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.

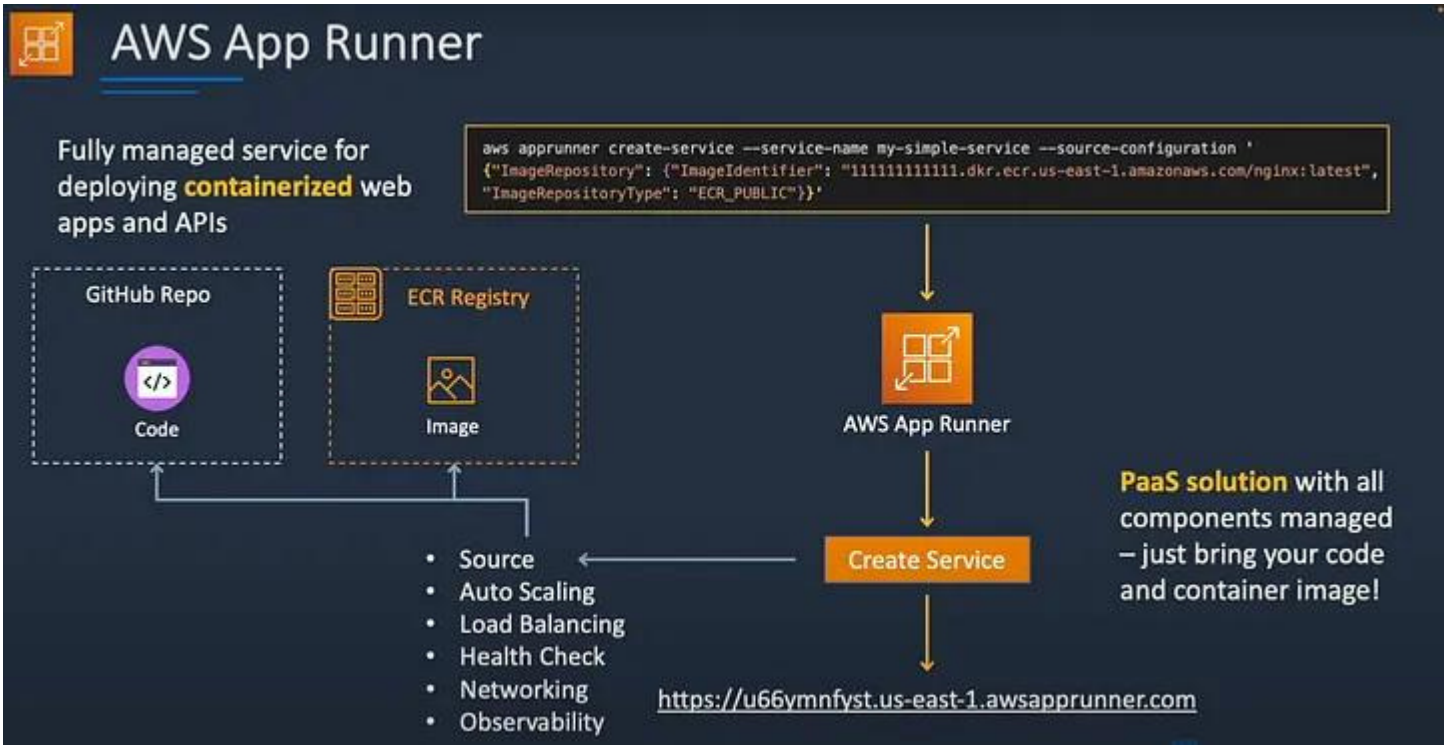
Stateless:

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.

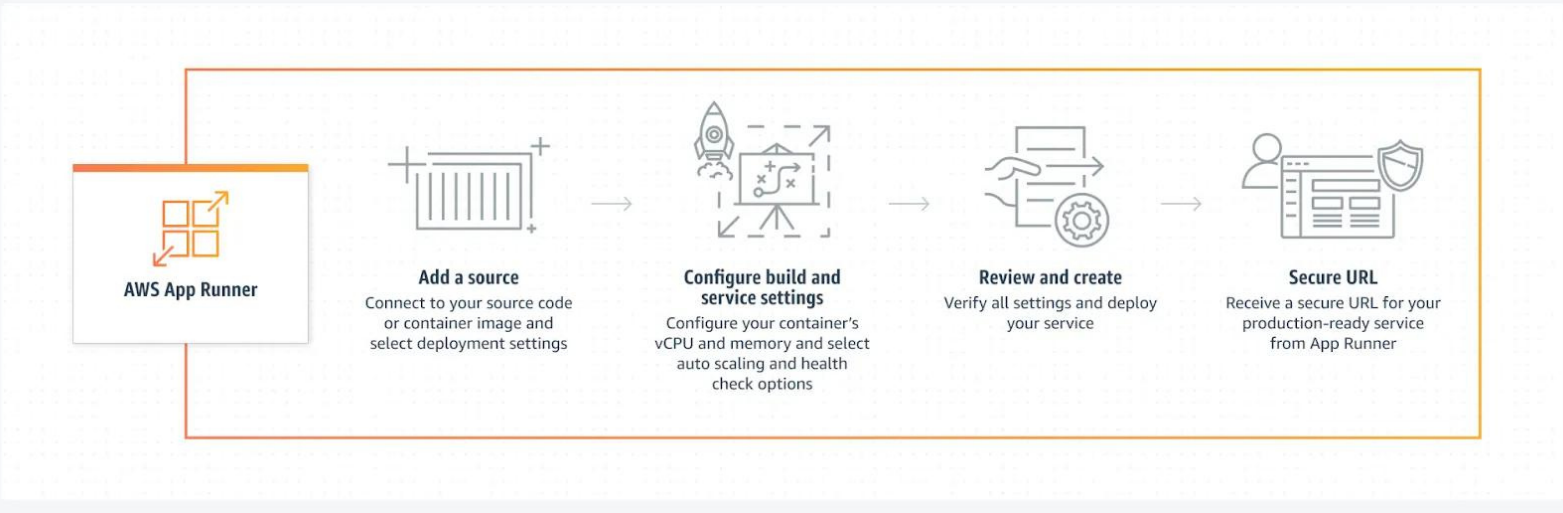


Why AWS App Runner ?

In traditional CICD, 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

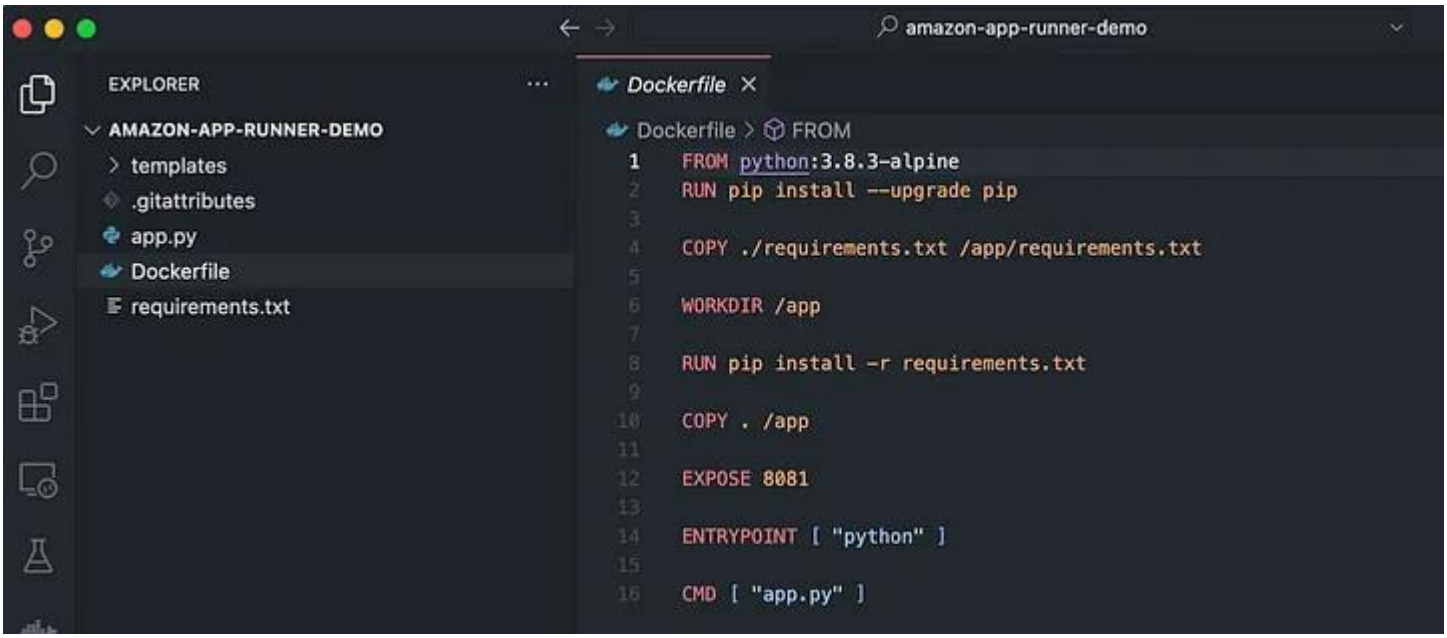


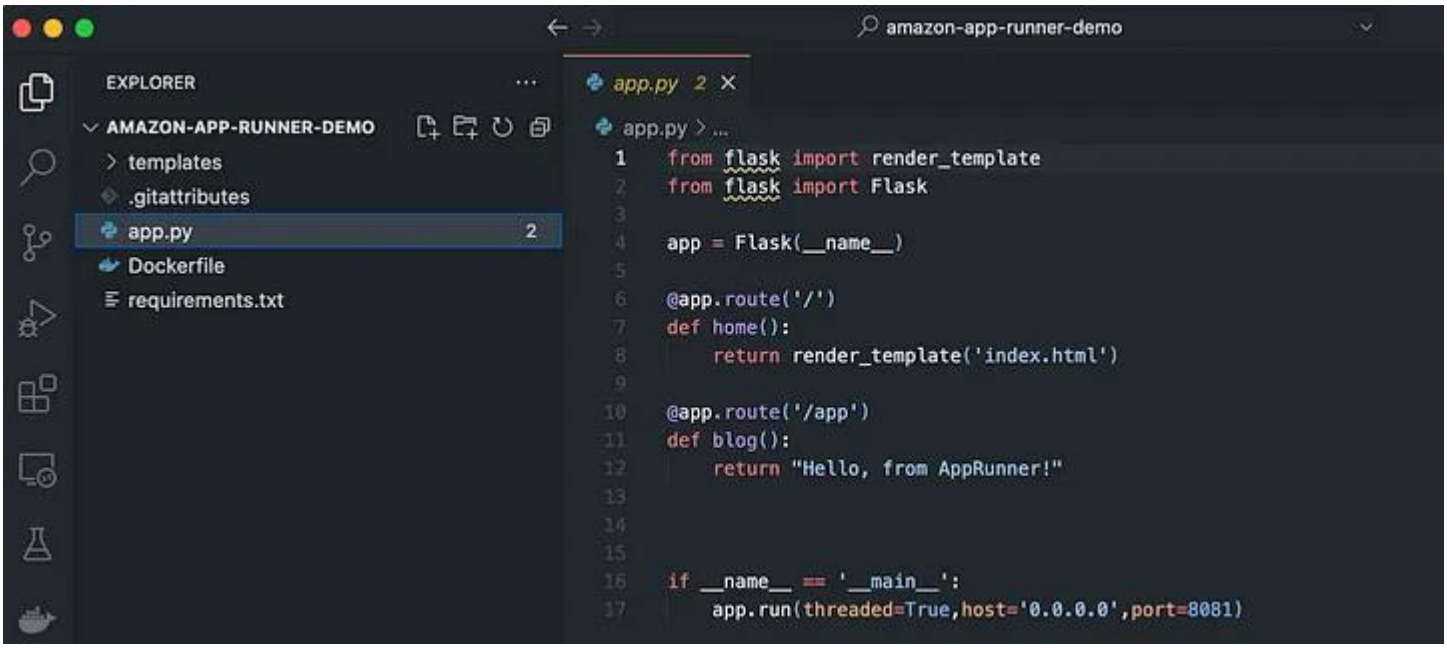
AWS App Runner is a fully managed container deployment service to build and run secure web applications at scale, without prior container or infrastructure experience.

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

Prerequisite :

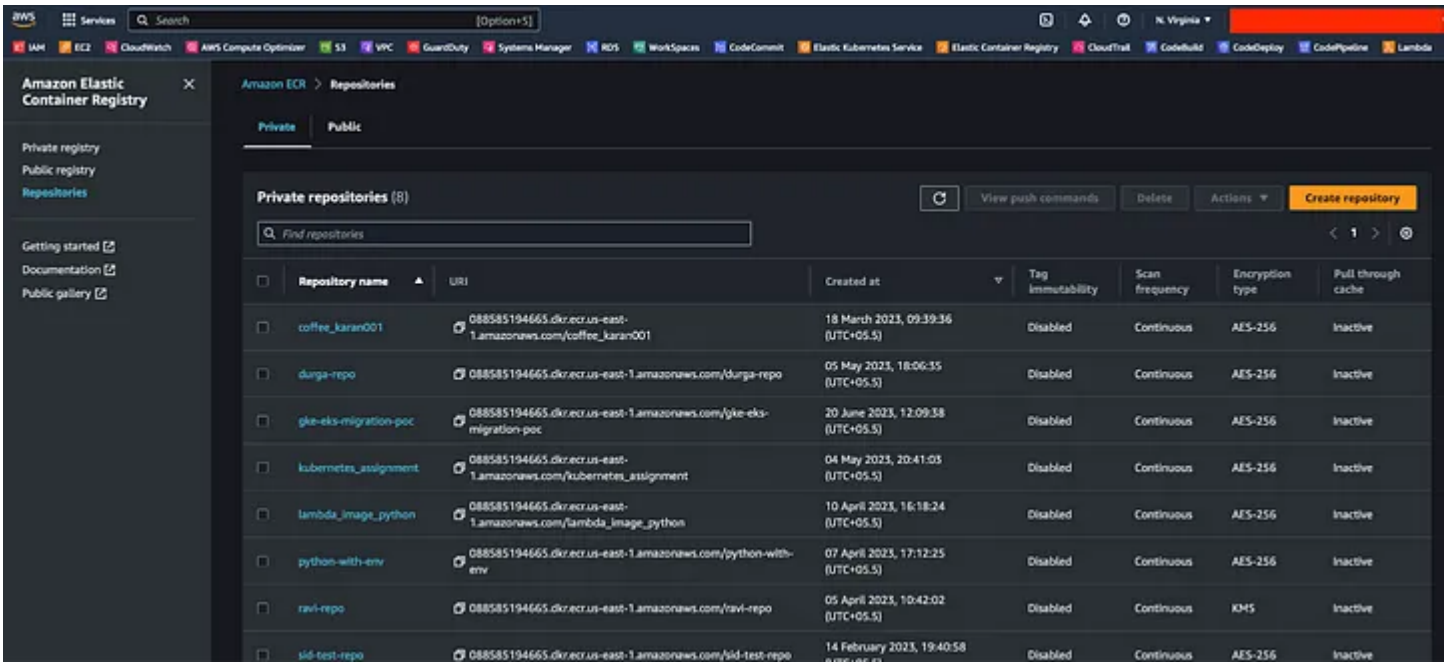
Build an Image with sample flask application or any





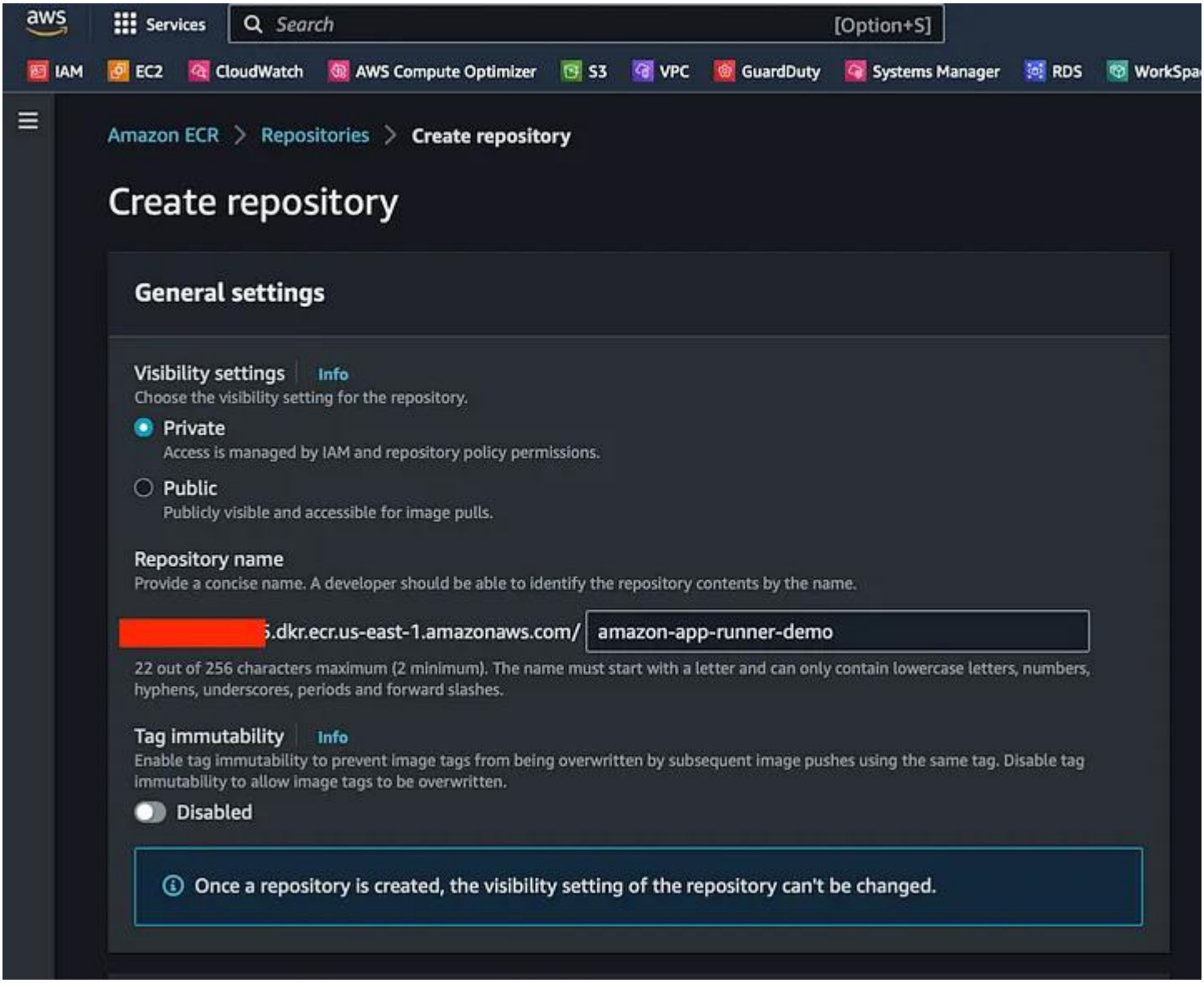
Create a ECR repository

Login to AWS Console, Go to **Amazon ECR**



Click on **Create Repository**

Provide repository name: amazon-app-runner-demo



22 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

Once a repository is created, the visibility setting of the repository can't be changed.

Image scan settings

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

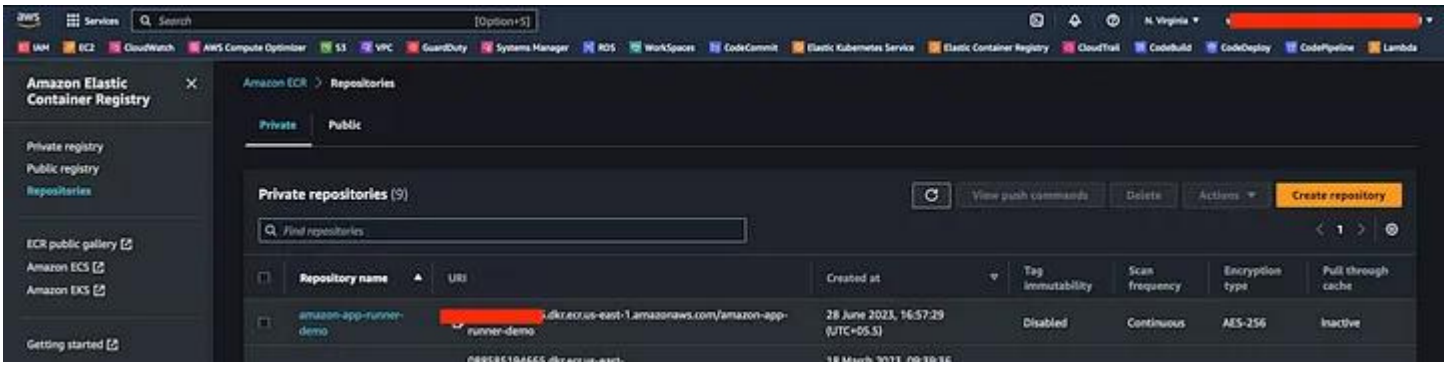
KMS encryption

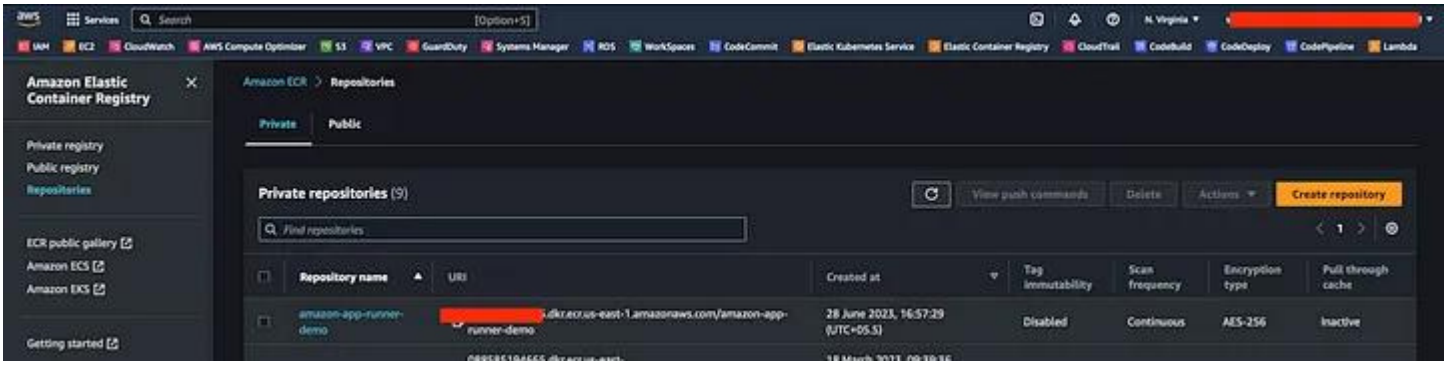
You can use AWS Key Management Service (KMS) to encrypt images stored in this repository, instead of using the default encryption settings.

Disabled

The KMS encryption settings cannot be changed or disabled after the repository is created.

Cancel | Create repository



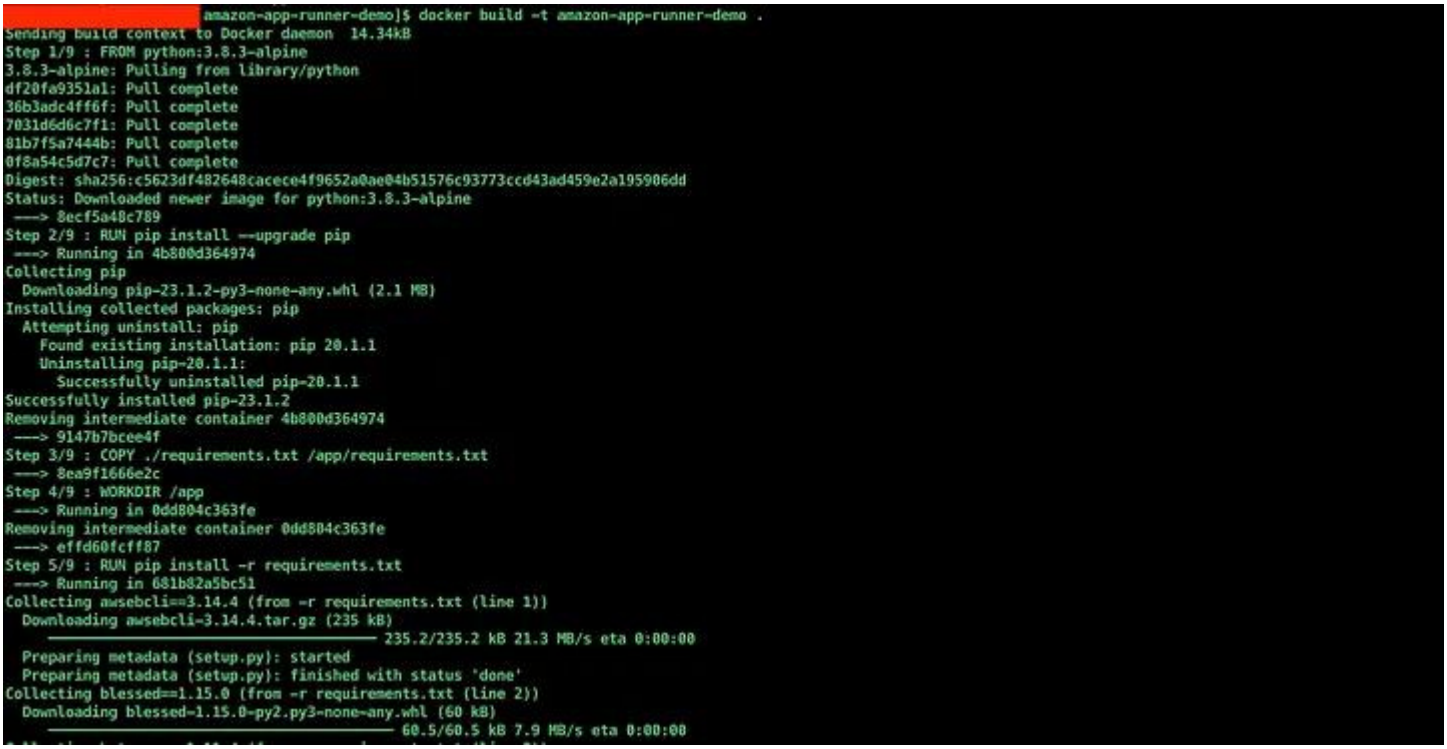


Build Docker Image and Push to ECR Repository

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 .

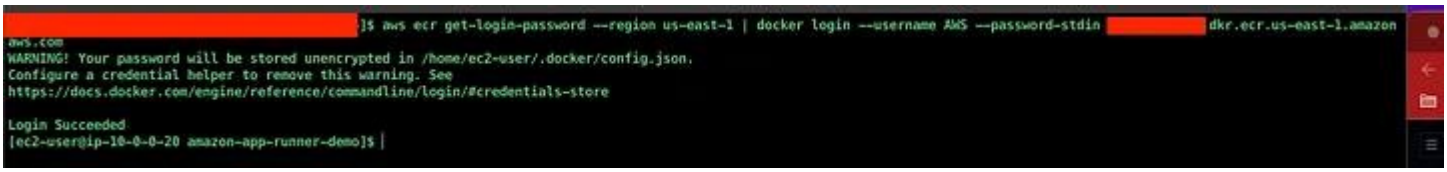


Push the docker image to ECR

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

Login ECR Repository

aws ecr get-login-password — region us-east-1 | docker login — username AWS — password-stdin xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com



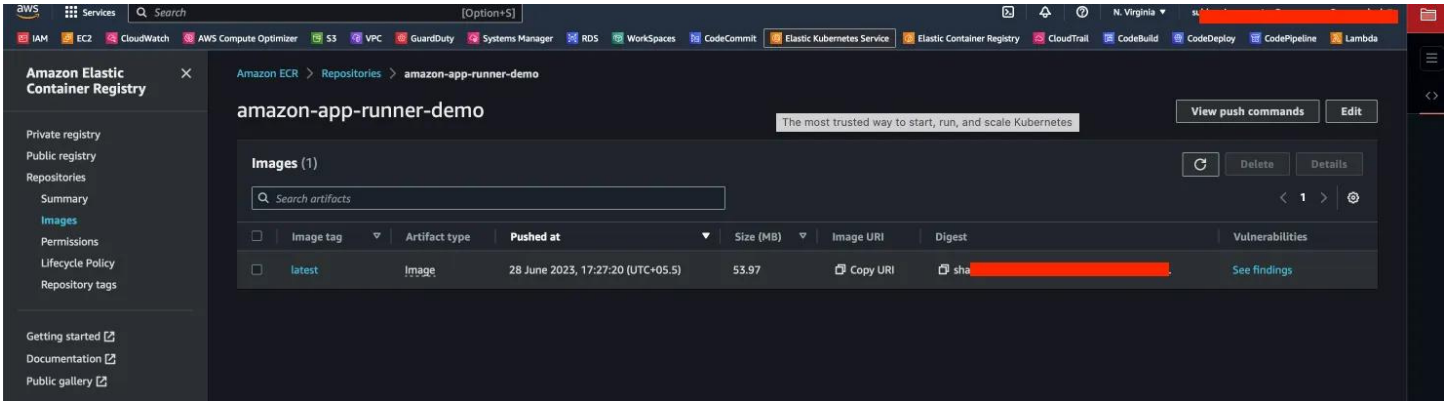
Tagging the image

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

Push this image to your newly created AWS repository:

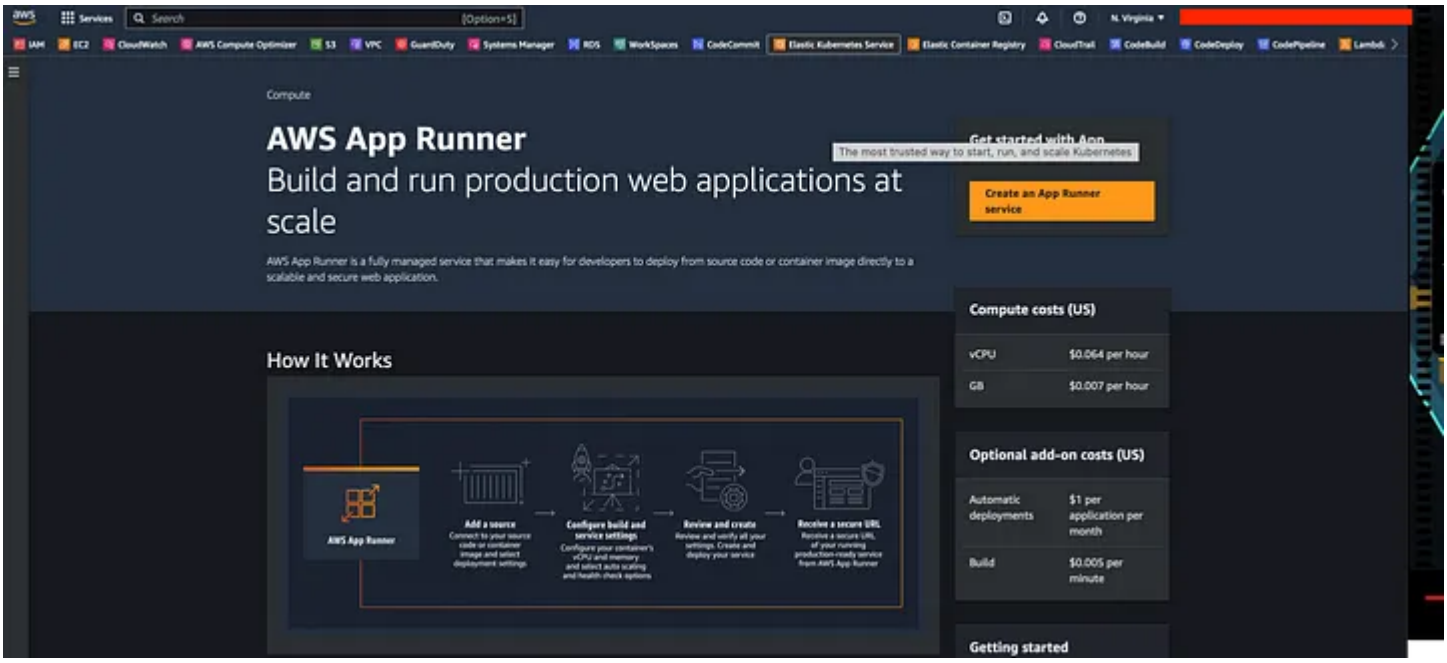
docker push xxxxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com/amazon-app-runner-demo:latest

Image has been pushed to ECR repository successfully



Deploy Application image Using App Runner

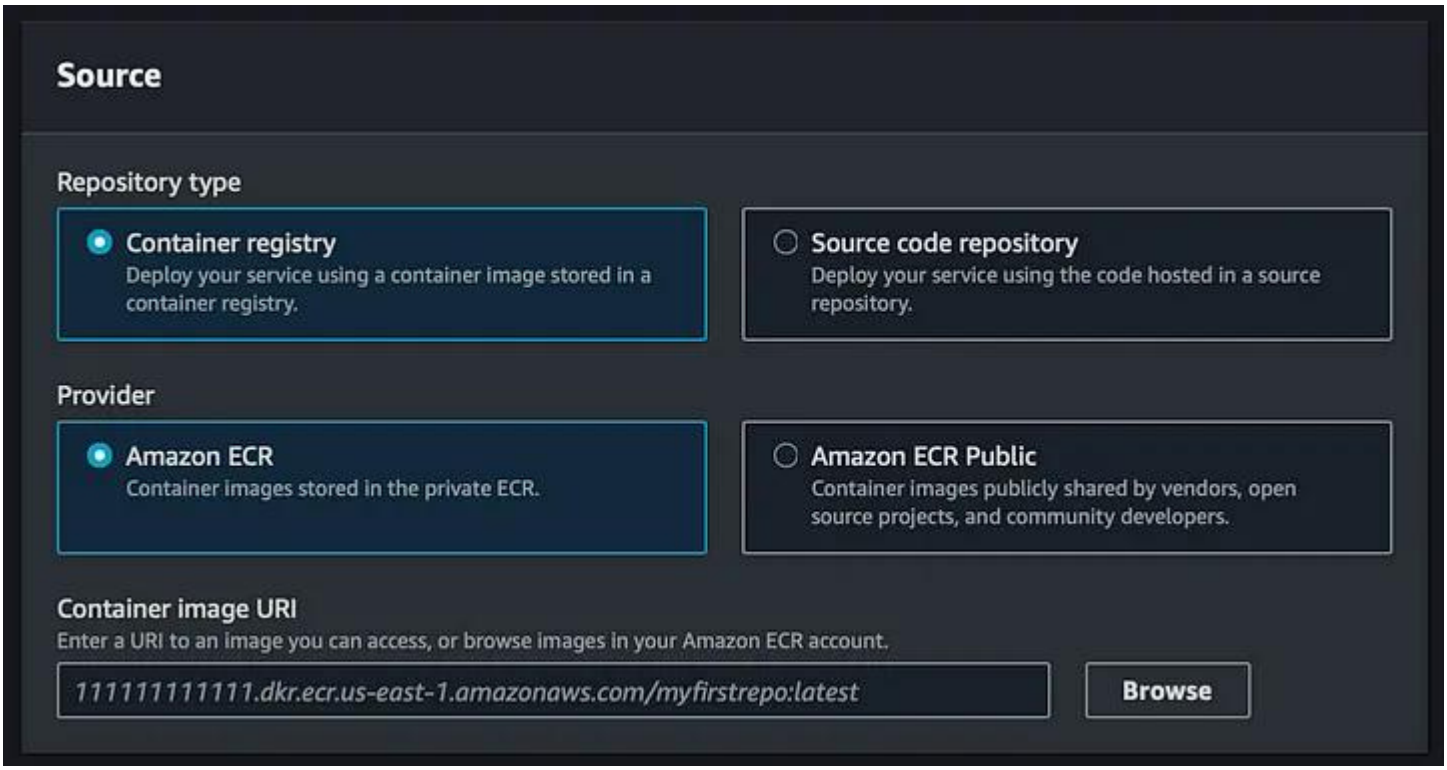
Go to APP Runner and Create an App Runner Service



Step 1: Source and deployment

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

Repository type:

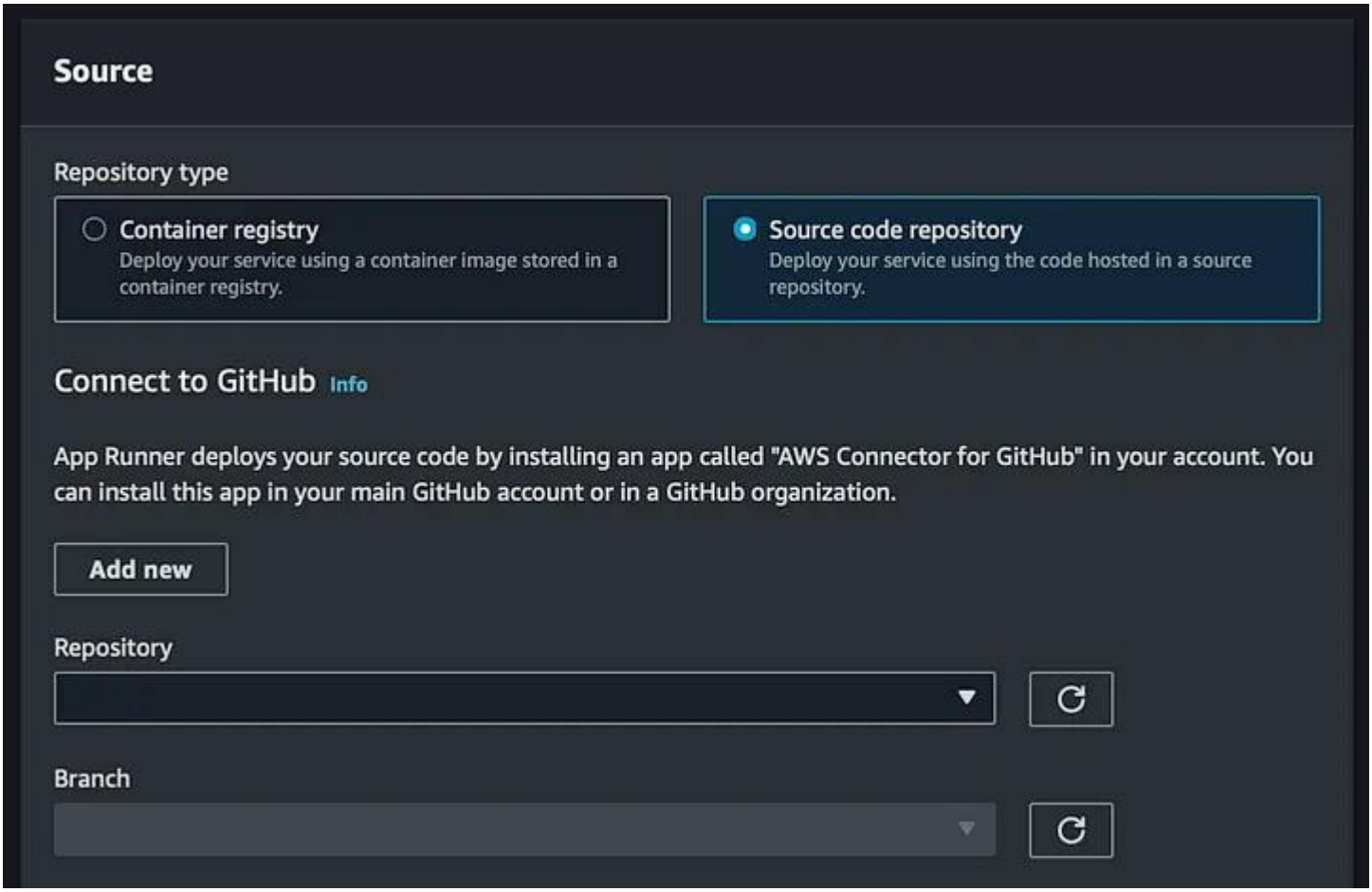


1. **Container Registry: EX ECR**

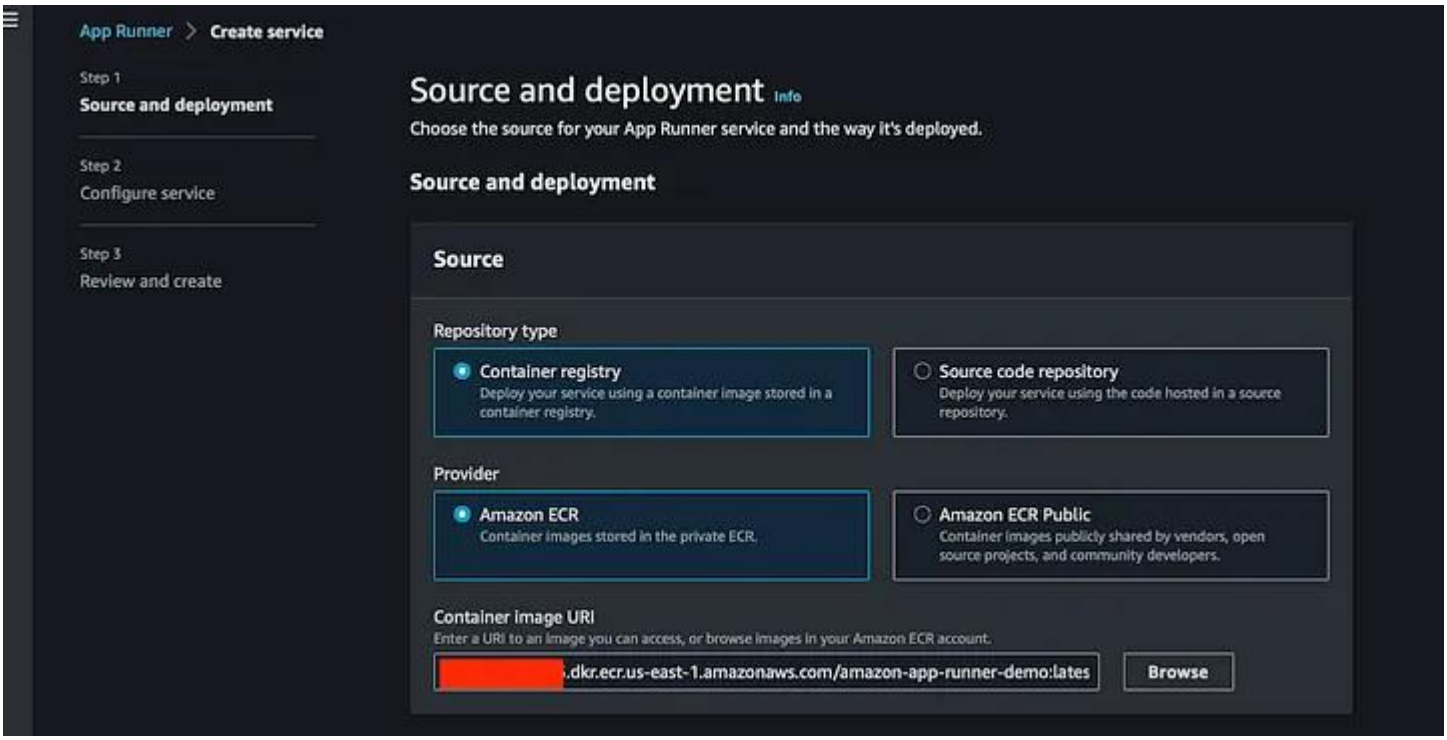
Deploy your service using a container image stored in a container registry.

2. **Source code repository**

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



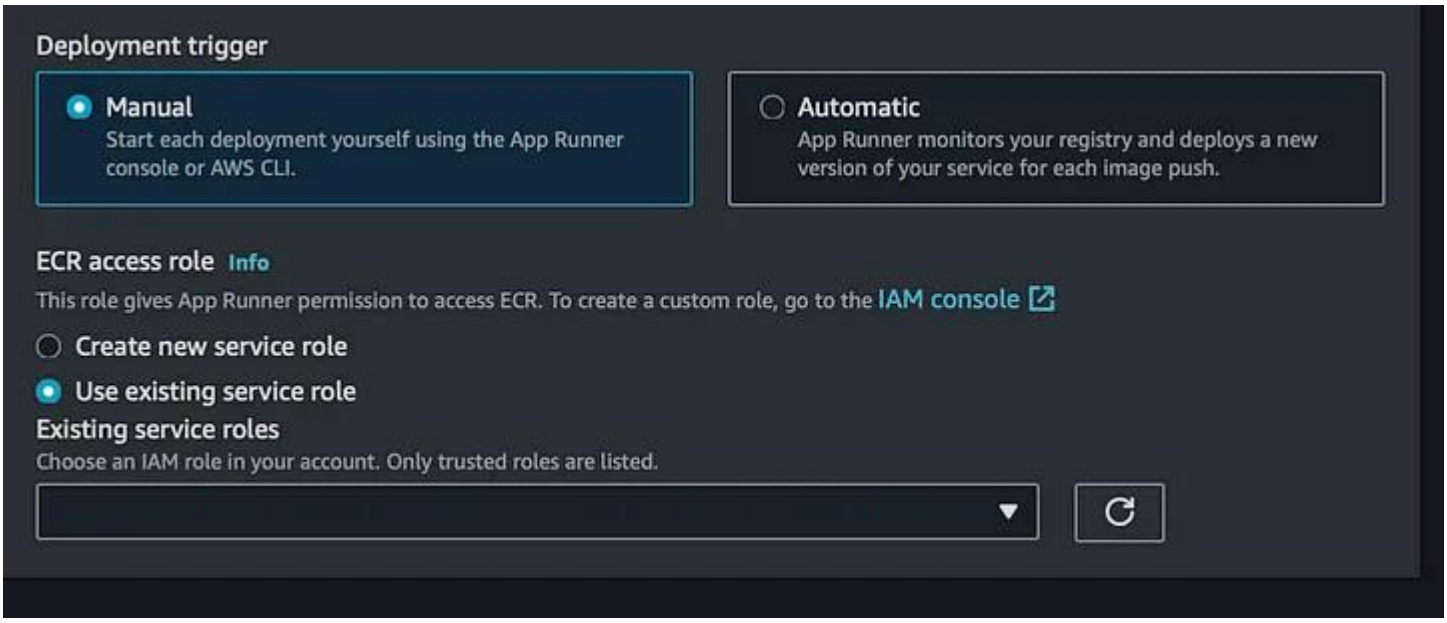
In this section, select “Repository type” and container registry and browse the “ECR Repository (Container Image URI)” which we have created in the previous step



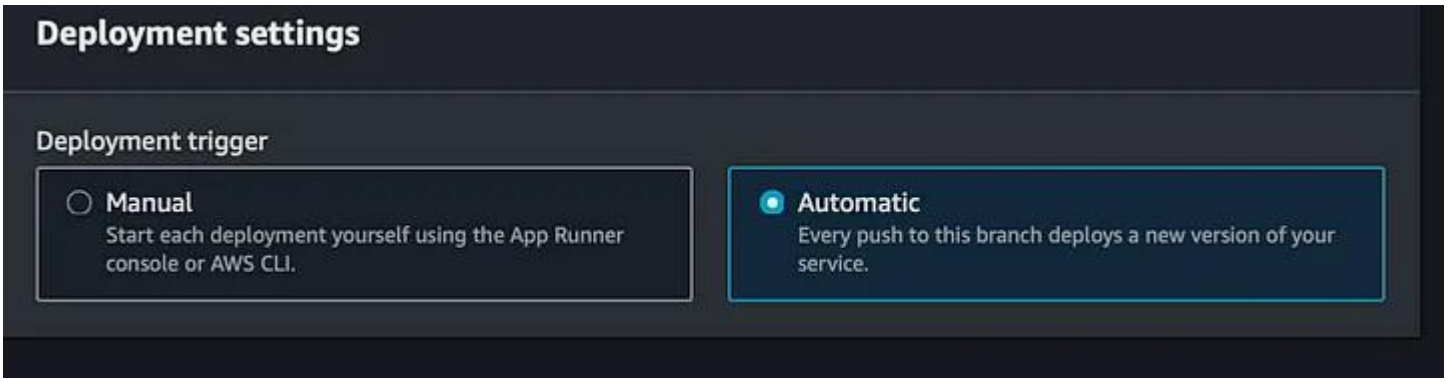
Deployment settings:

Deployment trigger

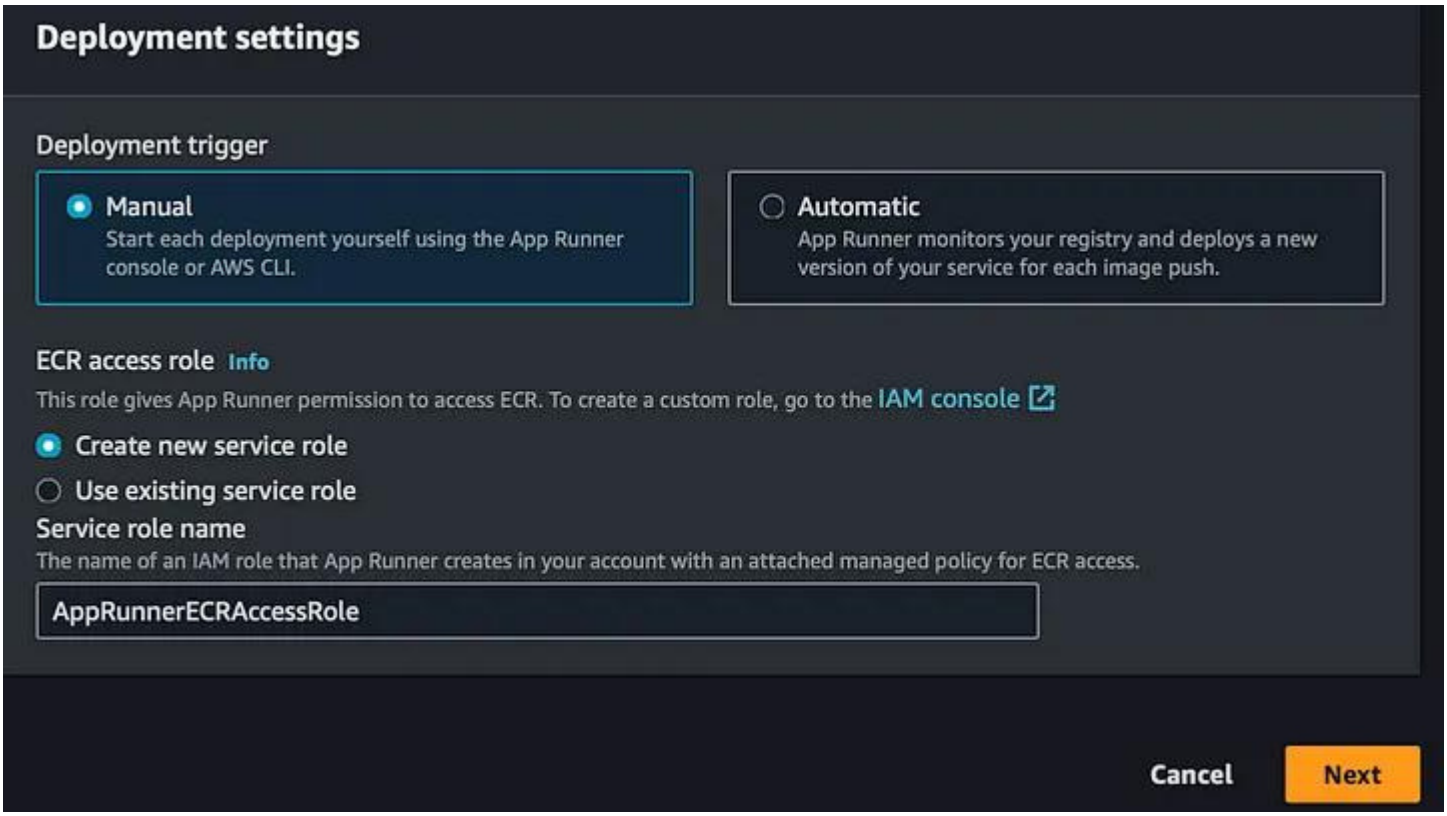
1. **Manual:** Start each deployment yourself using the App Runner console or AWS CLI.



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



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



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.

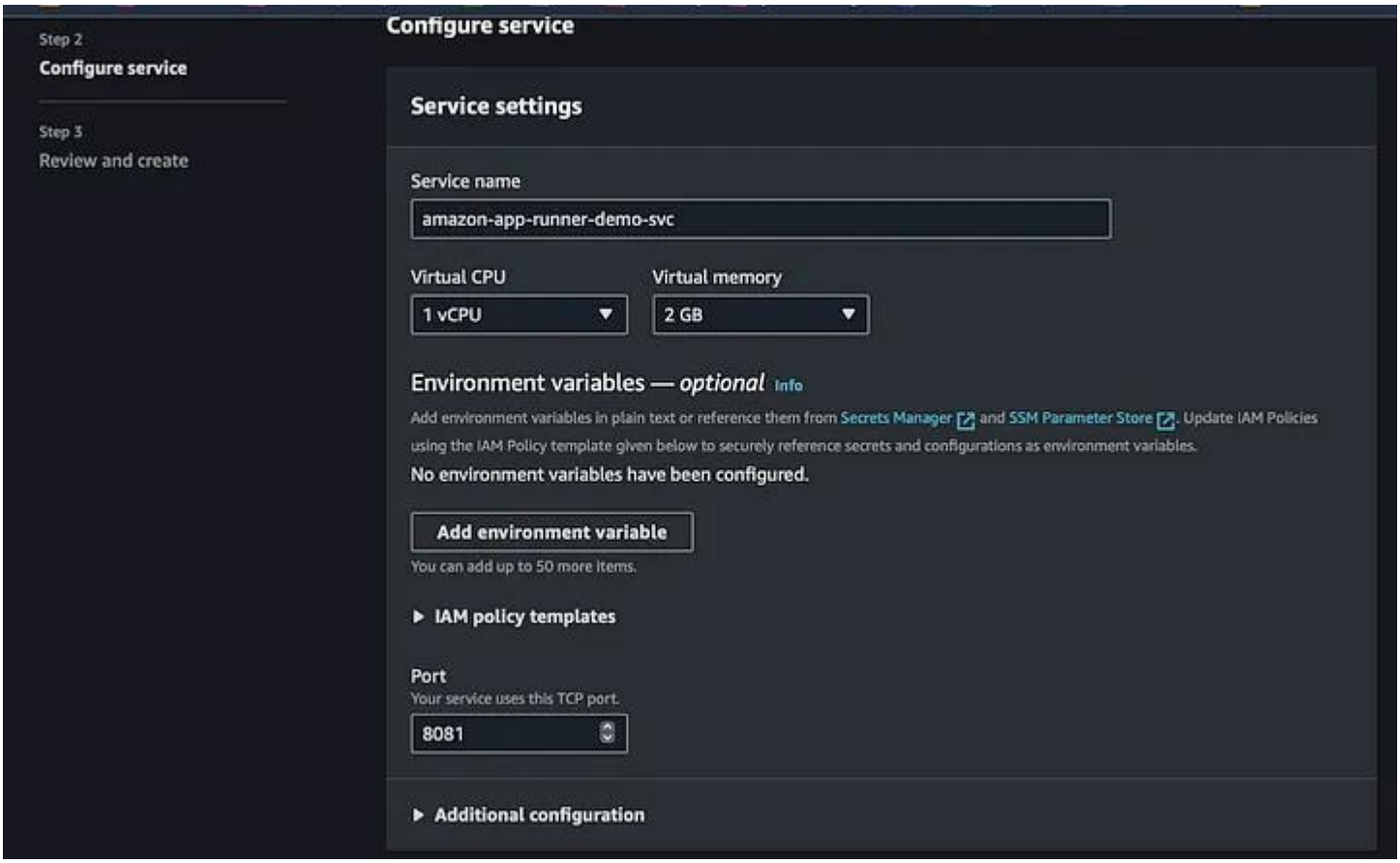
Port: 8081

Your service uses this TCP port.

Additional configuration:

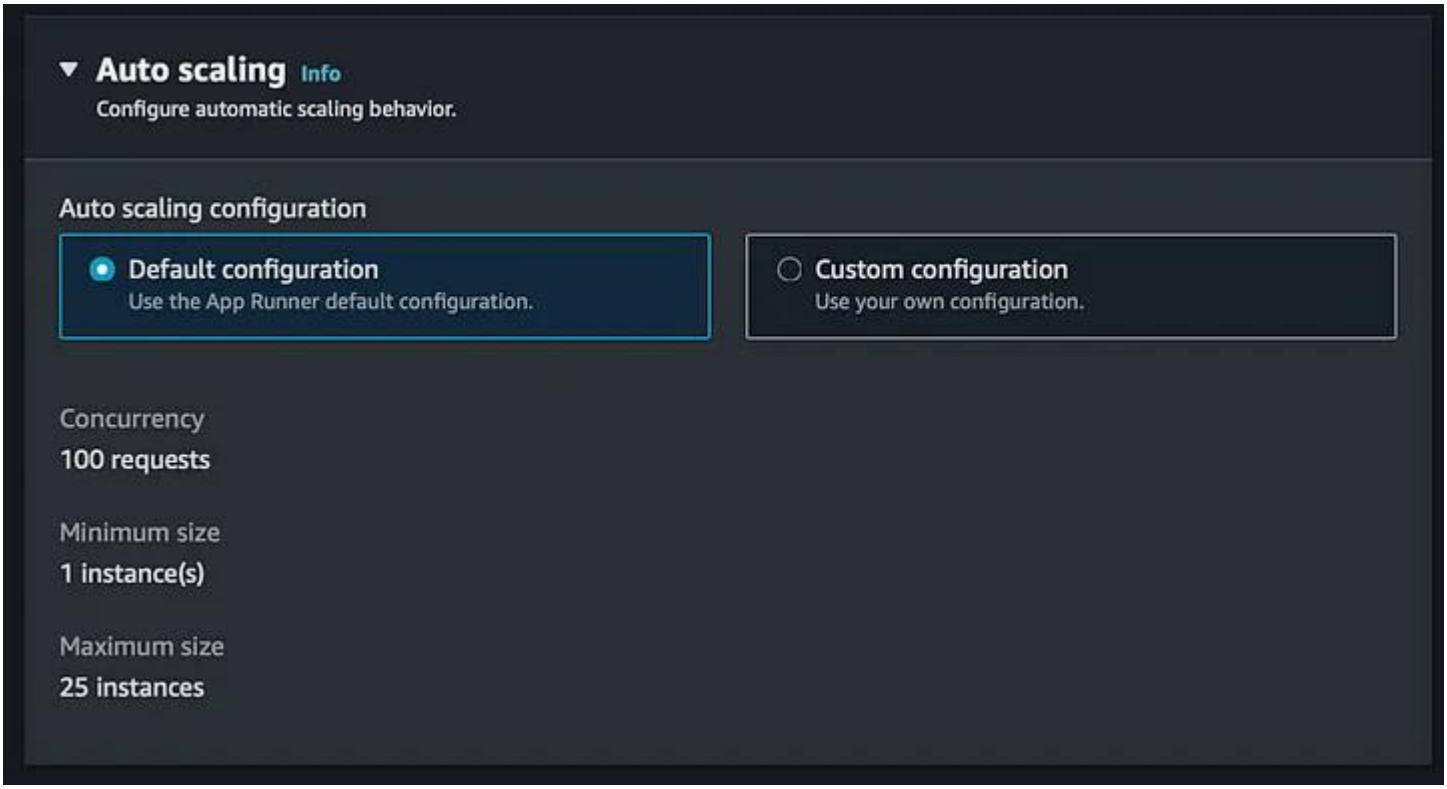
Start command — optional

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

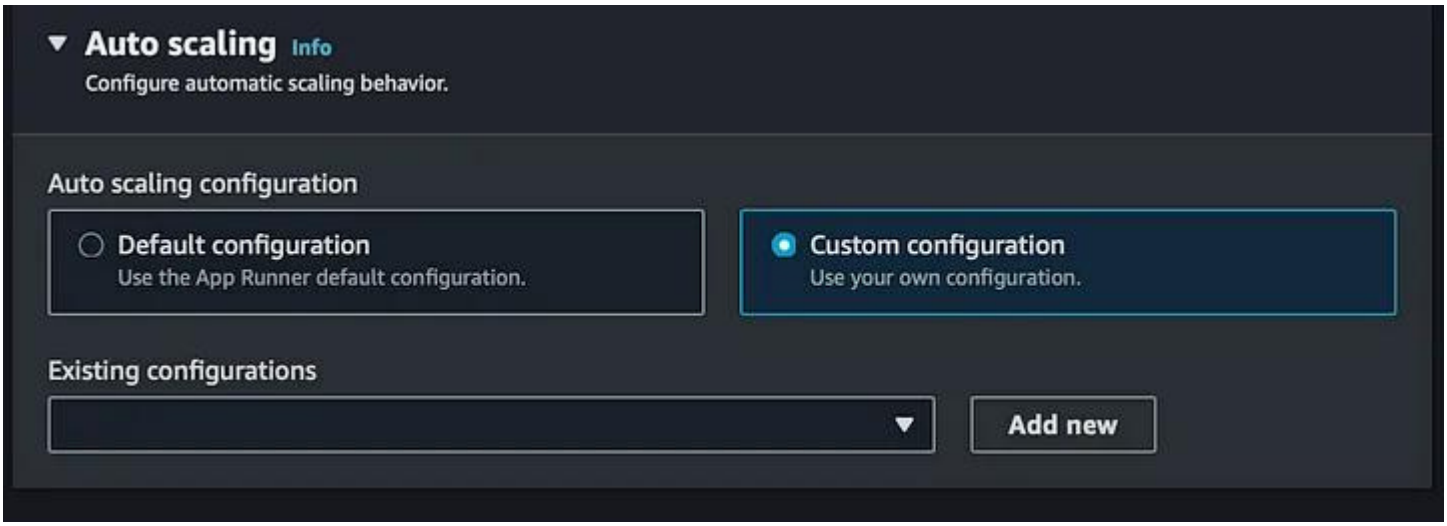


Auto scaling configuration

Default configuration: **Use the App Runner default configuration**

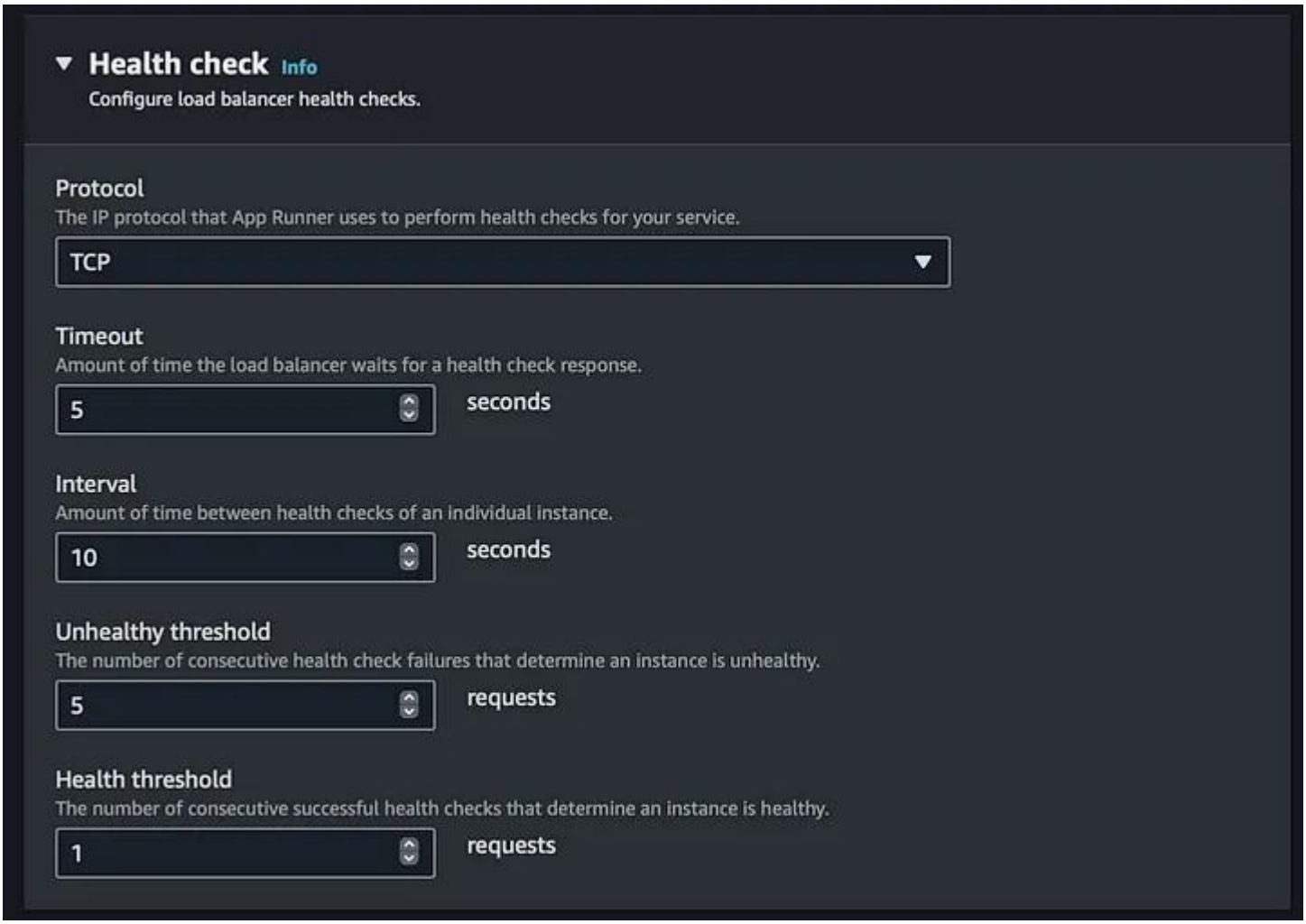


Custom configuration: **Use your own configuration.**



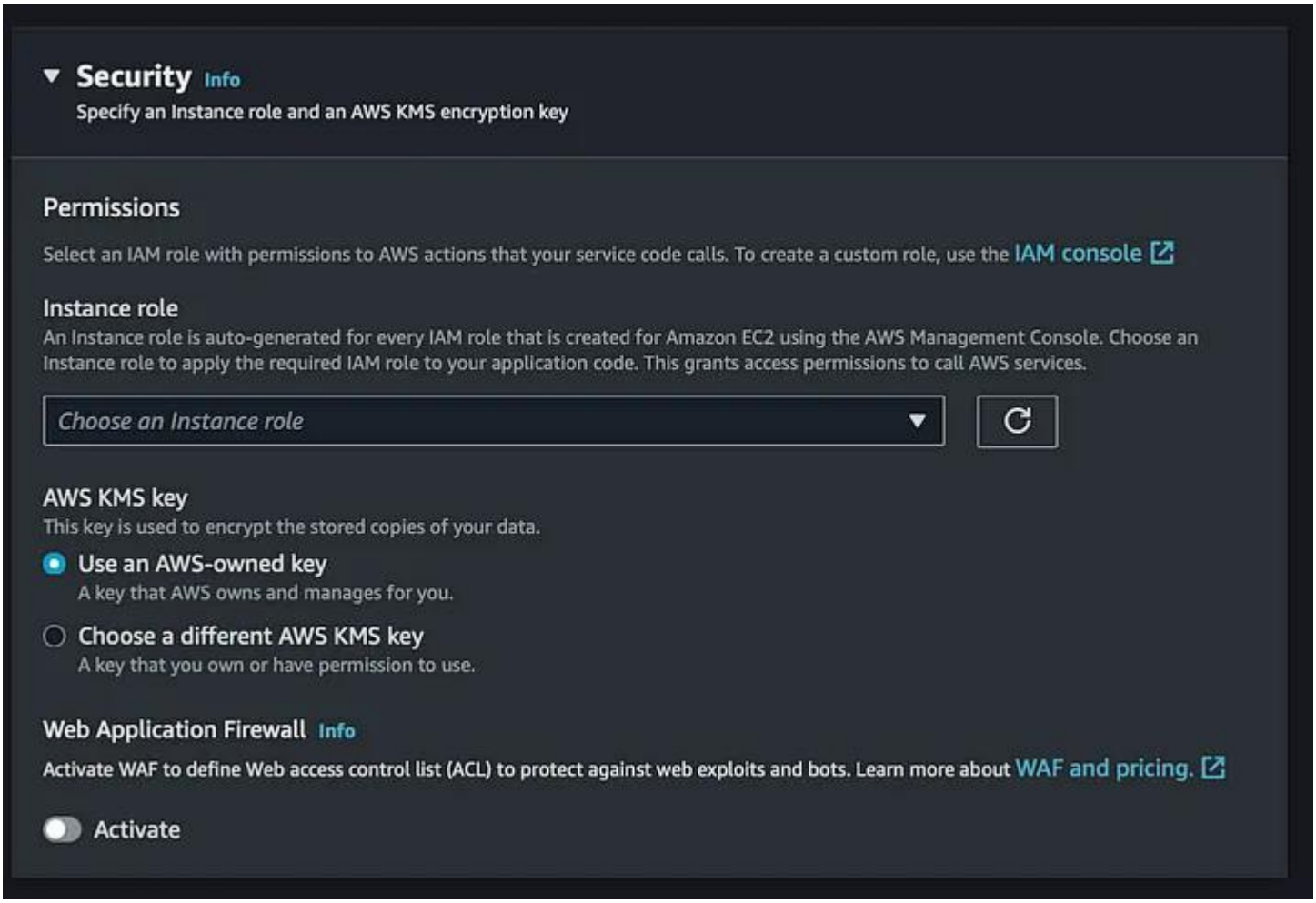
Health Check

Configure load balancer health checks



Security:

Specify an Instance role and an AWS KMS encryption key



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.

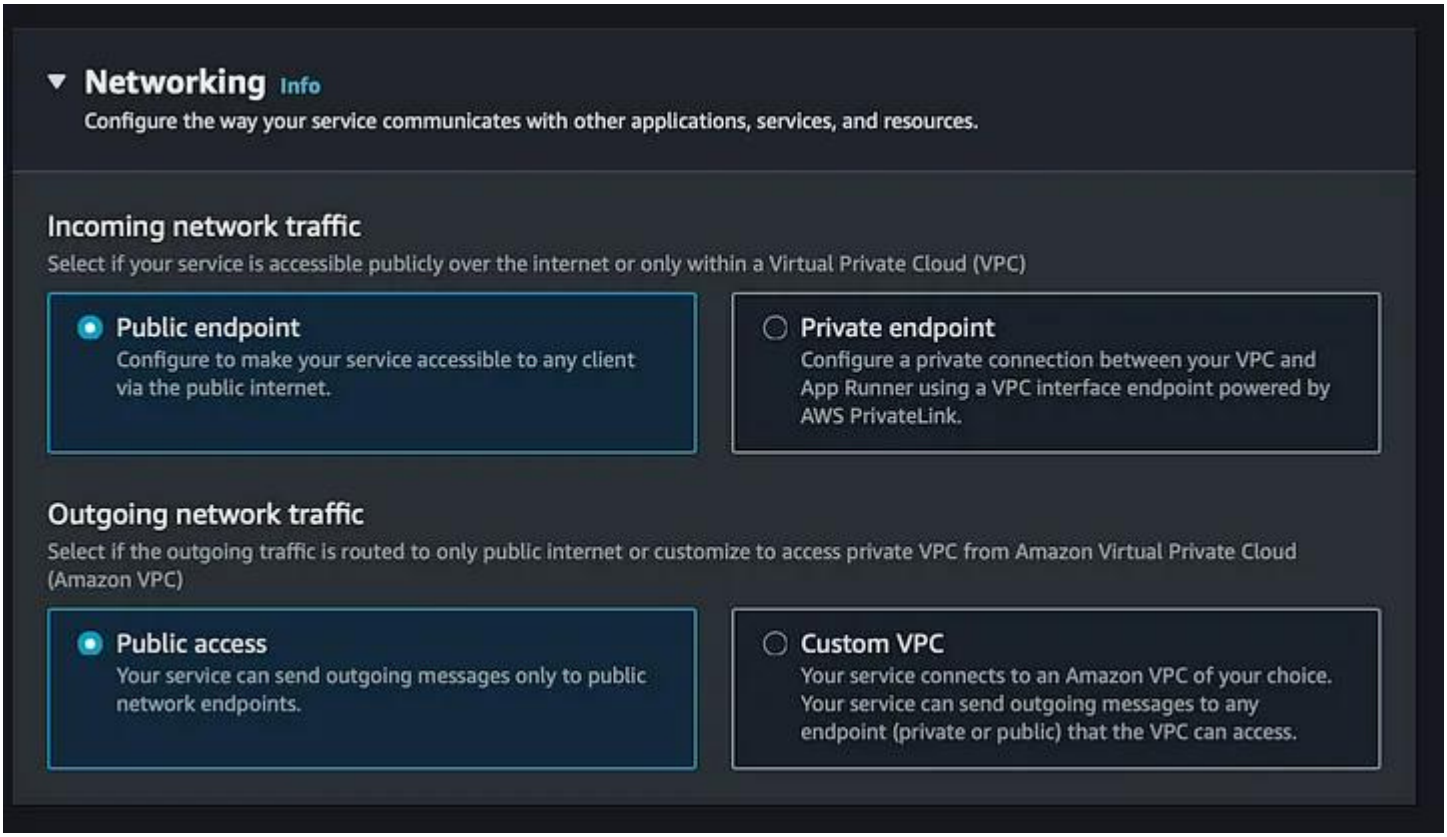
Outgoing network traffic

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.

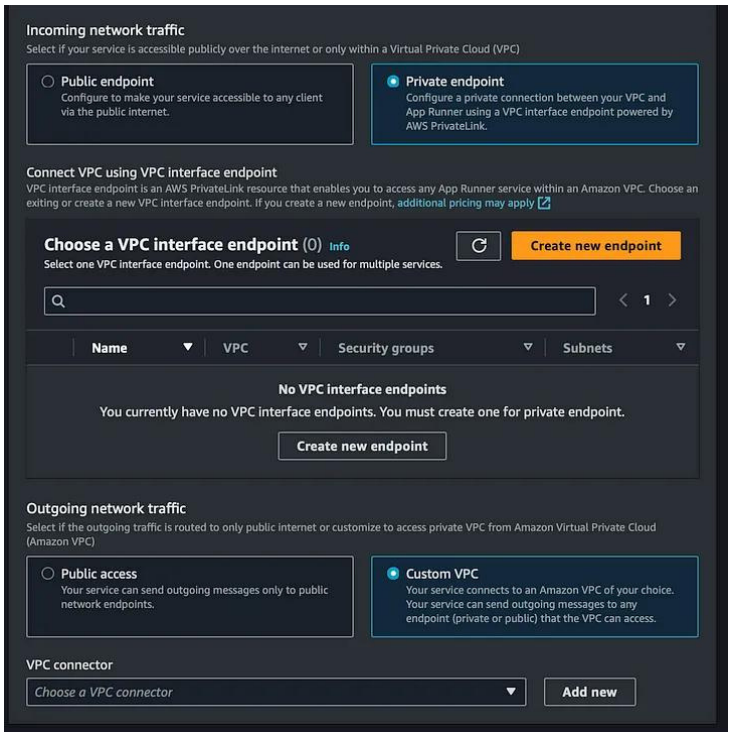


Incoming network traffic

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 VPCYour 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.

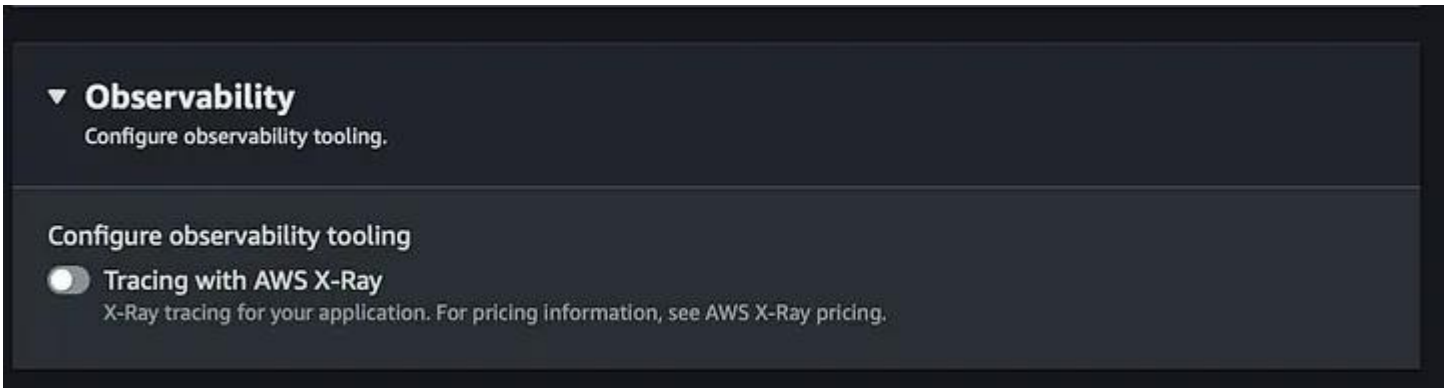


In this section, select the default but based on your requirement you can choose incoming traffic and outgoing network traffic as private

Observability:

Tracing with AWS X-Ray

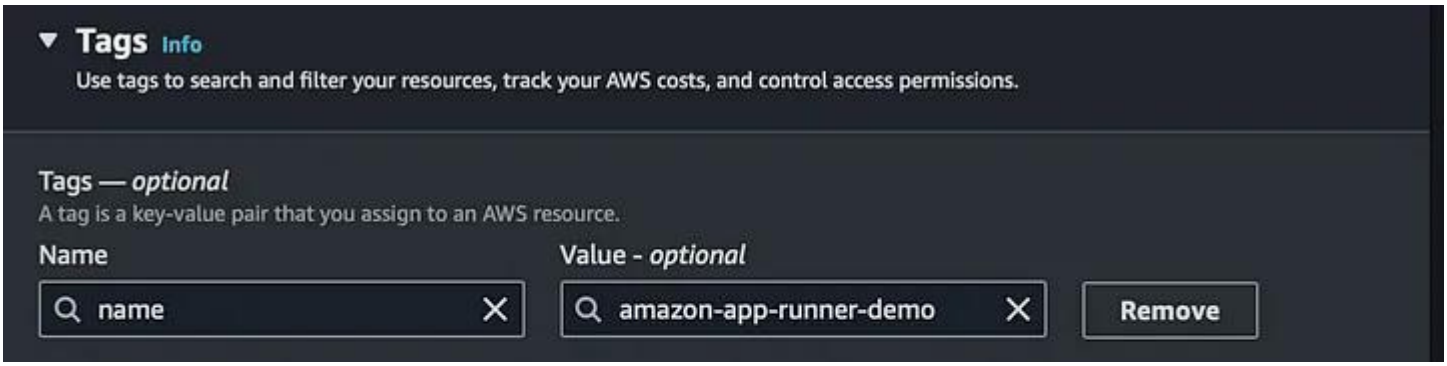
X-Ray tracing for your application. For pricing information, see AWS X-Ray pricing.



Tags

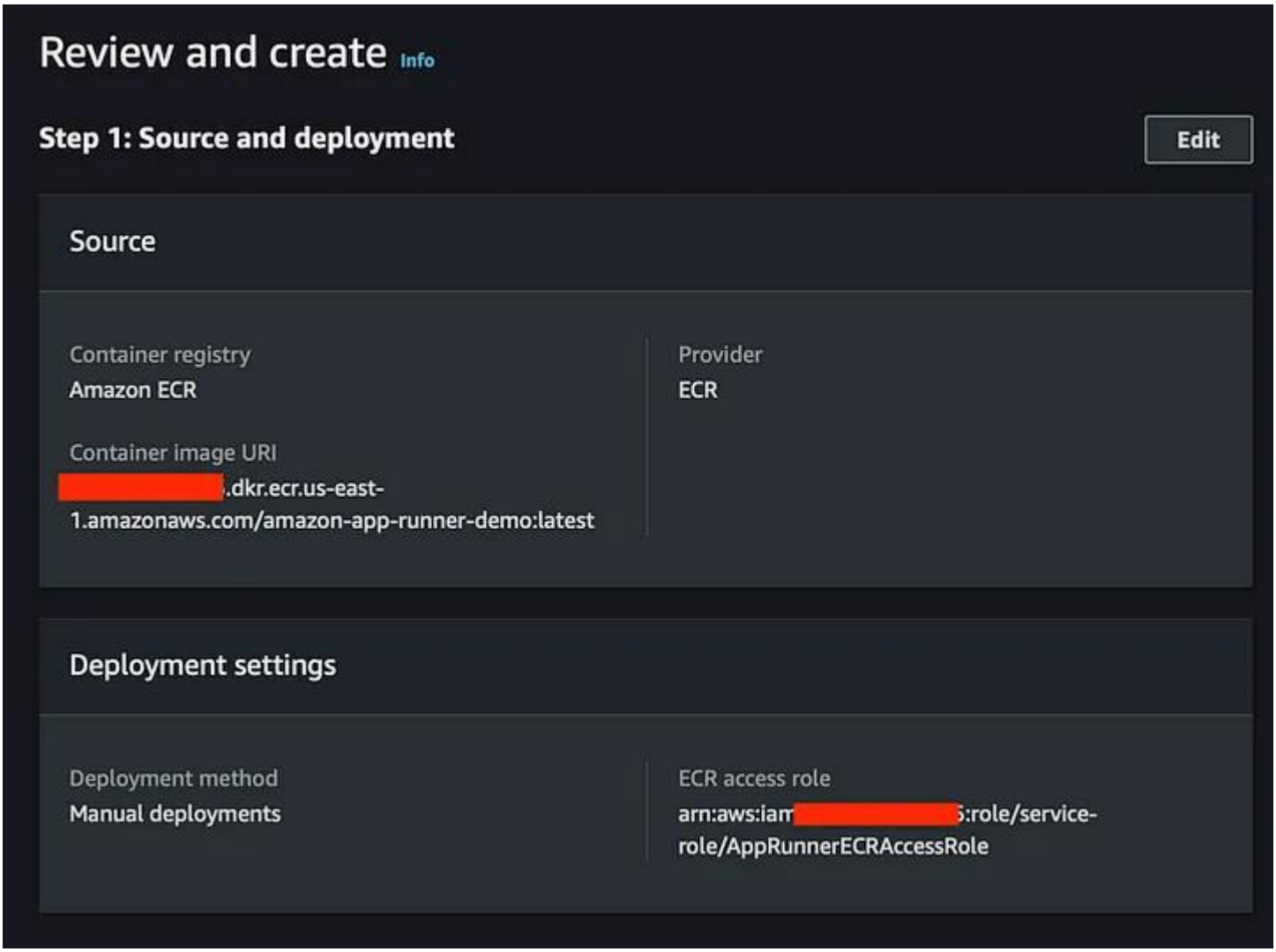
Use tags to search and filter your resources, track your AWS costs, and control access permissions.

Tags — optional



Step 3: Review and create

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



Step 2: Configure service

Edit

Service settings

Service name
amazon-app-runner-demo-svc

Port
8081

Virtual CPU & memory
1 vCPU & 2 GB

Environment variables

Name	Value
No environment variables have been configured.	

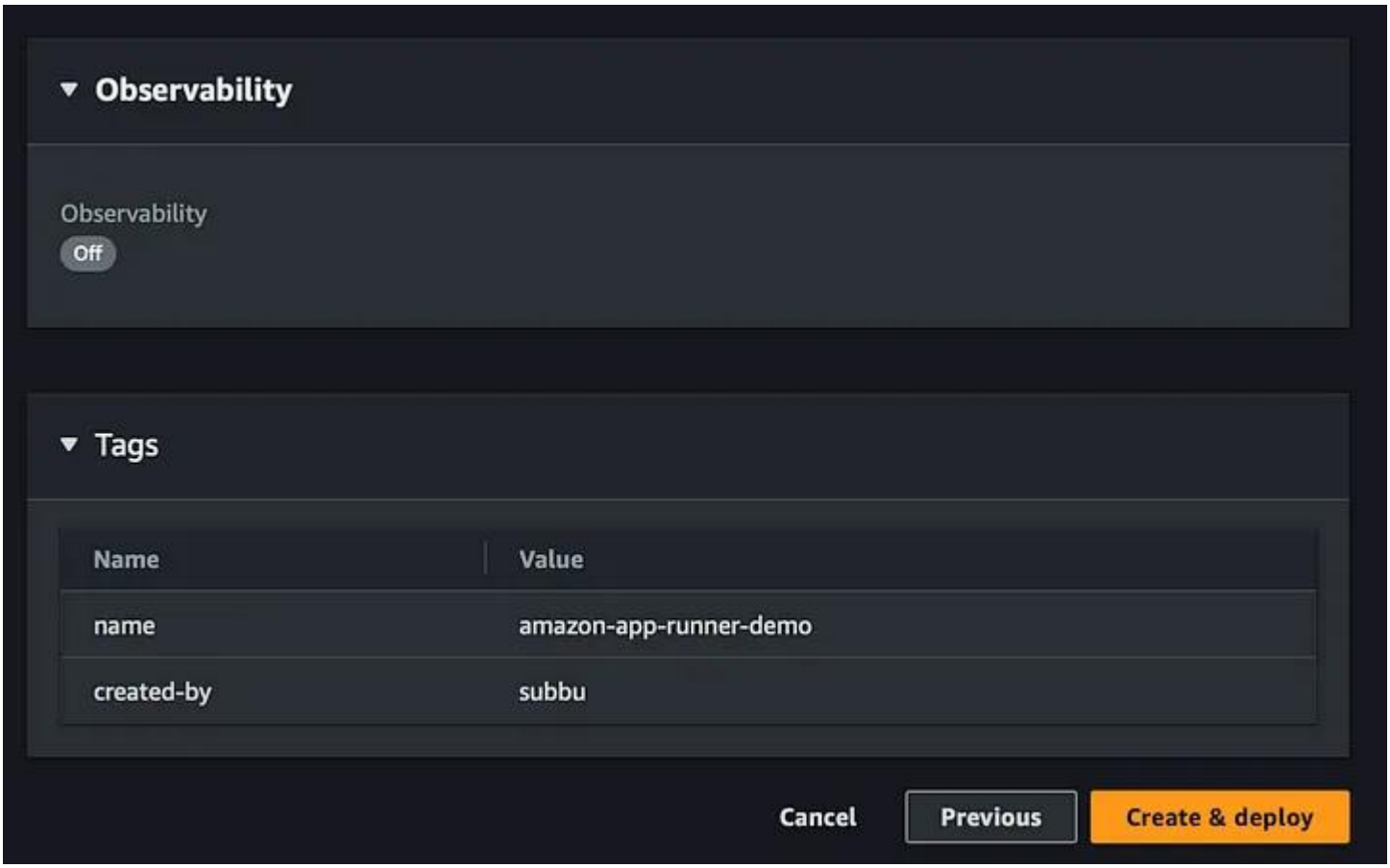
▶ Additional configuration

▼ Auto scaling

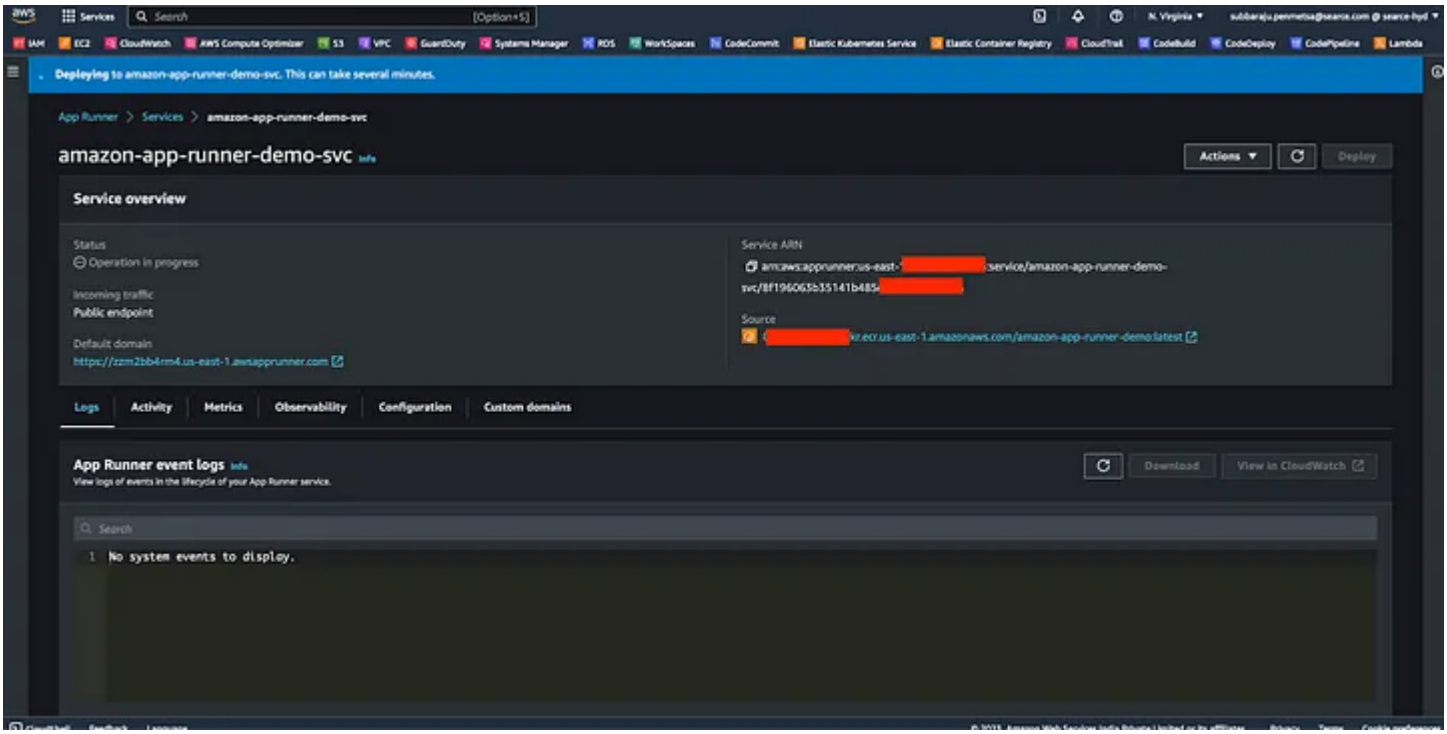
Concurrency
100

Minimum size
1

Maximum size
25

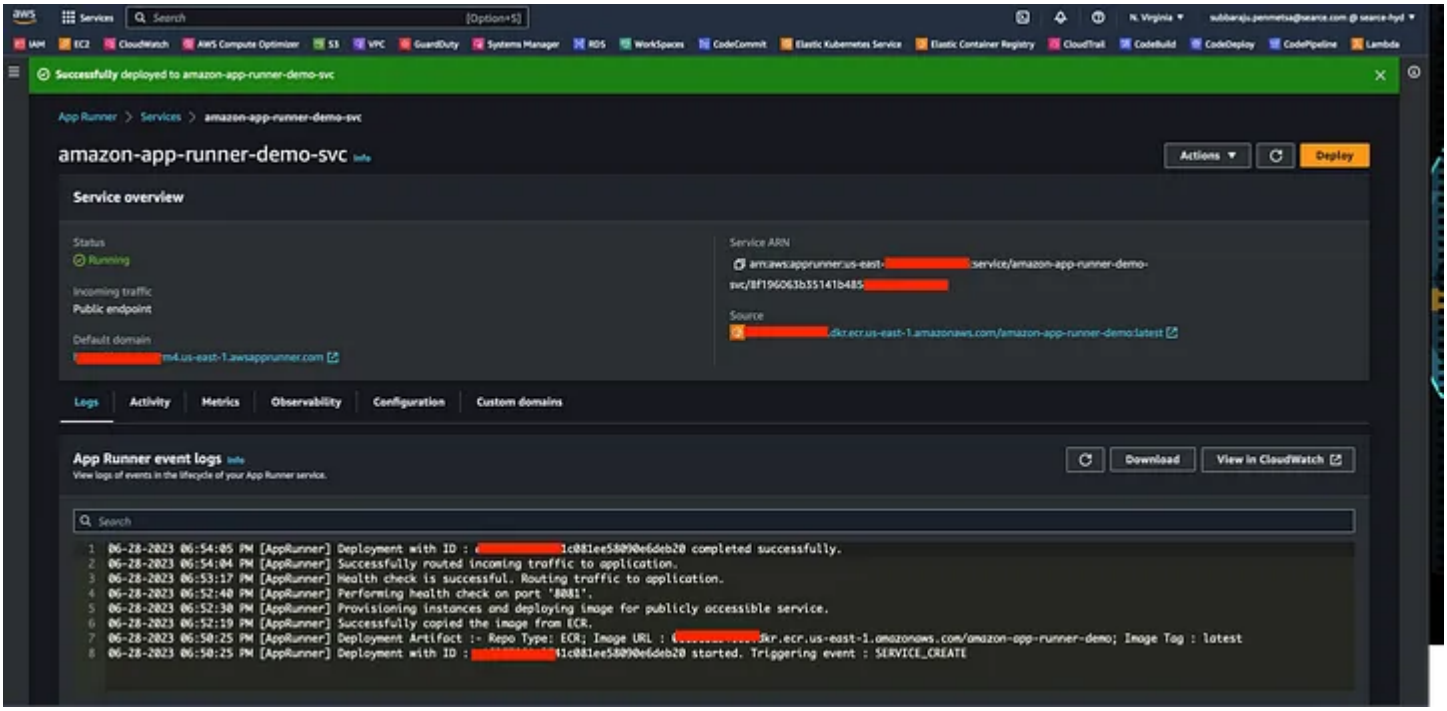


Click on **“Create & Deploy”**



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

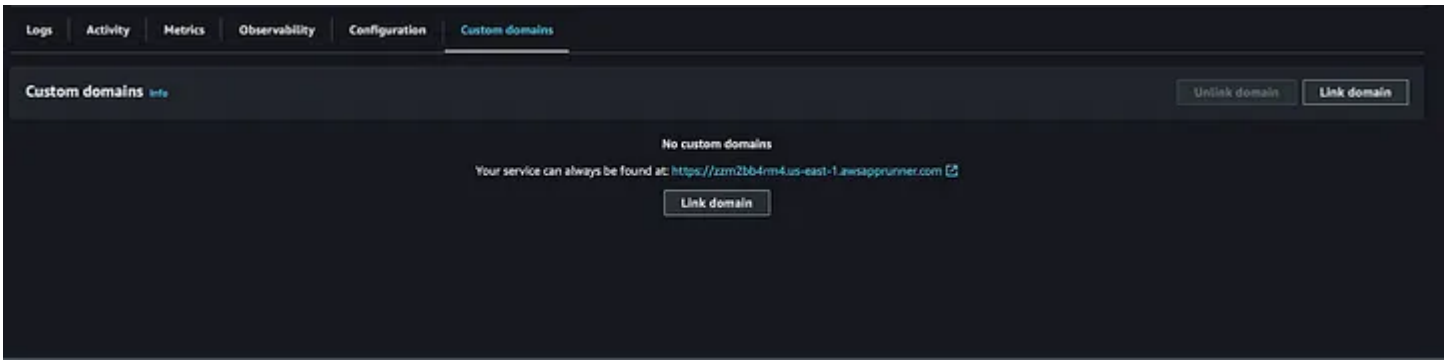
Go to “Logs” section and you can observe the creation process



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 provisoned 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 counsumed 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