Module 4: AST-3

TITLE: Continuous deployment of FastAPI app on EC2

LEARNING OBJECTIVES:

You will be able to understand and implement the following aspects:

1. Understand the working of an EC2 instance
2. Connect to the EC2 instance and install Docker on it
3. Configure the EC2 instance to add as GitHub Runner
4. Run deployment workflows using GitHub Actions

INTRODUCTION

Runners: A runner is a GitHub Actions server. They are the machines that execute jobs in a GitHub Actions workflow. It listens for available jobs, runs each in parallel, and reports back progress, logs, and results. For example, a runner can clone your repository locally, install testing software, and then run commands that evaluate your code.
Each runner can be hosted by GitHub or self-hosted on a localized server.

- GitHub Hosted runners: Provided by GitHub and are based on Ubuntu Linux, Windows, and macOS.
- Self-hosted runners: You can host your own runners and customize the environment used to run jobs in your GitHub Actions workflows.

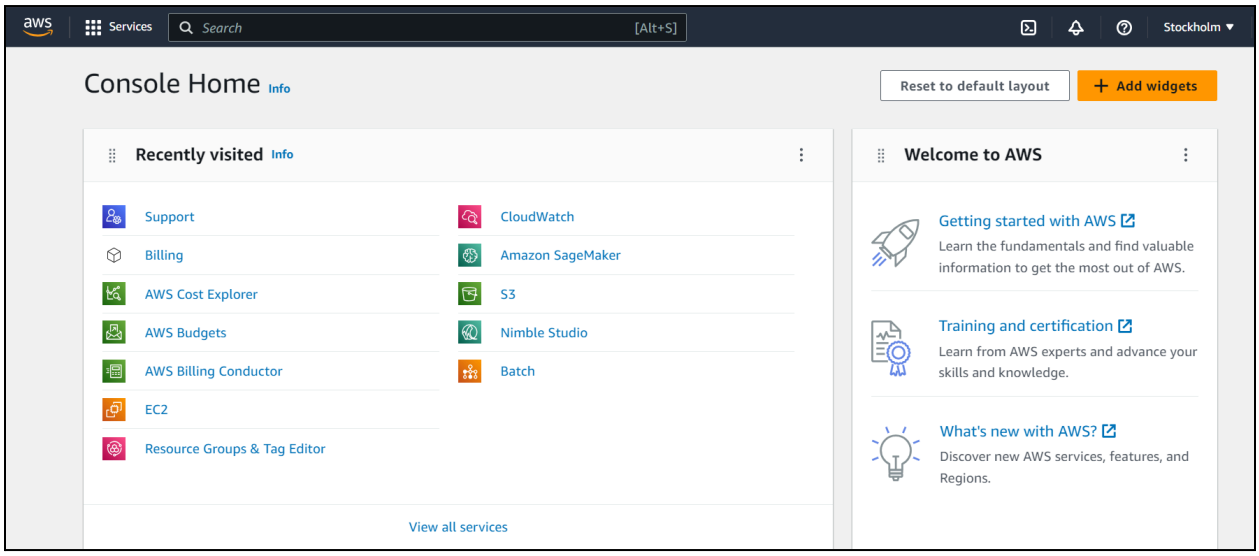
Docker Hub: is a service provided by Docker for finding and sharing container images.
It's the world's largest repository of container images with an array of content sources including container community developers, open source projects and independent software vendors (ISV) building and distributing their code in containers.

Procedure in a nutshell:

1. Login to AWS and create an EC2 instance
2. Add port to Security group
3. Connect to the EC2 Instance
4. Install Docker on EC2 instance
5. Add EC2 instance as a Self-hosted Github Runner
6. Create CD workflow and trigger it
7. Access App running on the public IP of the instance at the specified port

Login to AWS:

https://aws.amazon.com/console

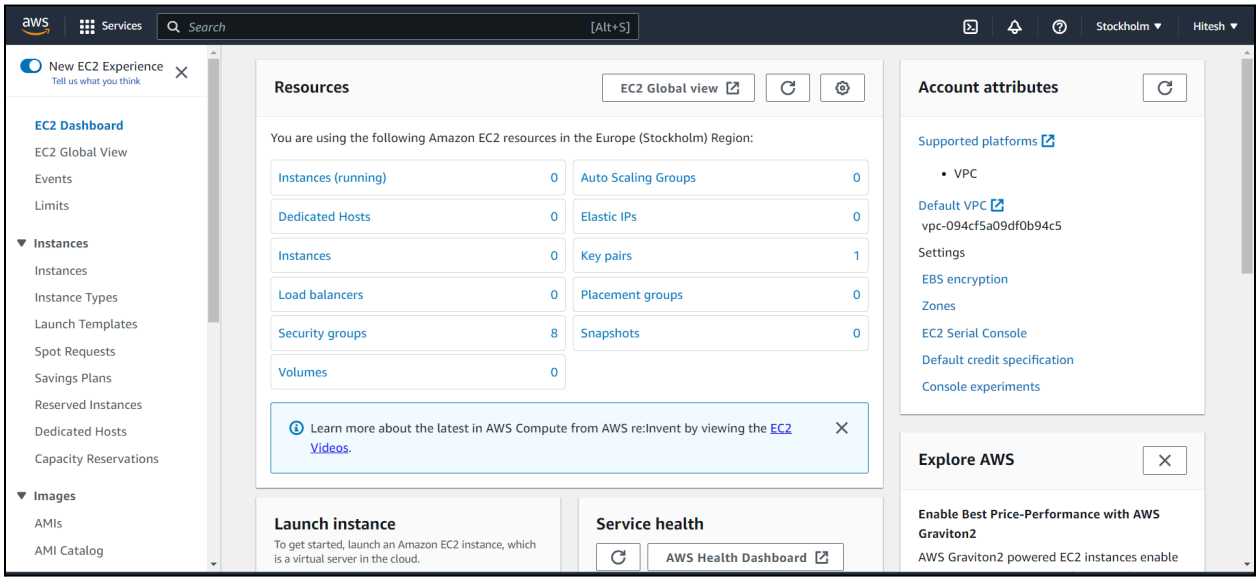


Create an EC2 instance:

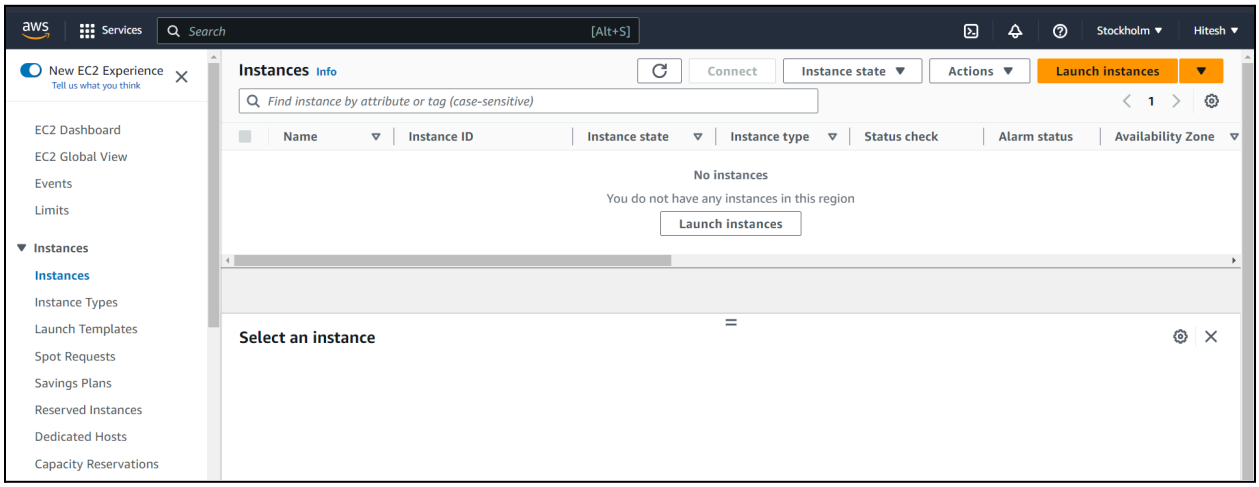
Amazon EC2 is a web service that provides scalable compute capacity in the cloud. It allows to quickly provision virtual servers, known as *instances*, and easily scale them up or down based on the needs. EC2 offers a wide selection of instance types to accommodate various workloads and applications.

Steps to create an EC2 instance:

1. Navigate to EC2 dashboard

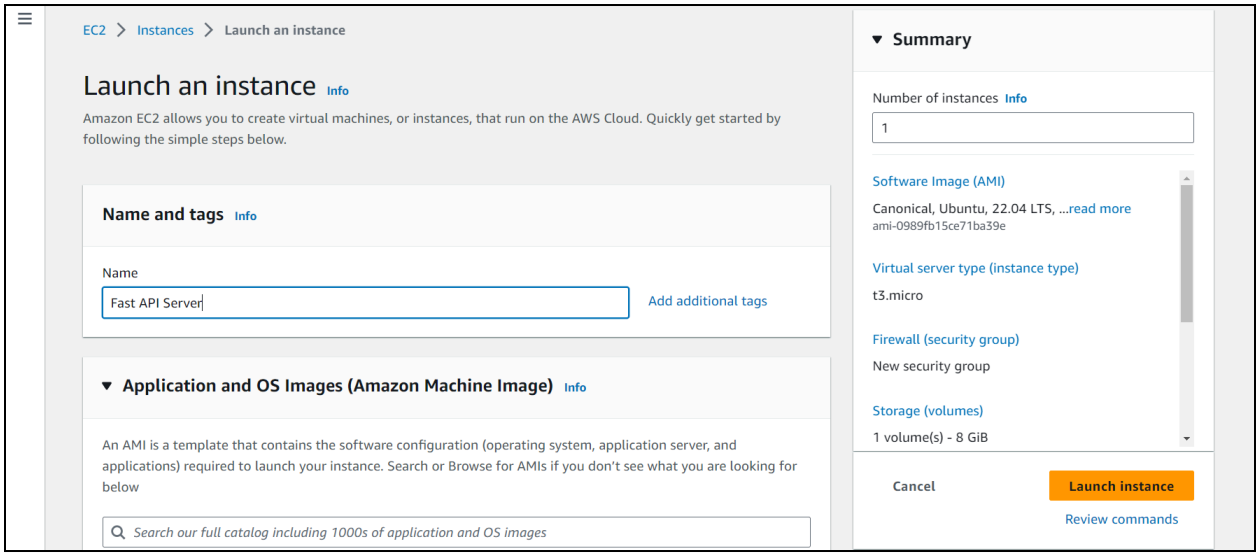


2. From left navigation panel go to *Instances*

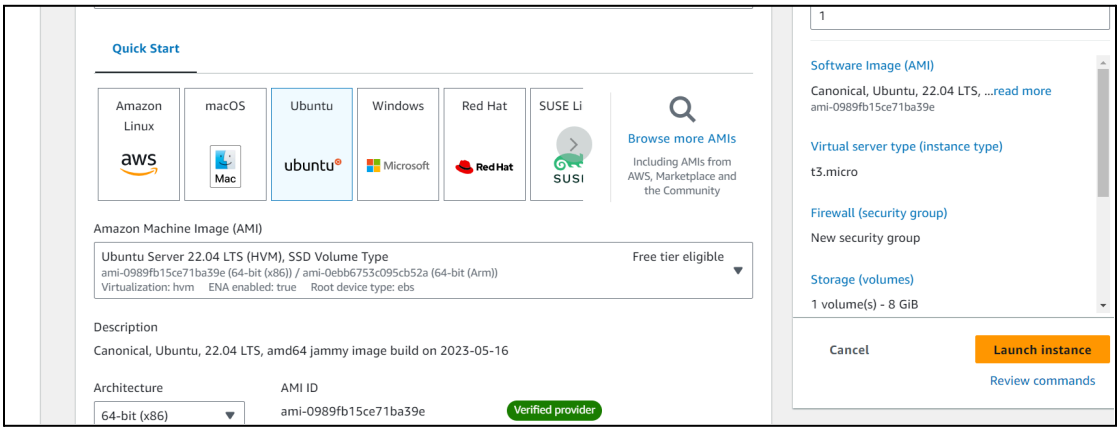


3. Click on *Launch instances*

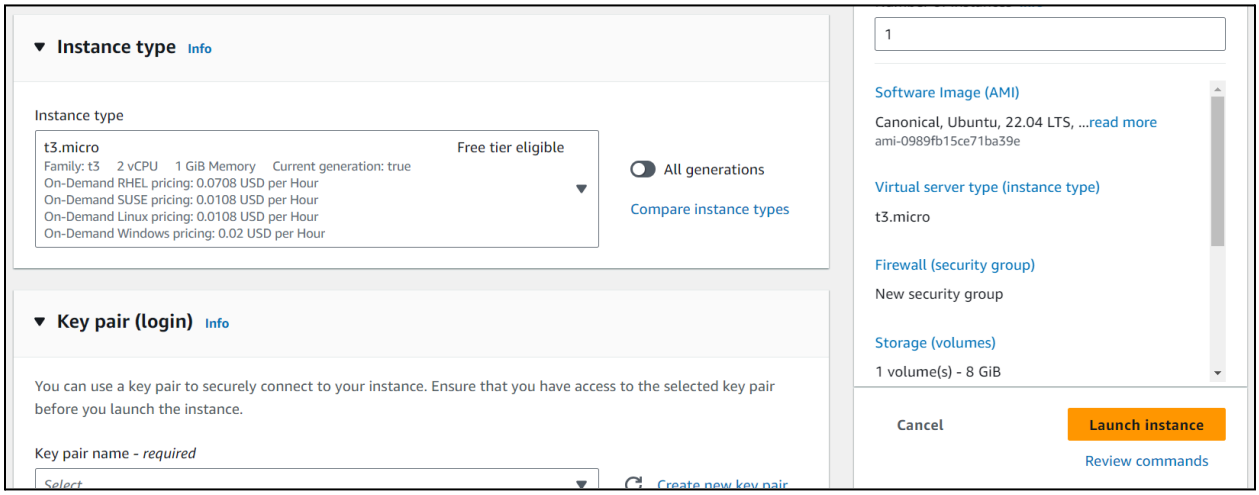
4. Give the name of the instance



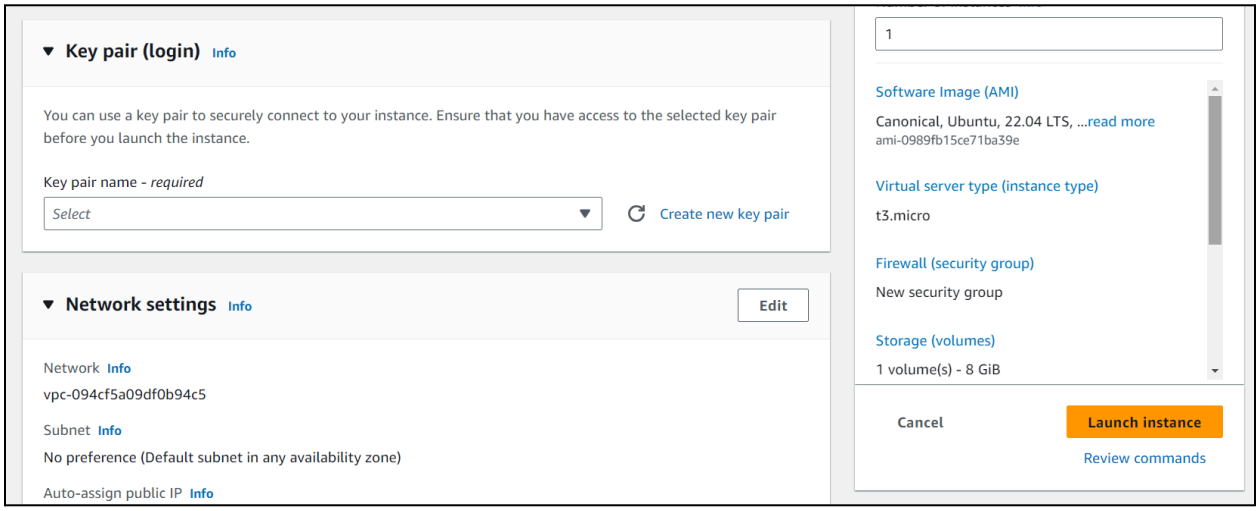
5. Select Ubuntu server (free-tier)



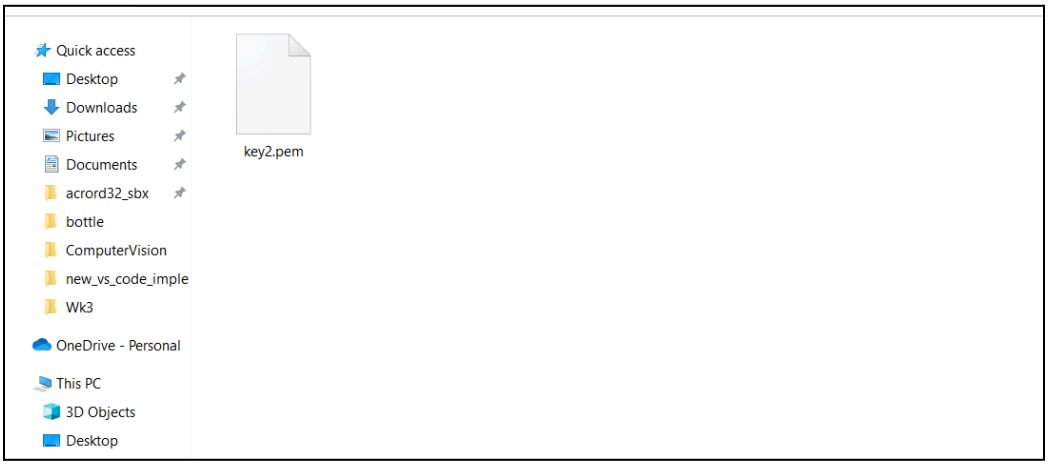
6. Select a free tier Instance Type: *t2.micro* (or *t3.micro* in the Regions in which *t2.micro* is unavailable)



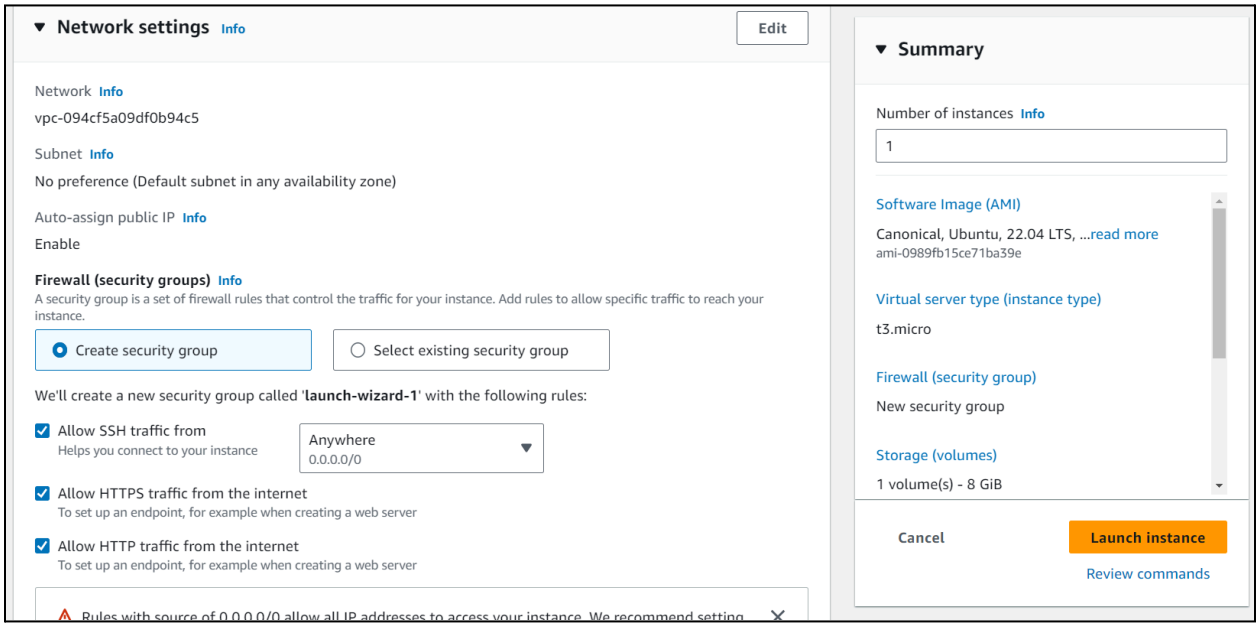
7. Create a new key pair. A key pair is used to securely connect to your instances.



8. Download and safely store the *.pem* file on your system

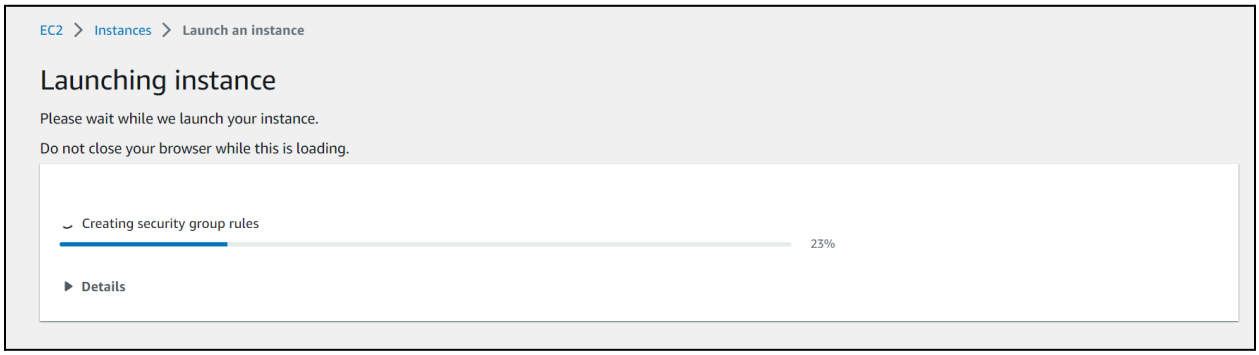


9. Under Network settings, select *Create security group*, and allow SSH, HTTPS, HTTP traffic

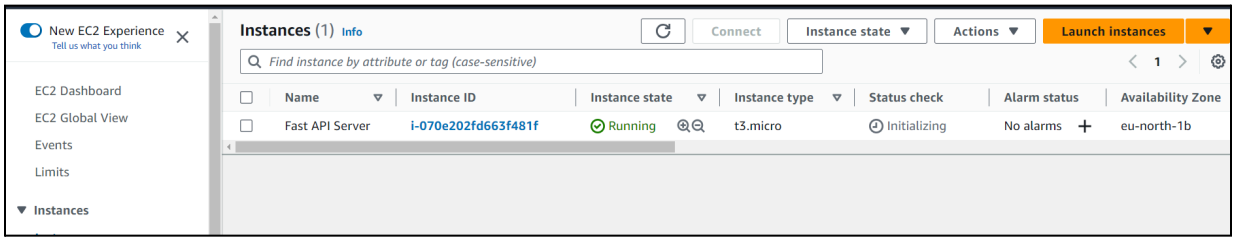


10. Recheck the configuration

11. Launch Instance



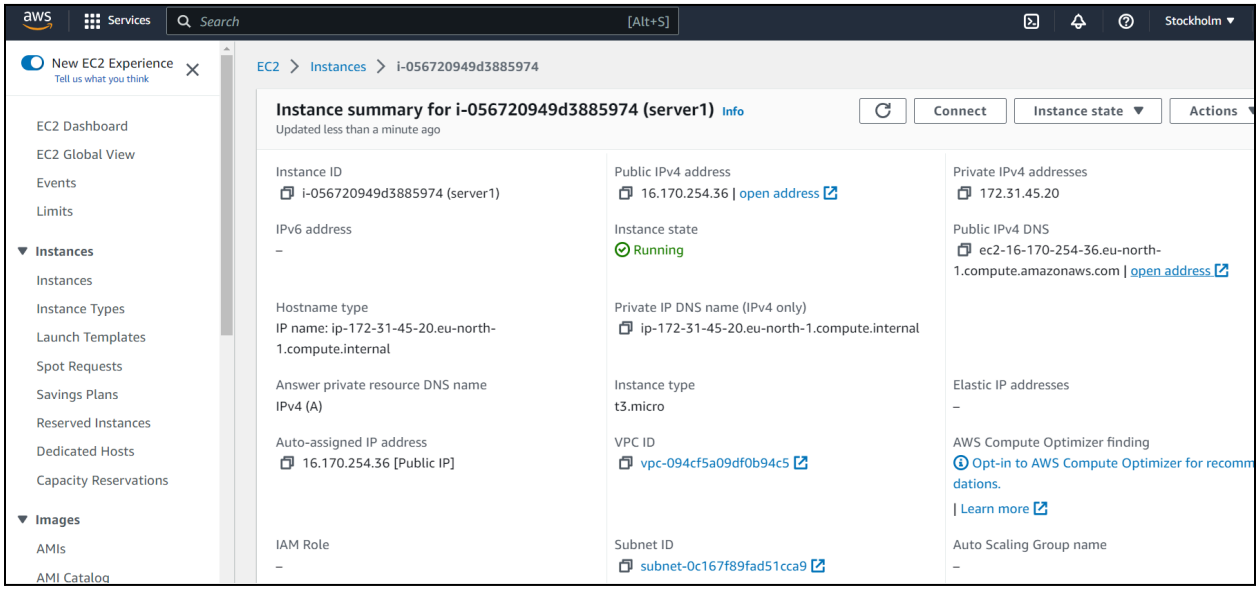
12. Check the status of the instance, it should be Running



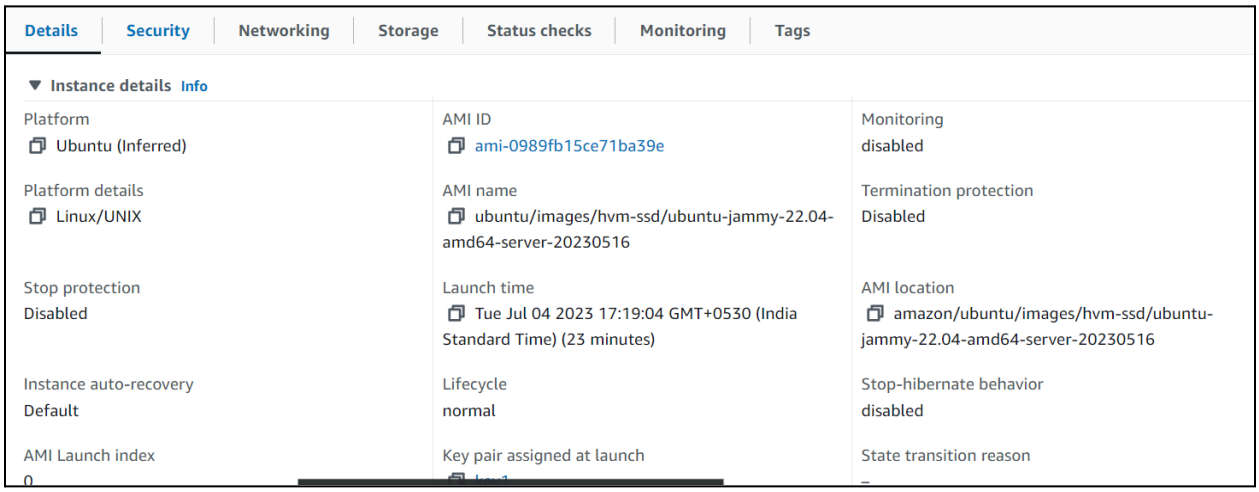
Add port to Security group:

When the FastAPI application will be launched on the EC2 instance, it will be running on port 8001 (as specified inside the titanic_model_api/app/main.py file). Once running, in order to access the application, the traffic needs to be allowed for this port.

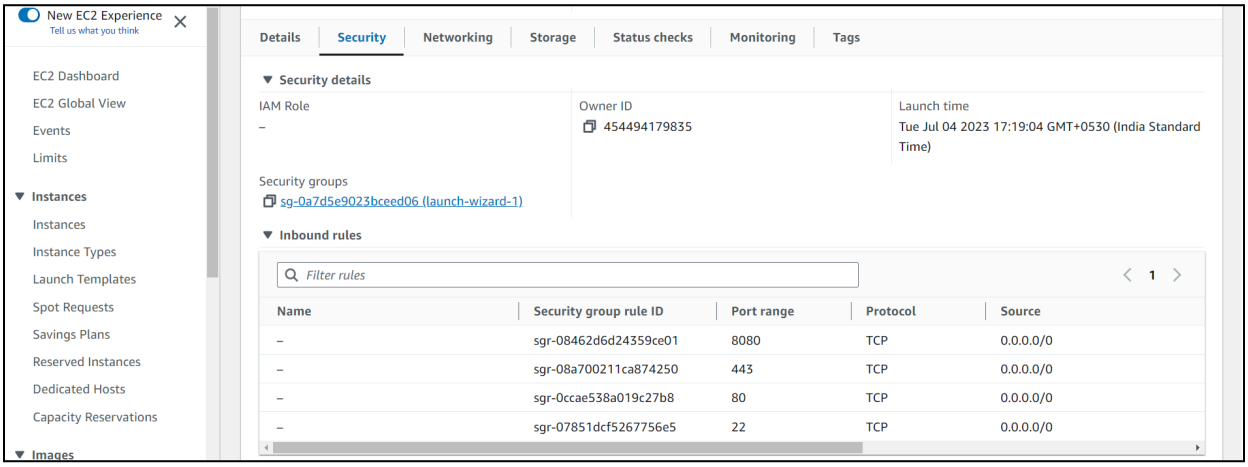
1. Click on the *instance ID* of your instance to see the details



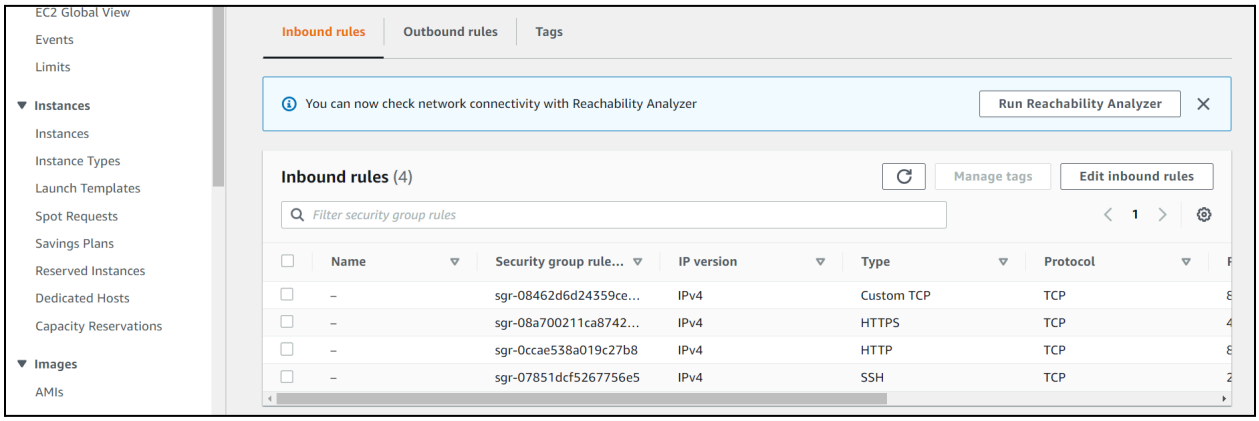
2. Scroll down and click on the *Security* tab



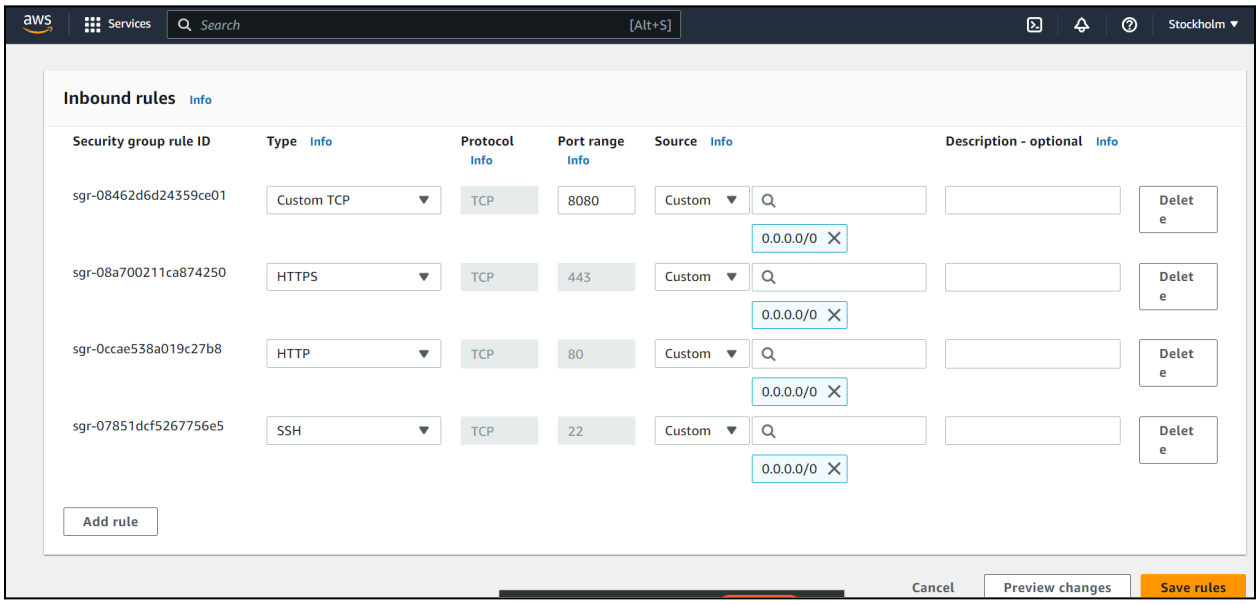
3. Select the security group associated with your instance, in this case launch-wizard-1



4. Scroll to find the inbound and outbound rules



5. Select *Edit inbound rules*



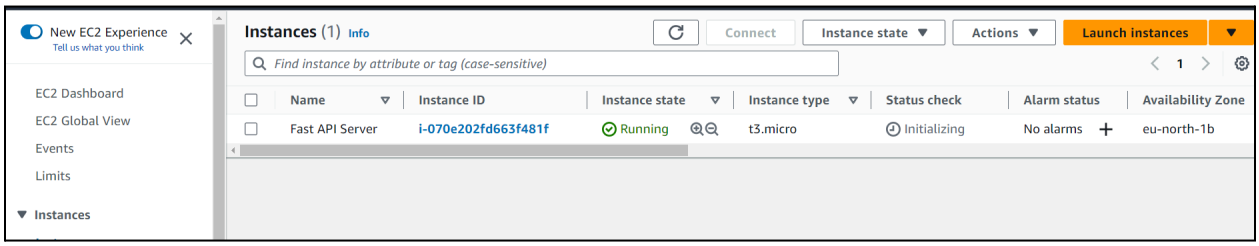
6. To add a new rule, click on *Add rule*. Provide information such as
   - Type: Custom TCP
   - Port range: 8001 (specifying this port as our FastAPI application will launch at this port)
   - Source: Anywhere-IPv4

7. Select *Save rules*

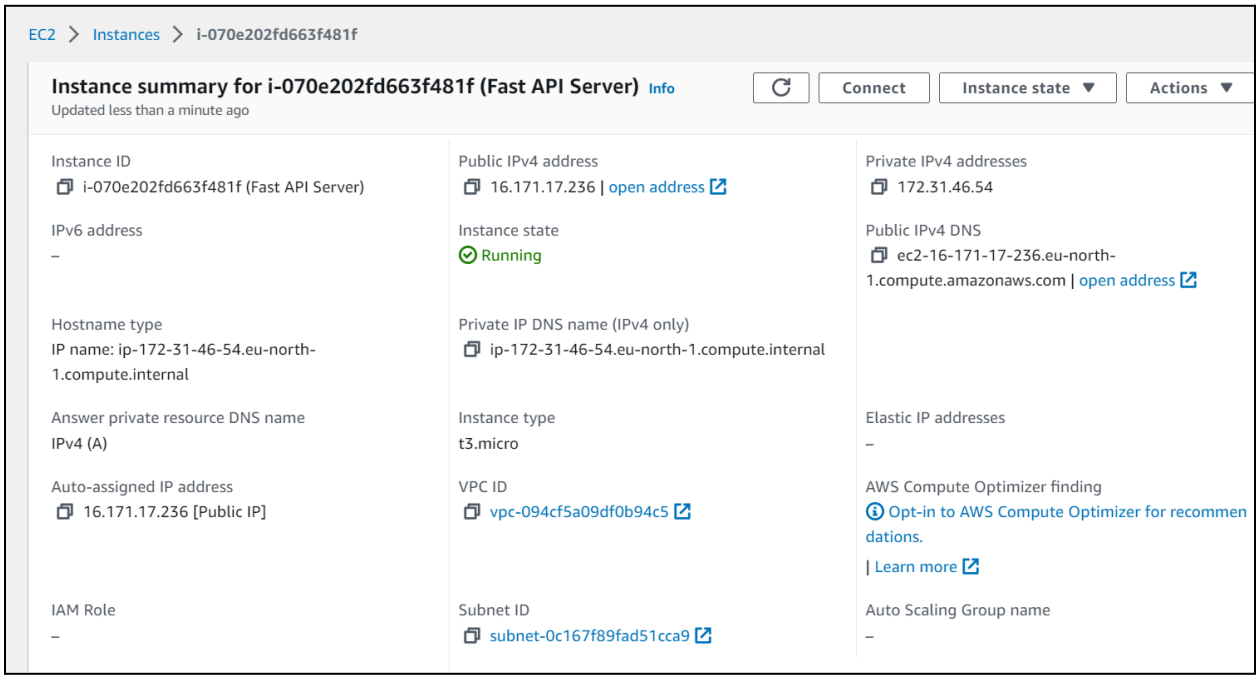
Connect to the EC2 Instance:

1. Go to *Instances* on EC2 dashboard

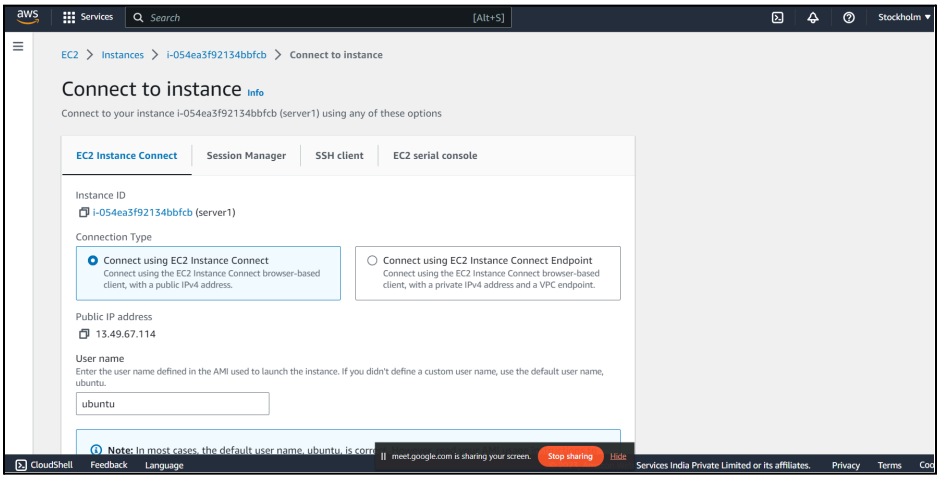
2. Click on the instance ID



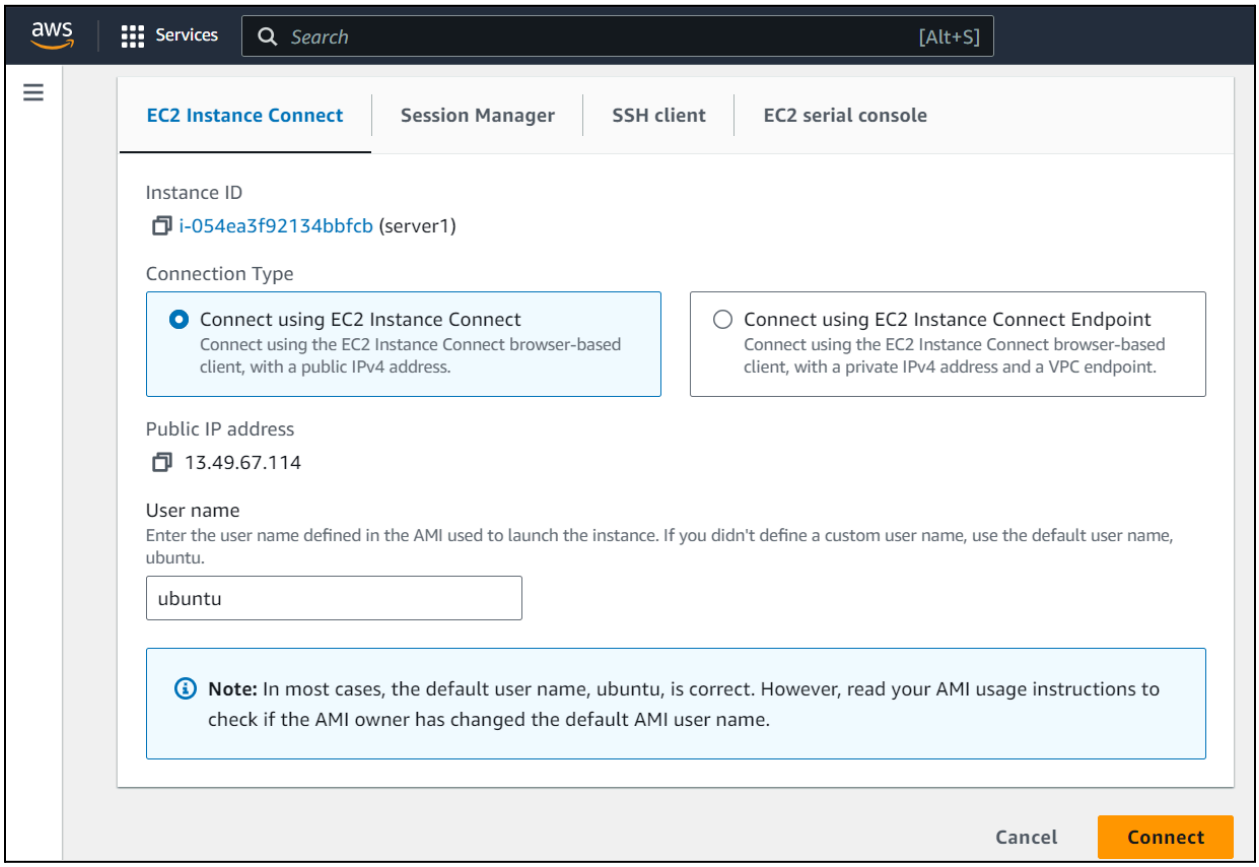
3. Click on *Connect*



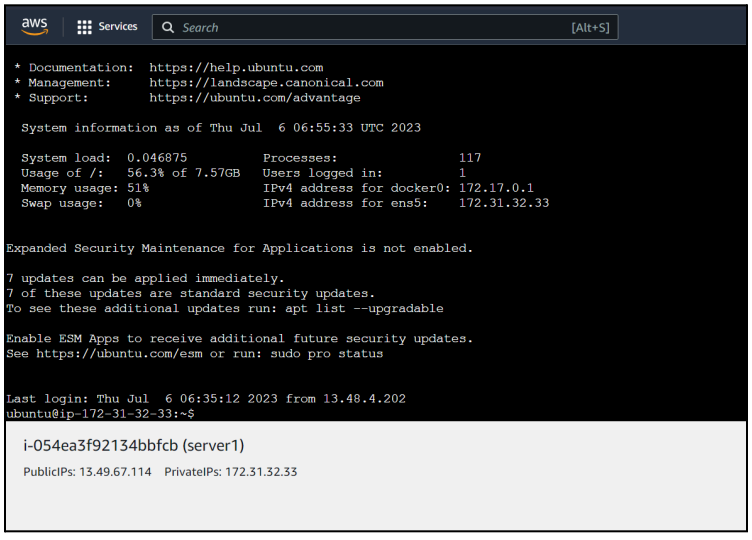
4. Select EC2 instance connect



5. Connect using EC2 instance connect and press the connect button



6. Shell for the instance will open in a new tab:



Install Docker on virtual machine / EC2 instance:

1. Open the terminal.
2. Remove any Docker files that are running in the system, using the following command:

   sudo apt-get remove docker docker-engine docker.io

   NOTE that we are using the *sudo* keyword along with commands.
   Using *sudo*, users no longer had to change to the root user or log into that account to run administrative commands (such as installing software). Users could run those admin activities through sudo with the same effect as if they were run from the root user account.

3. Get the system up-to-date using the following commands.
   It will update the package index files on the instance, which contain information about available packages and their versions.

   sudo apt-get update
   sudo apt-get upgrade -y
   Press enter, if a popup comes saying - Services will be restarted.

4. Install Docker using the following command:

   sudo apt install docker.io -y

5. Start docker service and enable the service to start at boot using the following commands:

   sudo systemctl start docker

   sudo systemctl enable docker

6. Before testing Docker, check the version installed using the following command:

   docker --version

   ```
   ubuntu@ip-172-31-82-246:~$ docker --version
   Docker version 20.10.12, build 20.10.12-0ubuntu4
   ```

   You can also check full version details for Docker Client and Server using:

   sudo docker version

   ```
   ubuntu@ip-172-31-45-39:~$ sudo docker version
   Client:
    Version:           24.0.5
    API version:       1.43
    Go version:        go1.20.3
    Git commit:        24.0.5-0ubuntu1~22.04.1
    Built:             Mon Aug 21 19:50:14 2023
    OS/Arch:           linux/amd64
    Context:           default
   ```

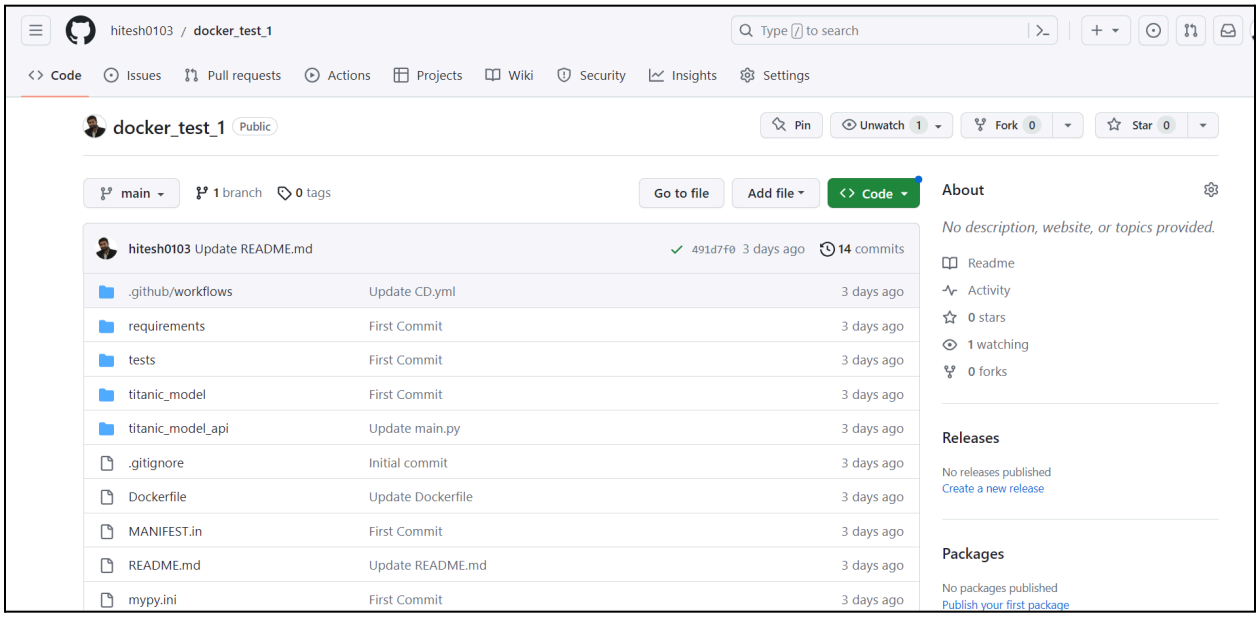
```
Server:
 Engine:
  Version:          24.0.5
  API version:      1.43 (minimum version 1.12)
  Go version:       go1.20.3
  Git commit:       24.0.5-0ubuntu1~22.04.1
  Built:            Mon Aug 21 19:50:14 2023
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.2
  GitCommit:
 runc:
  Version:          1.1.7-0ubuntu1~22.04.2
  GitCommit:
 docker-init:
  Version:          0.19.0
  GitCommit:
```

Add EC2 instance as a Self-hosted Github Runner:

A self-hosted runner is a system that you deploy and manage to execute jobs from GitHub Actions. They offer more control of hardware, operating system, and software tools than GitHub-hosted runners provide. With self-hosted runners, you can create custom hardware configurations that meet your needs with processing power or memory to run larger jobs, install software available on your local network, and choose an operating system not offered by GitHub-hosted runners.
Self-hosted runners can be physical, virtual, in a container, on-premises, or in a cloud.

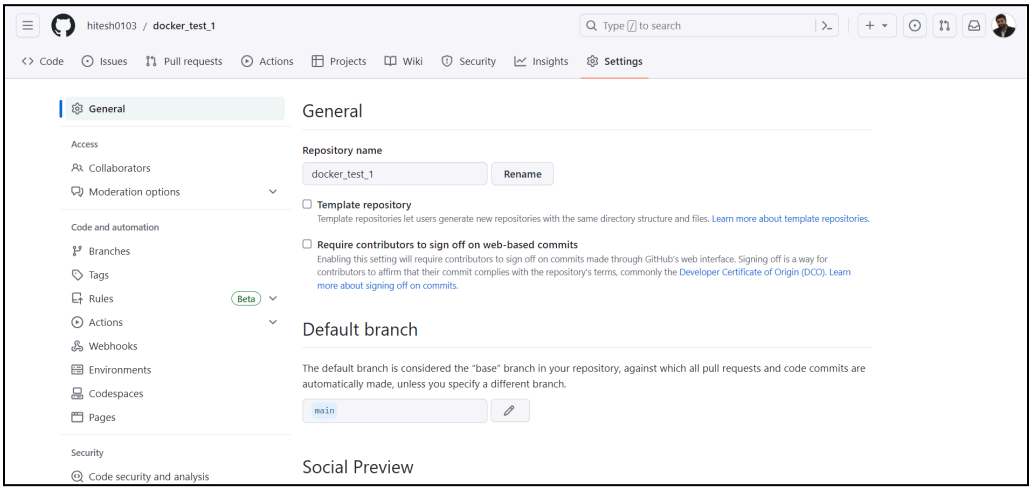
Steps to configure EC2 instance as a self-hosted runner:

1. Create a new repository or go to your existing GitHub repository

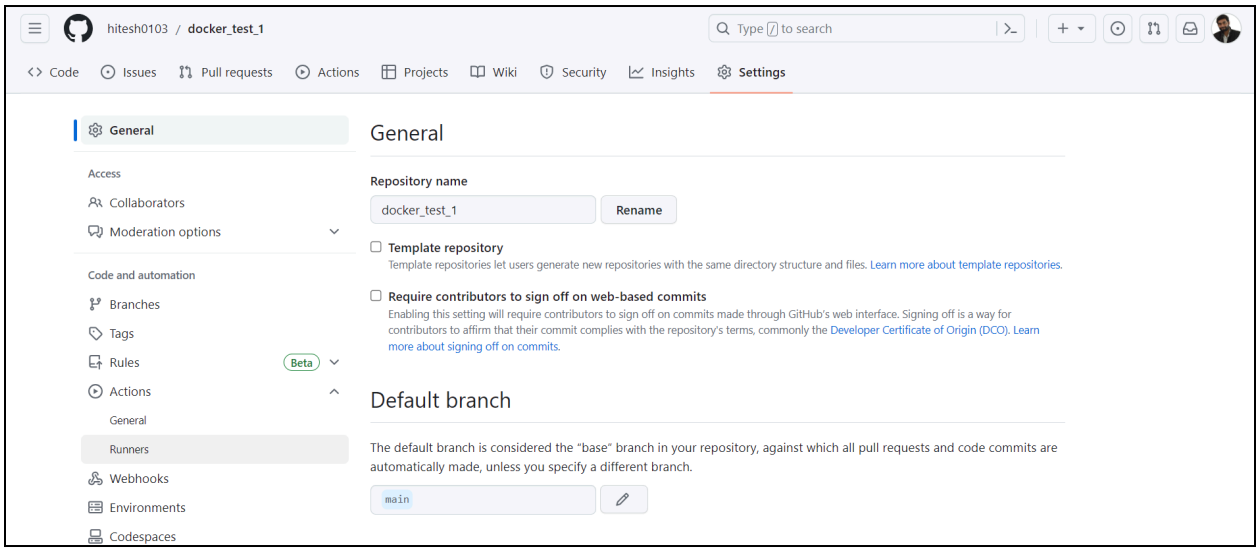


2. Add DockerHub credentials in the repository secrets

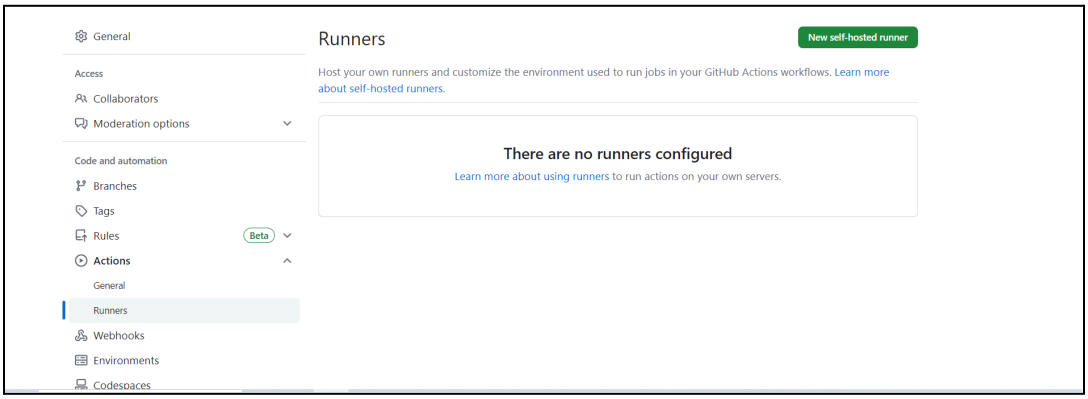
3. Go to *Settings*



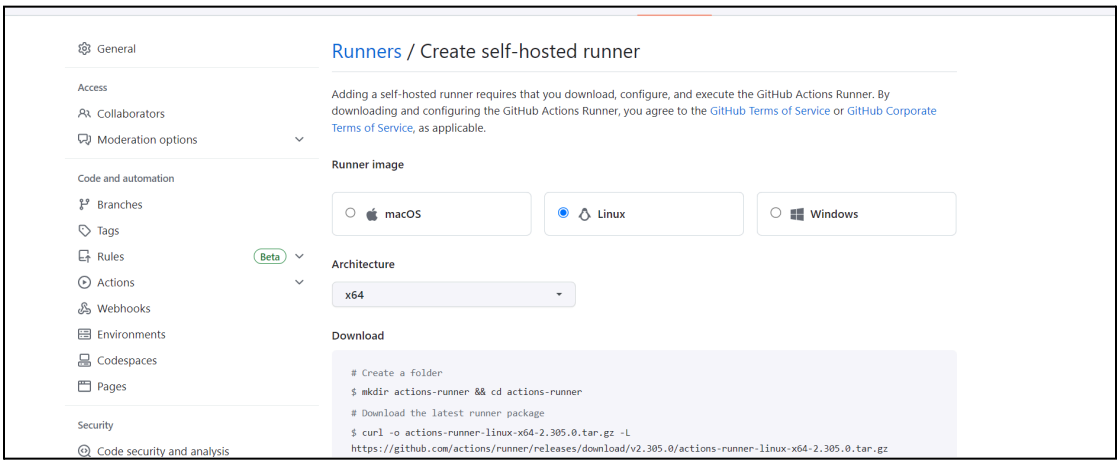
4. On the left navigation panel, select *Actions -> Runners*



5. Select *New self-hosted runner*



6. Select Runner Image: *Linux*, Architecture: *x64*



7. After that, commands related to downloading, configuration, and using the runners, will be listed below. These commands need to be executed one by one in the EC2 instance shell.
8. Download:
   Create a folder:

   mkdir actions-runner && cd actions-runner

   Download the latest runner package:

   Use the command showing in your repository and run

   Validate the hash (Optional):

   Use the command showing in your repository and run

   Extract the installer:

   Use the command showing in your repository and run

9. Configure:

Create the runner and start the configuration experience:

Use the command showing in your repository and run

It will require a few details. Press enter to use the defaults, recommended to use defaults.

10. Last step, run it!:

    `./run.sh &`

    '&' is used to execute it in the background

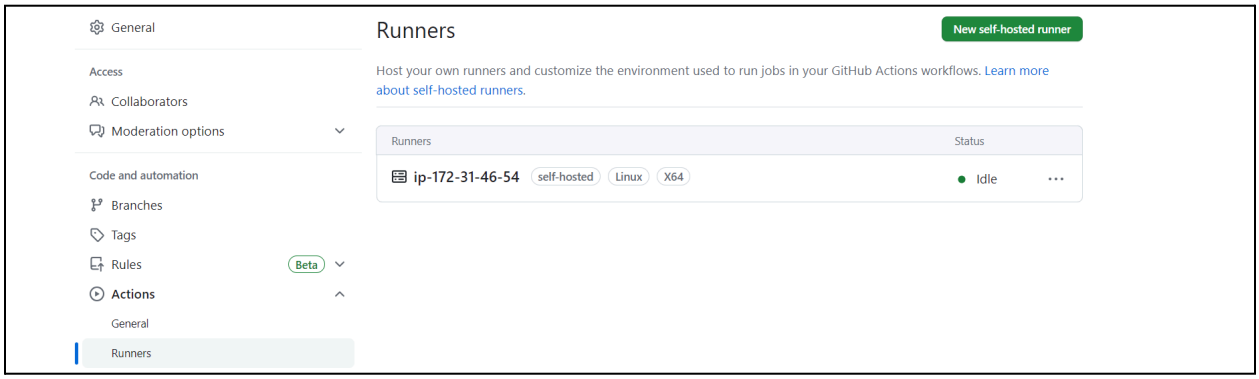
    It will output

    `Connected to GitHub`

    `Listening for Jobs`

```
ubuntu@ip-172-31-45-39:~/actions-runner$ ./run.sh &
[1] 14018
ubuntu@ip-172-31-45-39:~/actions-runner$
√ Connected to GitHub

Current runner version: '2.312.0'
2024-02-01 10:00:07Z: Listening for Jobs
```

11. Now, go to the Runners on GitHub, your EC2 instance should be listed as a runner



12. To use this self-hosted runner, just mention the below in your workflow yml file

    `runs-on: self-hosted`

    This will execute those jobs on the self-hosted runner, in this case EC2 instance.

Create CD workflow:

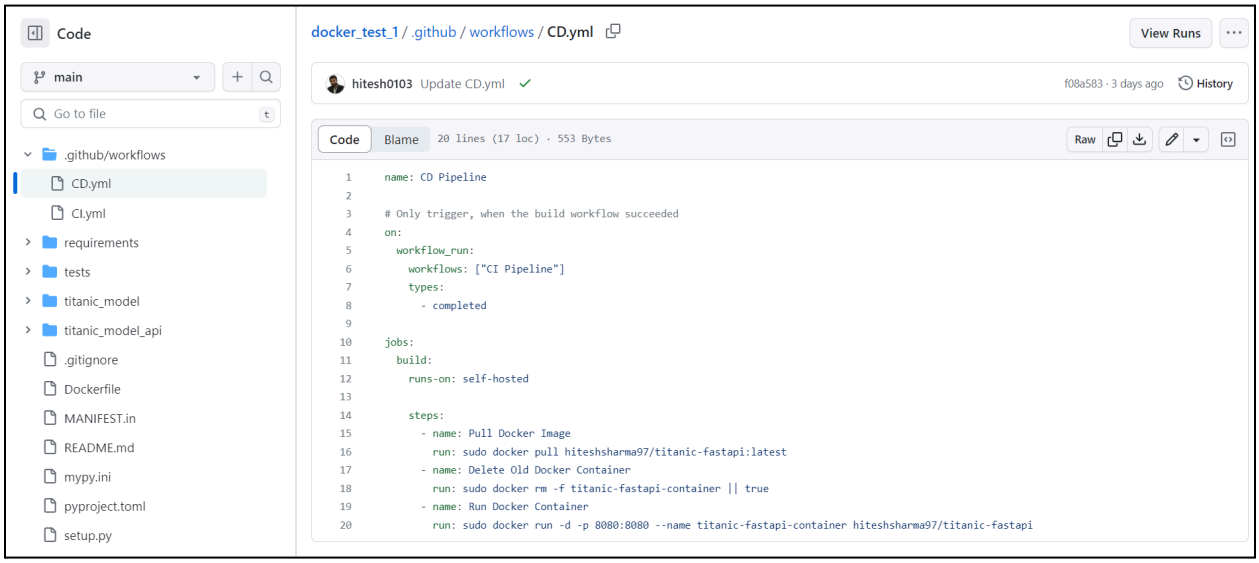
1. Configure workflow YML file of your GitHub repository to include the self-hosted runner.

   The Workflow file for this assignment is available in the drive inside '*EC2_Implementation*' folder.
   Workflow files: `.github/workflows/CI.yml`, and `.github/workflows/CD.yml`

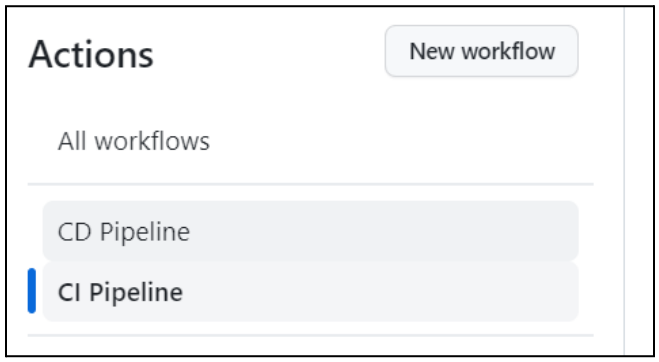
   Content of `CD.yml` file:

```yaml
name: CD Pipeline

# Only trigger, when the build workflow succeeded i.e. CI Pipeline
on:
  workflow_run:
    workflows: ["CI Pipeline"]
    types:
      - completed

jobs:
  deploy:
    runs-on: self-hosted
    steps:
    - name: Pull Docker Image
      env:
        DOCKER_USER: ${{ secrets.DOCKER_USER_NAME }}
      run: sudo docker pull $DOCKER_USER/titanic-fastapi:latest

    - name: Delete Old Docker Container
      run: sudo docker rm -f titanic-fastapi-container || true

    - name: Run Docker Container
      env:
        DOCKER_USER: ${{ secrets.DOCKER_USER_NAME }}
      run: sudo docker run -it -d -p 8001:8001 --name
titanic-fastapi-container $DOCKER_USER/titanic-fastapi:latest
```

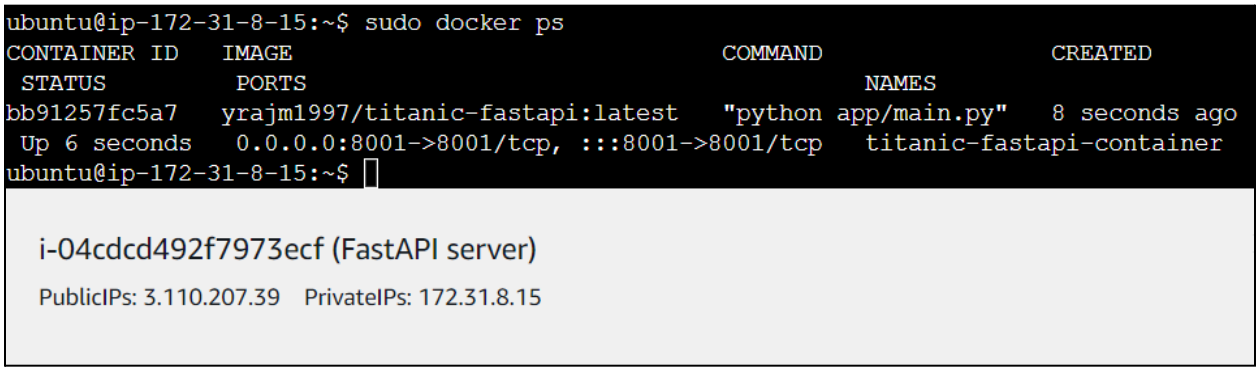
2. Add or upload the files to your GitHub repository including workflow files.



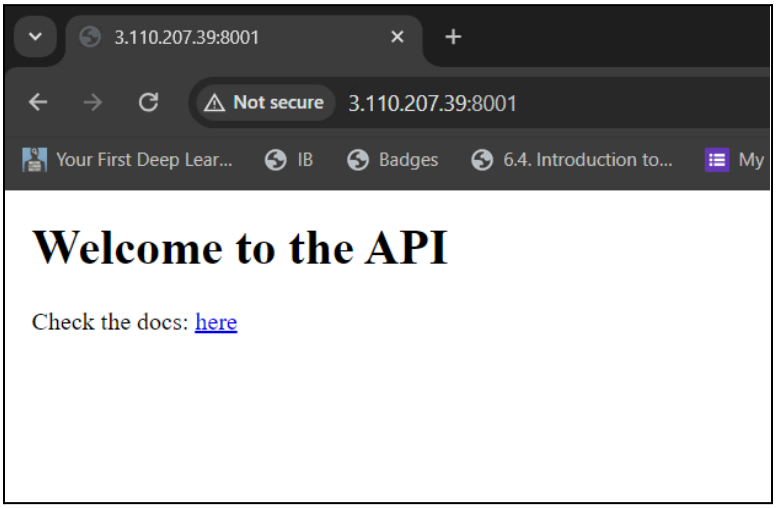
3. Once added, the workflow will run.
First the CI Pipeline workflow will run, and later the CD Pipeline will run.



4. After successful run, go to the EC2 instance and check, the container should be running
sudo docker ps



5. Access the application using the public IP address of the EC2 Instance and the port of your web app.
   For example: `http://<public-IP>:8001`



6. Once done, stop the container and exit the terminal

```
ubuntu@ip-172-31-8-15:~$ sudo docker stop titanic-fastapi-container
titanic-fastapi-container
ubuntu@ip-172-31-8-15:~$ exit
logout
```

7. Terminate the EC2 instance to prevent unnecessary charges