



App * Runner

AWS App Runner provides a highly abstracted and simple managed experience for running web applications and API hosting services.

Application Properties

Web Applications & API servers:

Applications that serve HTTP based requests.

Multi-concurrent

The application is long-running.

A single instance of the application may serve many requests during its lifetime.

Multiple requests maybe handled simultaneously.

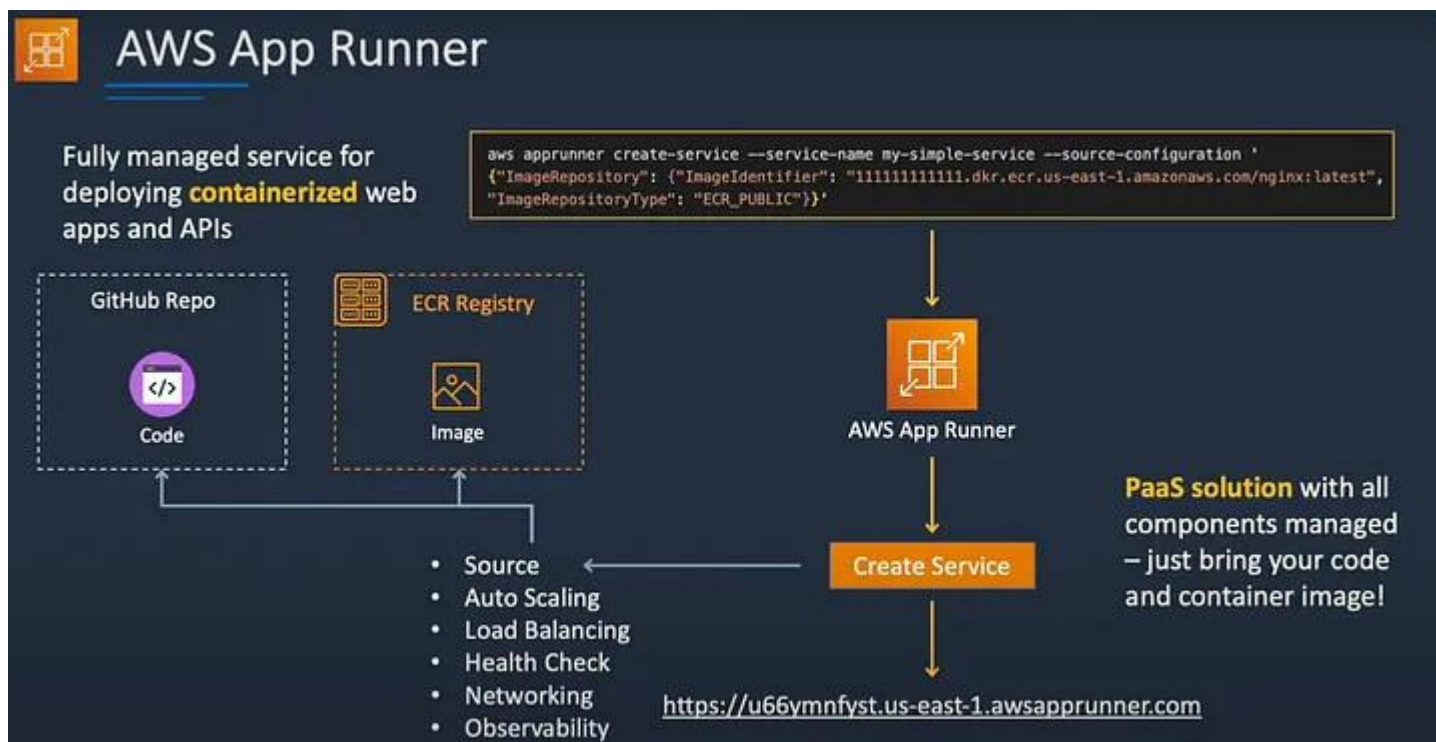
Stateless:

Requests are processed independently and do not depend on local state.

State maybe stored external to the application instance (e.g.: a DynamoDB table)

No background processing:

Any processing outside the context of a request must be limited.

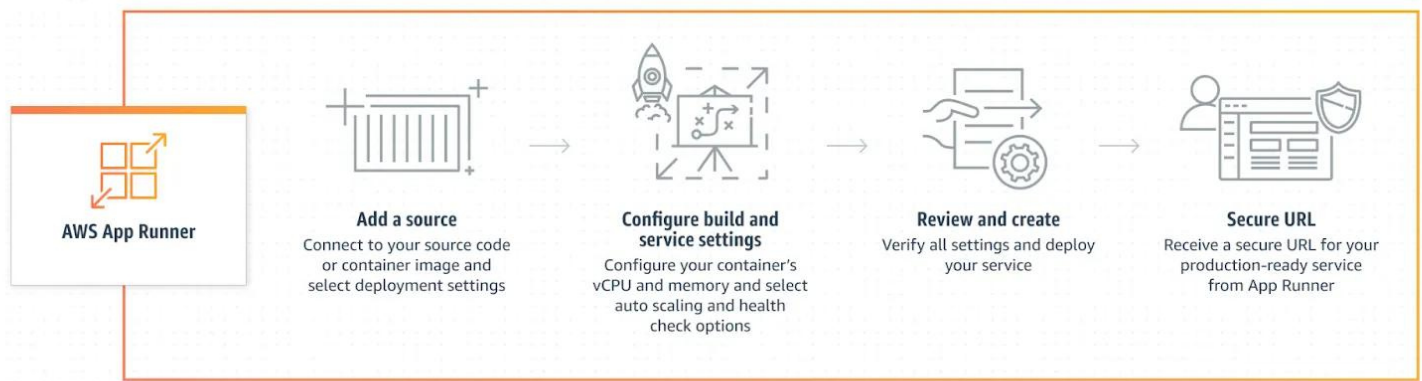


Why AWS App Runner ?

In traditional CI/CD, The below are the major components required if you want to set up a web application

1. Pushing the changes in Repository (GitHub, Code Commit,. etc.)
2. Code Pipeline
3. Auto Build
4. Deploy
5. Load Balancing
6. Auto Scaling Groups, Target Groups
7. Security Groups
8. Domains, Certificate Setup
9. Monitoring the Application Health
10. Logging

But using the AWS App Runner, we can simplify the process and we no need to do all the above traditional steps

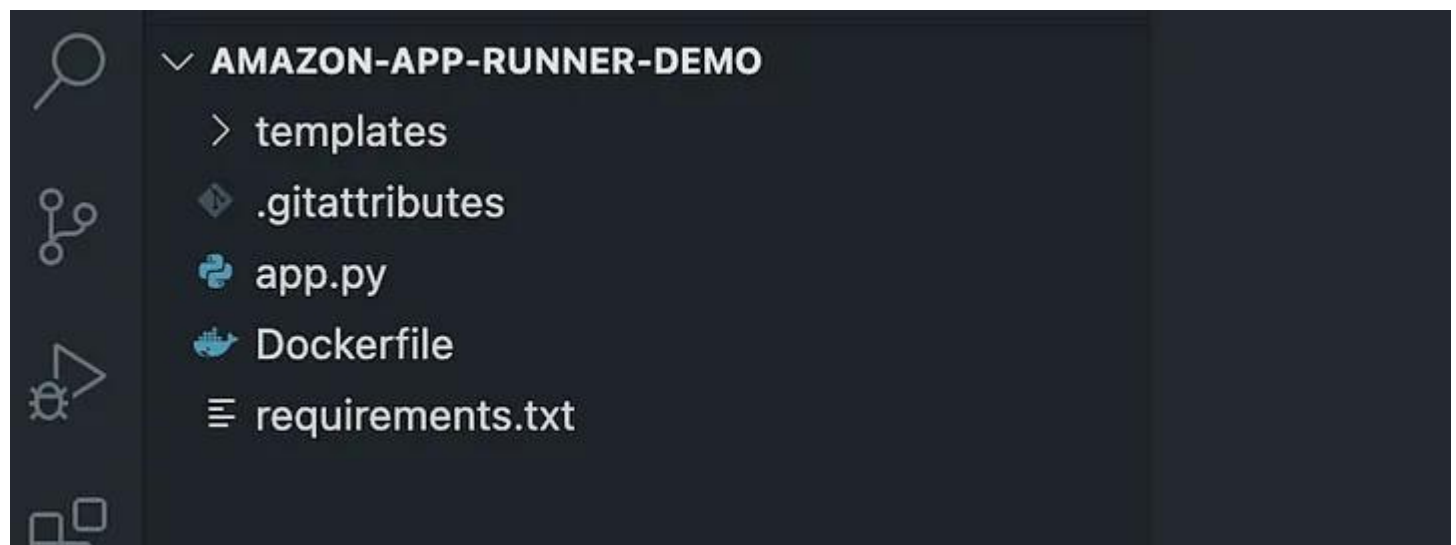


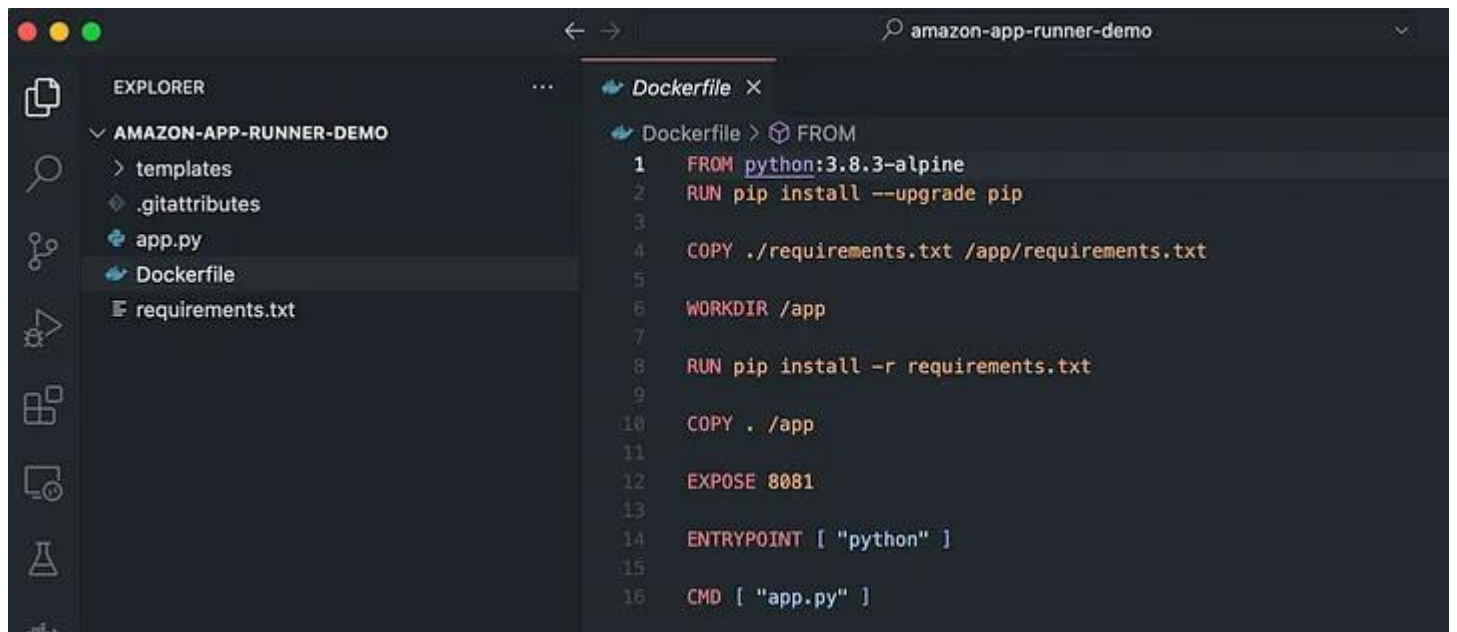
AWS App Runner is a fully managed container deployment service to build and run secure web applications at scale, without prior container or infrastructure experience.

- The below steps will help you to Use the AWS App Runner to deploy and manage containerized application easily

Prerequisite :

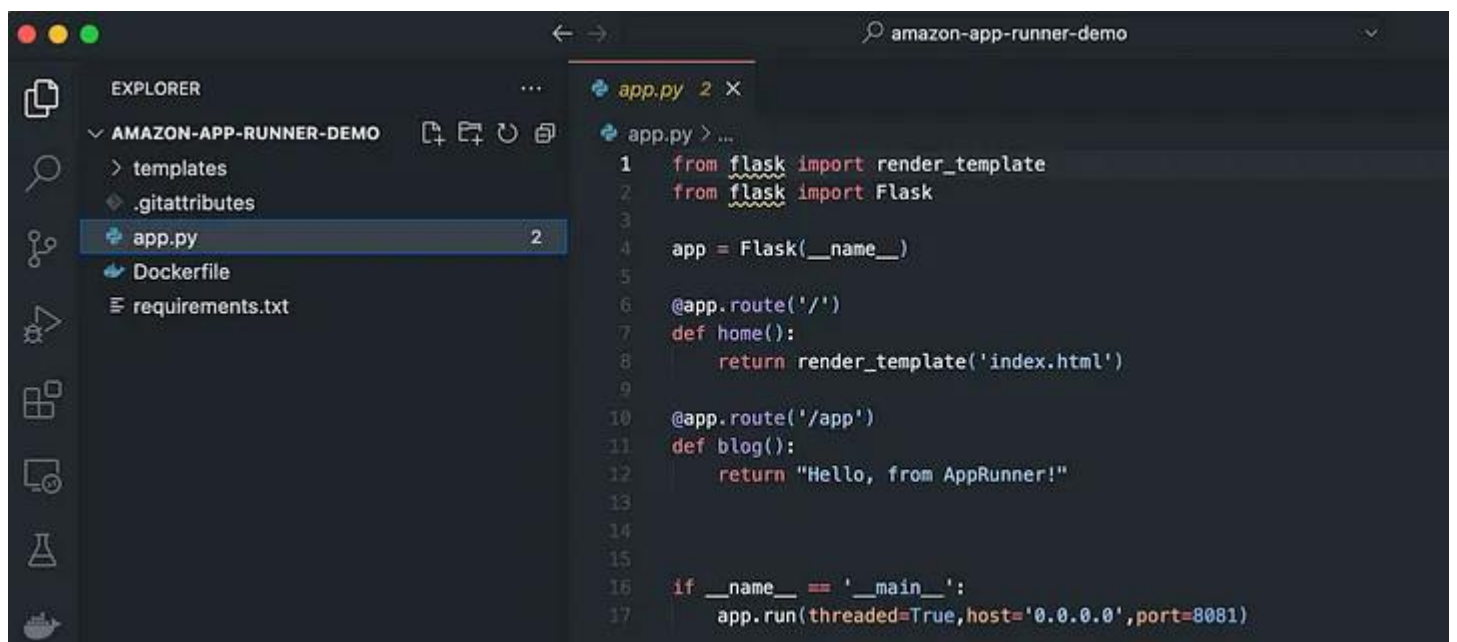
Build an Image with sample flask application or any





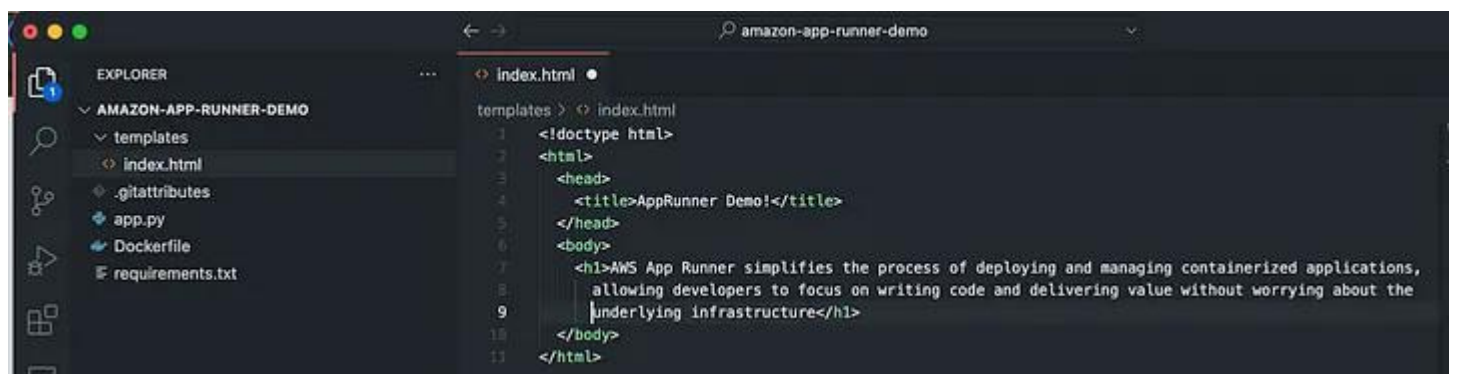
The screenshot shows the VS Code interface with the Explorer on the left and the Dockerfile editor on the right. The Explorer shows the project structure for 'AMAZON-APP-RUNNER-DEMO' with files: templates, .gitattributes, app.py, Dockerfile, and requirements.txt. The Dockerfile editor shows the following content:

```
Dockerfile > FROM
1 FROM python:3.8.3-alpine
2 RUN pip install --upgrade pip
3
4 COPY ./requirements.txt /app/requirements.txt
5
6 WORKDIR /app
7
8 RUN pip install -r requirements.txt
9
10 COPY . /app
11
12 EXPOSE 8081
13
14 ENTRYPOINT [ "python" ]
15
16 CMD [ "app.py" ]
```



The screenshot shows the VS Code interface with the Explorer on the left and the app.py editor on the right. The Explorer shows the project structure for 'AMAZON-APP-RUNNER-DEMO' with files: templates, .gitattributes, app.py, Dockerfile, and requirements.txt. The app.py editor shows the following content:

```
app.py > ...
1 from flask import render_template
2 from flask import Flask
3
4 app = Flask(__name__)
5
6 @app.route('/')
7 def home():
8     return render_template('index.html')
9
10 @app.route('/app')
11 def blog():
12     return "Hello, from AppRunner!"
13
14
15
16 if __name__ == '__main__':
17     app.run(threaded=True, host='0.0.0.0', port=8081)
```

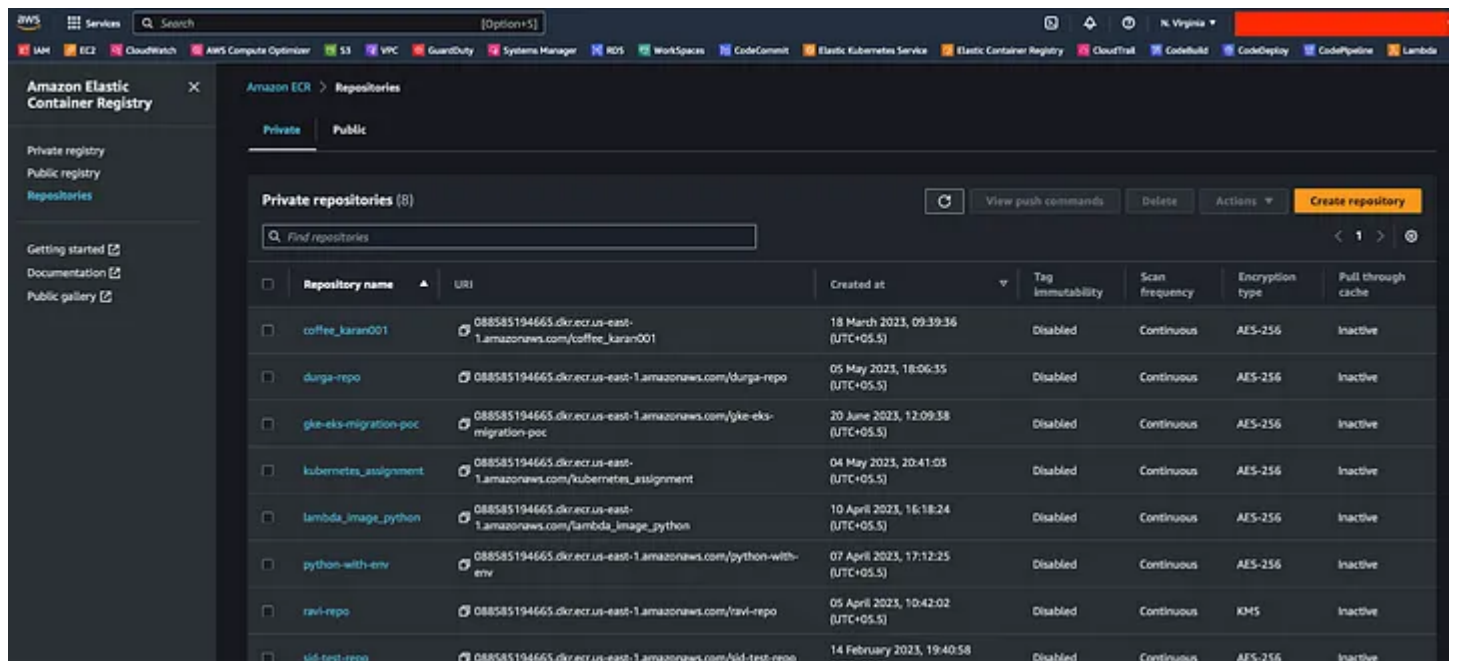


The screenshot shows the VS Code interface with the Explorer on the left and the index.html editor on the right. The Explorer shows the project structure for 'AMAZON-APP-RUNNER-DEMO' with files: templates, .gitattributes, app.py, Dockerfile, and requirements.txt. The index.html editor shows the following content:

```
templates > index.html
1 <!doctype html>
2 <html>
3 <head>
4 <title>AppRunner Demo!</title>
5 </head>
6 <body>
7 <h1>AWS App Runner simplifies the process of deploying and managing containerized applications,
8   allowing developers to focus on writing code and delivering value without worrying about the
9   underlying infrastructure</h1>
10 </body>
11 </html>
```

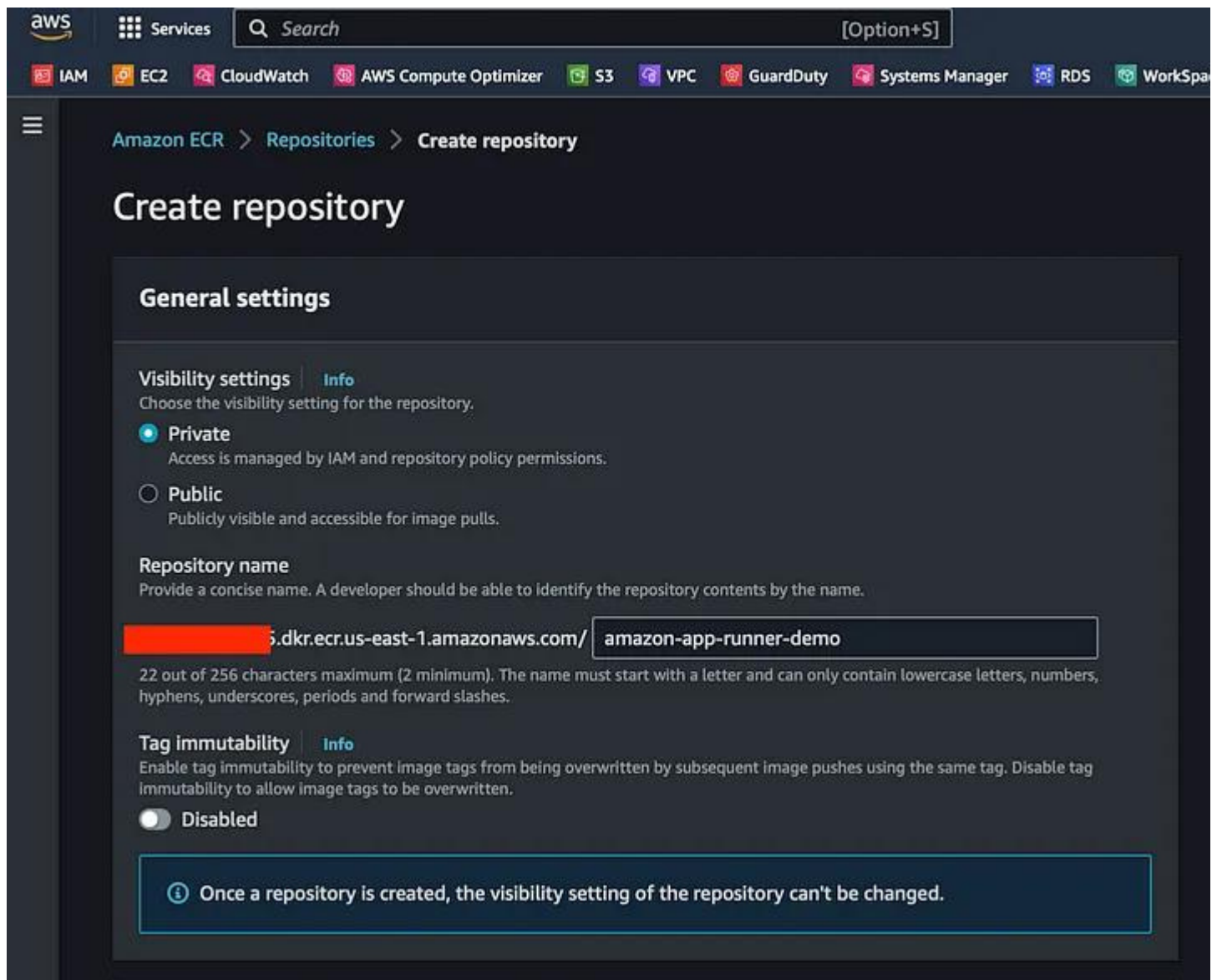
Create a ECR repository

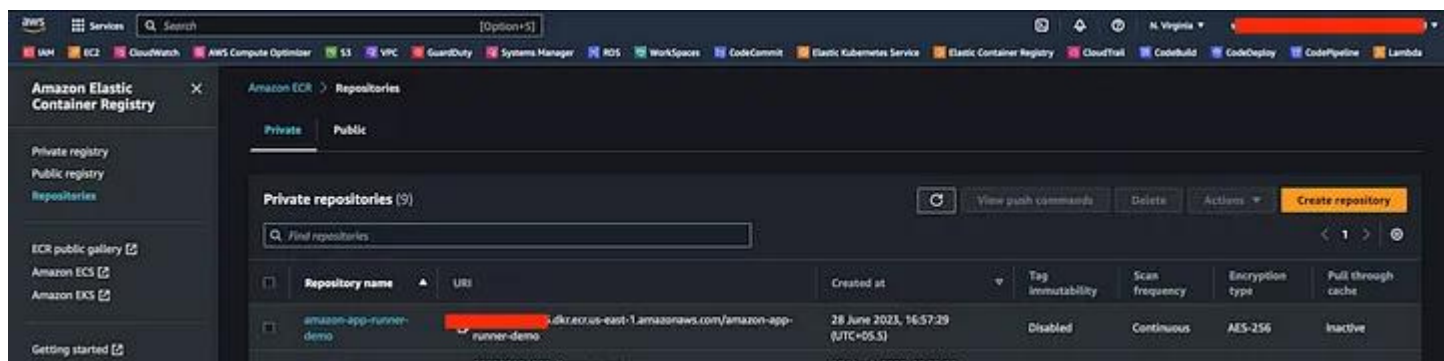
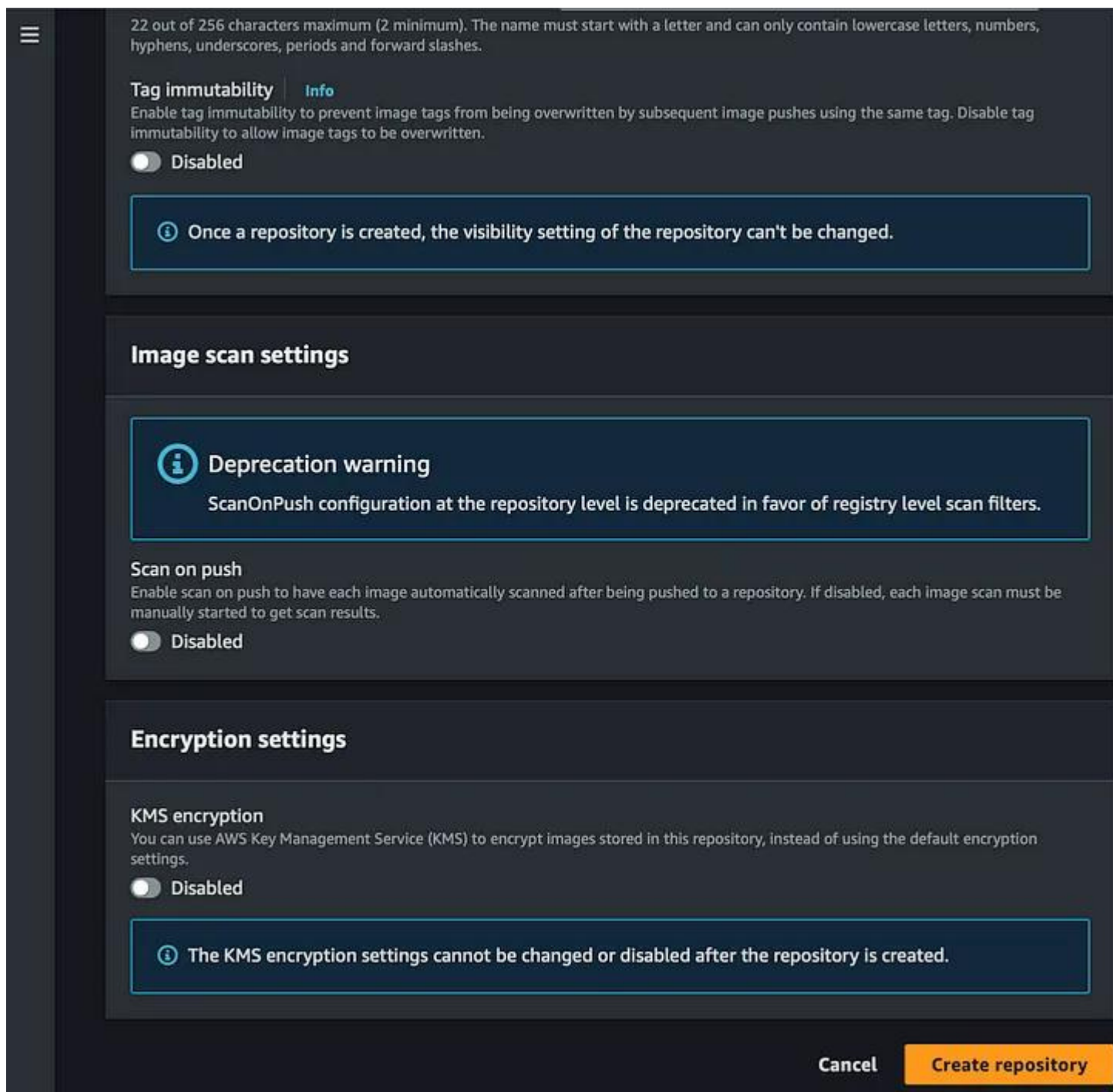
Login to AWS Console, Go to Amazon ECR

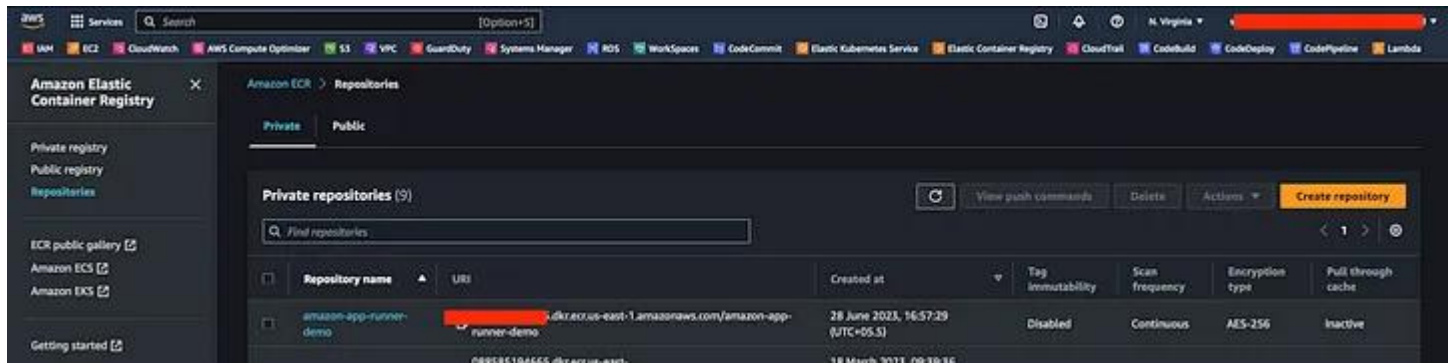


Click on Create Repository

Provide repository name: amazon-app-runner-demo





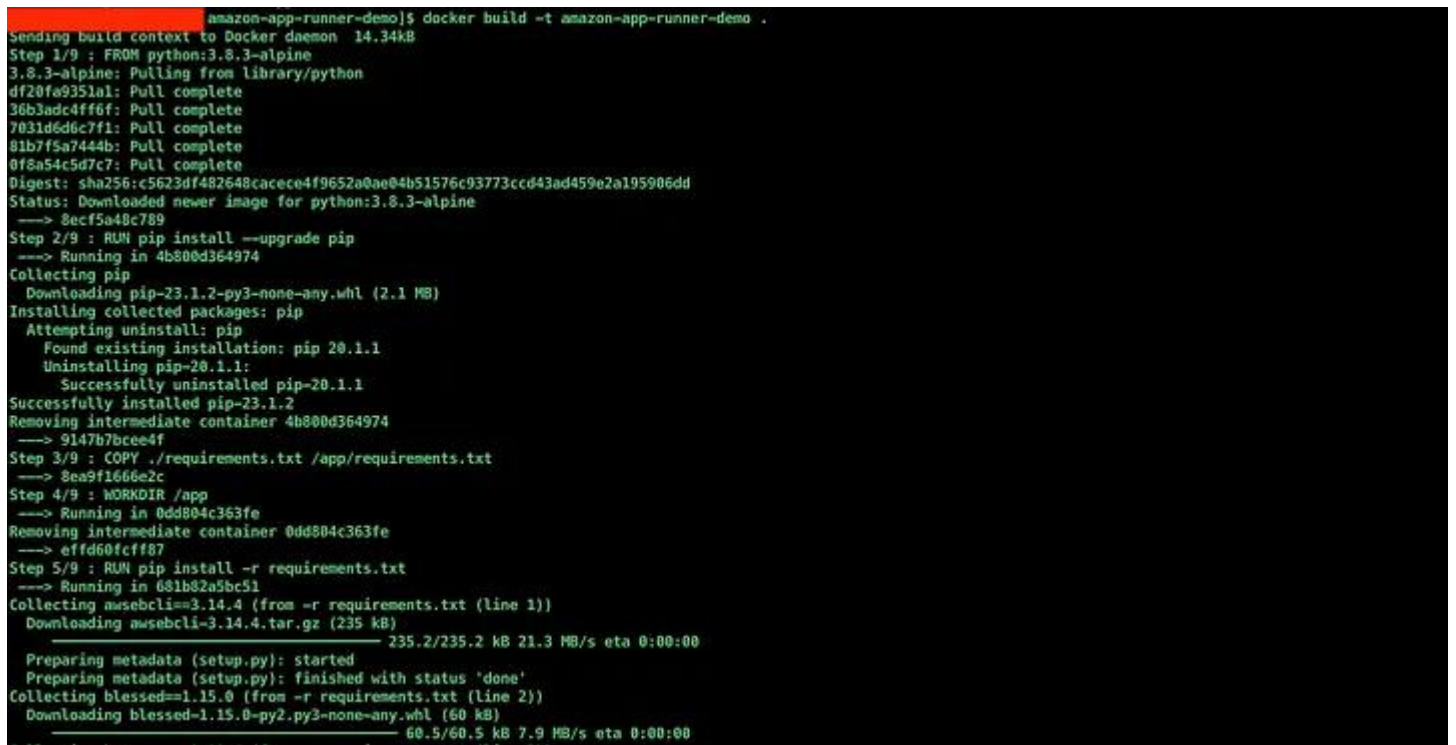


Build Docker Image and Push to ECR Repository

Build the docker image in local environment with the above Dockerfile and the sample application code

Build docker image

docker build -t amazon-app-runner-demo:latest .



Push the docker image to ECR

use the below commands to push the docker image from local to the ECR repository which we have created

Login ECR Repository

```
aws.com]$ aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin dkr.ecr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

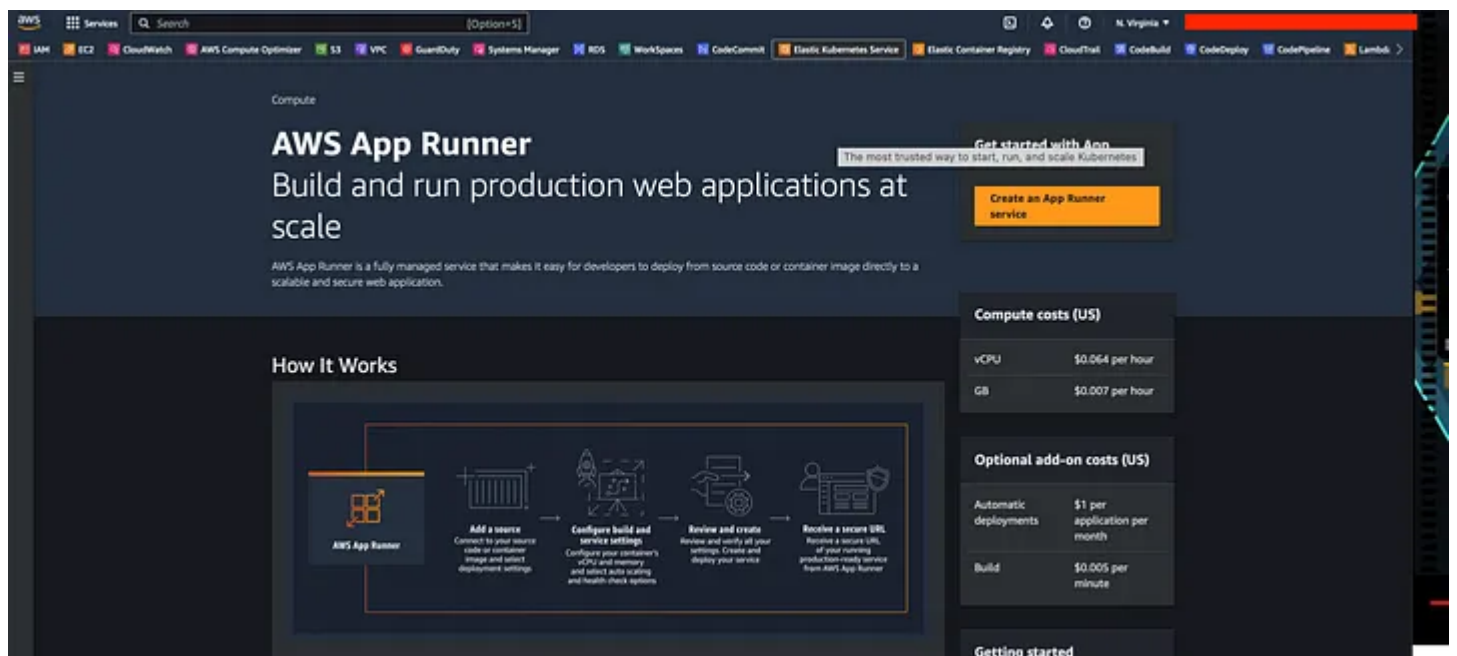
Login Succeeded
[ec2-user@ip-10-0-0-20 amazon-app-runner-demo]$
```

```
docker tag amazon-app-runner-demo:latest xxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/amazon-app-runner-demo:latest
```

```
docker push xxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/amazon-app-  
runner-demo:latest
```

The screenshot shows the Amazon ECR console interface. The top navigation bar includes the AWS logo, account information, and a list of services. The left sidebar contains navigation links for IAM, EC2, CloudWatch, AWS Compute Optimizer, S3, VPC, GuardDuty, Systems Manager, RDS, WorkSpaces, CodeCommit, Elastic Kubernetes Service, Elastic Container Registry, CloudTrail, CodeBuild, CodeDeploy, CodePipeline, and Lambda. The main content area displays the 'amazon-app-runner-demo' repository. The 'Images' tab is active, showing a table of images. The table has columns for Image tag, Artifact type, Pushed at, Size (MB), Image URI, Digest, and Vulnerabilities. One image is listed with the tag 'latest', pushed on June 23, 2023, at 17:27:20 UTC+05:51, with a size of 53.97 MB. The image URI is '763103331754.dkr.ecr.us-east-1.amazonaws.com/amazon-app-runner-demo:latest' and the digest is 'sha256:763103331754.dkr.ecr.us-east-1.amazonaws.com/amazon-app-runner-demo:latest'. A 'See findings' link is present for vulnerabilities.

Go to APP Runner and Create an App Runner Service



Step 1: Source and deployment

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

Repository type:

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.

1. Container Registry: EX ECR

Deploy your service using a container image stored in a container registry.

2. Source code repository

Deploy your service using the code hosted in a source repository.

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.

Connect to GitHub [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.

Add new

Repository

Branch

In this section, select “Repository type” and container registry and browse the “ECR Repository (Container Image URI)” which we have created in the previous step

App Runner > Create service

Step 1
Source and deployment

Step 2
Configure service

Step 3
Review and create

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.

Deployment settings:

Deployment trigger

1. Manual: Start each deployment yourself using the App Runner console or AWS CLI.

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

Existing service roles

Choose an IAM role in your account. Only trusted roles are listed.

2. Automatic: App Runner monitors your registry and deploys a new version of your service for each image push

Deployment settings

Deployment trigger

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

☒ **Automatic**
Every push to this branch deploys a new version of your service.

In this section, select manual deployment trigger, For ECR Access Role select “Create new service role”

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](#)

Click on “Next”

Step 2: Configure service

Provide Service Name: “amazon-app-runner-demo-svc”

Virtual CPU: “specify CPU, EX: 1 vCPU”

Virtual Memory: specify memory, EX: 3GB

Environment variables — optional

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.

IAM policy templates:

Copy the following IAM policy template to your Instance role to enable permissions to reference secrets and configurations from AWS Secrets Manager or SSM Parameter Store.

Port: 8081

Your service uses this TCP port.

Additional configuration:

Start command – optional

The application’s container runs this command on launch. Leave blank to use the entry point command defined in the container image.

The screenshot shows the 'Configure service' step in the AWS IAM console. On the left, a sidebar indicates the progress: 'Step 2: Configure service' is active, 'Step 1: Create role' is completed, and 'Step 3: Review and create' is next. The main area is titled 'Configure service' and contains several sections: 'Service settings' with a text input for 'Service name' (containing 'amazon-app-runner-demo-svc'), dropdowns for 'Virtual CPU' (1 vCPU) and 'Virtual memory' (2 GB); 'Environment variables — optional' with a link to 'Info', explanatory text about referencing secrets and configurations, and a message 'No environment variables have been configured.' with an 'Add environment variable' button; 'IAM policy templates' with a right-pointing arrow; 'Port' with a text input (8081) and a note 'Your service uses this TCP port.'; and 'Additional configuration' with a right-pointing arrow.

Auto scaling configuration

Default configuration: Use the App Runner default configuration

▼ Auto scaling [Info](#)
Configure automatic scaling behavior.

Auto scaling configuration

☒ **Default configuration**
Use the App Runner default configuration.

☐ **Custom configuration**
Use your own configuration.

Concurrency
100 requests

Minimum size
1 instance(s)

Maximum size
25 instances

Custom configuration: Use your own configuration.

▼ Auto scaling [Info](#)
Configure automatic scaling behavior.

Auto scaling configuration

☐ **Default configuration**
Use the App Runner default configuration.

☒ **Custom configuration**
Use your own configuration.

Existing configurations

▼ [Add new](#)

Health Check

Configure load balancer health checks

▼ 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

Security:

Specify an Instance role and an AWS KMS encryption key

▼ Security Info

Specify an Instance role and an AWS KMS encryption key

Permissions

Select an IAM role with permissions to AWS actions that your service code calls. To create a custom role, use the [IAM console](#)

Instance role

An Instance role is auto-generated for every IAM role that is created for Amazon EC2 using the AWS Management Console. Choose an Instance role to apply the required IAM role to your application code. This grants access permissions to call AWS services.

Choose an Instance role

▼

AWS KMS key

This key is used to encrypt the stored copies of your data.

☒ Use an AWS-owned key

A key that AWS owns and manages for you.

☐ Choose a different AWS KMS key

A key that you own or have permission to use.

Web Application Firewall Info

Activate WAF to define Web access control list (ACL) to protect against web exploits and bots. Learn more about [WAF and pricing](#).

☐ Activate

Networking:

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

Incoming network traffic

Select if your service is accessible publicly over the internet or only within a Virtual Private Cloud (VPC)

Public endpoint: Configure to make your service accessible to any client via the public internet.

Outgoing network traffic

Select if the outgoing traffic is routed to only public internet or customize to access private VPC from Amazon Virtual Private Cloud (Amazon VPC)

Public access:

Your service can send outgoing messages only to public network

endpoints.

▼ **Networking** Info

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

Incoming network traffic

Select if your service is accessible publicly over the internet or only within a Virtual Private Cloud (VPC)

☒ **Public endpoint**
Configure to make your service accessible to any client via the public internet.

☐ **Private endpoint**
Configure a private connection between your VPC and App Runner using a VPC interface endpoint powered by AWS PrivateLink.

Outgoing network traffic

Select if the outgoing traffic is routed to only public internet or customize to access private VPC from Amazon Virtual Private Cloud (Amazon VPC)

☒ **Public access**
Your service can send outgoing messages only to public network endpoints.

☐ **Custom VPC**
Your service connects to an Amazon VPC of your choice. Your service can send outgoing messages to any endpoint (private or public) that the VPC can access.

Incoming network traffic

Private endpoint: Configure a private connection between your VPC and App Runner using a VPC interface endpoint powered by AWS PrivateLink.

Outgoing network traffic

Custom VPC Your service connects to an Amazon VPC of your choice. Your service can send outgoing messages to any endpoint (private or public) that the VPC can access.

Incoming network traffic
Select if your service is accessible publicly over the internet or only within a Virtual Private Cloud (VPC)

☐ **Public endpoint**
Configure to make your service accessible to any client via the public internet.

☒ **Private endpoint**
Configure a private connection between your VPC and App Runner using a VPC interface endpoint powered by AWS PrivateLink.

Connect VPC using VPC interface endpoint
VPC interface endpoint is an AWS PrivateLink resource that enables you to access any App Runner service within an Amazon VPC. Choose an existing or create a new VPC interface endpoint. If you create a new endpoint, [additional pricing may apply](#).

Choose a VPC interface endpoint (0) [Info](#) ↻ Create new endpoint

Select one VPC interface endpoint. One endpoint can be used for multiple services.

< 1 >

Name	VPC	Security groups	Subnets
<p>No VPC interface endpoints</p> <p>You currently have no VPC interface endpoints. You must create one for private endpoint.</p> <p>Create new endpoint</p>			

Outgoing network traffic
Select if the outgoing traffic is routed to only public internet or customize to access private VPC from Amazon Virtual Private Cloud (Amazon VPC)

☐ **Public access**
Your service can send outgoing messages only to public network endpoints.

☒ **Custom VPC**
Your service connects to an Amazon VPC of your choice. Your service can send outgoing messages to any endpoint (private or public) that the VPC can access.

VPC connector

Add new

In this section, select the default but based on your requirement you can choose incoming traffic and outgoing network traffic as private

Observability:

Tracing with AWS X-Ray

X-Ray tracing for your application. For pricing information, see [AWS X-Ray pricing](#).

▼ **Observability**
Configure observability tooling.

Configure observability tooling

☒ **Tracing with AWS X-Ray**
X-Ray tracing for your application. For pricing information, see [AWS X-Ray pricing](#).

Tags

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

Tags — optional

▼ Tags Info

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

Tags — optional

A tag is a key-value pair that you assign to an AWS resource.

Name

Q name

×

Value - optional

Q amazon-app-runner-demo

×

Remove

Step 3: Review and create

Review all the values specified in the Source and deployment, Configure service steps. edit and make changes if required

Review and create Info

Step 1: Source and deployment

Edit

Source

Container registry

Amazon ECR

Container image URI

.dkr.ecr.us-east-

1.amazonaws.com/amazon-app-runner-demo:latest

Provider

ECR

Deployment settings

Deployment method

Manual deployments

ECR access role

arn:aws:iam:

5

:role/service-role/AppRunnerECRAccessRole

Step 2: Configure service

[Edit](#)

Service settings

Service name

amazon-app-runner-demo-svc

Virtual CPU & memory

1 vCPU & 2 GB

Port

8081

Environment variables

Name	Value
No environment variables have been configured.	

► Additional configuration

▼ Auto scaling

Concurrency

100

Maximum size

25

Minimum size

1

▼ Observability

Observability
Off

▼ Tags

Name	Value
name	amazon-app-runner-demo
created-by	subbu

Cancel

Previous

Create & deploy

Click on “Create & Deploy”

Deploying to amazon-app-runner-demo-svc. This can take several minutes.

App Runner > Services > amazon-app-runner-demo-svc

amazon-app-runner-demo-svc

Actions

Refresh

Deploy

Service overview

Status
Operation in progress

Incoming traffic
Public endpoint

Default domain
<https://tzm2bb4rm4.us-east-1.awsapprunner.com>

Service ARN
[arn:aws:apprunner:us-east-1:123456789012:service/amazon-app-runner-demo-svc/ff196063b355141b485](#)

Source
[123456789012.ecr.us-east-1.amazonaws.com/amazon-app-runner-demo:latest](#)

Logs

Activity

Metrics

Observability

Configuration

Custom domains

App Runner event logs

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

Refresh

Download

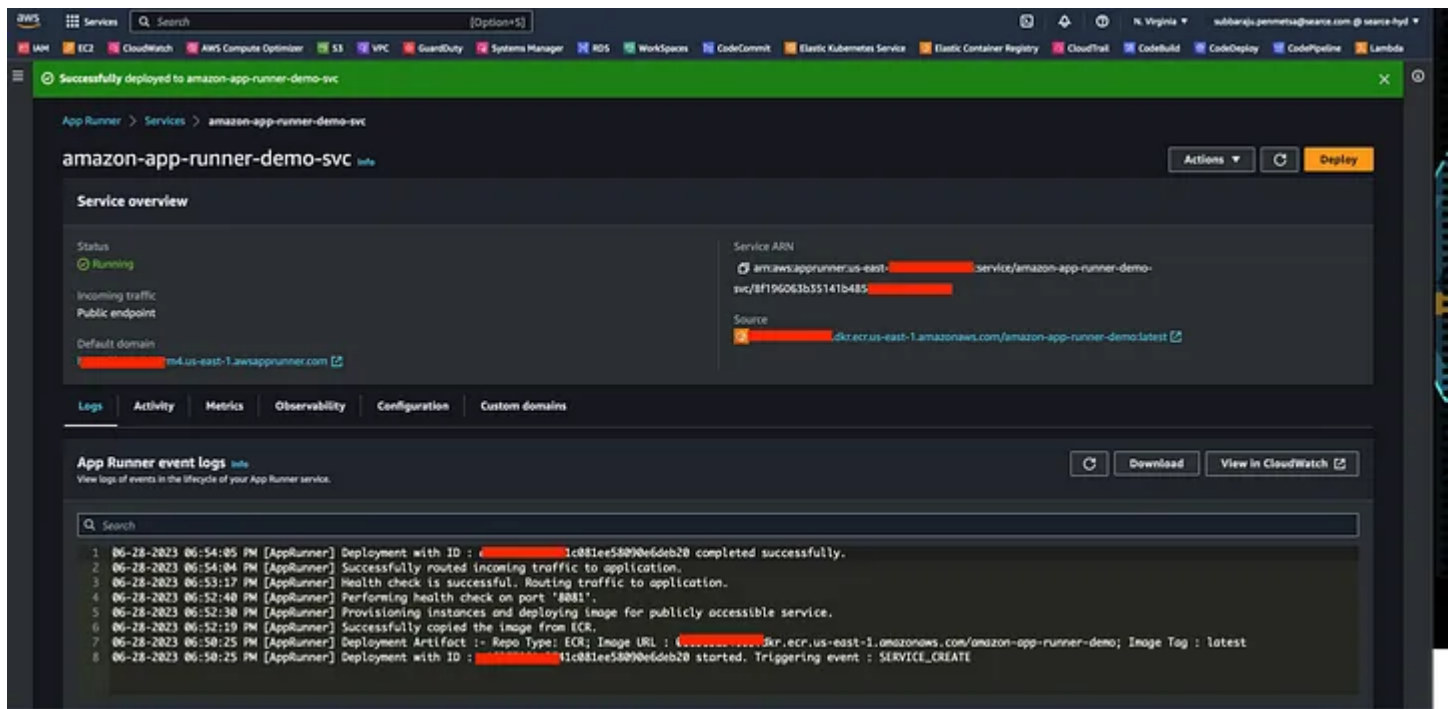
View in CloudWatch

Search

1 No system events to display.

It will take few minutes to create a service for your sample application

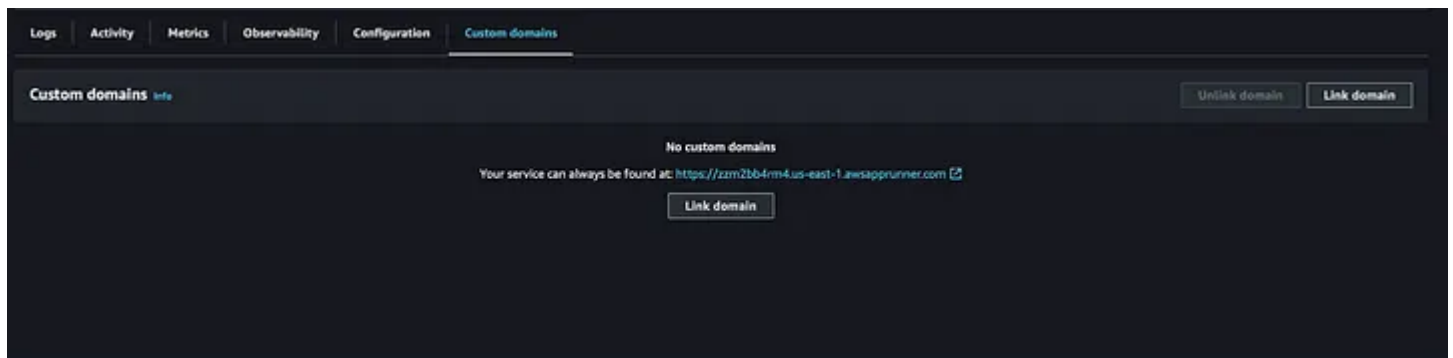
Go to “Logs” section and you can observe the creation process



Once the status as Running the you can access the application with default domain

Sample application is deployed success fully and able to see the login page with the default domain name generated by App Runner

You can attach this default domain to the route 53 custom domains from the custom domains section



Advantages of App Runner:

1. AWS App Runner builds and deploys containerized web applications automatically, balances traffic with encryption, scales to meet your traffic needs, and allows for the configuration of how services are accessed and communicated with other AWS applications in a private Amazon VPC.
2. **Developers** to focus on writing code and delivering value without worrying about the underlying infrastructure

3. **Easy deployment:** App Runner simplifies the deployment process by automatically detecting the language and framework used in your application and configuring the build and deployment settings accordingly
4. **Automatic scaling:** App Runner automatically scales your application based on incoming traffic and workload. It can handle traffic spikes and adjust the capacity to match the demand, ensuring that your application remains available and responsive
5. **Integrated load balancing:** The service includes built-in load balancing capabilities to distribute traffic across multiple instances of your application. It helps improve performance and ensures high availability by distributing the workload evenly.
6. **Security and compliance:** App Runner integrates with AWS Identity and Access Management (IAM) to manage user access and control permissions. It also supports integration with AWS Secrets Manager for securely storing and accessing sensitive information, such as database credentials.
7. **Monitoring and logging:** App Runner provides integration with AWS CloudWatch, allowing you to monitor your application's performance, set alarms, and collect logs for analysis and troubleshooting.
8. **Seamless integration with other AWS services:** App Runner integrates with other AWS services, such as Amazon RDS for managed databases, Amazon ElastiCache for in-memory caching, and Amazon S3 for object storage. This allows you to leverage the full capabilities of the AWS ecosystem.

AWS App Runner Pricing

Provisioned container instances: \$0.07/ GB-hour*

when the application is deployed, you pay for the memory provisioned in each container instance. Keeping your container instance's memory provisioned when your application is idle ensures it can deliver consistently low milli seconds.

Active Container Instances: \$0.064 / vCPU-hour*, \$0.007 / GB-hour*

When your application is processing requests, you switch from provisioned container instances to active container instances that consume both memory and compute resources. You pay for the compute and any additional memory consumed in excess of the memory allocated by your provisioned container instances. App Runner automatically scales the number of active container instances up and down to meet the processing requirements of your application. You can set a maximum limit on the number of active container instances your application uses so that costs do not exceed your budget. When your active container instances are idle, App Runner scales back to your provisioned container instances (the default is 1 provisioned container instance).

All container instance processing is billed per second, rounded up to the next nearest second. There is a one minute minimum charge for vCPU resources every time a provisioned container instance starts processing requests.

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

