Deploying an Application on AWS ECS with ECR and Docker

Creating an Amazon ECR repository, building and pushing a Docker image, and deploying the application on Amazon ECS using Fargate.

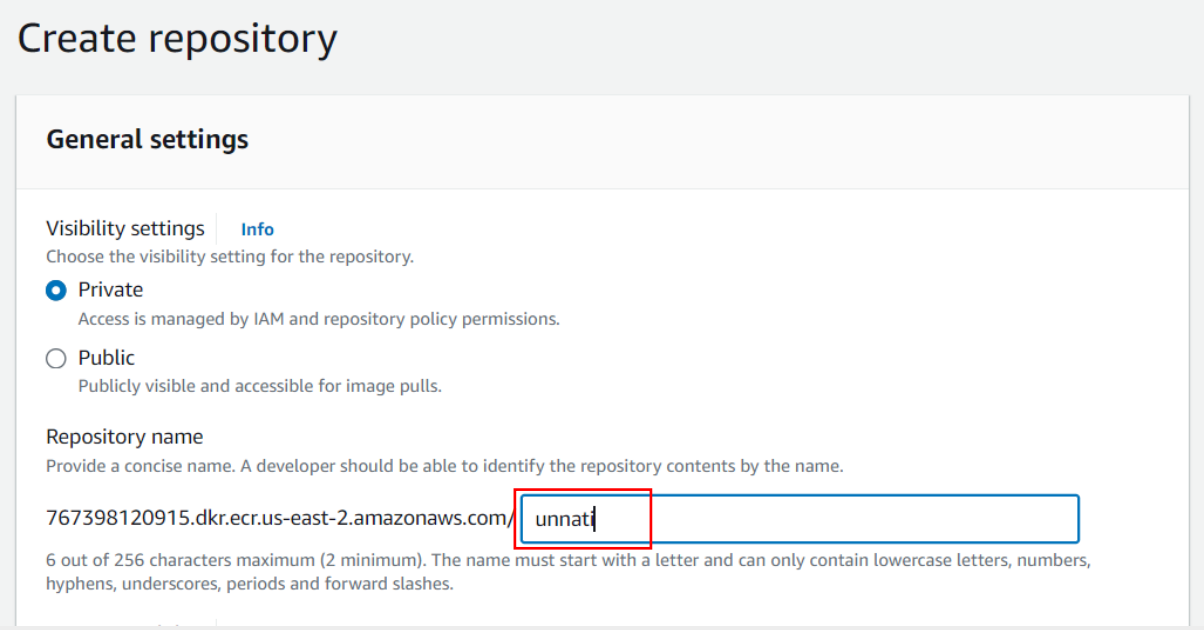
Steps

1. **Create an ECR Repository**

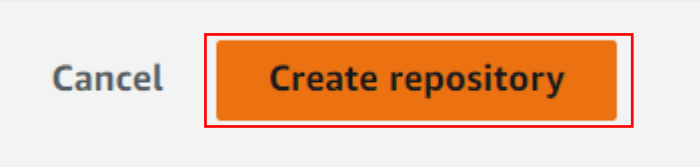
i. Navigate to the Amazon ECR console.

ii. Click on **Create repository**.

iii. Enter '**unnati**' as the repository name.



iv. Click on **Create repository**.



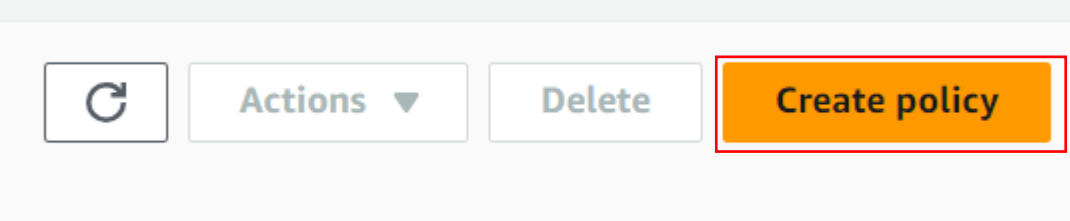
Pawar Divya

2. Create an IAM Policy

i. Navigate to the IAM console.

ii. Select **policies** from the sidebar.

iii. Click on **Create policy**.

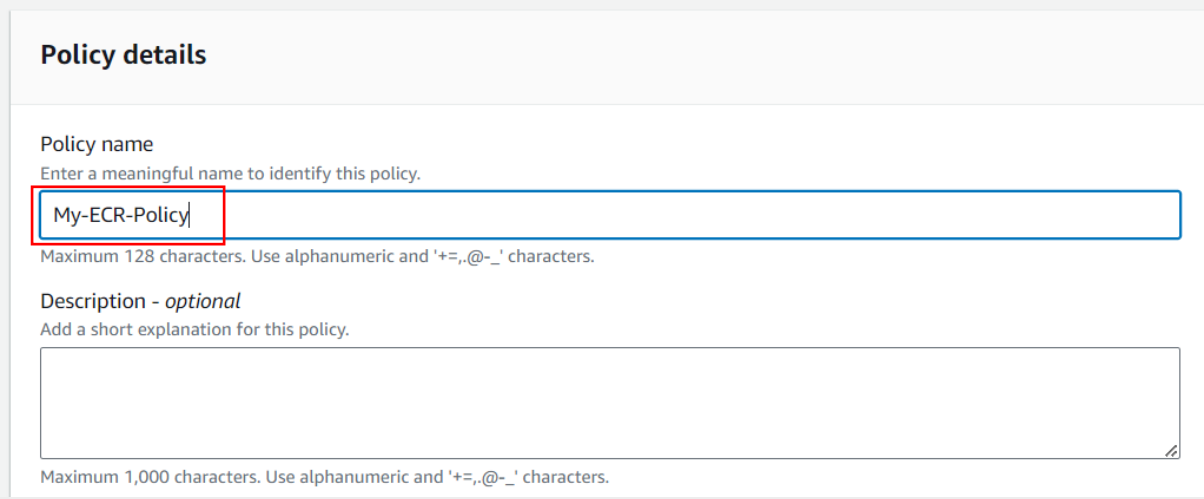


iv. Add the following JSON content to the policy document.

Policy editor

```
1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Effect": "Allow",
6              "Action": [
7                  "ecr:GetAuthorizationToken",
8                  "ecr:BatchCheckLayerAvailability",
9                  "ecr:GetDownloadUrlForLayer",
10                 "ecr:GetRepositoryPolicy",
11                 "ecr:DescribeRepositories",
12                 "ecr:ListImages",
13                 "ecr:DescribeImages",
14                 "ecr:BatchGetImage",
15                 "ecr:GetLifecyclePolicy",
16                 "ecr:GetLifecyclePolicyPreview",
17                 "ecr:ListTagsForResource",
18                 "ecr:PutImage",
19                 "ecr:UploadLayerPart",
20                 "ecr:InitiateLayerUpload",
21                 "ecr:DescribeImageScanFindings",
22                 "ecr:CompleteLayerUpload"
23             ],
24             "Resource": "*"
25         }
26     ]
27 }
28
```

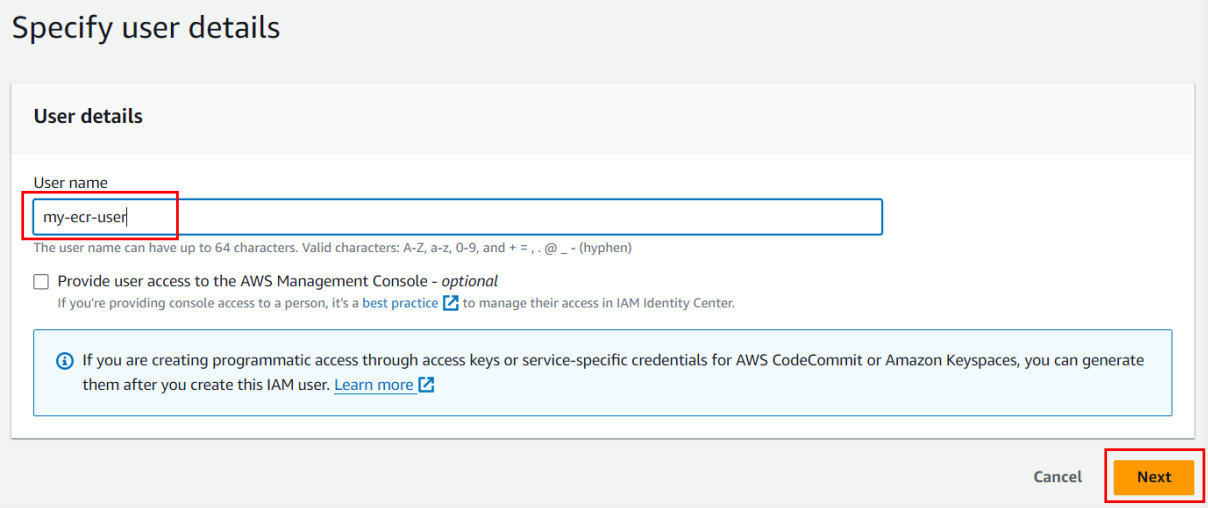
v. Give name for the policy.



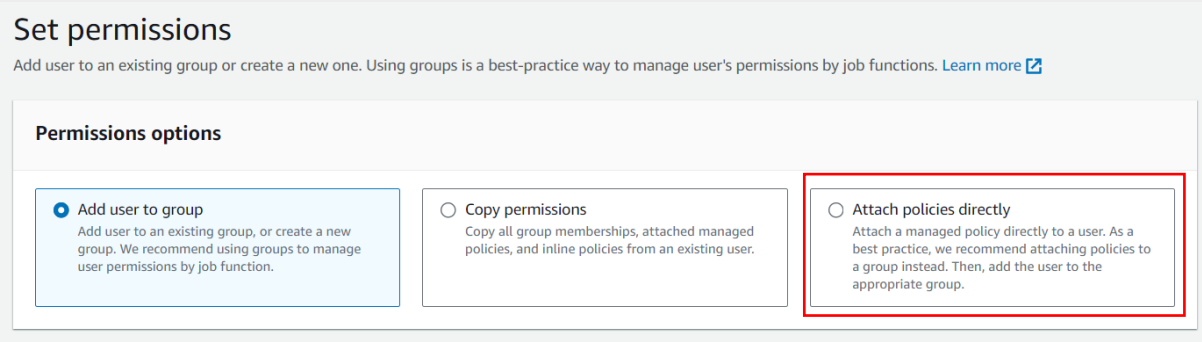
vi. Click on **Create policy**.

3. Create an IAM User and Attach Policy

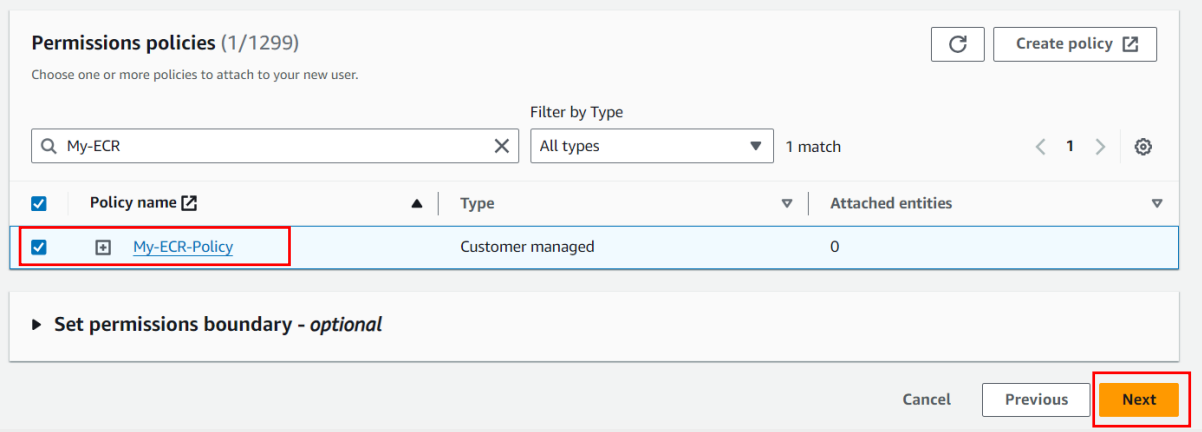
i. Navigate to the IAM console.
ii. Select **Users** from the sidebar.
iii. Click on **Add user**.
iv. Enter a username.



v. Click on **Next: Permissions**.
vi. Select **Attach Policy directly**.



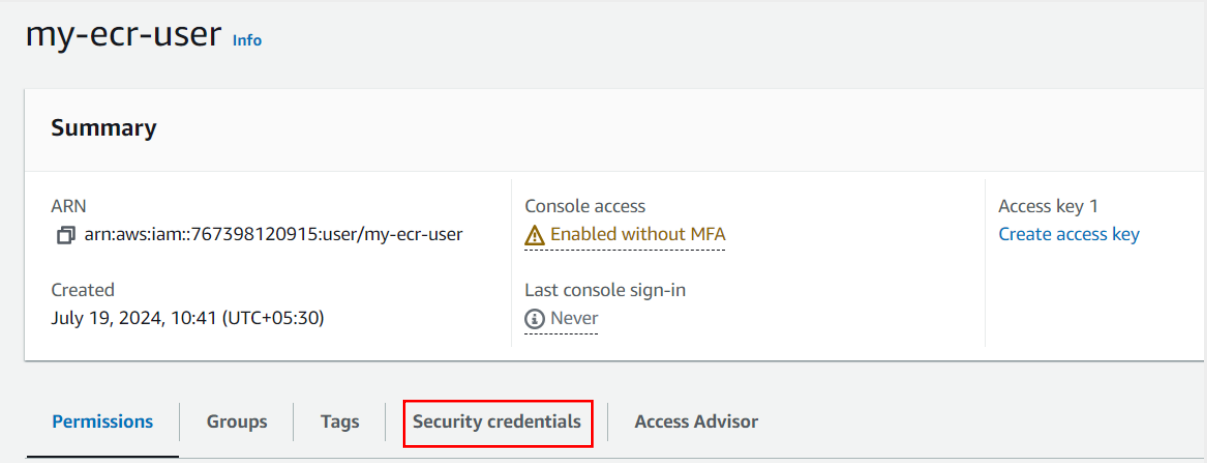
vii. Search and select the policy that we created earlier.



viii. Click on **Next: Review**.

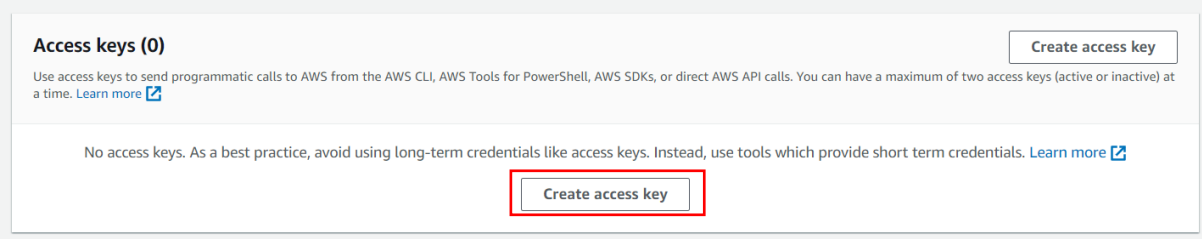
ix. Click on **Create user**.

x. Click on the created user and go to the **Security credentials** tab.



Pawar Divya

xi. Click on **Create access key**.



xii. Save the access key and secret key for later use.

4. Configure AWS CLI

i. Open a terminal (I'm using killercoda environment).

ii. Run the **aws configure** command:

iii. Enter the access key and secret key when prompted.

```
ubuntu $ aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-2
Default output format [None]: json
ubuntu $
```

5. Create a Dockerfile and index.html file

i. Create a '**Dockerfile**' with the following content:

ii. Create an '**index.html**' file with a simple message:

```
ubuntu $ echo "Hello from Divya!" > index.html
ubuntu $ cat Dockerfile
FROM docker.io/ubuntu
RUN apt update -y
RUN apt install apache2 -y
COPY index.html /var/www/html/
CMD ["apachectl","-D","FOREGROUND"]
ubuntu $
```

6. Build and Push Docker Image to ECR

i. Run the following command to authenticate Docker to your Amazon ECR registry:

```
ubuntu $ aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 76739
8120915.dkr.ecr.us-east-2.amazonaws.com
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu $
```

ii. Build the Docker image:

```
ubuntu $ docker build -t unnati .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  33.28kB
Step 1/5 : FROM docker.io/ubuntu
 ---> 35a88802559d
Step 2/5 : RUN apt update -y
 ---> Using cache
 ---> a6ef5a7835a3
Step 3/5 : RUN apt install apache2 -y
 ---> Using cache
 ---> cf6131a6d86a
Step 4/5 : COPY index.html /var/www/html/
 ---> Using cache
 ---> c78dd00cff5d
Step 5/5 : CMD ["apachectl","-D","FOREGROUND"]
 ---> Using cache
 ---> 802d531edd75
Successfully built 802d531edd75
Successfully tagged unnati:latest
ubuntu $
```

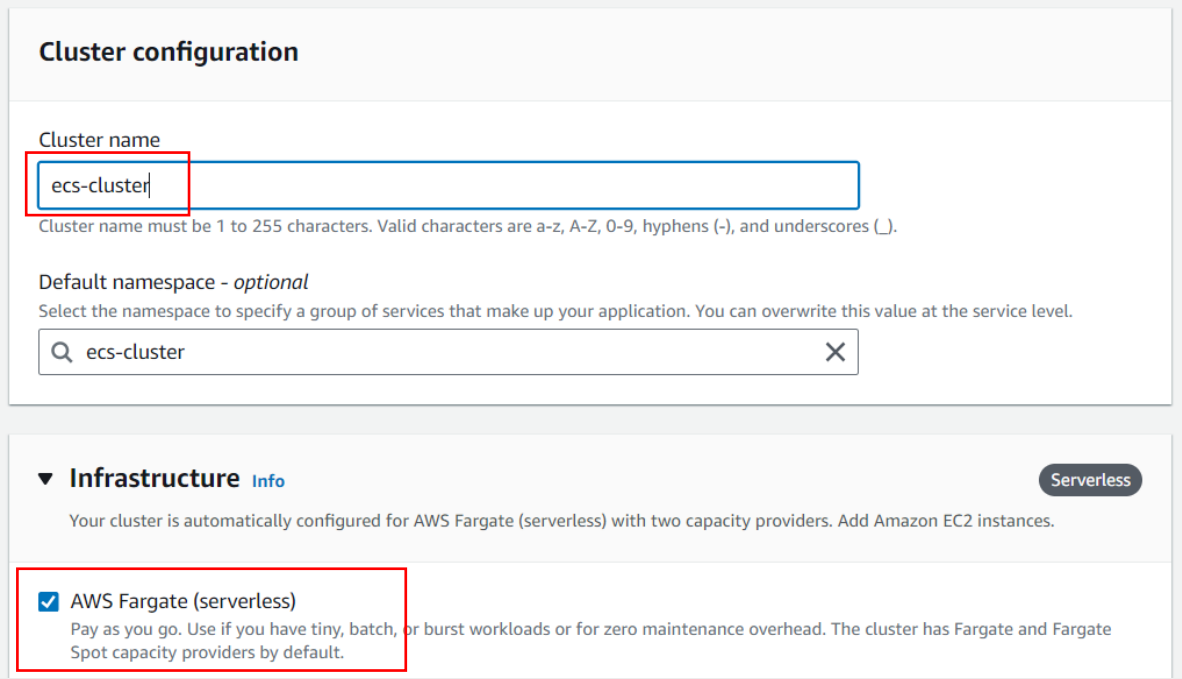
iii. Tag and Push the docker image to ECR:

```
ubuntu $ docker tag unnati:latest 767398120915.dkr.ecr.us-east-2.amazonaws.com/unnati:latest
ubuntu $ docker push 767398120915.dkr.ecr.us-east-2.amazonaws.com/unnati:latest
The push refers to repository [767398120915.dkr.ecr.us-east-2.amazonaws.com/unnati]
3047dad3da1a: Pushed
85705c77aae6: Pushed
d965f09cfbce: Pushed
a30a5965a4f7: Layer already exists
latest: digest: sha256:8ef0b7b1bd53085601bd3d6640385a37f67dd341e2d2b2c7890122ff9488a1f5 size: 1160
ubuntu $
```

7. Create an ECS Cluster

i. Navigate to the Amazon ECS console.
ii. Click on **Clusters** in the sidebar.
iii. Click on **Create Cluster**.

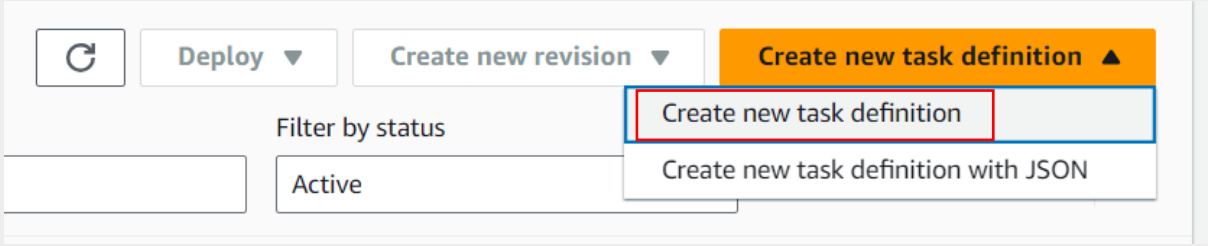
iv. Enter a name for the cluster.



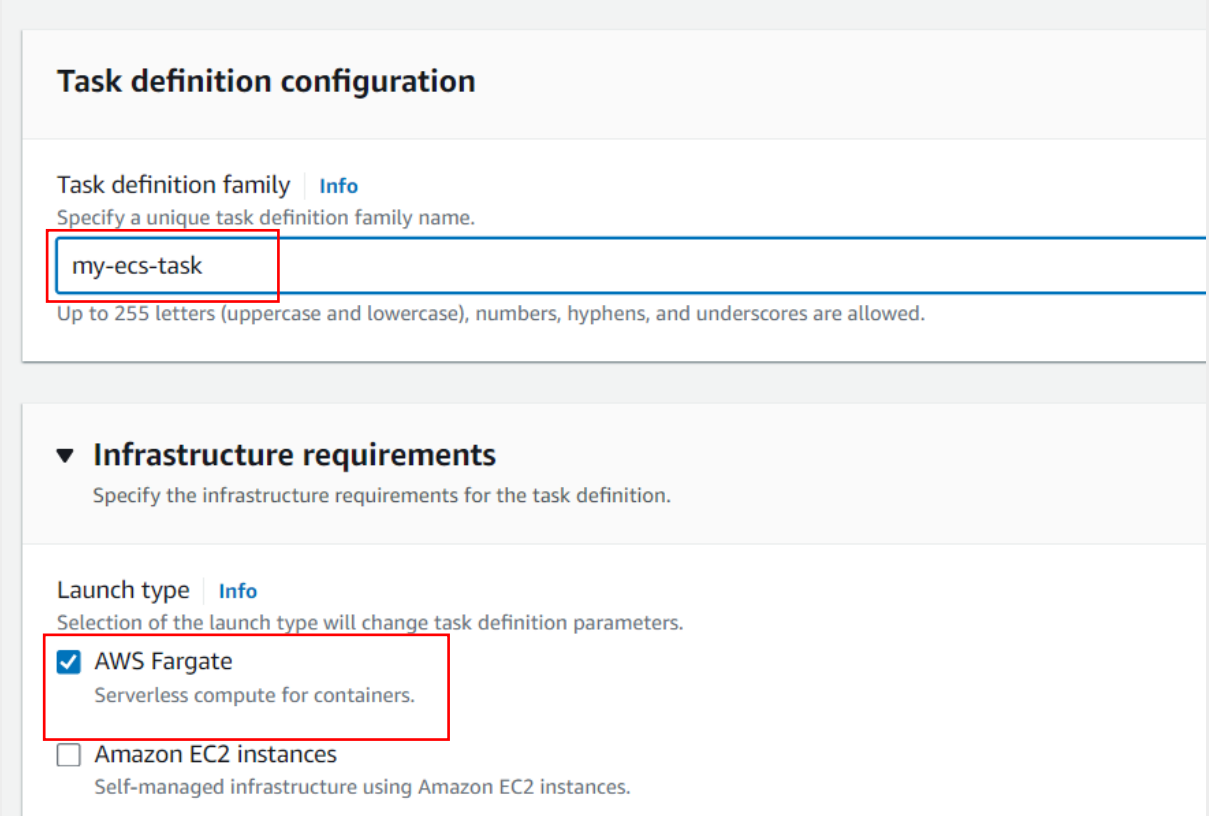
v. Click on **Create**.

8. Create a Task Definition

i. Navigate to the Amazon ECS console.
ii. Click on **Task Definition** in the sidebar.
iii. Click on **Create new Task Definition**.



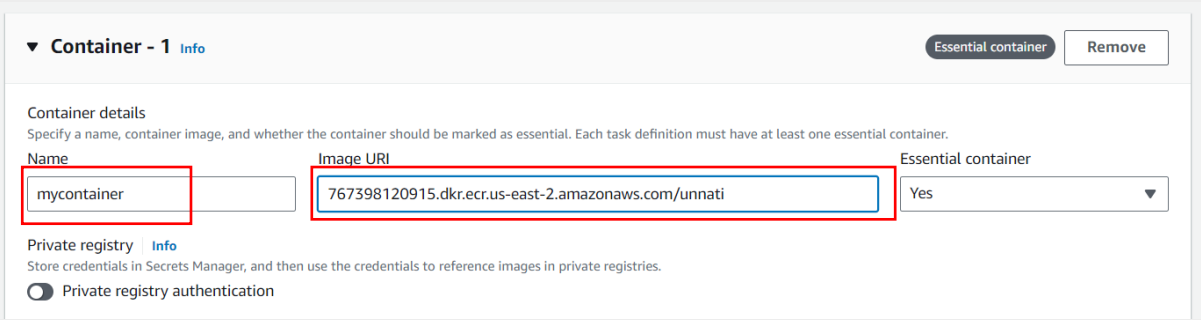
iv. Enter a name for task definition.



v. In the **Container** section, click on Add container.

vi. Enter a name for the container.

vii. In the **Image** field, enter the **URI** of the **ECR repository**.



viii. Click on **Create**.

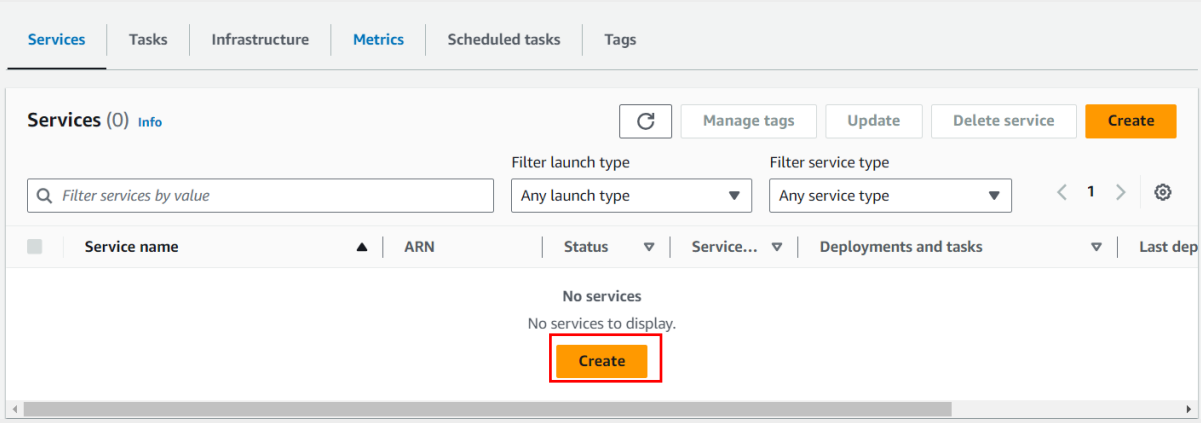
9. Create a Service

i. Navigate to ECS console.

ii. Click on **Clusters** in the sidebar.

iii. Select the cluster created earlier.

iv. Click on **Create** under **Services** tab.



v. Under **Environment**, select **Capacity provider strategy** and choose **FARGATE**.

Create Info

Environment

Existing cluster

ecs-cluster

▼ **Compute configuration *(advanced)***

Compute options Info

To ensure task distribution across your compute types, use appropriate compute options.

- (●) Capacity provider strategy
  Specify a launch strategy to distribute your tasks across one or more capacity providers.
- ( ) Launch type
  Launch tasks directly without the use of a capacity provider strategy.

Capacity provider strategy Info

Select either your cluster default capacity provider strategy or select the custom option to configure a different strategy.

- ( ) Use cluster default
  No default capacity provider strategy configured for this cluster.
- (●) Use custom (Advanced)

Capacity provider	Base Info	Weight Info
FARGATE ▼	0	1

Add capacity provider

vi. Under **Deployment configuration**, select **Task**.

vii. **Task definition**: Select the created task definition.

Deployment configuration

Application type Info

Specify what type of application you want to run.

- ( ) Service
  Launch a group of tasks handling a long-running computing work that can be stopped and restarted. For example, a web application.
- (●) Task
  Launch a standalone task that runs and terminates. For example, a batch job.

Task definition

Select an existing task definition. To create a new task definition, go to Task definitions.

☐ Specify the revision manually

Manually input the revision instead of choosing from the 100 most recent revisions for the selected task definition family.

Family	Revision
my-ecs-task ▼	1 (LATEST) ▼

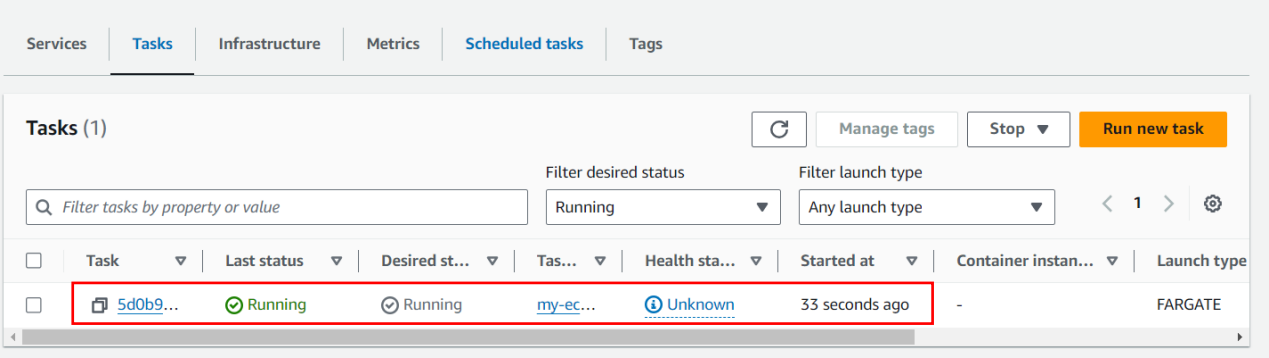
viii. Click **Next** and **Create service**.

10. Access the Deployed Application

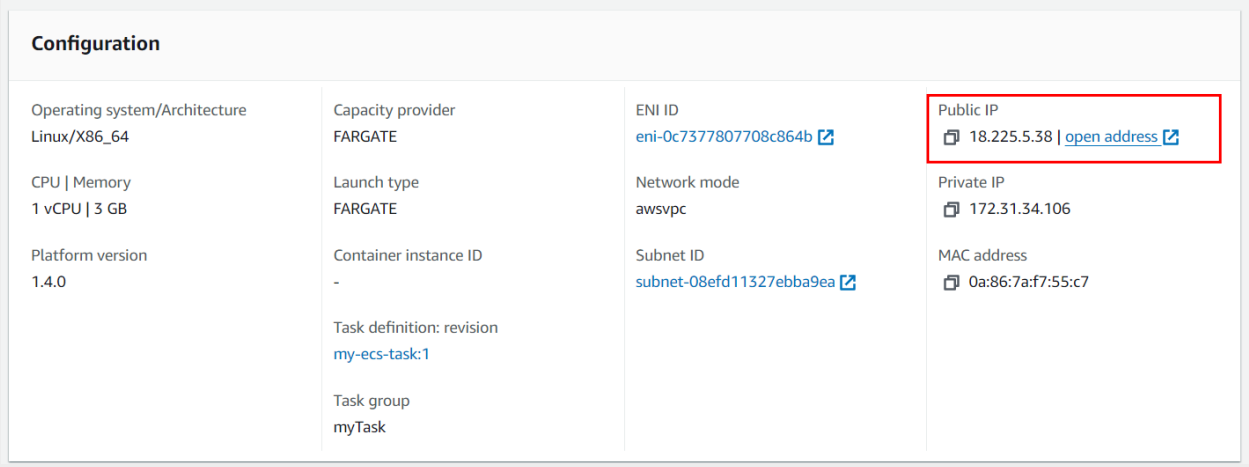
i. Navigate to the Amazon ECS console.

ii. Select the cluster and click on the service created earlier.

iii. Click on **Task** tab and select the running task.



iv. Under Configuration, click on Open address.



v. Open the address in a web browser to and navigate to the public IP address to view the '**index.html**' page created earlier.

By following these steps you've successfully deployed an application on AWS ECS using ECR and Docker.