# **Exploring Containers on AWS**

**Overview**

Docker containers have revolutionized application deployment by providing a lightweight, portable, and consistent environment for running applications. Amazon Web Services (AWS) offers a powerful container orchestration service called Amazon Elastic Container Service (ECS), which simplifies the management and scaling of Docker containers. In this workshop, we will walk through the process of deploying a Docker container in AWS ECS (Fargate), step by step.

AWS Fargate is a technology that you can use with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing.

A curated list of guides, development tools, and resources for:

1. [Amazon Elastic Container Service (ECS)](https://aws.amazon.com/ecs/)
2. [AWS Fargate](https://aws.amazon.com/fargate/)

**Prerequisites:**

Before we get started, we need to ensure we have the following prerequisites installed on our machine:

1. Docker — you can download it from the official website [here](https://www.docker.com/products/docker-desktop).
2. Node.js — you can download it from the official website [here](https://nodejs.org/).

LET’S GET STARTED

## Building the Docker Image

To build the docker image first, you need to create a Dockerfile for your application. The Dockerfile defines how to build your Docker image, including dependencies, configurations, and entry points. Here’s a sample Dockerfile for a simple web-based application:

# Use a node base image

FROM node:18-alpine

# Set the working directory in the container

WORKDIR /app

# Copy package.json and package-lock.json to the working directory

COPY package.json /app

# Install npm dependencies

RUN npm install

# Copy the entire project to the working directory

COPY . /app

# Build the React app

RUN npm run build

# Expose the port where the app runs

EXPOSE 3000

# # Command to start the React app

CMD ["npm", "start"]

This Dockerfile specifies a base image of node:alpine, which is a lightweight version of Node.js. We set the working directory to /app, copy the package.json and package-lock.json files, install the dependencies, copy the app files, build the app, and expose the port 3000. Finally, we set the command to run the app with npm start.

Refer to the Docker docs for details on the functionality of each command:

[Dockerfile reference](https://docs.docker.com/reference/dockerfile/) | [Docker Documentation](https://docs.docker.com/guides/docker-concepts/building-images/build-tag-and-publish-an-image/)

Next, use the Docker CLI to **build the Docker image** locally. Open your terminal and navigate to the root directory of your app, then run the following command:

$ docker build -t weather-forecasting-app .

## Pushing the Docker Container Image to [DockerHub](https://hub.docker.com/).

1. First, make sure you have a Docker Hub account. If you don't have one, you can sign up for free at <https://hub.docker.com/signup>.

DockerHub is a service where we store our container images. Consider it as [npmjs](https://www.npmjs.com/) of containers instead of JS packages.

1. Log in to Docker Hub using the docker login command in your terminal.

docker login

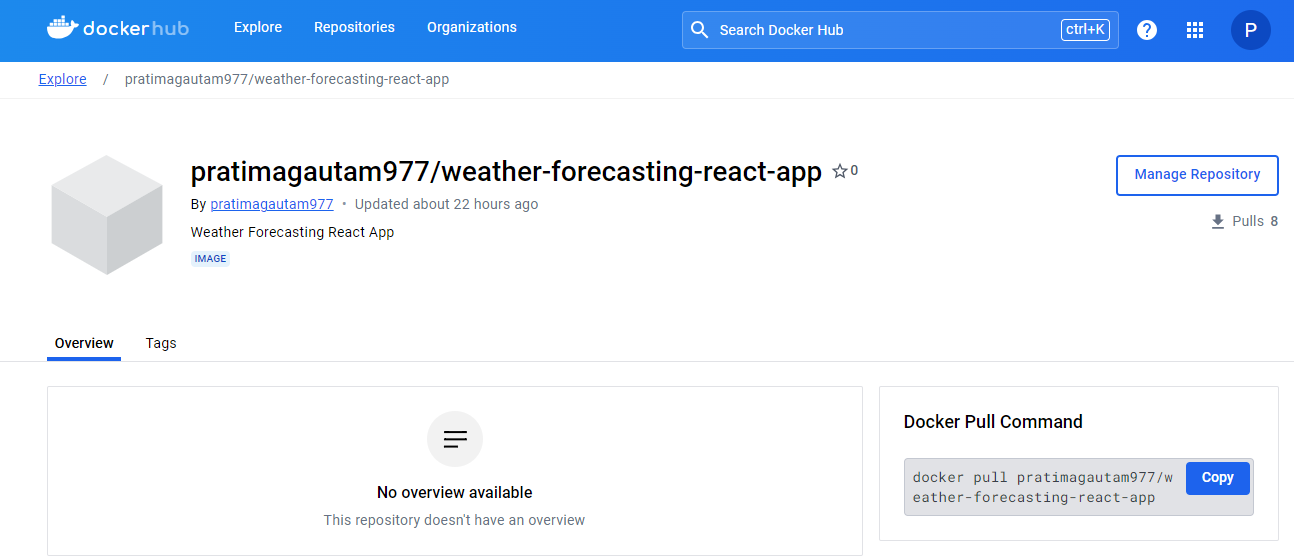
1. Enter your Docker Hub username and password when prompted.
2. Tag the Docker image with your Docker Hub username and the app name using the following command:

docker tag weather-forecasting-app pratimagautam977/weather-forecasting-react-app

1. Push the Docker image to Docker Hub using the following command:

$ docker push pratimagautam977/weather-forecasting-react-app:latest

Your Docker image should now be available on Docker Hub.



## Deploy on AWS ECS (AWS Fargate)

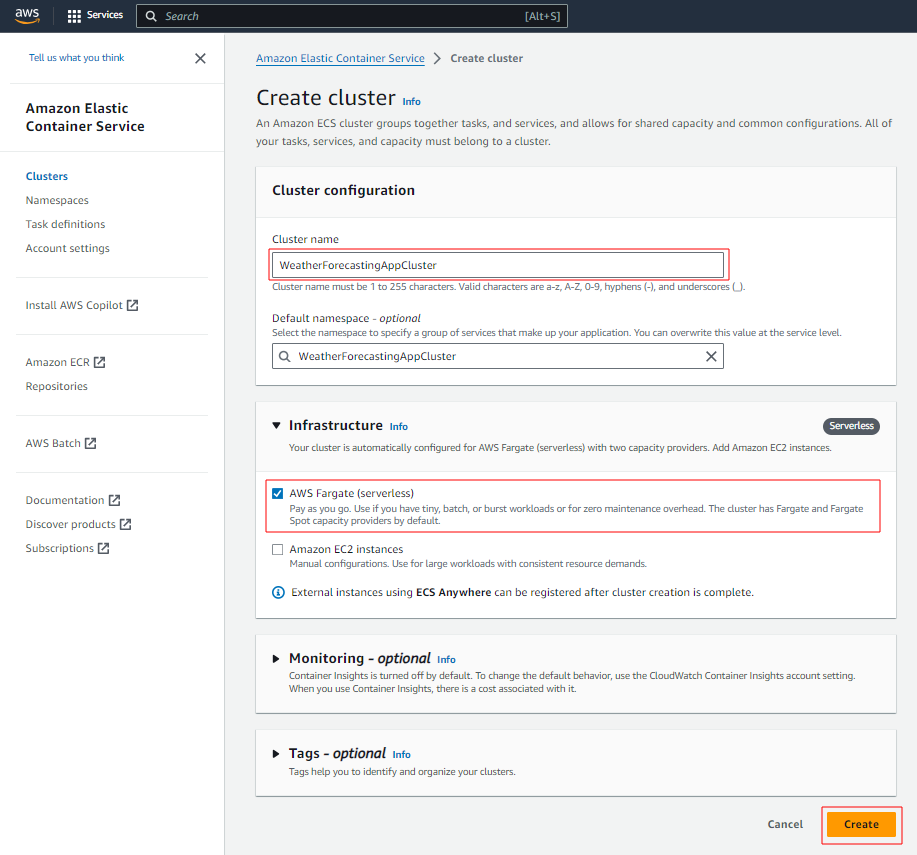
AWS Fargate cannot be configured directly as it is more an underlying technology to run serverless applications on Amazon AWS. In the next part we are going to step into each part that needs configuration to get a React application running.

Once our container image has been pushed, we are now ready to use ECS and all the awesomeness it has to offer.

Head over to Amazon Elastic Container Service (Amazon ECS) and create a cluster.

### Create the ECS cluster

1. Go to the [AWS Academy Learner Lab](https://awsacademy.instructure.com/courses/78105) and click on ‘Start Lab’
2. Once the AWS Environment is ready, click on the link to Head over to your AWS Management Console, open *Services*, type *ECS* and click on *Elastic Container Service*.
3. On the left side menu click on *Amazon ECS > Clusters* and hit the *Create Cluster* button.

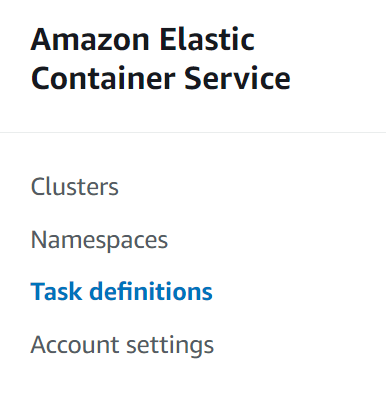


1. Once the Cluster has been created, select 'View Cluster'.

### Task Definition for ECS

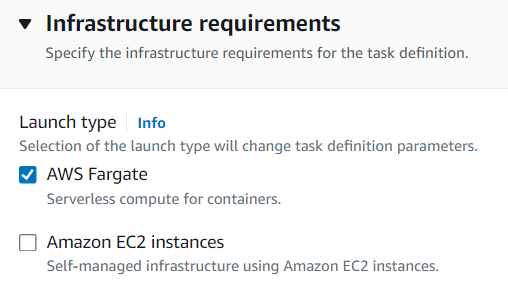
In ECS, the basic unit of a deployment is a task, a logical construct that models one or more containers. This means that the ECS APIs operate on tasks rather than individual containers. In ECS, you can’t run a container: rather, you run a task, which, in turns, runs your container(s). A task contains one or more containers.

In the navigation pane, choose “Task Definitions”.

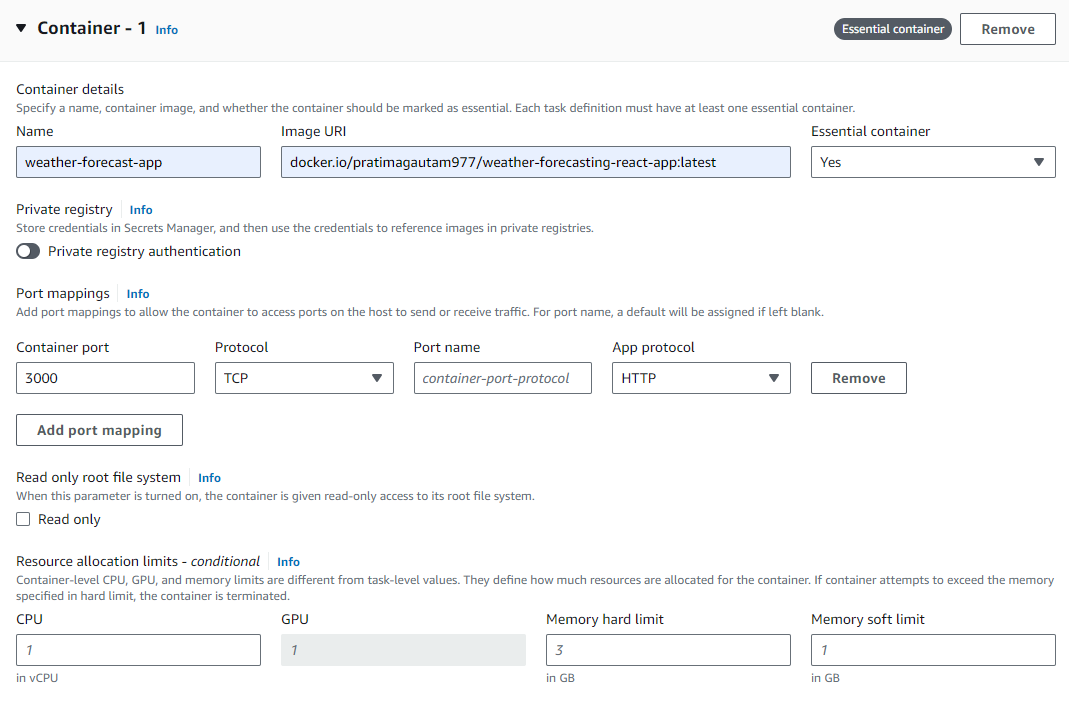


#### Task definition Configuration

1. On the Create new task definition menu, choose "Create new Task Definition".
2. For the Task definition family, specify a unique name for the task definition. e.g., **weather-forcast-task-def**
3. For the Launch type, choose the application environment as **AWS Fargate** (which is serverless). Amazon ECS uses this value to perform validation to ensure that the task definition parameters are valid for the infrastructure type.



1. For Operating system/Architecture, keep the **default** for the task.
2. For **Network mode**, the default is **awsvpc** mode.
3. Expand the Task roles section to configure the AWS Identity and Access Management (IAM) roles for the task:
   1. For the Task role, choose the IAM role **‘LabRole’** to assign to the task. A task IAM role provides permissions for the containers in a task to call AWS API operations.
   2. For the Task execution role, choose the role **‘LabRole’**

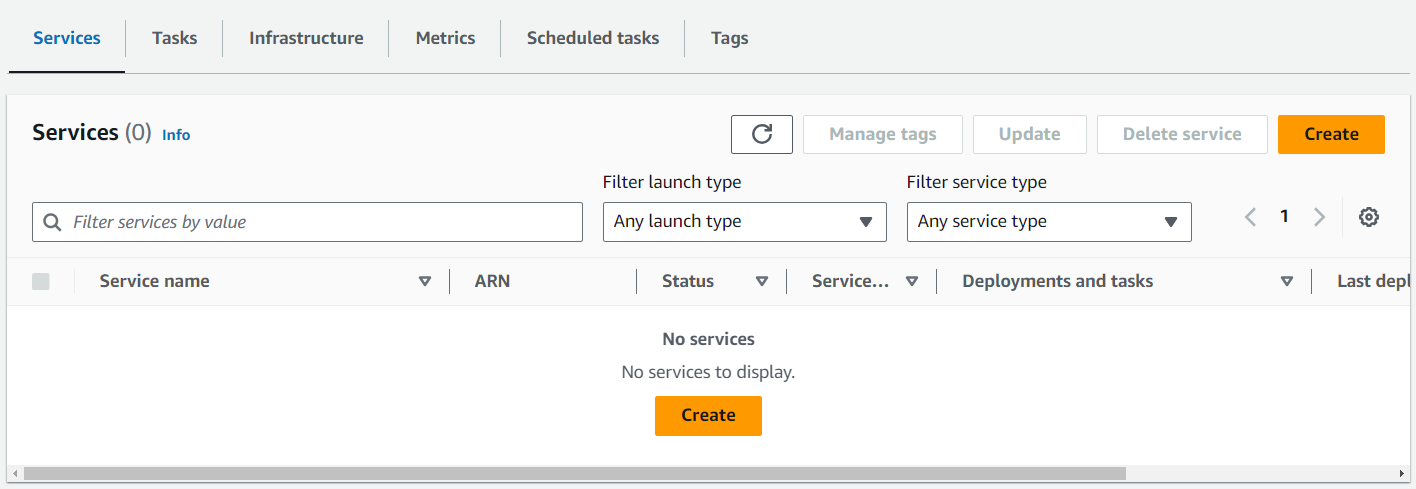


Configure the container details in your task definition. Set the values as follows:

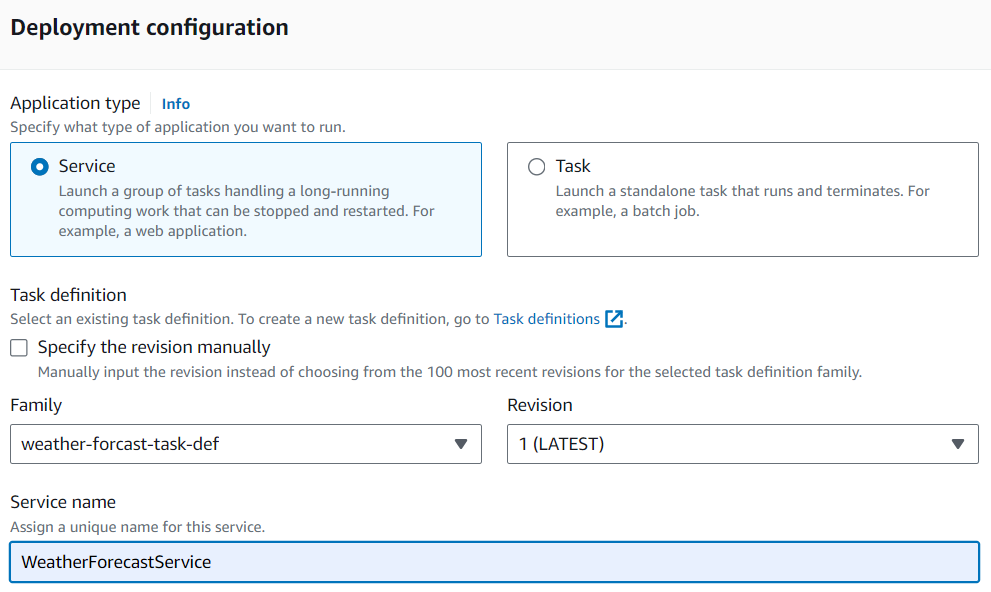
1. For **Name**, enter a name for the container. e.g. **weather-forecast-app**
2. For **Image URI**, enter the image location to use to start a container. Images in the DockerHub can be specified by [docker.io/dockerhub\_username/dockerhub\_repository:latest](http://docker.io/dockerhub_username/dockerhub_repository:latest).   
   e.g [docker.io/pratimagautam977/weather-forecasting-react-app:latest](http://docker.io/pratimagautam977/weather-forecasting-react-app:latest)
3. A port mapping allows the container to access ports on the host to send or receive traffic. Under Port mappings, use container port  **'3000'** with **TCP** protocol for the React application.
4. Once done, click “Create.

### Create ECS Service to deploy the Docker container.

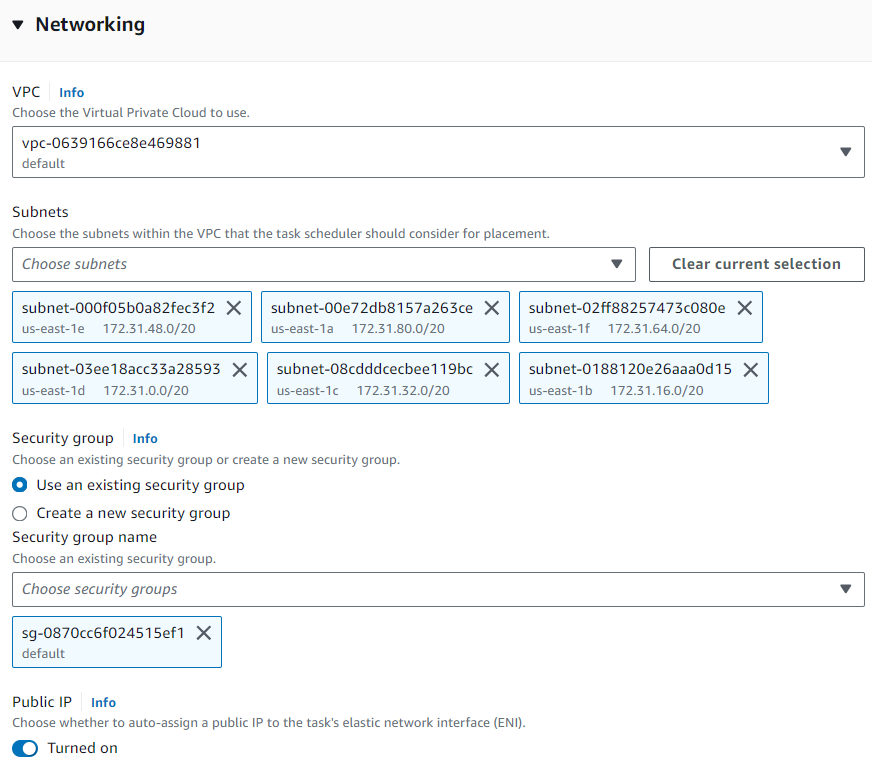
1. Go back to your ECS cluster, and select the cluster created on **section 3.1**
2. Click on “Create” in the “Services” section of your ECS cluster.



1. Keep everything as default, and go to “Deployment Configuration”.
2. At “Family”, choose the task definition you created in the previous step, set the desired number of tasks (for this example you can keep it as one), assign a “Service Name”, and configure the load balancing options if necessary.



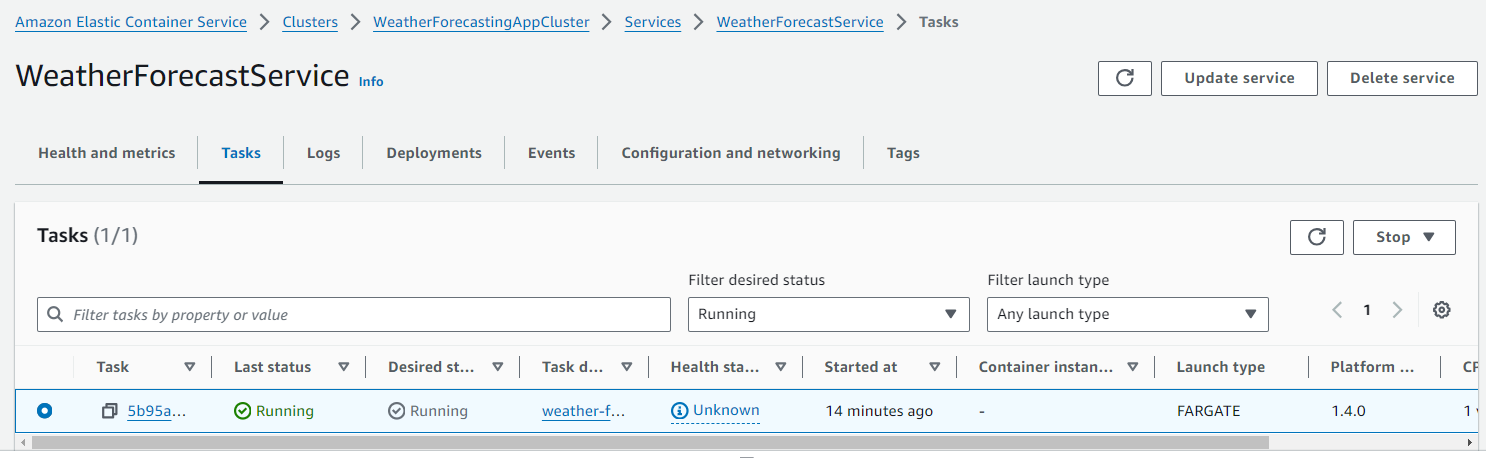
1. Go to “Networking”, and choose the right “Subnets” and “Security Groups”.



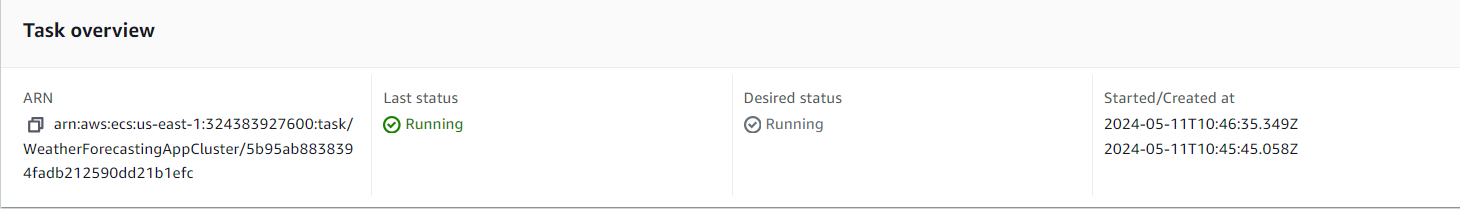
Note: Make sure that the security group allows the right “L4 ports” to connect to your container.

1. Click “Create Service.”

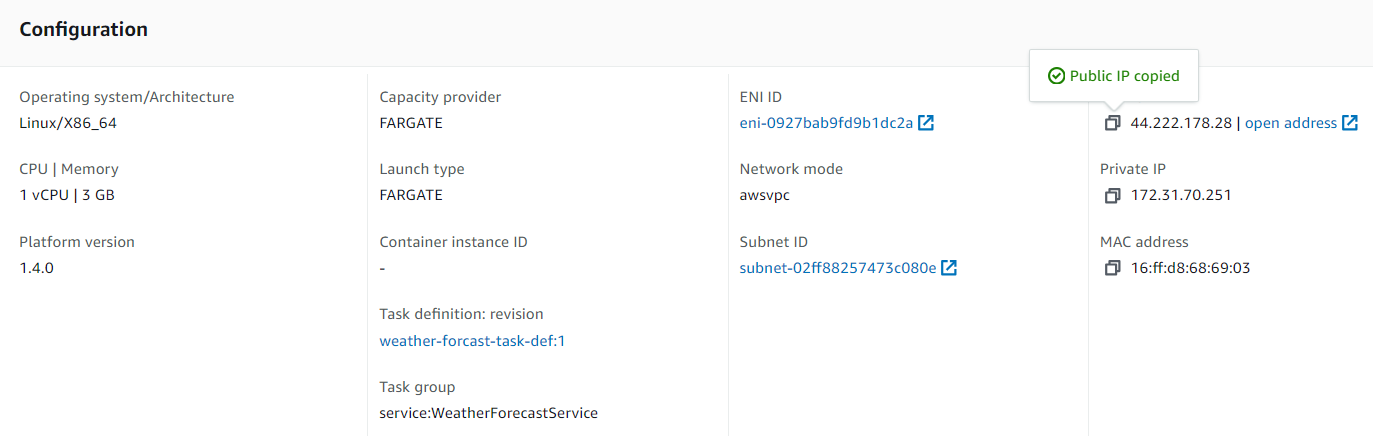
Once the Service gets deployed successfully, it will create a task. Click on ‘View Service’ and you can check the task by going to the “Task” tab, and click on the Task ID.



This will show you the state of the task:

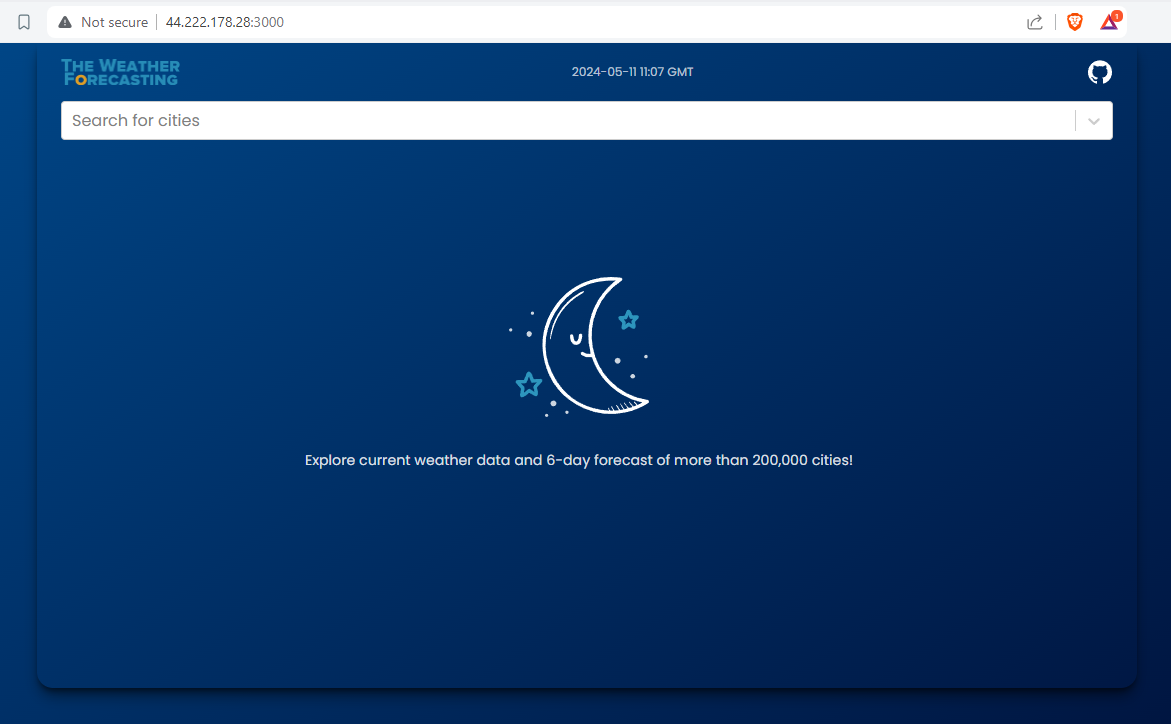


Once the Task is running, go to “Configuration” and connect to your container via the Public IP:



If your container is running a web application, go to your browser and make sure you select the right port (if you are not able to connect, check the security group inbound rules):

This should load your website properly:



Congratulations! You’ve successfully deployed a Docker container in AWS using ECS and Fargate. Docker containers and AWS ECS provide an efficient and scalable way to manage and run applications.