

## 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.

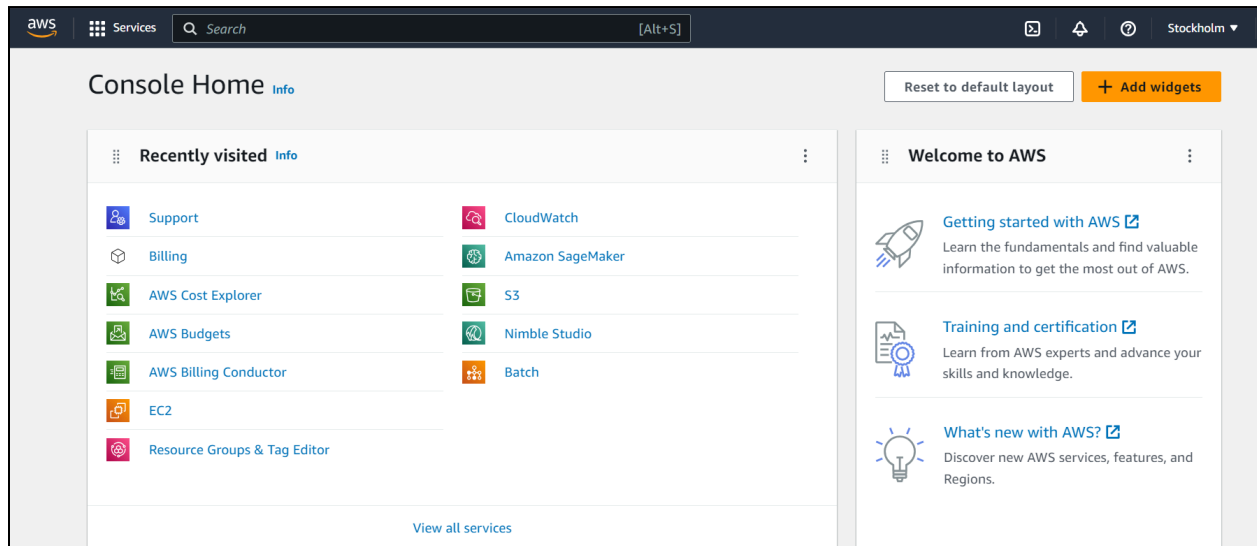
It's the world's largest repository of container images with an array of content sources including container community developers, open source projects and independent software vendors (ISV) building and distributing their code in containers.

### 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>

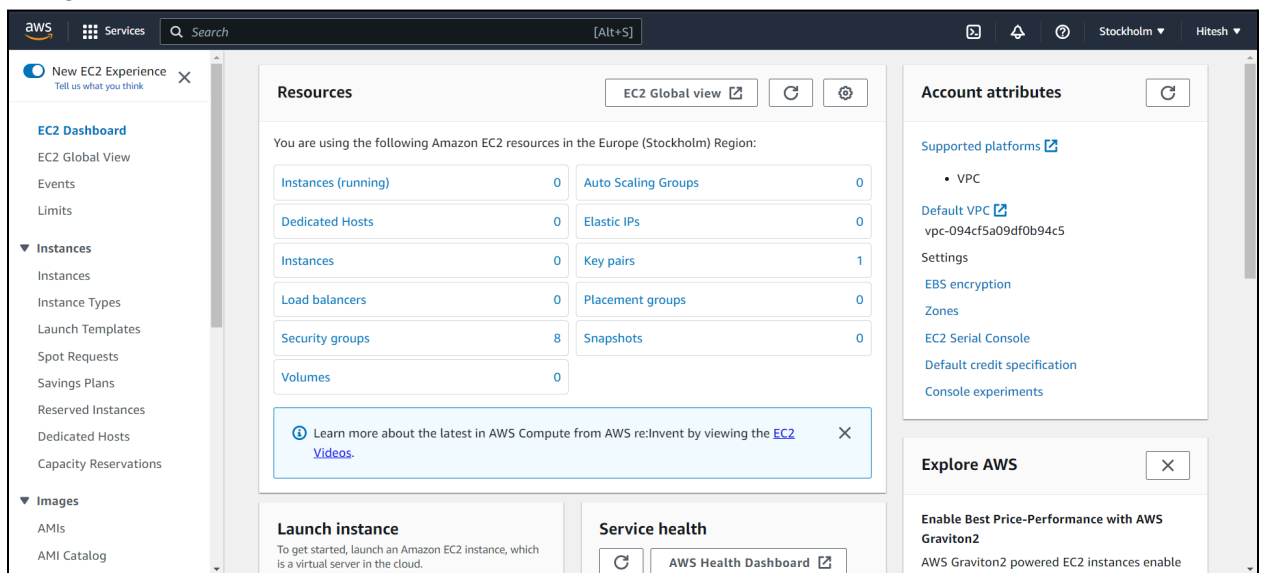


## Create an EC2 instance:

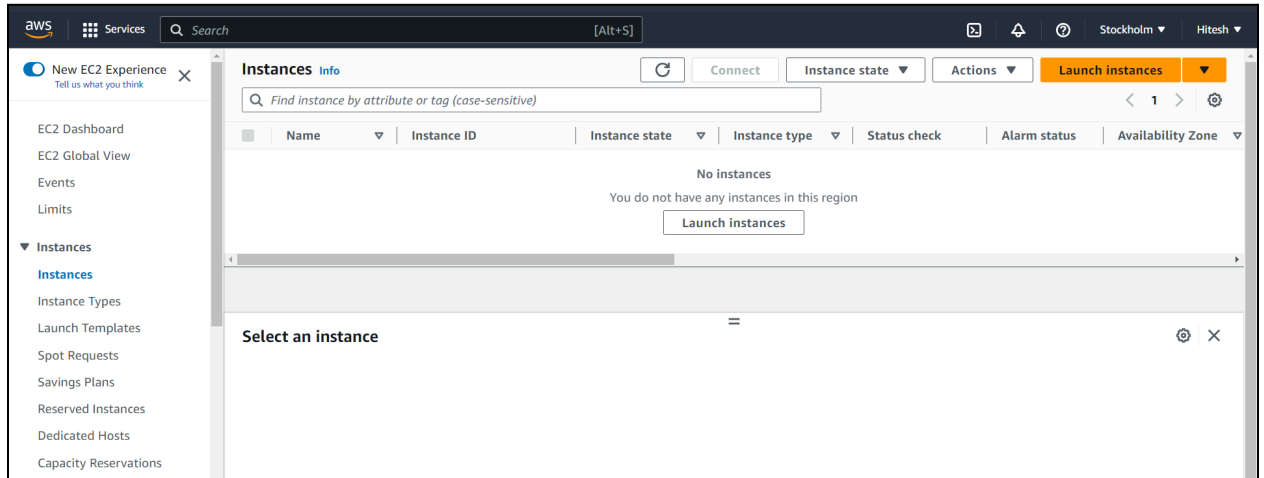
Amazon EC2 is a web service that provides scalable compute capacity in the cloud. It allows to quickly provision virtual servers, known as *instances*, and easily scale them up or down based on the needs. EC2 offers a wide selection of instance types to accommodate various workloads and applications.

Steps to create an EC2 instance:

1. Navigate to EC2 dashboard

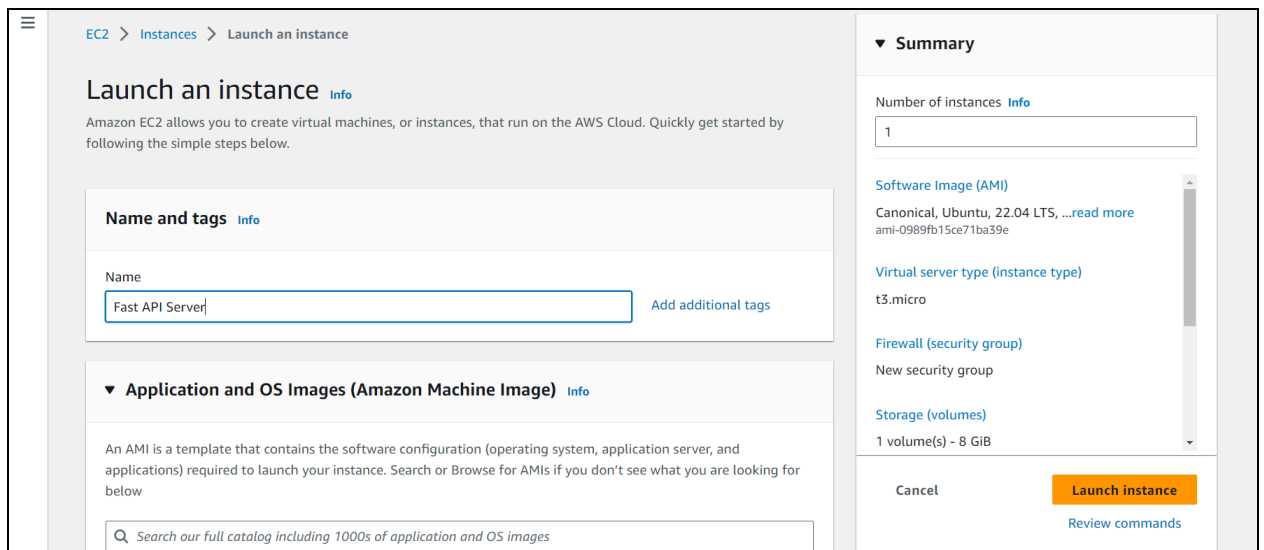


2. From left navigation panel go to *Instances*

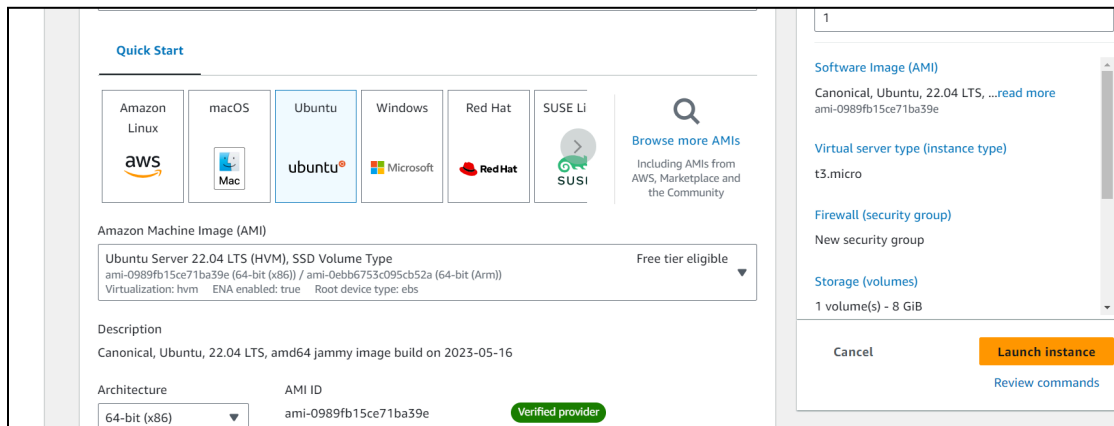


3. Click on *Launch instances*

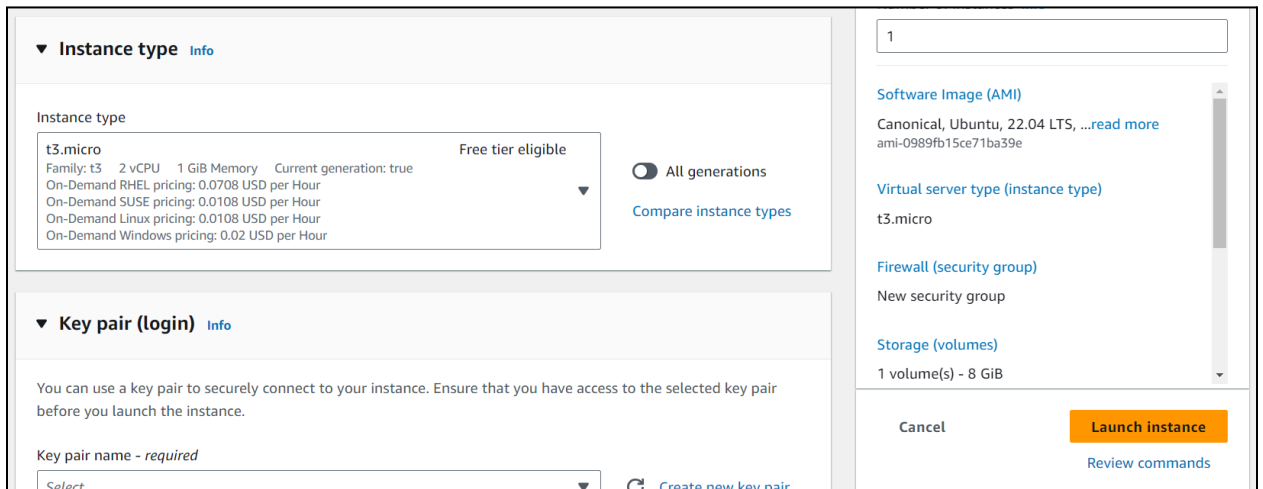
4. Give the name of the instance



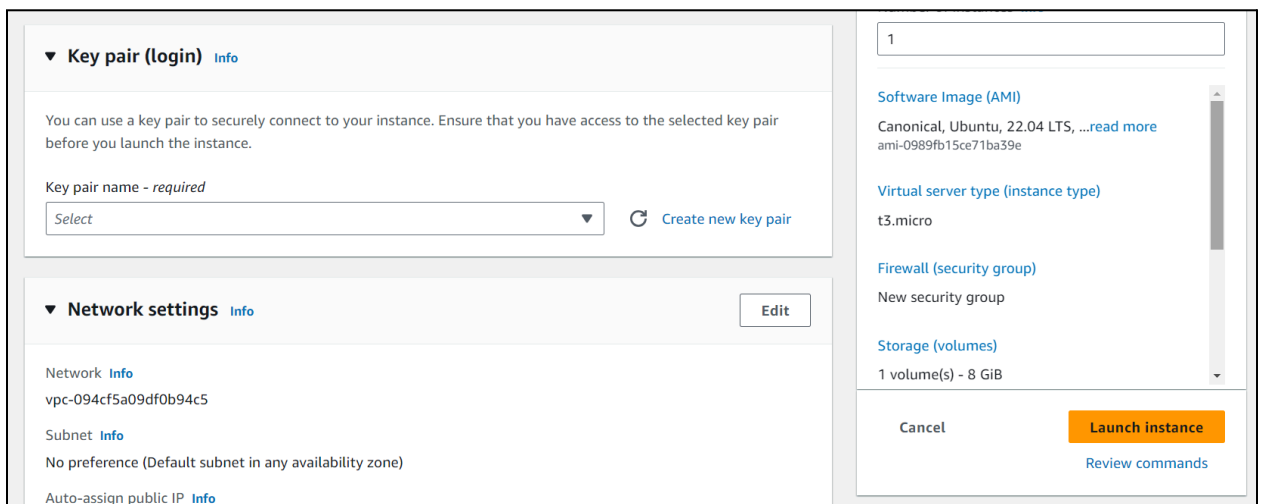
## 5. Select Ubuntu server (free-tier)



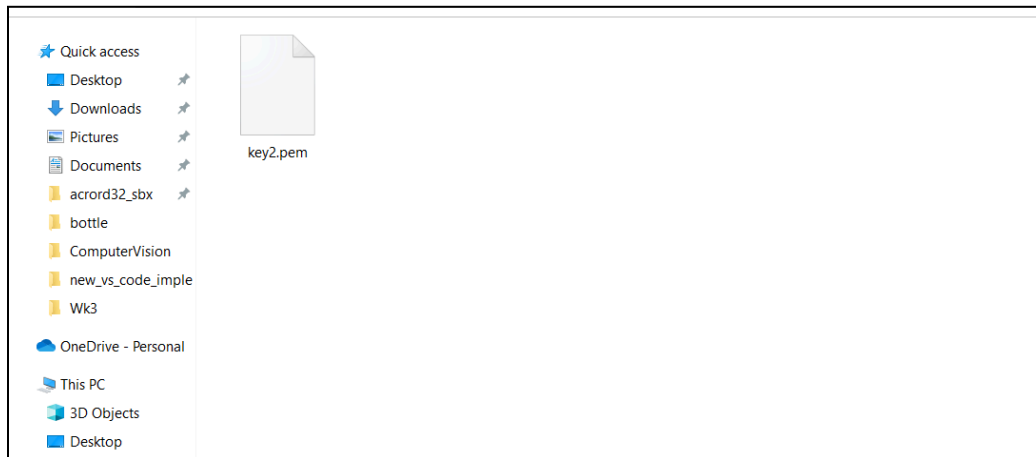
## 6. Select a free tier Instance Type: *t2.micro* (or *t3.micro* in the Regions in which *t2.micro* is unavailable)



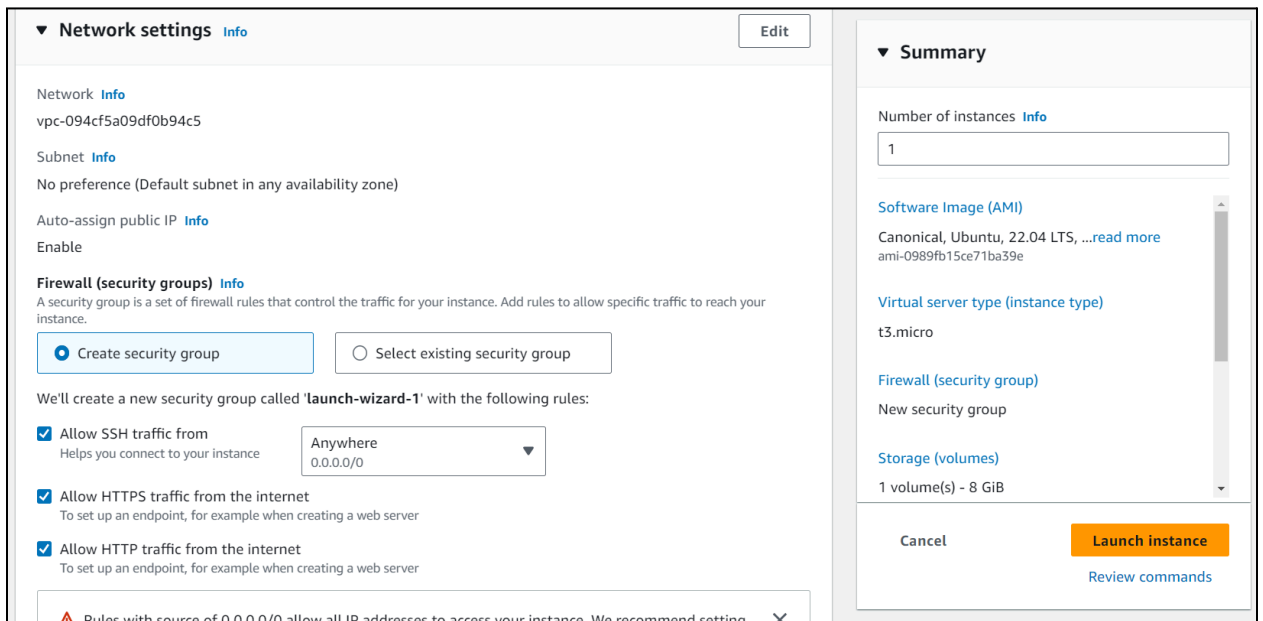
## 7. Create a new key pair. A key pair is used to securely connect to your instances.



## 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

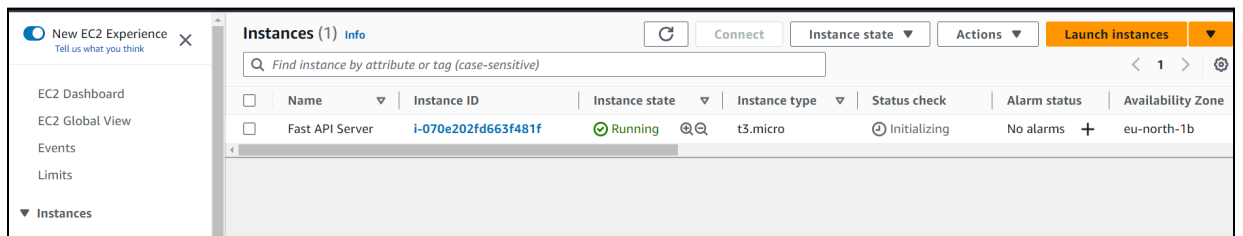


## 10. Recheck the configuration

## 11. Launch Instance



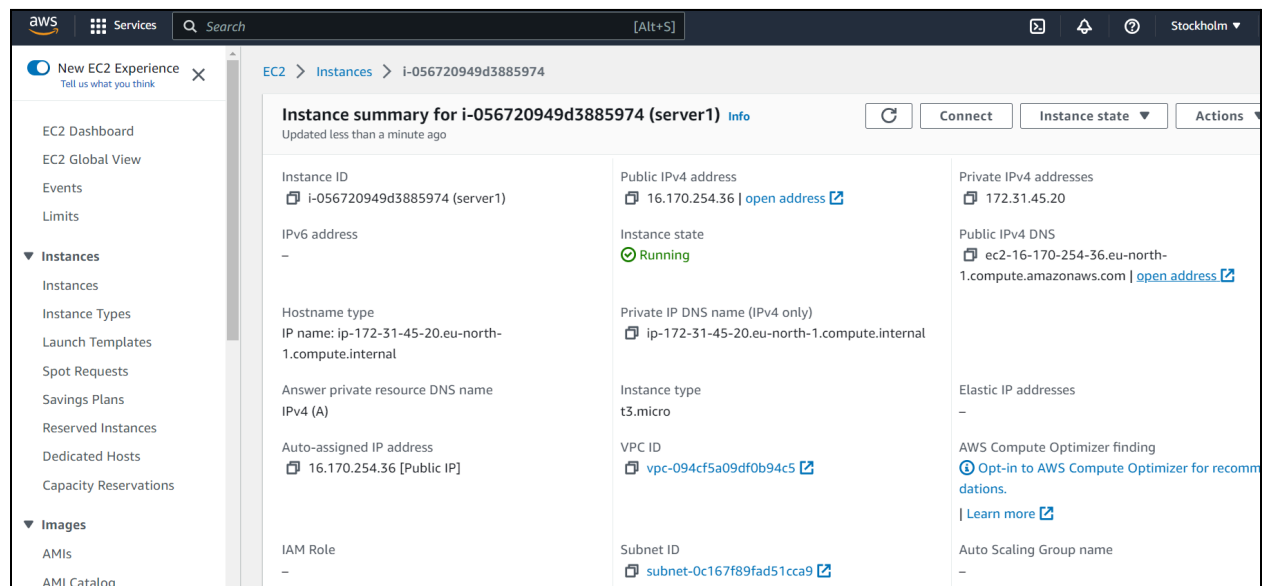
## 12. Check the status of the instance, it should be Running



## 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

Details	Security	Networking	Storage	Status checks	Monitoring	Tags
▼ Instance details Info						
Platform Ubuntu (Inferred)	AMI ID ami-0989fb15ce71ba39e	Monitoring disabled				
Platform details Linux/UNIX	AMI name ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20230516	Termination protection Disabled				
Stop protection Disabled	Launch time Tue Jul 04 2023 17:19:04 GMT+0530 (India Standard Time) (23 minutes)	AMI location amazon/ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-20230516				
Instance auto-recovery Default	Lifecycle normal	Stop-hibernate behavior disabled				
AMI Launch index 0	Key pair assigned at launch Launch	State transition reason -				

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

New EC2 Experience

Tell us what you think

EC2 Dashboard

EC2 Global View

Events

Limits

Instances

Instances

Instance Types

Launch Templates

Spot Requests

Savings Plans

Reserved Instances

Dedicated Hosts

Capacity Reservations

Images

DetailsSecurityNetworkingStorageStatus checksMonitoringTags

▼ Security details

IAM Role

-

Owner ID

454494179835

Launch time

Tue Jul 04 2023 17:19:04 GMT+0530 (India Standard Time)

Security groups

sg-0a7d5e9023bcecd06 (launch-wizard-1)

▼ Inbound rules

Filter rules

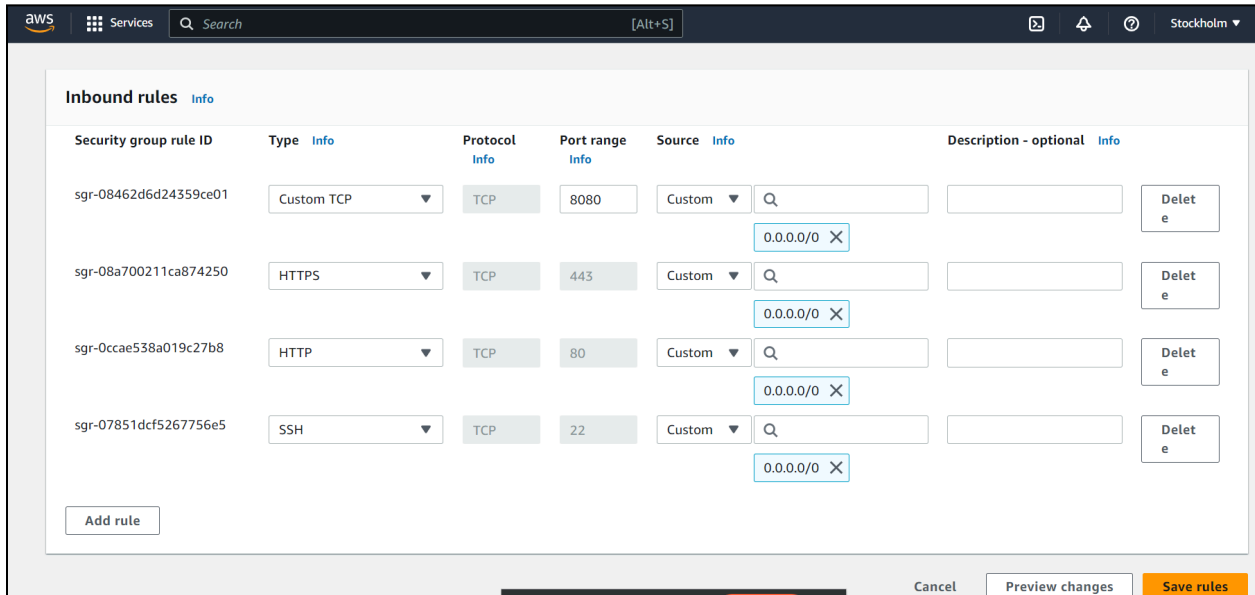
< 1 >

Name	Security group rule ID	Port range	Protocol	Source
-	sgr-08462d6d24359ce01	8080	TCP	0.0.0.0/0
-	sgr-08a700211ca874250	443	TCP	0.0.0.0/0
-	sgr-0ccae538a019c27b8	80	TCP	0.0.0.0/0
-	sgr-07851dcf5267756e5	22	TCP	0.0.0.0/0

## 4. Scroll to find the inbound and outbound rules

EC2 Global View Events Limits ▼ Instances Instances Instance Types Launch Templates Spot Requests Savings Plans Reserved Instances Dedicated Hosts Capacity Reservations ▼ Images AMIs	Inbound rules	Outbound rules	Tags
	You can now check network connectivity with Reachability Analyzer		
	Run Reachability Analyzer		
	Inbound rules (4)		
	Filter security group rules		
	<input type="checkbox"/>	Name	Security group rule...
	<input type="checkbox"/>	-	sgr-08462d6d24359ce...
	<input type="checkbox"/>	-	sgr-08a700211ca8742...
	<input type="checkbox"/>	-	sgr-0ccae538a019c27b8
	<input type="checkbox"/>	-	sgr-07851dcf5267756e5

## 5. Select *Edit inbound rules*



The screenshot shows the AWS Management Console 'Inbound rules' page for a security group. It displays a table of existing rules with columns for Security group rule ID, Type, Protocol, Port range, Source, and Description. Below the table is an 'Add rule' button. At the bottom right are 'Cancel', 'Preview changes', and 'Save rules' buttons.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional	Actions
sgr-08462d6d24359ce01	Custom TCP	TCP	8080	Custom		Delete
sgr-08a700211ca874250	HTTPS	TCP	443	Custom		Delete
sgr-0ccae538a019c27b8	HTTP	TCP	80	Custom		Delete
sgr-07851dcf5267756e5	SSH	TCP	22	Custom		Delete

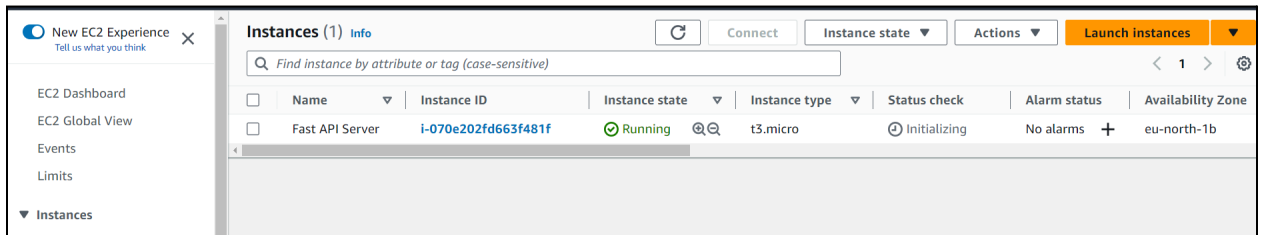
6. To add a new rule, click on *Add rule*. Provide information such as

- 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



The screenshot shows the AWS Management Console 'Instances' page. It displays a table of EC2 instances with columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, and Availability Zone. The 'Fast API Server' instance is shown in a 'Running' state.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
Fast API Server	i-070e202fd663f481f	Running	t3.micro	Initializing	No alarms	eu-north-1b



### 3. Click on *Connect*

EC2 > Instances > i-070e202fd663f481f

**Instance summary for i-070e202fd663f481f (Fast API Server)** Info

Updated less than a minute ago

[Refresh](#) [Connect](#) [Instance state ▼](#) [Actions ▼](#)

Instance ID i-070e202fd663f481f (Fast API Server)	Public IPv4 address 16.171.17.236   <a href="#">open address</a>	Private IPv4 addresses 172.31.46.54
IPv6 address –	Instance state <span>Running</span>	Public IPv4 DNS ec2-16-171-17-236.eu-north-1.compute.amazonaws.com   <a href="#">open address</a>
Hostname type IP name: ip-172-31-46-54.eu-north-1.compute.internal	Private IP DNS name (IPv4 only) ip-172-31-46-54.eu-north-1.compute.internal	Elastic IP addresses –
Answer private resource DNS name IPv4 (A)	Instance type t3.micro	AWS Compute Optimizer finding <a href="#">Opt-in to AWS Compute Optimizer for recommendations.</a> <a href="#">Learn more</a>
Auto-assigned IP address 16.171.17.236 [Public IP]	VPC ID vpc-094cf5a09df0b94c5	Auto Scaling Group name –
IAM Role –	Subnet ID subnet-0c167f89fad51cca9	

### 4. Select EC2 instance connect

aws Services Search [Alt+S] Stockholm

EC2 > Instances > i-054ea3f92134bbfcb > Connect to instance

**Connect to instance** Info

Connect to your instance i-054ea3f92134bbfcb (server1) using any of these options

[EC2 Instance Connect](#) [Session Manager](#) [SSH client](#) [EC2 serial console](#)

Instance ID  
i-054ea3f92134bbfcb (server1)

Connection Type

☒ **Connect using EC2 Instance Connect**  
Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.

☐ **Connect using EC2 Instance Connect Endpoint**  
Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

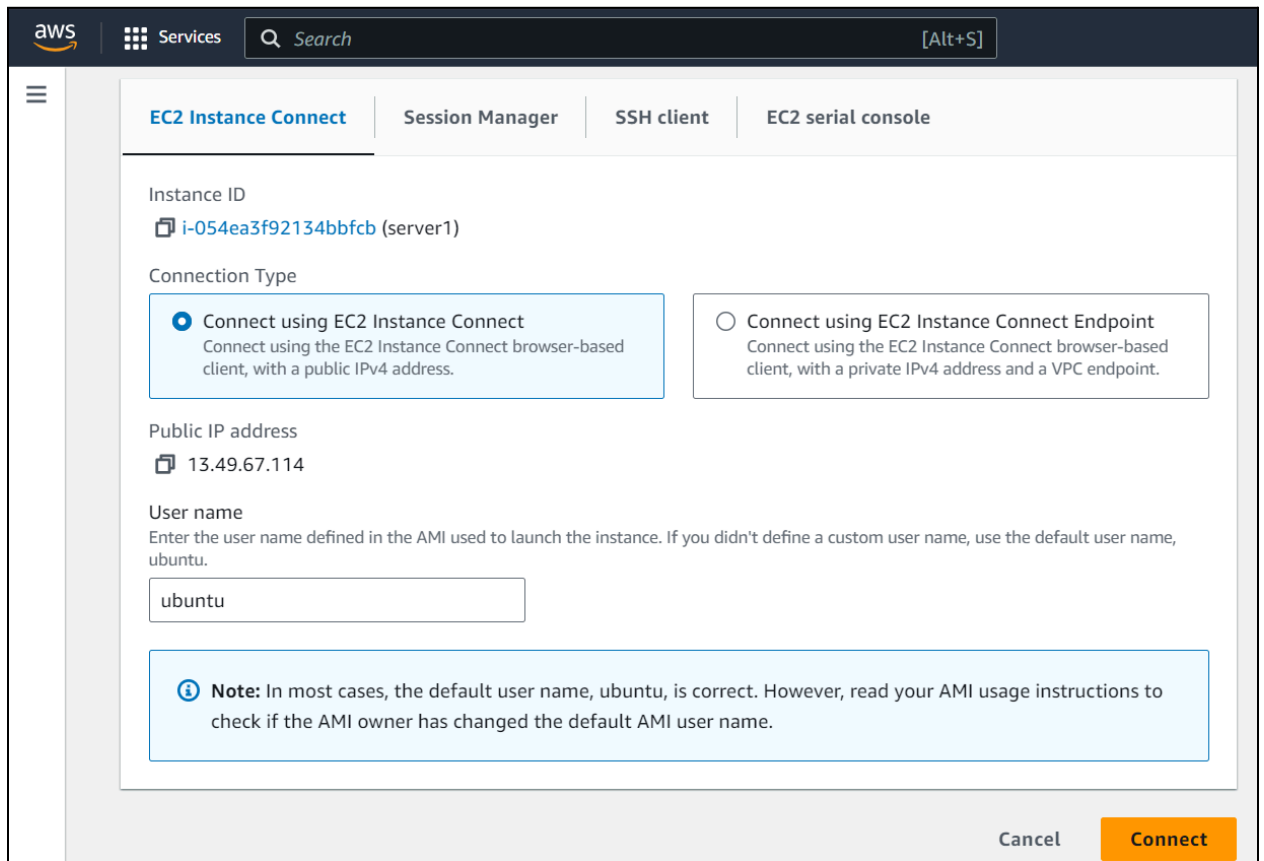
Public IP address  
13.49.67.114

User name  
Enter the user name defined in the AMI used to launch the instance. If you didn't define a custom user name, use the default user name, ubuntu.  
ubuntu

Note: In most cases, the default user name, ubuntu, is correct.

CloudShell Feedback Language meet.google.com is sharing your screen. Stop sharing Hide Services India Private Limited or its affiliates. Privacy Terms

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



aws Services Search [Alt+S]

EC2 Instance Connect Session Manager SSH client EC2 serial console

Instance ID  
i-054ea3f92134bbfcb (server1)

Connection Type

☒ Connect using EC2 Instance Connect  
 Connect using the EC2 Instance Connect browser-based client, with a public IPv4 address.

☐ Connect using EC2 Instance Connect Endpoint  
 Connect using the EC2 Instance Connect browser-based client, with a private IPv4 address and a VPC endpoint.

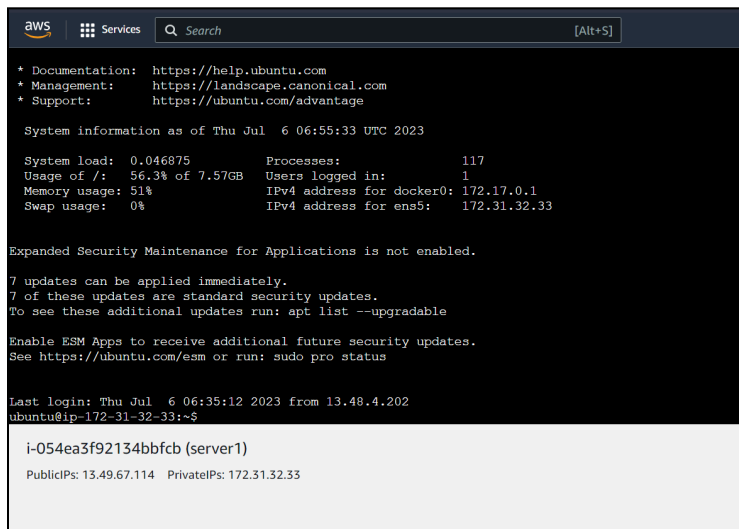
Public IP address  
13.49.67.114

User name  
Enter the user name defined in the AMI used to launch the instance. If you didn't define a custom user name, use the default user name, ubuntu.  
ubuntu

**Note:** In most cases, the default user name, ubuntu, is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI user name.

Cancel Connect

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



```

aws Services Search [Alt+S]

* Documentation: https://help.ubuntu.com
* Management:   https://landscape.canonical.com
* Support:      https://ubuntu.com/advantage

System information as of Thu Jul 6 06:55:33 UTC 2023

System load: 0.046875   Processes:            117
Usage of /:  56.3% of 7.57GB   Users logged in:     1
Memory usage: 51%          IPv4 address for docker0: 172.17.0.1
Swap usage:  0%            IPv4 address for ens5:  172.31.32.33

Expanded Security Maintenance for Applications is not enabled.

7 updates can be applied immediately.
7 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Thu Jul 6 06:35:12 2023 from 13.48.4.202
ubuntu@ip-172-31-32-33:~$

i-054ea3f92134bbfcb (server1)
PublicIPs: 13.49.67.114 PrivateIPs: 172.31.32.33
  
```

## 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:
  
```

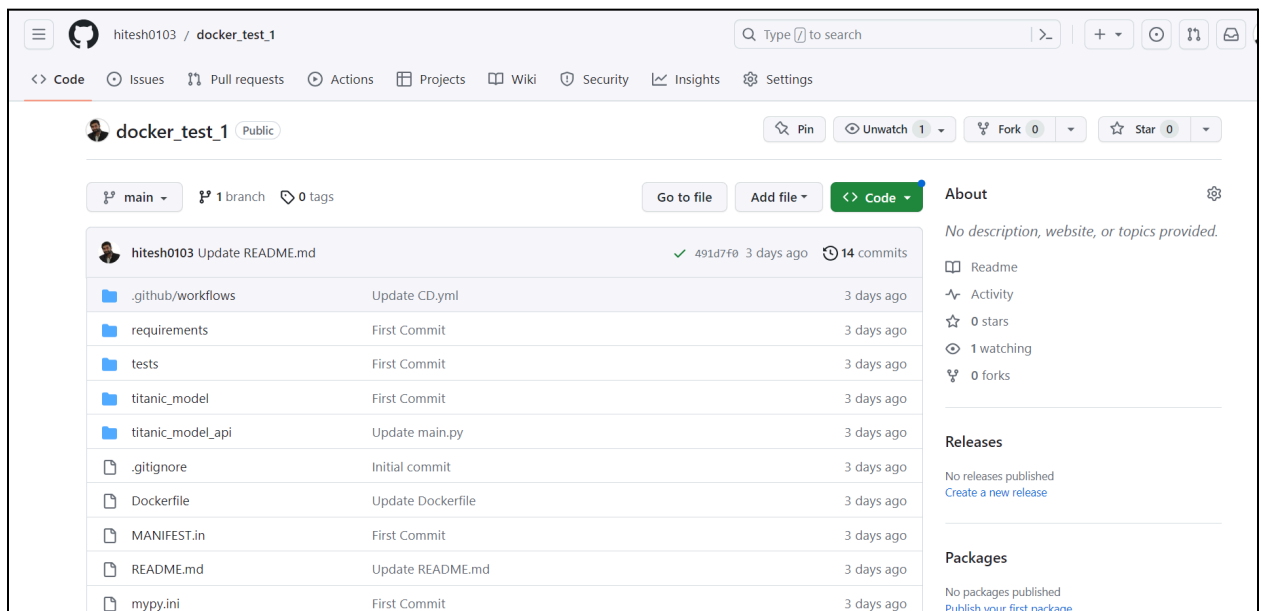
## Add EC2 instance as a Self-hosted Github Runner:

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



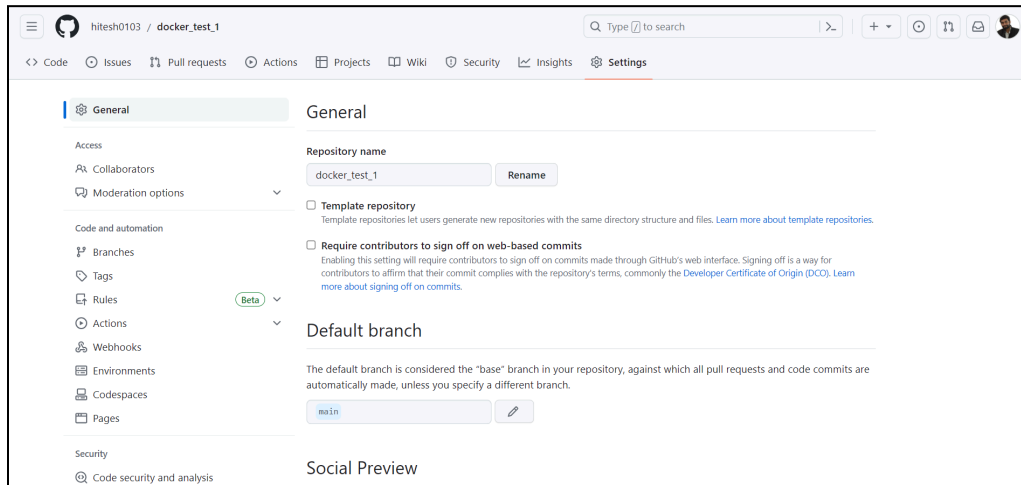
The screenshot shows a GitHub repository named 'docker\_test\_1' owned by 'hitesh0103'. The repository is public and has 1 branch (main) and 0 tags. The file list includes:

- .github/workflows (Update CD.yml, 3 days ago)
- requirements (First Commit, 3 days ago)
- tests (First Commit, 3 days ago)
- titanic\_model (First Commit, 3 days ago)
- titanic\_model\_api (Update main.py, 3 days ago)
- .gitignore (Initial commit, 3 days ago)
- Dockerfile (Update Dockerfile, 3 days ago)
- MANIFEST.in (First Commit, 3 days ago)
- README.md (Update README.md, 3 days ago)
- mypy.ini (First Commit, 3 days ago)

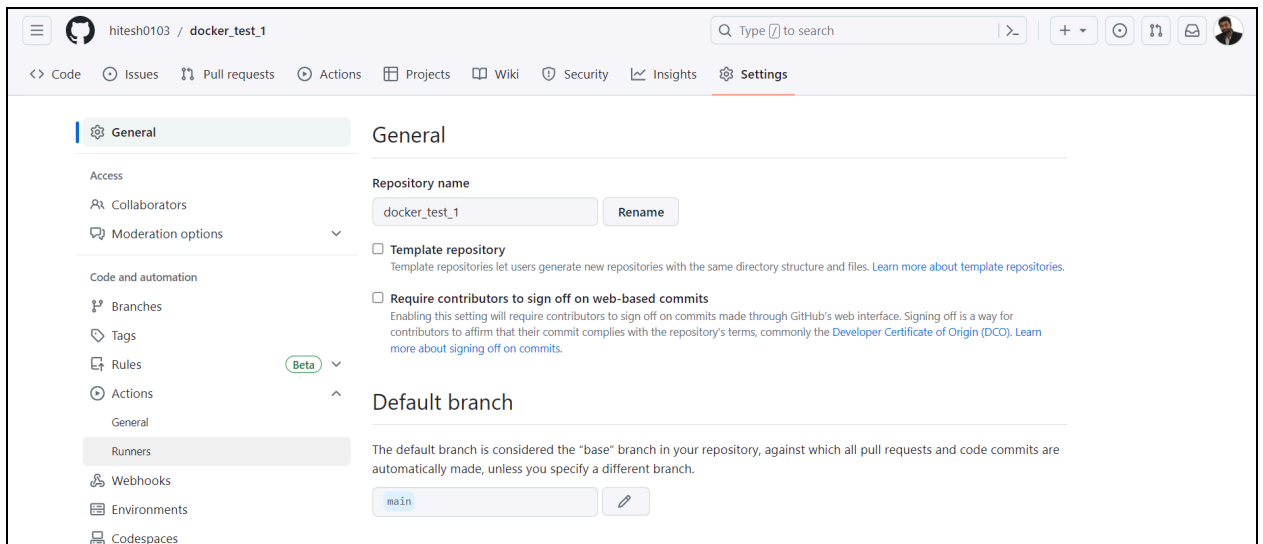
The right sidebar shows the repository's metadata: 0 stars, 1 watching, and 0 forks. It also includes sections for 'About' (no description), 'Releases' (no releases published), and 'Packages' (no packages published).

## 2. Add DockerHub credentials in the repository secrets

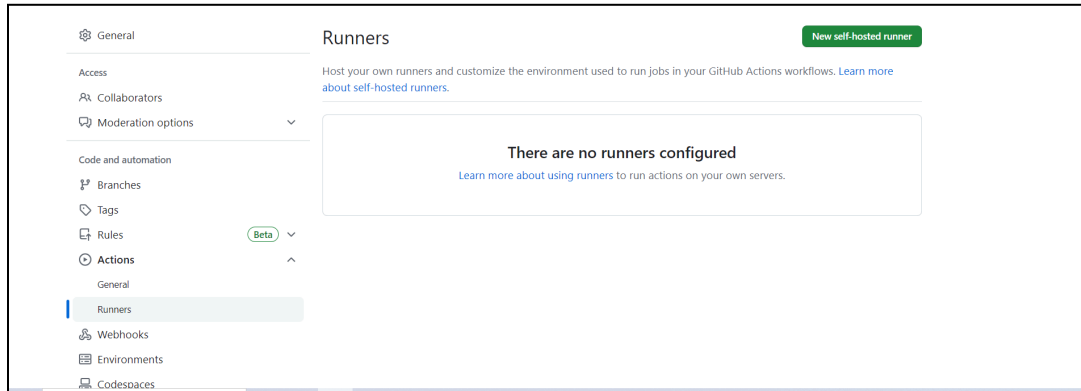
## 3. Go to *Settings*



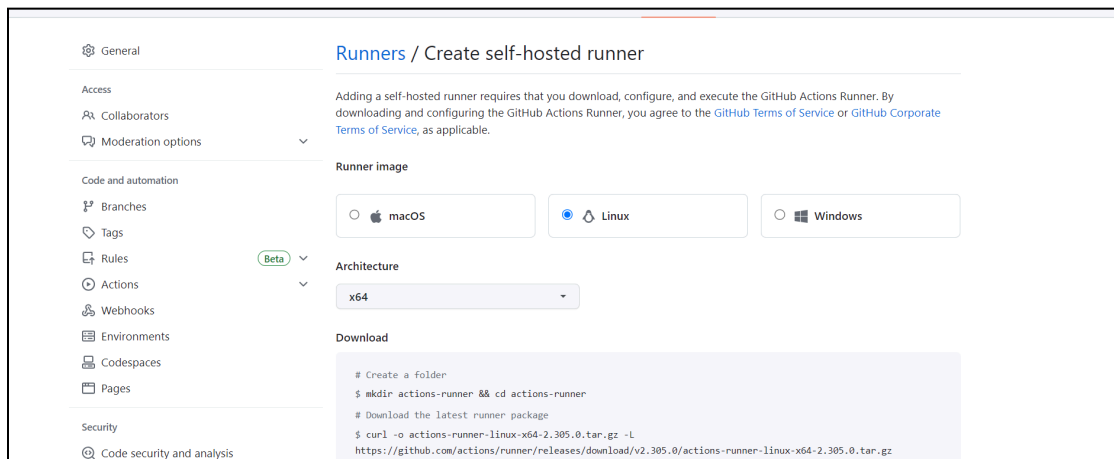
## 4. On the left navigation panel, select *Actions* -> *Runners*



## 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

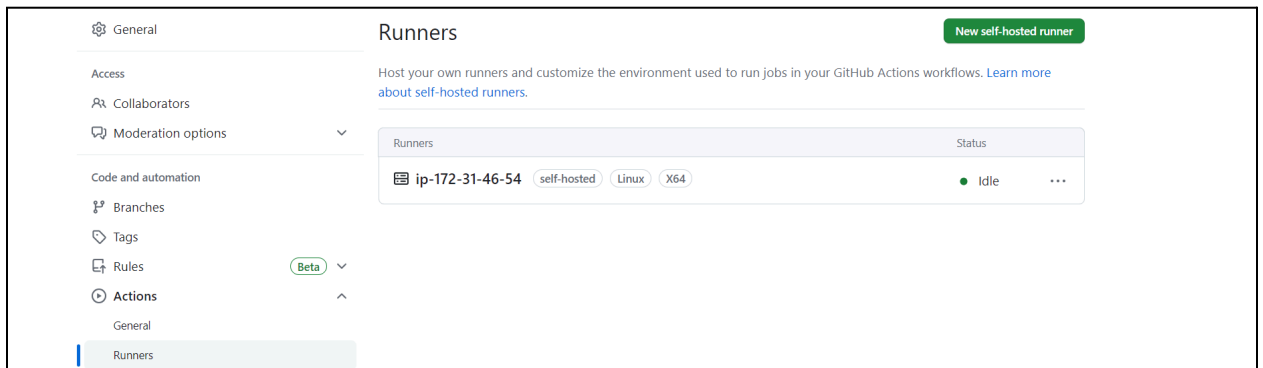
Connected to GitHub

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



The screenshot shows the GitHub 'Runners' page. On the left is a sidebar with navigation links: General, Access, Collaborators, Moderation options, Code and automation, Branches, Tags, Rules (marked Beta), Actions, and Runners (selected). The main content area is titled 'Runners' and includes a 'New self-hosted runner' button. Below this is a table with one runner listed: 'ip-172-31-46-54' with status 'self-hosted', architecture 'Linux', and platform 'X64'. The status is 'Idle' with a green dot and a menu icon.

Runners	Status
ip-172-31-46-54 self-hosted Linux X64	Idle ...

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 YAML 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:

```
name: CD Pipeline

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

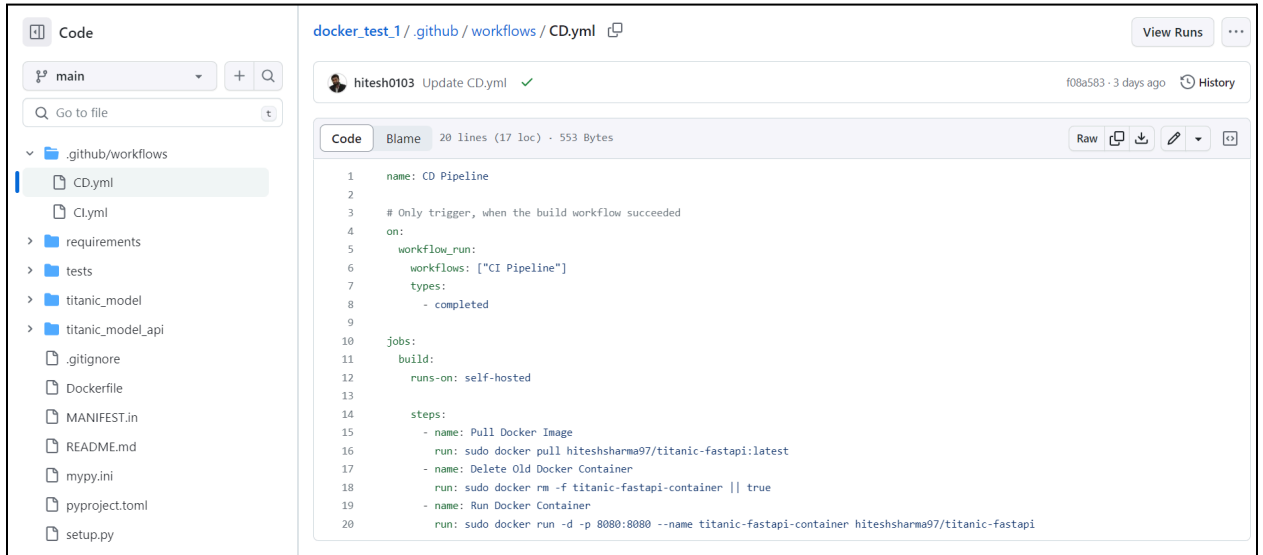
jobs:
  deploy:
    runs-on: self-hosted
    steps:
      - name: Pull Docker Image
        env:
          DOCKER_USER: ${ secrets.DOCKER_USER_NAME }
        run: sudo docker pull $DOCKER_USER/titanic-fastapi:latest

      - 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.

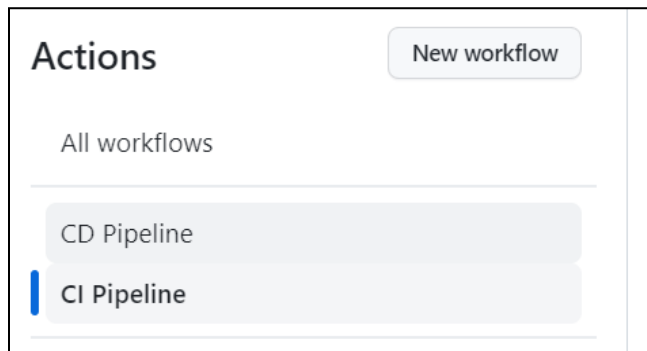


The screenshot shows a GitHub repository interface. On the left, the file explorer shows a directory structure with files like .gitignore, Dockerfile, MANIFEST.in, README.md, mypy.ini, pyproject.toml, and setup.py. The right pane displays the content of the file `docker_test_1 / .github / workflows / CD.yml`. The file content is as follows:

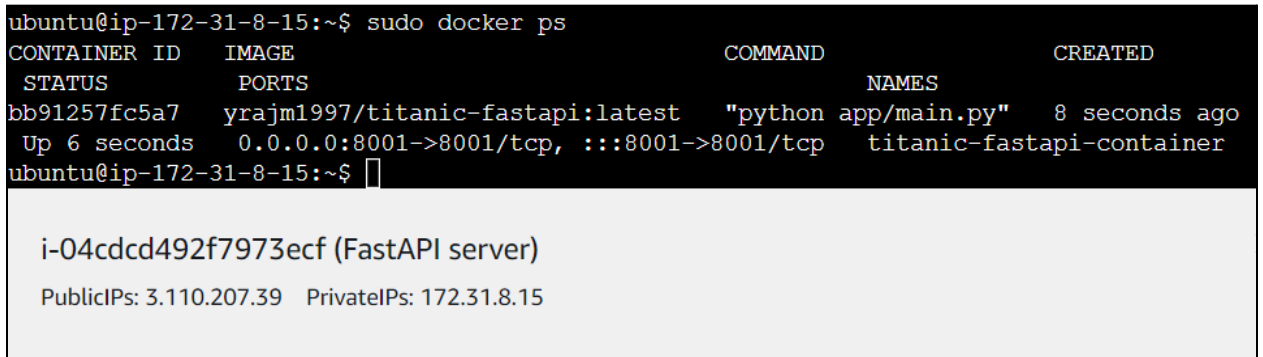
```

1  name: CD Pipeline
2
3  # Only trigger, when the build workflow succeeded
4  on:
5    workflow_run:
6      workflows: ["CI Pipeline"]
7      types:
8        - completed
9
10 jobs:
11   build:
12     runs-on: self-hosted
13
14   steps:
15     - name: Pull Docker Image
16       run: sudo docker pull hiteshsharma97/titanic-fastapi:latest
17     - name: Delete Old Docker Container
18       run: sudo docker rm -f titanic-fastapi-container || true
19     - name: Run Docker Container
20       run: sudo docker run -d -p 8080:8080 --name titanic-fastapi-container hiteshsharma97/titanic-fastapi
  
```

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`



The screenshot shows a terminal window with the following output:

```

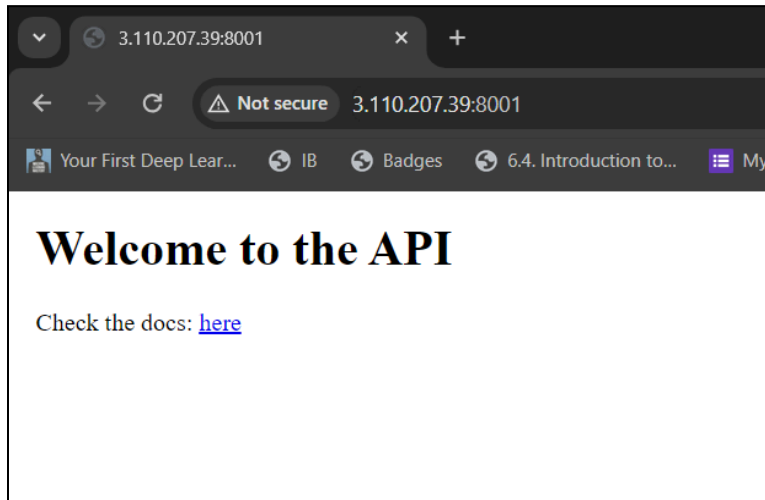
ubuntu@ip-172-31-8-15:~$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED
STATUS        PORTS
bb91257fc5a7   yrajm1997/titanic-fastapi:latest    "python app/main.py"    8 seconds ago
Up 6 seconds   0.0.0.0:8001->8001/tcp, :::8001->8001/tcp  titanic-fastapi-container
ubuntu@ip-172-31-8-15:~$
  
```

Below the terminal output, the EC2 instance details are shown:

i-04cdcd492f7973ecf (FastAPI server)  
 PublicIPs: 3.110.207.39 PrivateIPs: 172.31.8.15

- 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`



- 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
```

- Terminate the EC2 instance to prevent unnecessary charges

