



Hands-on tutorials

Deploy a Web App on Nginx Server using AWS App Runner



Deploy a Web App on Nginx Server using AWS App Runner: Hands-on tutorials

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Deploy a Web App on Nginx Server using AWS App Runner	i
Overview	1
What you will accomplish	1
Prerequisites	2
Implementation	2
Congratulations	19

Deploy a Web App on Nginx Server using AWS App Runner

AWS experience	Beginner
Time to complete	20 minutes
Cost to complete	Less than \$0.16 if completed within two hours and you delete your resources at the end of the tutorial.
Get help	Troubleshooting AWS CLI Troubleshooting Docker commands and ECR
Last updated	June 1, 2022

Overview

In this tutorial, you will learn how to deploy a sample containerized application on a Nginx server using AWS App Runner.

AWS App Runner is a fully managed service that makes it easy for developers to quickly deploy containerized web applications and APIs, at scale and with no prior infrastructure experience required. Start with your source code or a container image. App Runner automatically builds and deploys the web application and load balances traffic with encryption. App Runner also scales up or down automatically to meet your traffic needs.

What you will accomplish

In this tutorial, you will:

- Create a container image for your web app
- Push the image to Amazon Elastic Container Registry
- Create an AWS App Runner service

- Clean up your resources

Prerequisites

Before starting this tutorial, you will need:

- An AWS account: if you don't already have one follow the [Setup Your Environment](#) tutorial.
- [AWS Command Line Interface](#) **installed** and **configured**.
- [Docker Engine](#) **installed**, and the application **started**.
- Visual Studio Code **installed**.

Implementation

Use the following step-by-step written tutorial or watch the video to learn how to Deploy a Web Application on Nginx Server using AWS App Runner.

Step 1: Push container image to Amazon ECR

In this step, you will create a private repository in Amazon ECR and push the container image you built in previous module to the newly created repository.

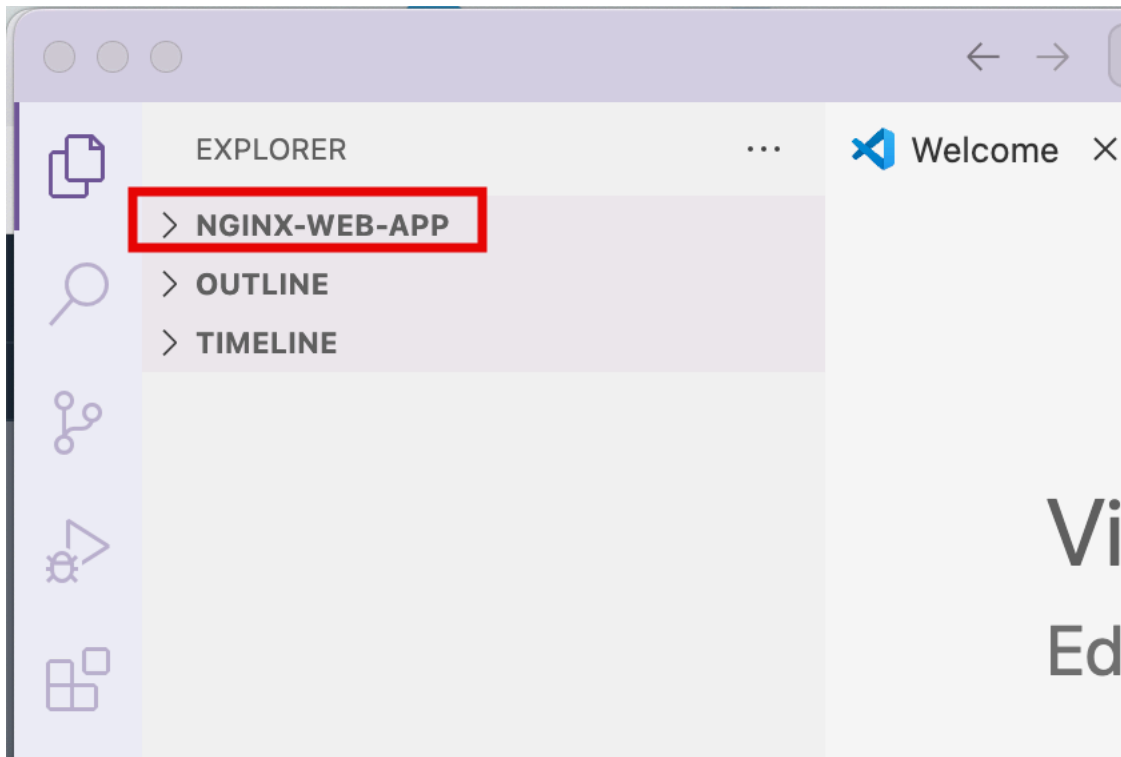
1. Create a project directory

In a new terminal window, **run** the following commands to **create** a new folder called **nginx-web-app**, and **navigate** to the folder.

```
mkdir nginx-web-appcd nginx-web-app
```

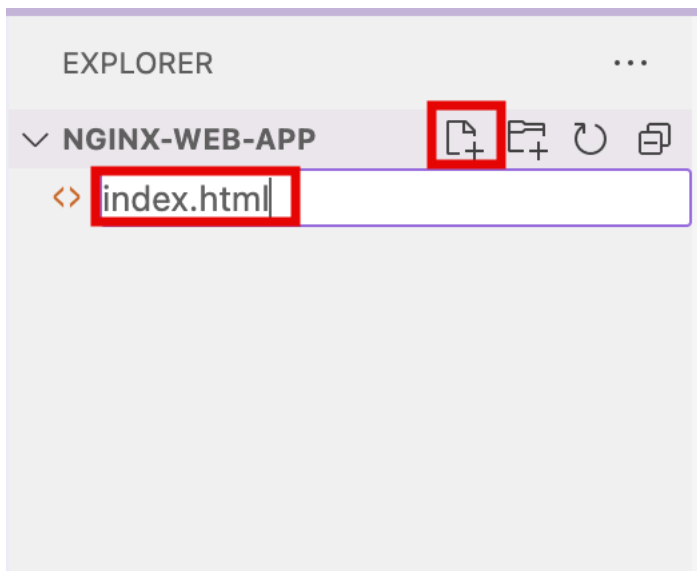
2. Open Visual Studio Code

On your local machine, **navigate** to the Visual Studio Code application, and open the **nginx-web-app** folder.



3. Create an HTML file

In the **Explorer** section, select the **+New file** icon, and enter **index.html** for the file name.

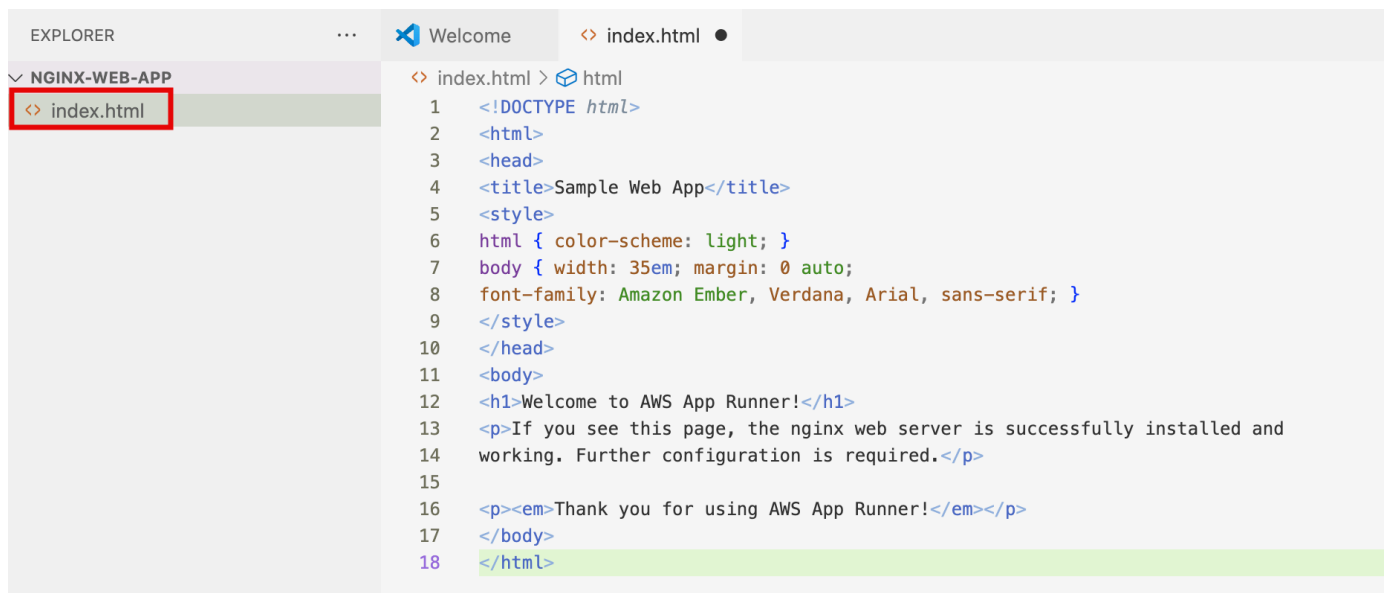


4. Add HTML content

Select the **index.html** file, and **update** it with the following code. Then, **save** the file.

```
<!DOCTYPE html>
```

```
<html>
<head>
  <title>Sample Web App</title>
  <style>
    html {
      color-scheme: light;
    }
    body {
      width: 35em;
      margin: 0 auto;
      font-family: Amazon Ember, Verdana, Arial, sans-serif;
    }
  </style>
</head>
<body>
  <h1>Welcome to AWS App Runner!</h1>
  <p>If you see this page, the nginx web server is successfully installed and
  working. Further configuration is required.</p>
  <p><em>Thank you for using AWS App Runner!</em></p>
</body>
</html>
```

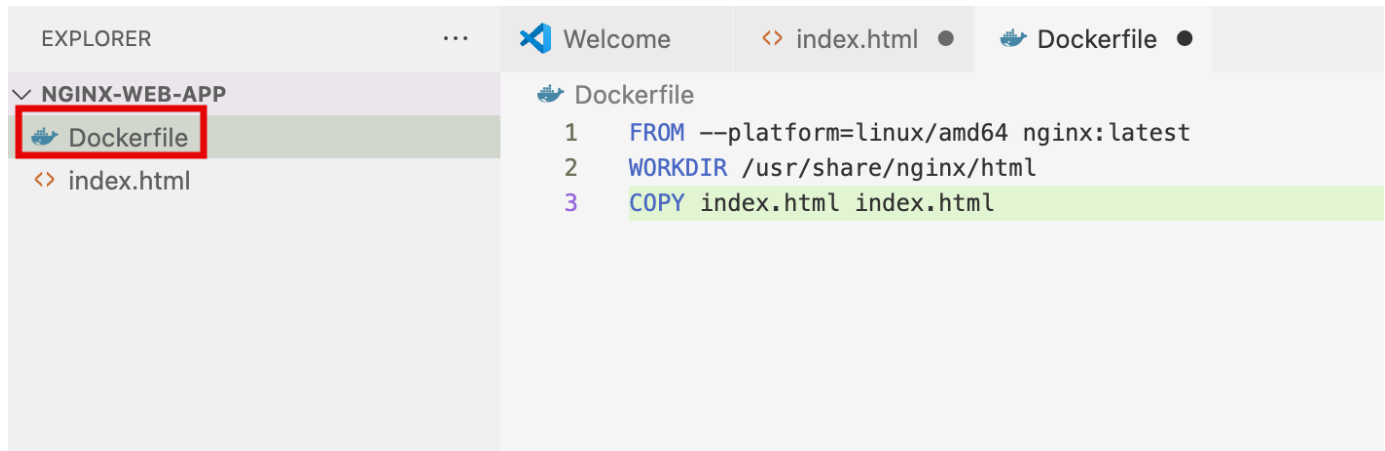
A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a folder named 'NGINX-WEB-APP' with a file 'index.html' highlighted by a red box. The main editor area shows the content of 'index.html' with line numbers from 1 to 18. The code is identical to the one shown in the previous block, but with syntax highlighting: HTML tags are blue, CSS properties are green, and text content is black. Line 18 is highlighted in light green.

```
<> index.html > html
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title>Sample Web App</title>
5  <style>
6  html { color-scheme: light; }
7  body { width: 35em; margin: 0 auto;
8  font-family: Amazon Ember, Verdana, Arial, sans-serif; }
9  </style>
10 </head>
11 <body>
12 <h1>Welcome to AWS App Runner!</h1>
13 <p>If you see this page, the nginx web server is successfully installed and
14 working. Further configuration is required.</p>
15
16 <p><em>Thank you for using AWS App Runner!</em></p>
17 </body>
18 </html>
```

5. Create Dockerfile

Create another file named **Dockerfile**, and **update** it with the following code. Then, **save** the file.

```
FROM --platform=linux/amd64 nginx:latest
WORKDIR /usr/share/nginx/html
COPY index.html index.html
```



6. Build a container

In the open terminal window, **run** the following command to create container image.

```
docker build -t nginx-web-app .
```

```

~/nginx-web-app — aws-testing — -zsh — zsh — Basic — ttys000 — 102x31
[\aws-testing] docker build -t nginx-web-app .
[+] Building 9.2s (9/9) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile            0.0s
=> => transferring dockerfile: 411B                            0.0s
=> [internal] load metadata for docker.io/library/nginx:latest 1.0s
=> [auth] library/nginx:pull token for registry-1.docker.io    0.0s
=> [internal] load .dockerignore                              0.0s
=> => transferring context: 2B                                  0.0s
=> [1/3] FROM docker.io/library/nginx:latest@sha256:6af79ae5de407283dcea 8.0s
=> => resolve docker.io/library/nginx:latest@sha256:6af79ae5de407283dcea 0.0s
=> => sha256:6af79ae5de407283dcea8b00d5c37ace95441fd58 10.27kB / 10.27kB 0.0s
=> => sha256:a72860cb95fd59e9c696c66441c64f18e66915fa26b 7.30kB / 7.30kB 0.0s
=> => sha256:baa881b012a49e3c2cd6ab9d80f9fcd2962a98af8ed 2.29kB / 2.29kB 0.0s
=> => sha256:efc2b5ad9eec05befa54239d53feeae3569ccbef6 29.13MB / 29.13MB 6.5s
=> => sha256:8fe9a55eb80f3167f7b3a9c39f90b9eacf833841e 41.83MB / 41.83MB 4.2s
=> => sha256:045037a63be803c1d446a5239439580a49cd8a8682a5add 627B / 627B 0.1s
=> => sha256:3dfc528a4df9e1be9b2817271a35cef87f001e699e5b8ef 394B / 394B 0.2s
=> => sha256:7111b42b4bfa1b5273abcc4b138983f48f9cb96bb3f896a 955B / 955B 0.2s
=> => sha256:9e891cdb453be97c53e1ddbe4b955ee71099f18f16e 1.21kB / 1.21kB 0.3s
=> => sha256:0f11e17345c583a30e9cc89b80b1423b7b52b0e36cd 1.40kB / 1.40kB 0.4s
=> => extracting sha256:efc2b5ad9eec05befa54239d53feeae3569ccbef689aa5e5 0.7s
=> => extracting sha256:8fe9a55eb80f3167f7b3a9c39f90b9eacf833841e5a9f8d6 0.5s
=> => extracting sha256:045037a63be803c1d446a5239439580a49cd8a8682a5addf 0.0s
=> => extracting sha256:7111b42b4bfa1b5273abcc4b138983f48f9cb96bb3f896a6 0.0s
=> => extracting sha256:3dfc528a4df9e1be9b2817271a35cef87f001e699e5b8ef9 0.0s
=> => extracting sha256:9e891cdb453be97c53e1ddbe4b955ee71099f18f16e68e70 0.0s
=> => extracting sha256:0f11e17345c583a30e9cc89b80b1423b7b52b0e36cda9a6d 0.0s
=> [internal] load build context                                0.0s
=> => transferring context: 2.16kB                                0.0s
=> [2/3] WORKDIR /usr/share/nginx/html                        0.1s
=> [3/3] COPY index.html index.html                          0.0s
=> exporting to image                                        0.0s
=> => exporting layers                                            0.0s
=> => writing image sha256:cc25f42be9011e8571e4cb9156ee7e2f415ef5653e212 0.0s
=> => naming to docker.io/library/nginx-web-app                 0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/'nnyvumd2on
23ay
[\aws-testing] $

```

Step 2: Create an AWS App Runner Service

In this module, you will create an AWS App Runner service using the container image you built in previous module.

1. Create an Amazon ECR repository

Sign in to the AWS Management console in a new browser window, and **open** the Amazon Elastic Container Registry at <https://console.aws.amazon.com/ecr/home>.

For **Create a repository**, choose **Create**.

Containers

Amazon Elastic Container Registry

Share and deploy container software, publicly or privately

Create a repository



2. Configure repository settings

On the **Create repository** page, for **Repository name** enter **nginx-web-app**, leave the default selections, and select **Create repository**.

Create repository

General settings

Visibility settings | [Info](#)

Choose the visibility setting for the repository.

Private

Access is managed by IAM and repository policy permissions.

Public

Publicly visible and accessible for image pulls.

Repository name

Provide a concise name. A developer should be able to identify the repository contents by the name.

64C .dkr.ecr.sa-east-1.amazonaws.com/

13 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes.

Tag immutability | [Info](#)

Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

Disabled

[i](#) Once a repository is created, the visibility setting of the repository can't be changed.

Image scan settings

[i](#) **Deprecation warning**

ScanOnPush configuration at the repository level is deprecated in favor of registry level scan filters.

Scan on push

Enable scan on push to have each image automatically scanned after being pushed to a repository. If disabled, each image scan must be manually started to get scan results.

Disabled

Encryption settings

KMS encryption

You can use AWS Key Management Service (KMS) (KMS) to encrypt images stored in this repository, instead of using the default encryption settings.

Disabled

[i](#) The KMS encryption settings cannot be changed or disabled after the repository is created.

[Cancel](#)

[Create repository](#)

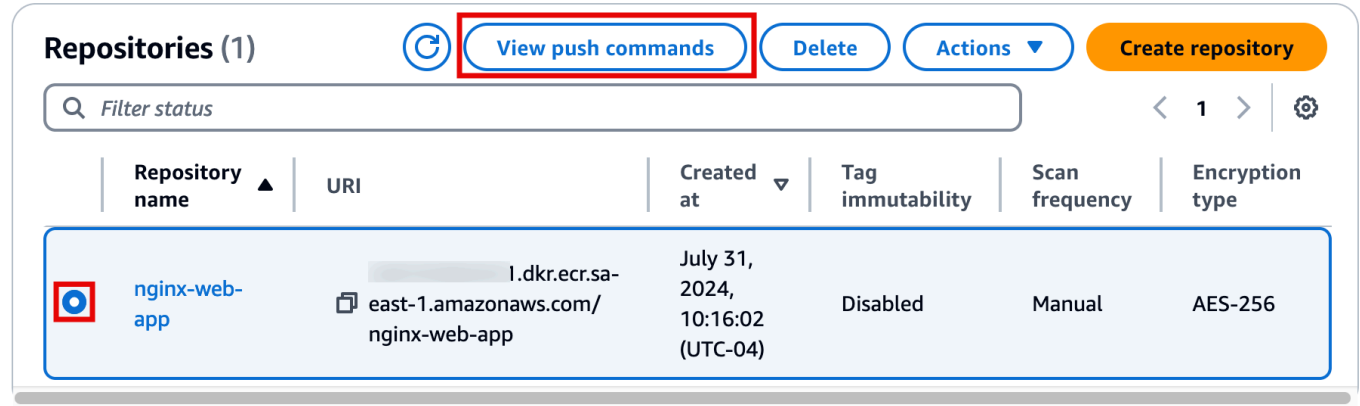
3. View push commands

Once the repository has been created, select the **radio button** for the repository, and then select **View push commands**.

✔ **Created private repository**
nginx-web-app has been successfully created in private registry

[Amazon ECR](#) > [Private registry](#) > [Repositories](#)

Private repositories



Repositories (1) [View push commands](#) [Delete](#) [Actions](#) [Create repository](#)

Filter status

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
nginx-web-app	east-1.dkr.ecr.sa-east-1.amazonaws.com/nginx-web-app	July 31, 2024, 10:16:02 (UTC-04)	Disabled	Manual	AES-256

4. Push your image

Follow all the steps in the pop-up window, to **authenticate** and **push** the image to the repository.

Push commands for nginx-web-app



macOS / Linux

Windows

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see [Getting Started with Amazon ECR](#).

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see [Registry Authentication](#).

1. Retrieve an authentication token and authenticate your Docker client to your registry. Use the AWS CLI:

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin  
[REDACTED].dkr.ecr.us-west-2.amazonaws.com
```

Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.

2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions [here](#). You can skip this step if your image is already built:

```
docker build -t nginx-web-app .
```

3. After the build completes, tag your image so you can push the image to this repository:

```
docker tag nginx-web-app:latest [REDACTED].dkr.ecr.us-west-2.amazonaws.com/nginx-web-app:latest
```

4. Run the following command to push this image to your newly created AWS repository:

```
docker push [REDACTED].dkr.ecr.us-west-2.amazonaws.com/nginx-web-app:latest
```

Close

Step 3: Create a container image

In this step, you will create a container image of a sample web app.

1. Create an App Runner service

In the Source and deployment section, leave the default selections for Repository type and Provider. For Container image URI, select Browse.

Compute

AWS App Runner

Build and run production web applications at scale

AWS App Runner is a fully managed service that makes it easy for developers to deploy from source code or container image directly to a scalable and secure web application.

Get started with App Runner

Create an App Runner service

2. Configure the source

In the **Source and deployment** section, leave the default selections for **Repository type** and **Provider**. For **Container image URI**, select **Browse**.

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

Amazon ECR
Container images stored in the private ECR.

Amazon ECR Public
Container images publicly shared by vendors, open source projects, and community developers.

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

3. Select the container image

In the pop-up window, for **Image repository**, select **nginx-web-app**, and choose **Continue**.

Select Amazon ECR container image ✕

Choose an image repository and tag in the Amazon ECR registry of your AWS account **assumed-role%2FAdmin%2F** to deploy to your App Runner service. To deploy a container image of a different AWS Account, cancel and paste the image URI.

Image repository

nginx-web-app ▼

Image tag

latest ▼

Cancel **Continue**

4. Set up ECR access

In the **Deployment settings** section, for **ECR access role**, select **Create new service role**, and choose **Next**.

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**

5. Configure service settings

On the **Configure service** page, for **Service name** enter **nginx-web-app-service**, and change the **Port** to **80**. Leave the rest as default, and select **Next**.

Configure service [Info](#)

Configure service

Service settings

Service name

Virtual CPU

Virtual memory

Runtime 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 for secrets

Port

Your service uses this TCP port.

▶ 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

▶ Networking [Info](#)

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

▶ Observability

Configure observability tooling.

▶ Tags [Info](#)

6. Review and deploy

On the **Review and create** page, review all inputs, and choose **Create & deploy**.

Review and create [Info](#)

Step 1: Source and deployment

[Edit](#)

Source

Container registry
Amazon ECR

Provider
ECR

Container image URI
[redacted].dkr.ecr.us-east-1.amazonaws.com/nginx-
web-app:latest

Deployment settings

Deployment method
Manual deployments

ECR access role
arn:aws:iam::[redacted]:role/service-role/
AppRunnerECRAccessRole

Step 2: Configure service

[Edit](#)

Service settings

Service name
nginx-web-app-service

Virtual CPU & memory
1 vCPU & 2 GB

Port
80

Runtime environment variables

Name	Value
------	-------

No environment variables have been configured.

▶ **Additional configuration**

▶ **Auto scaling**

▶ **Health check**

▶ **Security**

▶ **Networking**

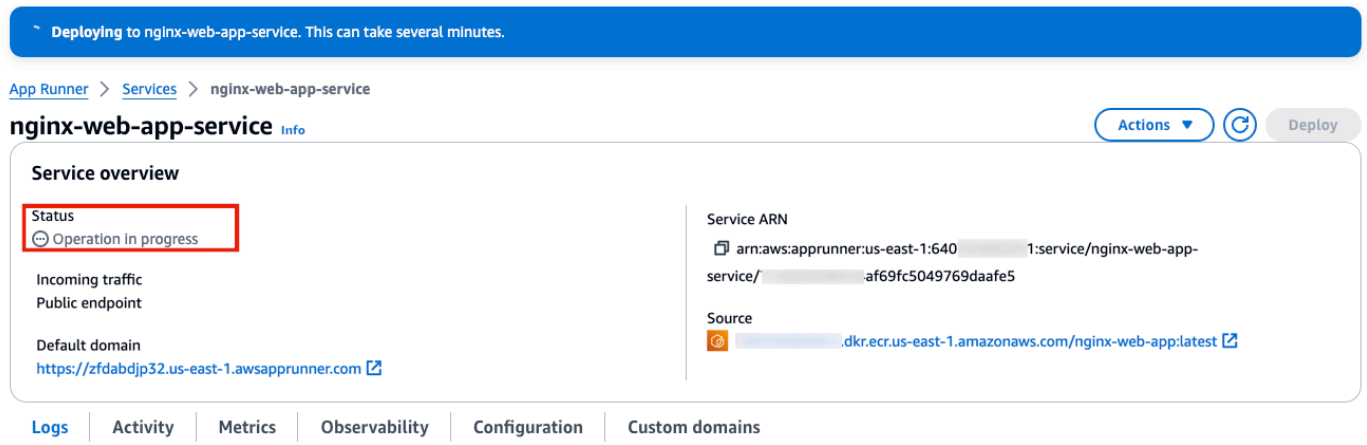
▶ **Observability**

▶ **Tags**

[Cancel](#)[Previous](#)[Create & deploy](#)

7. Monitor deployment

It will take several minutes for the service to be deployed. You can **view** the event logs for progress.



Deploying to nginx-web-app-service. This can take several minutes.

App Runner > Services > nginx-web-app-service

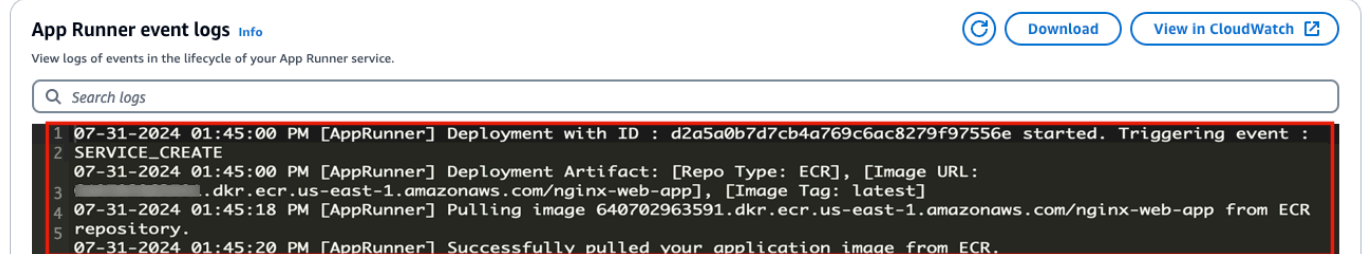
nginx-web-app-service Info

Actions ⌂ Deploy

Service overview

Status ⌂ Operation in progress	Service ARN arn:aws:apprunner:us-east-1:640...:service/nginx-web-app-service/af69fc5049769daafe5
Incoming traffic	Source ...dkr.ecr.us-east-1.amazonaws.com/nginx-web-app:latest
Public endpoint	
Default domain https://zfdabdjp32.us-east-1.awsapprunner.com	

Logs | Activity | Metrics | Observability | Configuration | Custom domains



App Runner event logs Info

View logs of events in the lifecycle of your App Runner service.

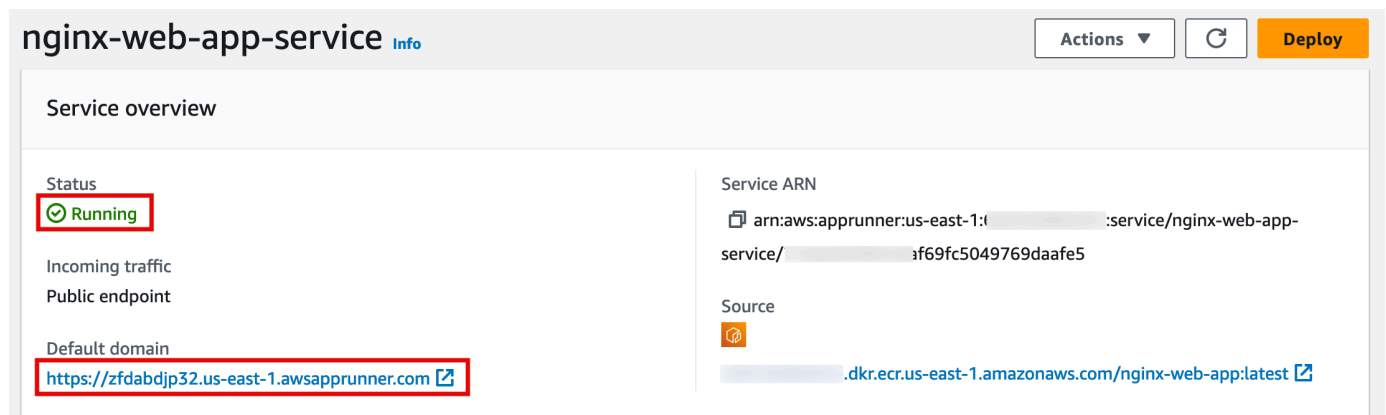
Download View in CloudWatch

Search logs

```
1 07-31-2024 01:45:00 PM [AppRunner] Deployment with ID : d2a5a0b7d7cb4a769c6ac8279f97556e started. Triggering event :
2 SERVICE_CREATE
3 07-31-2024 01:45:00 PM [AppRunner] Deployment Artifact: [Repo Type: ECR], [Image URL:
4 ...dkr.ecr.us-east-1.amazonaws.com/nginx-web-app], [Image Tag: latest]
5 07-31-2024 01:45:18 PM [AppRunner] Pulling image 640702963591.dkr.ecr.us-east-1.amazonaws.com/nginx-web-app from ECR
6 repository.
7 07-31-2024 01:45:20 PM [AppRunner] Successfully pulled your application image from ECR.
```

8. Access the application

Once the status updates to **Running**, choose the default domain name URL to view the web app.



nginx-web-app-service Info

Actions ⌂ Deploy

Service overview

Status ✔ Running	Service ARN arn:aws:apprunner:us-east-1:640...:service/nginx-web-app-service/af69fc5049769daafe5
Incoming traffic	Source ...dkr.ecr.us-east-1.amazonaws.com/nginx-web-app:latest
Public endpoint	
Default domain https://zfdabdjp32.us-east-1.awsapprunner.com	

9. Verify the deployment

The Welcome page and confirmation message should look like the image on the right.



Welcome to AWS App Runner!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

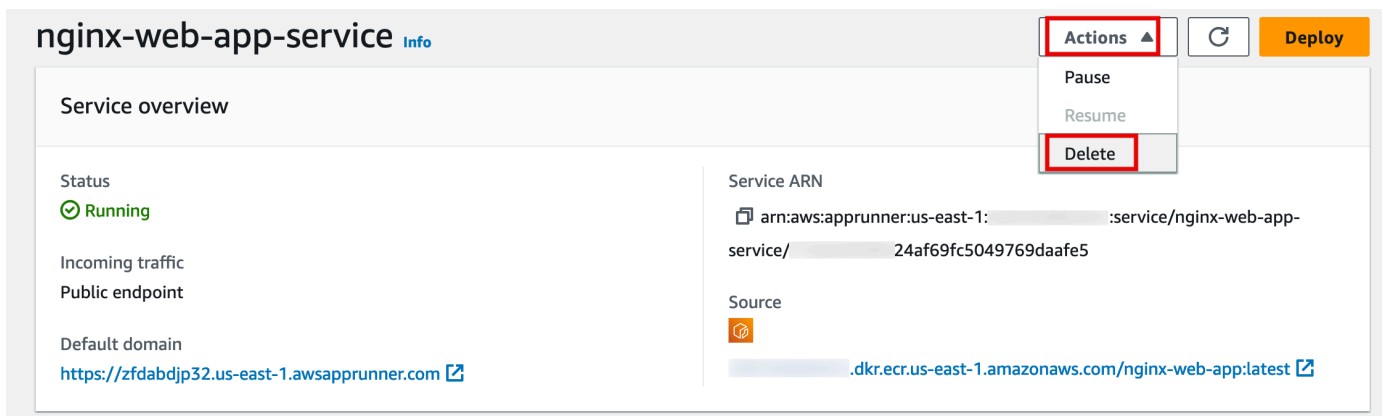
Thank you for using AWS App Runner!

Clean up resources

In this step, you will go through the steps to delete all the resources you created throughout this tutorial. It is a best practice to delete resources you are no longer using to avoid unwanted charges.

1. Delete the App Runner service

In the AWS App Runner console, navigate to the **nginx-web-app-service**, choose **Actions**, and select **Delete**.



2. Confirm the deletion

Follow the prompts in the pop-up window to **confirm** deletion of the service.

Delete nginx-web-app-service



Deleting this App Runner service will also delete all associations with resources (e.g. custom domains).

To confirm deletion, type *delete* in the field.

Cancel

Delete

3. Delete the ECR repository

In the Amazon ECR console, select the radio button next to the **nginx-web-app repository**, and choose **Delete**.

Private repositories

Repositories (2)

< 1 >

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryp
<input checked="" type="radio"/> nginx-web-app	east-1.amazonaws.com/nginx-web-app	July 31, 2024, 13:39:42 (UTC-04)	Disabled	Manual	AES

Congratulations

You have containerized a sample web app running on a Nginx server and pushed the image to Amazon Elastic Container Registry. Then, you created an AWS App Runner service using the image.