM roles

App Runner identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. App Runner supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the *IAM User Guide*.

Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in App Runner use the following prefix before the action: `apprunner:`. For example, to grant someone permission to run an Amazon EC2 instance with the Amazon EC2

`RunInstances` API operation, you include the `ec2:RunInstances` action in their policy. Policy statements must include either an `Action` or `NotAction` element. App Runner defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

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

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

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

To see a list of App Runner actions, see [Actions defined by AWS App Runner](#) in the *Service Authorization Reference*.

Resources

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its [Amazon Resource Name (ARN)](#). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

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

App Runner resources have the following ARN structure:

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

For more information about the format of ARNs, see [Amazon Resource Names (ARNs) and AWS Service Namespaces](#) in the *AWS General Reference*.

For example, to specify the `my-service` service in your statement, use the following ARN:

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

To specify all services that belong to a specific account, use the wildcard (*):

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

Some App Runner actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

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

To see a list of App Runner resource types and their ARNs, see Resources defined by AWS App Runner in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS App Runner.

Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Condition element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

App Runner supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

App Runner defines a set of service-specific condition keys. In addition, App Runner supports tag-based access control, which is implemented using condition keys. For details, see the section called "Authorization based on App Runner tags".

To see a list of App Runner condition keys, see Condition keys for AWS App Runner in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS App Runner.

Examples

To view examples of App Runner identity-based policies, see App Runner identity-based policy examples.

App Runner resource-based policies

App Runner does not support resource-based policies.

Authorization based on App Runner tags

You can attach tags to App Runner resources or pass tags in a request to App Runner. To control access based on tags, you provide tag information in the [condition element](#) of a policy using the apprunner:ResourceTag/*key-name*, aws:RequestTag/*key-name*, or aws:TagKeys condition keys. For more information about tagging App Runner resources, see [the section called "Configuration"](#).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see [Controlling access to App Runner services based on tags](#).

App Runner user permissions

To use App Runner, IAM users need permissions to App Runner actions. A common way to grant permissions to users is by attaching a policy to IAM users or groups. For more information about managing user permissions, see [Changing permissions for an IAM user](#) in the *IAM User Guide*.

App Runner provides two managed policies that you can attach to your users.

- [AWSAppRunnerReadOnlyAccess](#) – Grants permissions to list and view details about App Runner resources.

- [AWSAppRunnerFullAccess](#)– Grants permissions to all App Runner actions.

For more granular control of user permissions, you can create a custom policy and attach it to your users. For details, see [Creating IAM policies](#) in the *IAM User Guide*.

For examples of user policies, see [the section called "User policies"](#).

App Runner IAM roles

An [IAM role](#) is an entity within your AWS account that has specific permissions.

Service-linked roles

[Service-linked roles](#) allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

App Runner supports service-linked roles. For information about creating or managing App Runner service-linked roles, see the section called "Using service-linked roles".

Service roles

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM user can change the permissions for this role. However, doing so might break the functionality of the service.

App Runner supports a few service roles.

Access role

The access role is a role that App Runner uses for accessing images in Amazon Elastic Container Registry (Amazon ECR) in your account. It's required to access an image in Amazon ECR, and isn't required with Amazon ECR Public.

Before creating a service based on an image in Amazon ECR, use IAM to create a service role. Use the managed policy AWSAppRunnerServicePolicyForECRAccess in your service role. You can then pass this role to App Runner when you call the CreateService API in the AuthenticationConfiguration member of the SourceConfiguration parameter, or when you use the App Runner console to create a service.

> **ⓘ Note**
>
> If you create your own custom policy for your access role, be sure to specify "Resource": "*" for the ecr:GetAuthorizationToken action. Tokens can be used to access any Amazon ECR registry that you have access to.

When you create your access role, be sure to add a trust policy that declares the App Runner service principal build.apprunner.amazonaws.com as a trusted entity.

Trust policy for an access role

JSON

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

```
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "build.apprunner.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

If you use the App Runner console to create a service, the console can automatically create an access role for you and choose it for the new service. The console also lists other roles in your account, and you can select a different role if you like.

Instance role

The instance role is an optional role that App Runner uses to provide permissions to AWS service actions that your service's compute instances need. You need to provide an instance role to App Runner if your application code calls AWS actions (APIs). Either embed the required permissions in your instance role or create your own custom policy and use it in the instance role. We have no way to anticipate which calls your code uses. Therefore, we don't provide a managed policy for this purpose.

Before creating an App Runner service, use IAM to create a service role with the required custom or embedded policies. You can then pass this role to App Runner as the instance role when you call the [CreateService](#) API in the `InstanceRoleArn` member of the [InstanceConfiguration](#) parameter, or when you use the App Runner console to create a service.

When you create your instance role, be sure to add a trust policy that declares the App Runner service principal `tasks.apprunner.amazonaws.com` as a trusted entity.

Trust policy for an instance role

JSON

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

```
      "Principal": {
        "Service": "tasks.apprunner.aws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

If you use the App Runner console to create a service, the console lists the roles in your account, and you can select the role that you created for this purpose.

For information about creating a service, see the section called "Creation".

App Runner identity-based policy examples

By default, IAM users and roles don't have permission to create or modify AWS App Runner resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the *IAM User Guide*.

For other App Runner security topics, see *Security*.

Topics

- Policy best practices
- User policies
- Controlling access to App Runner services based on tags

Policy best practices

Identity-based policies determine whether someone can create, access, or delete App Runner resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies*

that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.

- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.

- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

User policies

To access the App Runner console, IAM users must have a minimum set of permissions. These permissions must allow you to list and view details about the App Runner resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for users with that policy.

App Runner provides two managed policies that you can attach to your users.

- AWSAppRunnerReadOnlyAccess – Grants permissions to list and view details about App Runner resources.

- AWSAppRunnerFullAccess – Grants permissions to all App Runner actions.

To ensure that users can use the App Runner console, attach, at a minimum, the AWSAppRunnerReadOnlyAccess managed policy to the users. You can attach the AWSAppRunnerFullAccess managed policy instead, or add specific additional permissions, to allow users to create, modify, and delete resource. For more information, see Adding Permissions to a User in the *IAM User Guide*.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that you want to allow users to perform.

The following examples demonstrate custom user policies. You can use them as starting points to defining your own custom user policies. Copy the example, and or remove actions, scope down resources, and add conditions.

Example: console and connection management user policy

This example policy enables console access and allows connection creation and management. It doesn't allow App Runner service creation and management. It can be attached to a user whose role is to manage App Runner service access to source code assets.

JSON

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

```
    }
```

Example: user policies that use condition keys

The examples in this section demonstrate conditional permissions that depend on some resource properties or action parameters.

This example policy enables creating an App Runner service but denies using a connection named `prod`.

JSON

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

This example policy enables updating an App Runner service named `preprod` only with an auto scaling configuration named `preprod`.

JSON

```
{
    "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:us-east-1:*:autoscalingconfiguration/preprod/*"
                        }
                    }
                }
            ]
        }
```

Controlling access to App Runner services based on tags

You can use conditions in your identity-based policy to control access to App Runner resources based on tags. This example shows how you might create a policy that allows deleting an App Runner service. However, permission is granted only if the service tag Owner has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "ListServicesInConsole",
      "Effect": "Allow",
      "Action": "apprunner:ListServices",
      "Resource": "*"
    },
    {
      "Sid": "DeleteServiceIfOwner",
      "Effect": "Allow",
      "Action": "apprunner:DeleteService",
      "Resource": "arn:aws:apprunner:us-east-1:*:service/*",
      "Condition": {
        "StringEquals": {"aws:ResourceTag/Owner": "${aws:username}"}
      }
```

```
        }
    ]
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to delete an App Runner service, the service must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON Policy Elements: Condition in the *IAM User Guide*.

Using service-linked roles for App Runner

AWS App Runner uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to App Runner. Service-linked roles are predefined by App Runner and include all the permissions that the service requires to call other AWS services on your behalf.

Topics

- Using roles for management
- Using roles for networking

Using roles for management

AWS App Runner uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to App Runner. Service-linked roles are predefined by App Runner and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up App Runner easier because you don't have to manually add the necessary permissions. App Runner defines the permissions of its service-linked roles, and unless defined otherwise, only App Runner can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your App Runner resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS Services That Work](#) [with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for App Runner

App Runner uses the service-linked role named **AWSServiceRoleForAppRunner**.

The role allows App Runner to perform the following tasks:

- Push logs to Amazon CloudWatch Logs log groups.
- Create Amazon CloudWatch Events rules to subscribe to Amazon Elastic Container Registry (Amazon ECR) image pushes.
- Send tracing information to AWS X-Ray.

The AWSServiceRoleForAppRunner service-linked role trusts the following services to assume the role:

- `apprunner.amazonaws.com`

The permissions policies of the AWSServiceRoleForAppRunner service-linked role contain all of the permissions that App Runner needs to complete actions on your behalf:

- Managed policy [AppRunnerServiceRolePolicy](#)
- Policy for X-Ray tracing – See the following policy content.

Policy for X-Ray tracing

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
   {
     "Effect": "Allow",
     "Action": [
       "xray:PutTraceSegments",
       "xray:PutTelemetryRecords",
```

```
          "xray:GetSamplingRules",
          "xray:GetSamplingTargets",
          "xray:GetSamplingStatisticSummaries"
        ],
        "Resource": [
          "*"
        ]
      }
    ]
}
```

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Creating a service-linked role for App Runner

You don't need to manually create a service-linked role. When you create an App Runner service in the AWS Management Console, the AWS CLI, or the AWS API, App Runner creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create an App Runner service, App Runner creates the service-linked role for you again.

Editing a service-linked role for App Runner

App Runner does not allow you to edit the AWSServiceRoleForAppRunner service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a service-linked role for App Runner

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning up a service-linked role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

In App Runner, this means deleting all App Runner services in your account. To learn about deleting App Runner services, see the section called "Deletion".

> **ⓘ Note**
>
> If the App Runner service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

Manually delete the service-linked role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAppRunner service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported regions for App Runner service-linked roles

App Runner supports using service-linked roles in all of the regions where the service is available. For more information, see AWS App Runner endpoints and quotas in the *AWS General Reference*.

Using roles for networking

AWS App Runner uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to App Runner. Service-linked roles are predefined by App Runner and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up App Runner easier because you don't have to manually add the necessary permissions. App Runner defines the permissions of its service-linked roles, and unless defined otherwise, only App Runner can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your App Runner resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for App Runner

App Runner uses the service-linked role named **AWSServiceRoleForAppRunnerNetworking**.

The role allows App Runner to perform the following tasks:

- Attach a VPC to your App Runner service and manage network interfaces.

The AWSServiceRoleForAppRunnerNetworking service-linked role trusts the following services to assume the role:

- `networking.apprunner.amazonaws.com`

The role permissions policy named AppRunnerNetworkingServiceRolePolicy contains all of the permissions that App Runner needs to complete actions on your behalf.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-Linked Role Permissions in the *IAM User Guide.*

Creating a service-linked role for App Runner

You don't need to manually create a service-linked role. When you create a VPC connector in the AWS Management Console, the AWS CLI, or the AWS API, App Runner creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you create a VPC connector, App Runner creates the service-linked role for you again.

Editing a service-linked role for App Runner

App Runner does not allow you to edit the AWSServiceRoleForAppRunnerNetworking service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a Service-Linked Role in the *IAM User Guide.*

Deleting a service-linked role for App Runner

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up your service-linked role before you can manually delete it.

Cleaning Up a Service-Linked Role

Before you can use IAM to delete a service-linked role, you must first delete any resources used by the role.

In App Runner, this means disassociating VPC connectors from all App Runner services in your account, and deleting the VPC connectors. For more information, see the section called "Outgoing traffic".

> ⓘ **Note**
>
> If the App Runner service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

Manually Delete the Service-Linked Role

Use the IAM console, the AWS CLI, or the AWS API to delete the AWSServiceRoleForAppRunnerNetworking service-linked role. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Supported regions for App Runner service-linked roles

App Runner supports using service-linked roles in all of the regions where the service is available. For more information, see AWS App Runner endpoints and quotas in the *AWS General Reference*.

AWS managed policies for AWS App Runner

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

App Runner updates to AWS managed policies

View details about updates to AWS managed policies for App Runner since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the App Runner Document history page.

Change	Description	Date
AWSAppRunnerReadOnlyAccess – New policy	App Runner added a new policy to allow users to list and view details about App Runner resources.	Feb 24, 2022
AWSAppRunnerFullAccess – Update to an existing policy	App Runner updated the resource list for the `iam:CreateServiceLinkedRole` action to allow creation of `AWSServiceRoleForAppRunnerNetworking` service-linked role.	Feb 8, 2022
AppRunnerNetworkingServiceRolePolicy – New policy	App Runner added a new policy to allow App Runner to make calls to Amazon	Feb 8, 2022

Change	Description	Date
	Virtual Private Cloud to attach a VPC to your App Runner service and manage network interfaces on behalf of App Runner services. The policy is used in the `AWSServiceRoleForAppRunnerN etworking` service-linked role.	
[AWSAppRunnerFullAccess](#) – New policy	App Runner added a new policy to allow users to perform all App Runner actions.	Jan 10, 2022
[AppRunnerServiceRolePolicy](#) – New policy	App Runner added a new policy to allow App Runner to make calls to Amazon CloudWatch Logs and Amazon CloudWatch Events on behalf of App Runner services. The policy is used in the `AWSServiceRoleForAppRunner` service-linked role.	Mar 1, 2021
[AWSAppRunnerServicePolicyFo rECRAccess](#) – New policy	App Runner added a new policy to allow App Runner to access Amazon Elastic Container Registry (Amazon ECR) images in your account.	Mar 1, 2021
App Runner started tracking changes	App Runner started tracking changes for its AWS managed policies.	Mar 1, 2021

Troubleshooting App Runner identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS App Runner and IAM.

For other App Runner security topics, see *Security*.

Topics

- [I'm not authorized to perform an action in App Runner](#)
- [I want to allow people outside of my AWS account to access my App Runner resources](#)

I'm not authorized to perform an action in App Runner

If the AWS Management Console tells you that you're not authorized to perform an action, contact your administrator for assistance. Your administrator is the person that provided you with your AWS sign-in credentials.

The following example error occurs when an IAM user named `marymajor` tries to use the console to view details about an App Runner service but doesn't have `apprunner:DescribeService` permissions.

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

In this case, Mary asks her administrator to update her policies to allow her to access the *my-example-service* resource using the `apprunner:DescribeService` action.

I want to allow people outside of my AWS account to access my App Runner resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether App Runner supports these features, see [How App Runner works with IAM](#).

- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.

- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.

- To learn how to provide access through identity federation, see [Providing access to externally authenticated users (identity federation)](#) in the *IAM User Guide*.

- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Logging and monitoring in App Runner

Monitoring is an important part of maintaining the reliability, availability, and performance of your AWS App Runner service. Collecting monitoring data from all parts of your AWS solution allows you to more easily debug a failure if one occurs. App Runner integrates with several AWS tools for monitoring your App Runner services and responding to potential incidents.

Amazon CloudWatch alarms

With Amazon CloudWatch alarms, you can watch a service metric over a time period that you specify. If the metric exceeds a given threshold for a given number of periods, you receive a notification.

App Runner collects a variety of metrics about the service as a whole and the instances (scaling units) that run your web service. For more information, see Metrics (CloudWatch).

Application logs

App Runner collects the output of your application code and streams it to Amazon CloudWatch Logs. What's in this output is up to you. For example, you could include detailed records of requests made to your web service. These log records might prove useful in security and access audits. For more information, see Logs (CloudWatch Logs).

AWS CloudTrail action logs

App Runner is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in App Runner. CloudTrail captures all API calls for App Runner as events. You can view the most recent events in the CloudTrail console, and you can create a trail to enable continuous delivery of CloudTrail events to an Amazon Simple Storage Service (Amazon S3) bucket. For more information, see API actions (CloudTrail).

Compliance validation for App Runner

Third-party auditors assess the security and compliance of AWS App Runner as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see [AWS Security Documentation](#).

For other App Runner security topics, see [*Security*](#).

Resilience in App Runner

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

AWS App Runner manages and automates the use of the AWS global infrastructure on your behalf. When using App Runner, you benefit from the availability and fault tolerance mechanisms that AWS offers.

For other App Runner security topics, see [*Security*](#).

Infrastructure security in AWS App Runner

As a managed service, AWS App Runner is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use AWS published API calls to manage and operate App Runner through the network. Clients that call App Runner APIs must support Transport Layer Security (TLS) 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes. These requirements do not apply to endpoints from App Runner applications.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

For other App Runner security topics, see *Security*.

Using App Runner with VPC endpoints

Your AWS application might integrate AWS App Runner services with other AWS services that run in a VPC from Amazon Virtual Private Cloud (Amazon VPC). Parts of your application might make requests to App Runner from within the VPC. For example, you might use AWS CodePipeline to continuously deploy to your App Runner service. One way to improve the security of your application is to send these App Runner requests (and requests to other AWS services) over a VPC endpoint.

Using a *VPC endpoint*, you can privately connect your VPC to supported AWS services and VPC endpoint services that are powered by AWS PrivateLink. You don't need an internet gateway, NAT device, VPN connection, or Direct Connect connection.

Resources in your VPC don't use public IP addresses to interact with App Runner resources. Traffic between your VPC and App Runner doesn't leave the Amazon network. For more information about VPC endpoints, see VPC endpoints in the *AWS PrivateLink Guide*.

> ⓘ **Note**
>
> By default, the web application in your App Runner service runs in a VPC that App Runner provides and configures. This VPC is public. It means that it's connected to the internet. You can optionally associate your application with a custom VPC. For more information, see the section called "Outgoing traffic".
>
> You can configure your services to access the internet, including AWS APIs, even when your service is connected to a VPC. For instructions on how to enable public internet access for VPC outbound traffic, see the section called "Considerations when selecting a subnet".
>
> App Runner doesn't support creating a VPC endpoint for your application.

Setting up a VPC endpoint for App Runner

To create the interface VPC endpoint for the App Runner service in your VPC, follow the [Create an interface endpoint](#) procedure in the *AWS PrivateLink Guide*. For **Service Name**, choose `com.amazonaws.`*`region`*`.apprunner`.

VPC network privacy considerations

> ⚠️ **Important**
>
> Using a VPC endpoint for App Runner doesn't ensure that all traffic from your VPC stays off of the internet. The VPC might be public. Moreover, some parts of your solution might not use VPC endpoints to make AWS API calls. For example, AWS services might call other services using their public endpoints. If traffic privacy is required for the solution in your VPC, read this section.

To ensure privacy of network traffic in your VPC, consider the following:

- *Enable DNS name* – Parts of your application might still send requests to App Runner over the internet using the `apprunner.`*`region`*`.amazonaws.com` public endpoint. If your VPC is configured with internet access, these requests succeed with no indication to you. You can prevent this by ensuring that **Enable DNS name** is enabled when you create the endpoint. By default, it's set to true. This adds a DNS entry in your VPC that maps the public service endpoint to the interface VPC endpoint.

- *Configure VPC endpoints for additional services* – Your solution might send requests to other AWS services. For example, AWS CodePipeline might send requests to AWS CodeBuild. Configure VPC endpoints for these services, and enable DNS names on these endpoints.

- *Configure a private VPC* – If possible (if your solution doesn't need internet access at all), set up your VPC as private, which means that it has no internet connection. This ensures that a missing VPC endpoint causes a visible error, so that you can add the missing endpoint.

Using endpoint policies to control access with VPC endpoints

VPC endpoint policies are supported for App Runner. By default, full access to App Runner is allowed through the interface endpoint. VPC endpoint policies can be used to control which AWS principals can access the App Runner endpoint. Alternatively, you can associate a security

group with the endpoint network interfaces to control traffic to App Runner through the interface endpoint.

Integrating with interface endpoint

App Runner supports AWS PrivateLink, which provides private connectivity to App Runner and eliminates exposure of traffic to the internet. To enable your application to send requests to App Runner using AWS PrivateLink, configure a type of VPC endpoint known as an *interface endpoint*. For more information, see Interface VPC endpoints (AWS PrivateLink) in the *AWS PrivateLink Guide*.

Configuration and vulnerability analysis in App Runner

AWS and our customers share responsibility for achieving a high level of software component security and compliance. For more information, see the AWS shared responsibility model.

Patch container images

Patching the container image is part of the customer's responsibility in the shared security model. The image owner is responsible for updating and regularly patching the container image. We recommend establishing a routine schedule for checking and applying updates to your container images. For more information on how to scan your images for vulnerabilities, see the AWS App Runner Documentation

For other App Runner security topics, see *Security*.

Security best practices for App Runner

AWS App Runner provides several security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations, not prescriptions.

For other App Runner security topics, see *Security*.

Preventive security best practices

Preventive security controls attempt to prevent incidents before they occur.

Implement least privilege access

App Runner provides AWS Identity and Access Management (IAM) managed policies for IAM users and the access role. These managed policies specify all permissions that might be necessary for the correct operation of your App Runner service.

Your application might not require all the permissions in our managed policies. You can customize them and grant only the permissions that are required for your users and your App Runner service to perform their tasks. This is particularly relevant to user policies, where different user roles might have different permission needs. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

Scan your images for vulnerabilities

You can use the Amazon ECR's APIs to help identify software vulnerabilities in your container images. For more information, see the Amazon ECR documentation.

Detective security best practices

Detective security controls identify security violations after they have occurred. They can help you detect a potential security threat or incident.

Implement monitoring

Monitoring is an important part of maintaining the reliability, security, availability, and performance of your App Runner solutions. AWS provides several tools and services to help you monitor your AWS services.

The following are some examples of items to monitor:

- *Amazon CloudWatch metrics for App Runner* – Set alarms for key App Runner metrics and for your application's custom metrics. For details, see Metrics (CloudWatch).
- *AWS CloudTrail entries* – Track actions that might impact availability, like `PauseService` or `DeleteConnection`. For details, see API actions (CloudTrail).

AWS Glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS Glossary Reference*.