

PROJECT 6: SETTING UP A CONTINUOUS DELIVERY PIPELINE WITH GIT, JENKINS, DOCKER, AND AWS ECS

- **Create a sample application with a Dockerfile:**
 - ➔ Create a new directory for the application files.
 - ➔ Create a new Dockerfile that specifies the application dependencies and configuration.
- **Configure Jenkins to build the Docker image, push it to a Docker registry, and deploy it to AWS ECS:**
 - ➔ Install Jenkins on a separate server or locally.
 - ➔ Install the necessary plugins for Docker and AWS ECS.
 - ➔ Create a new Jenkins job and configure it to build the Docker image, push it to a Docker registry, and deploy it to AWS ECS.
- **Set up AWS ECS to run the Docker container and automatically scale the service based on traffic:**
 - ➔ Create a new ECS cluster and task definition that specifies the Docker image to run.
 - ➔ Create a new ECS service that runs the task definition and automatically scales based on traffic.
 - ➔ Test the pipeline by making changes to the application code and verifying that the changes are automatically deployed to the production environment.

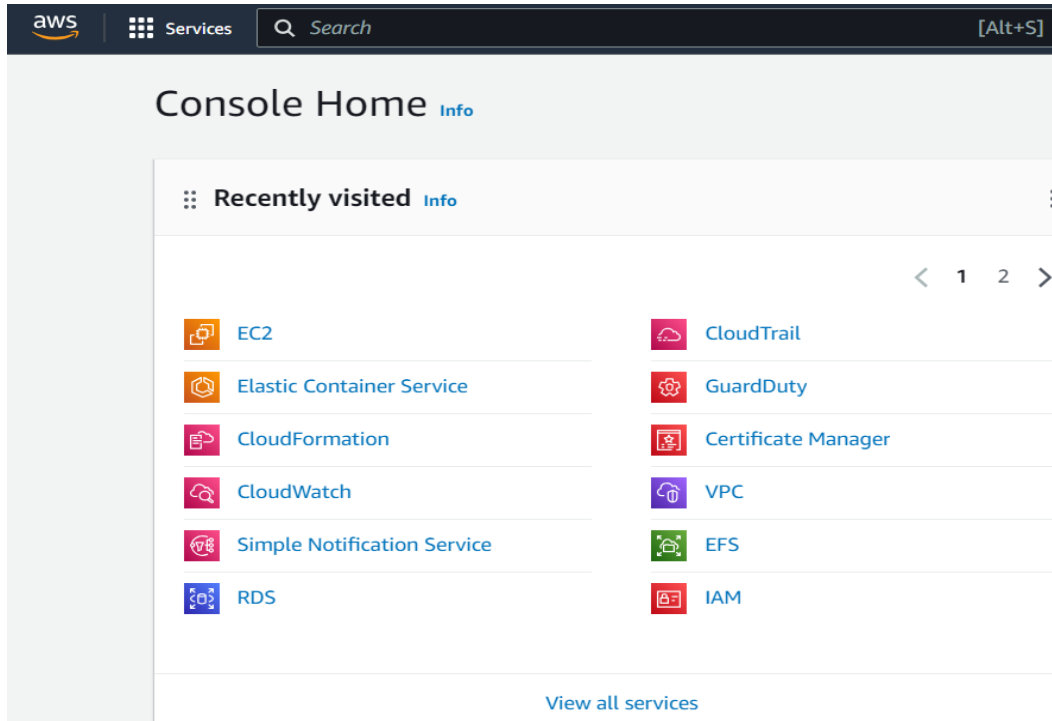
Project Requirements:

- Cloud : **AWS**
- Services : **Amazon EC2**
Amazon ECS
- Source code : **Git & GitHub**
- CI / CD : **Jenkins**
- Containerization : **Docker**

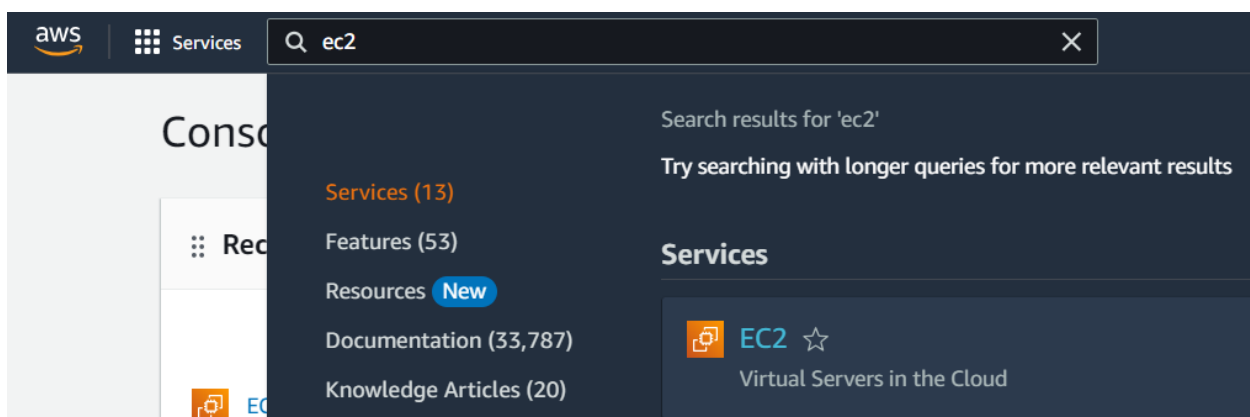
SOLUTION:

Step:1 – Creating an EC2 Instance:

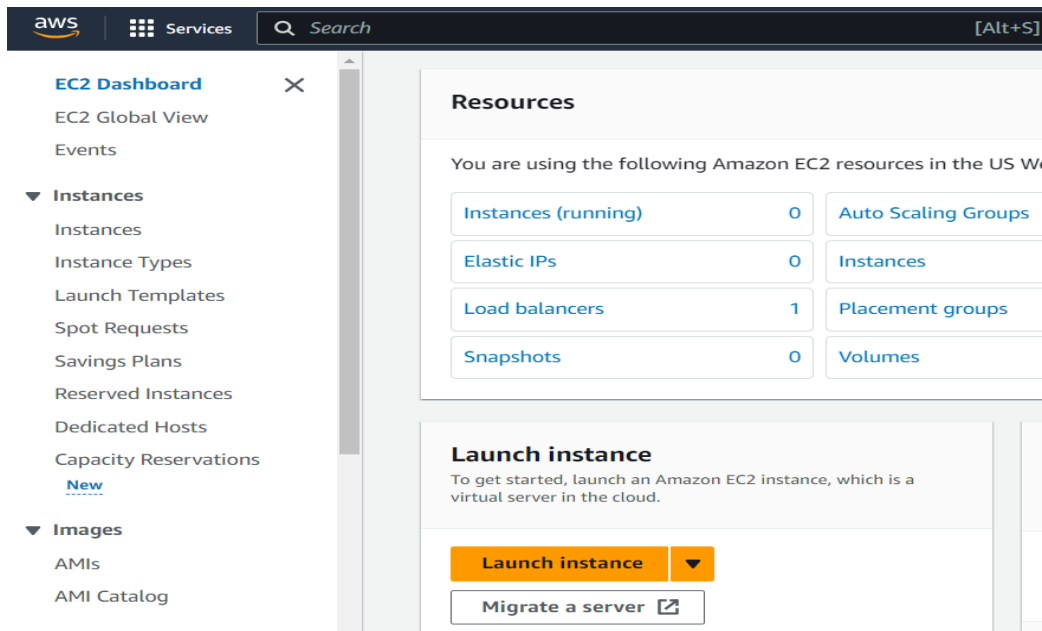
- First login into your [AWS Management console](#):



- Next under search panel search **EC2** & click that one:



- Then EC2 Management console will appear, on that click **launch instances**:



- Then **naming the instances** according to the project:

Launch an instance [Info](#)

Amazon EC2 allows you to create virtual machines, following the simple steps below.

Name and tags [Info](#)

Name

- Then **selecting operating system [OS]**, here I am selecting **Ubuntu 20.04 OS**:

Quick Start

Amazon Linux

macOS

Ubuntu

Windows

Red Hat

SUSE Linux

[Browse more AMIs](#)
Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Ubuntu Server 20.04 LTS (HVM), SSD Volume Type	Free tier eligible
ami-08e2c1a8d17c2fe17 (64-bit (x86)) / ami-05f290e7e87696c29 (64-bit (Arm))	
Virtualization: hvm ENA enabled: true Root device type: ebs	

- Then selecting the instance type: Here I am selecting **t2.micro** instance type which contains **1vpc, 1gb ram**:

Instance type

t2.micro Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true

On-Demand Linux base pricing: 0.0124 USD per Hour

On-Demand Windows base pricing: 0.017 USD per Hour

On-Demand RHEL base pricing: 0.0724 USD per Hour

On-Demand SUSE base pricing: 0.0124 USD per Hour

Additional costs apply for AMIs with pre-installed software

- Then selecting the **keypair**, for secure login into your instances:

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - *required*

vockey

 [Create new key pair](#)

- Then selecting the default **VPC & default subnet**:

▼ Network settings [Info](#)

VPC - *required* [Info](#)

vpc-0faeefcc1359d13d0
172.31.0.0/16

(default) ▼

Subnet [Info](#)

subnet-0582956f7d764a33a

VPC: vpc-0faeefcc1359d13d0 Owner: 293628418374 Availability Zone: us-west-2c
IP addresses available: 4091 CIDR: 172.31.0.0/20

▼

- Then naming the **security group & description**:

Security group name - *required*

Project-6

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and ._-:/()#,@[]+=&:{}!\$*

Description - *required* [Info](#)

Project-6

- Then adding the ssh port under security group:

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

Remove

Type [Info](#)

ssh

Protocol [Info](#)

TCP

Port range [Info](#)

22

Source type [Info](#)

Anywhere

Source [Info](#)

🔍 Add CIDR, prefix list or security

0.0.0.0/0 ✕

Description - *optional* [Info](#)

e.g. SSH for admin desktop

- Then keeping the default storage and launching the instances:

▼ Configure storage [Info](#)

[Advanced](#)

1x 8 GiB gp2 Root volume (Not encrypted)

📘 Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage ✕

Add new volume

The selected AMI contains more instance store volumes than the instance allows. Only the first 0 instance store volumes from the AMI will be accessible from the instance

0 x File systems

[Edit](#)

► [Advanced details](#) [Info](#)

Canonical, Ubuntu, 20.04 LTS, ...[read more](#)
ami-08e2c1a8d17c2fe17

[Virtual server type \(instance type\)](#)
t2.medium

[Firewall \(security group\)](#)
New security group

[Storage \(volumes\)](#)
1 volume(s) - 8 GiB

📘 **Free tier:** In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which

Cancel

Launch instance

[Review commands](#)

- Instance creating has been initiated:

[EC2](#) > [Instances](#) > Launch an instance

🔄 **Launching instance**
Creating security group rules

21%

- The instance has been created successfully:

Instances (1) Info 🔄				
🔍 Find Instance by attribute or tag (case-sensitive)				
<input type="checkbox"/>	Name ✎	Instance ID	Instance state	Instance type
<input type="checkbox"/>	PROJECT-6	i-0059986d3b353f505	🟢 Running 🔍 🔍	t2.medium

Step:2 – Installing docker on newly created instances:

- First connect the instance with putty or with instance connect:

```

🔒 Authenticating with public key "imported-openssh-key"

• MobaXterm Personal Edition v23.2 •
  (SSH client, X server and network tools)

➤ SSH session to ubuntu@54.213.232.44
  • Direct SSH : ✓
  • SSH compression : ✓
  • SSH-browser : ✓
  • X11-forwarding : ✓ (remote display is forwarded through SSH)
➤ For more info, ctrl+click on help or visit our website.

Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1048-aws x86_64)

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

System information as of Mon Nov 13 14:42:13 UTC 2023

System load:  0.05          Processes:           115
Usage of /:   21.0% of 7.57GB Users logged in:       0
Memory usage: 5%           IPv4 address for eth0: 172.31.6.219
Swap usage:   0%

```

- Installing docker on it with the command:

apt-get update

apt-get install -y docker.io

```

Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.
Setting up dnsmasq-base (2.80-1.1ubuntu1.7) ...
Setting up ubuntu-fan (0.12.13ubuntu0.1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/ubuntu-fan.service → /lib/systemd/system/ubuntu-fan.service.
Processing triggers for systemd (245.4-4ubuntu3.22) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for dbus (1.12.16-2ubuntu2.3) ...
Processing triggers for libc-bin (2.31-0ubuntu9.12) ...

```

- Checking whether docker is installed or not by using the command:

docker --version

docker --info

```
root@ip-172-31-6-219:/home/ubuntu# docker --version
Docker version 24.0.5, build 24.0.5-0ubuntu1~20.04.1
root@ip-172-31-6-219:/home/ubuntu# docker info
Client:
 Version:      24.0.5
 Context:      default
 Debug Mode:   false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
```

Step:3 – Creating application source code and dockerfile:

- Creating app.py which contains main application code:

Source code contains:

```
# app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return '<b>Hello,</b><br><b>this python application runs<br>from Amazon ECS with the help of Jenkins CI/CD<br>Pipeline!!!</b>'
```

```
if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

```
root@ip-172-31-6-219:/home/ubuntu# vi app.py
root@ip-172-31-6-219:/home/ubuntu# ls
app.py
root@ip-172-31-6-219:/home/ubuntu#
```

- Then creating requirements.txt file for dependencies to support this application:

Requirements.txt contains:

```
Flask==2.0.1
Werkzeug==2.0.1
```

```
root@ip-172-31-6-219:/home/ubuntu# vi requirements.txt
root@ip-172-31-6-219:/home/ubuntu# ls
app.py  requirements.txt
root@ip-172-31-6-219:/home/ubuntu#
```

- Creating dockerfile for this application:

Dockerfile contains:

```
# Dockerfile
#choosing the base image:
FROM python:3.8-alpine

#choosing working directory for the application:
WORKDIR /app

#copying the requirements.txt file to app directory and
installing packages:
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

#copying the rest of application code to the working
directory:
COPY . .

#exposing the application:
```



```
EXPOSE 5000
```

```
#Executing the application after creating image:
```

```
CMD ["python", "app.py"]
```

```
root@ip-172-31-6-219:/home/ubuntu# vi dockerfile
root@ip-172-31-6-219:/home/ubuntu# ls
app.py  dockerfile  requirements.txt
root@ip-172-31-6-219:/home/ubuntu#
```

Step:4 – Building a dockerimage and testing the application:

- Building the docker image from the dockerfile, with the command:

docker build -t ravivarman46/python:app .

I am building the docker image with my Docker hub id, so that it will be useful while pushing the image to the docker hub:

```
root@ip-172-31-6-219:/home/ubuntu# docker build -t ravivarman46/python:app .
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  15.36kB
Step 1/7 : FROM python:3.8-alpine
3.8-alpine: Pulling from library/python
96526aa774ef: Pull complete
430548f4d4bf: Pull complete
43f3c7ab6662: Pull complete
14d88fea9a04: Pull complete
be855da05668: Pull complete
```

Checking with the command: **docker images**

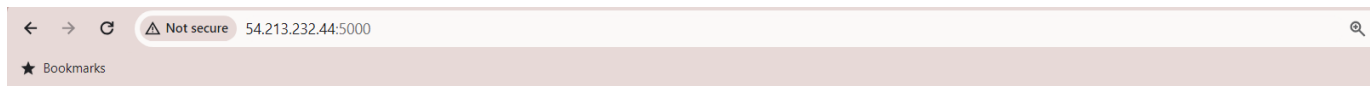
```
root@ip-172-31-6-219:/home/ubuntu# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ravivarman46/python  app                6ca7d401e940       29 seconds ago     60.4MB
python              3.8-alpine         4bf70beea733       4 weeks ago        50MB
root@ip-172-31-6-219:/home/ubuntu#
```

- Testing the docker image by running it:

docker run -d -it -p 5000:5000 ravivarman46/python:app

```
root@ip-172-31-6-219:/home/ubuntu# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
root@ip-172-31-6-219:/home/ubuntu# docker run -d -it -p 5000:5000 ravivarman46/python:app
2dc1cad850c34daf762bad10ac5d2e8d2047fb2282cb096b43ac8306984ce14e
root@ip-172-31-6-219:/home/ubuntu# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
2dc1cad850c3   ravivarman46/python:app   "python app.py"         3 seconds ago   Up 2 seconds   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   interesting_mestorf
root@ip-172-31-6-219:/home/ubuntu#
```

- Checking the output from the browser by pasting the public ip along with port number:



Hello,

this python application runs from Amazon ECS with the help of Jenkins CI/CD Pipeline!!!

The container created from docker image is working fine:

Step: 5 – Pushing Docker image to docker hub:

- First logging into docker hub with docker hub credentials:

```
root@ip-172-31-6-219:/home/ubuntu# docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID,
to create one.
Username: ravivarman46
Password:
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
root@ip-172-31-6-219:/home/ubuntu#
```


- Pushing the docker image to docker hub with the command: **docker push ravivarman46/python:app**

```


root@ip-172-31-6-219:/home/ubuntu# docker push ravivarman46/python:app
The push refers to repository [docker.io/ravivarman46/python]
f8ad28457d31: Pushed
fd7883c8c44a: Pushed
b50bd2d52e3e: Pushed
a9a8ae30353a: Pushed
2b863a82abca: Layer already exists
9618769d4452: Layer already exists
cdea479ac6b8: Layer already exists
6f25d7d19389: Layer already exists
cc2447e1835a: Layer already exists
app: digest: sha256:ffcabb627d42925e6ee5f486e4828d0ac2b990cd4bc6408fff4ecf1d1b1efeb size: 2200
root@ip-172-31-6-219:/home/ubuntu#


```

- Checking on docker hub:

 **ravivarman46 / python**

Description

This repository does not have a description 

 Last pushed: 2 minutes ago

Tags

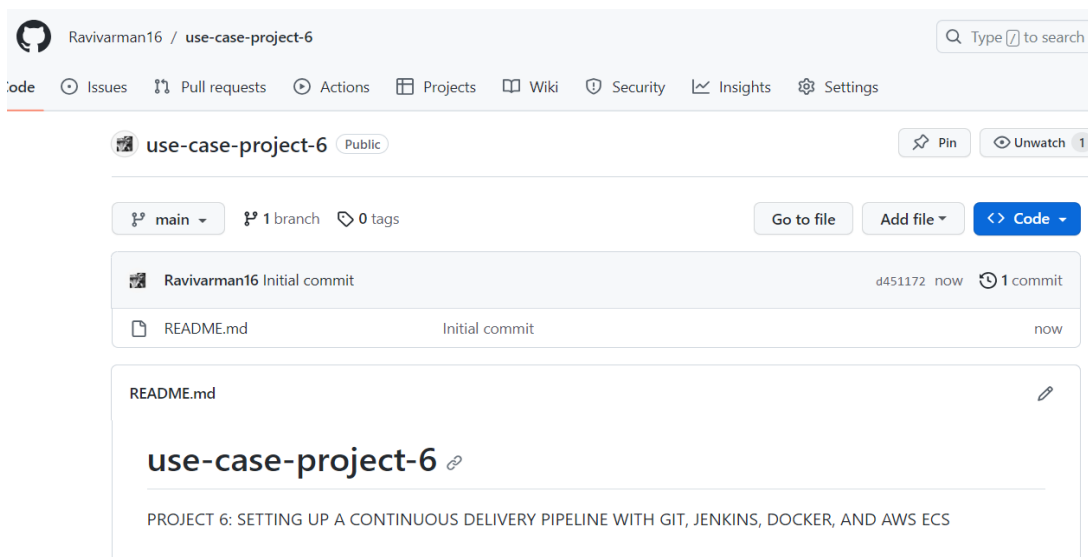
This repository contains 2 tag(s).

Tag	OS	Type	Pulled	Pushed
 app		Image	---	2 minutes ago

Step: 6 – Pushing application code to Github:

- Creating repository under GitHub:

<https://github.com/Ravivarman16/use-case-project-6.git>



The screenshot shows the GitHub interface for a repository named 'use-case-project-6' under the user 'Ravivarman16'. The repository is public and has 1 branch (main) and 0 tags. It shows an initial commit by 'Ravivarman16' with the message 'Initial commit' and a file named 'README.md'. The README content is visible, showing the title 'use-case-project-6' and the description 'PROJECT 6: SETTING UP A CONTINUOUS DELIVERY PIPELINE WITH GIT, JENKINS, DOCKER, AND AWS ECS'.

- On the command line first initializing git in the current working directory:

```
root@ip-172-31-6-219:/home/ubuntu# pwd
/home/ubuntu
root@ip-172-31-6-219:/home/ubuntu# git init
Initialized empty Git repository in /home/ubuntu/.git/
root@ip-172-31-6-219:/home/ubuntu# git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .Xauthority
        .bash_logout
        .bashrc
        .cache/
        .profile
        .ssh/
        .sudo_as_admin_successful
        app.py
        dockerfile
        requirements.txt

nothing added to commit but untracked files present (use "git add" to track)
root@ip-172-31-6-219:/home/ubuntu#
```

- Then staging and committing the required files for this project:

```
root@ip-172-31-6-219:/home/ubuntu# git add app.py requirements.txt dockerfile
root@ip-172-31-6-219:/home/ubuntu# git status
On branch master

No commits yet

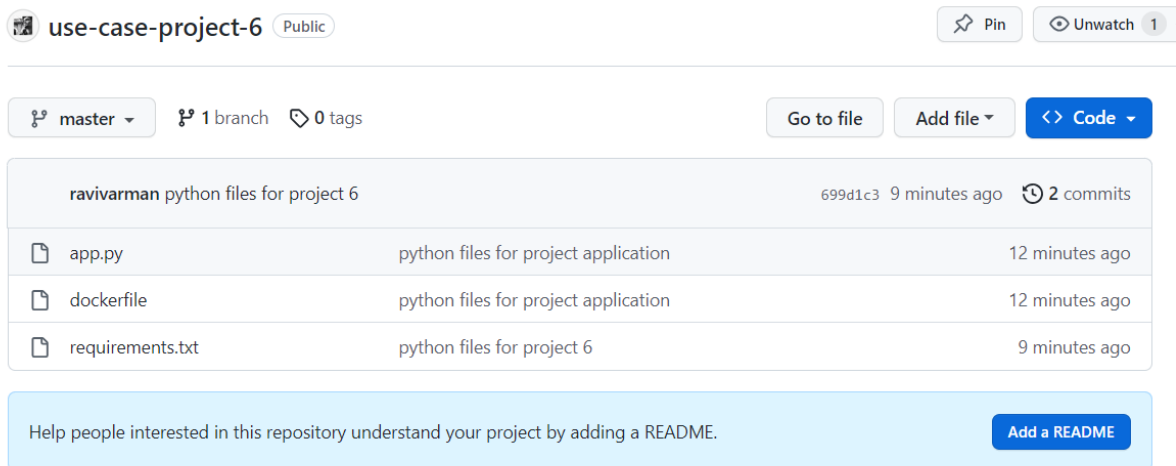
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   app.py
        new file:   dockerfile
        new file:   requirements.txt
```

```
root@ip-172-31-6-219:/home/ubuntu# git config user.name "ravivarman"
root@ip-172-31-6-219:/home/ubuntu# git config user.email "jravi1605@gmail.com"
root@ip-172-31-6-219:/home/ubuntu#
root@ip-172-31-6-219:/home/ubuntu# git commit -m "python files for project application"
[master (root-commit) 6d327b4] python files for project application
Committer: ravivarman <root@ip-172-31-6-219.us-west-2.compute.internal>
Your name and email address were not found, so I automatically based it on the
```

- Then pushing it to the remote repository:

```
root@ip-172-31-6-219:/home/ubuntu# git remote add python https://github.com/Ravivarman16/use-case-project-6.git
root@ip-172-31-6-219:/home/ubuntu# git push python master
Username for 'https://github.com': Ravivarman16
Password for 'https://Ravivarman16@github.com':
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 1017 bytes | 1017.00 KiB/s, done.
Total 8 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
remote:
remote: Create a pull request for 'master' on GitHub by visiting:
remote:   https://github.com/Ravivarman16/use-case-project-6/pull/new/master
remote:
To https://github.com/Ravivarman16/use-case-project-6.git
 * [new branch]      master -> master
root@ip-172-31-6-219:/home/ubuntu#
```

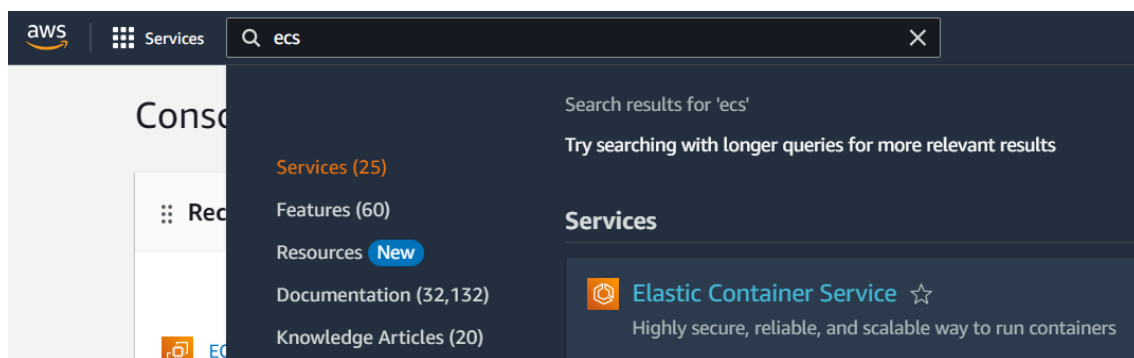
- Checking it on remote repository:



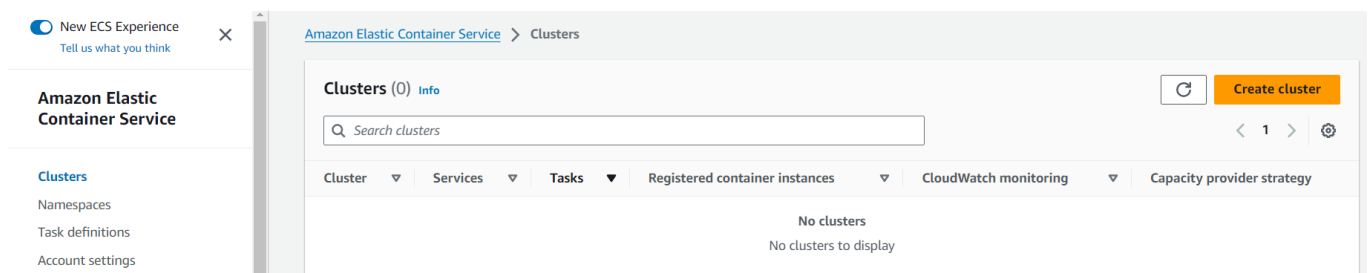
Step:7 – Deploying the docker image to Amazon ECS service:

Creating a ECS Cluster:

- On the AWS management console, under service panel search **ECS**, click that one:



- Then we can able to see ECS management console, on left hand side we could able to the **cluster** option, click that to create a new cluster:



- Then **naming the cluster & creating namespaces**:

Cluster configuration

Cluster name

py-app

There can be a maximum of 255 characters. The valid characters are letters (uppercase and lowercase), numbers, hyphens, and underscores.

Default namespace - *optional*

Select the namespace to specify a group of services that make up your application. You can overwrite this value at the service level.

Q py-app



- Then under infrastructure, selecting **AWS Fargate** option, which is a serverless option:

▼ Infrastructure [Info](#)

Serverless

Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances, or external instances using ECS Anywhere.

☒ **AWS Fargate (serverless)**

Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

☐ **Amazon EC2 instances**

Manual configurations. Use for large workloads with consistent resource demands.

☐ **External instances using ECS Anywhere**

Manual configurations. Use to add data center compute.

- Remaining options, I am keeping as the default, click create option:

► Monitoring - *optional* [Info](#)

Container Insights is off by default. When you use Container Insights, there is a cost associated with it.

► Tags - *optional* [Info](#)

Tags help you to identify and organize your clusters.

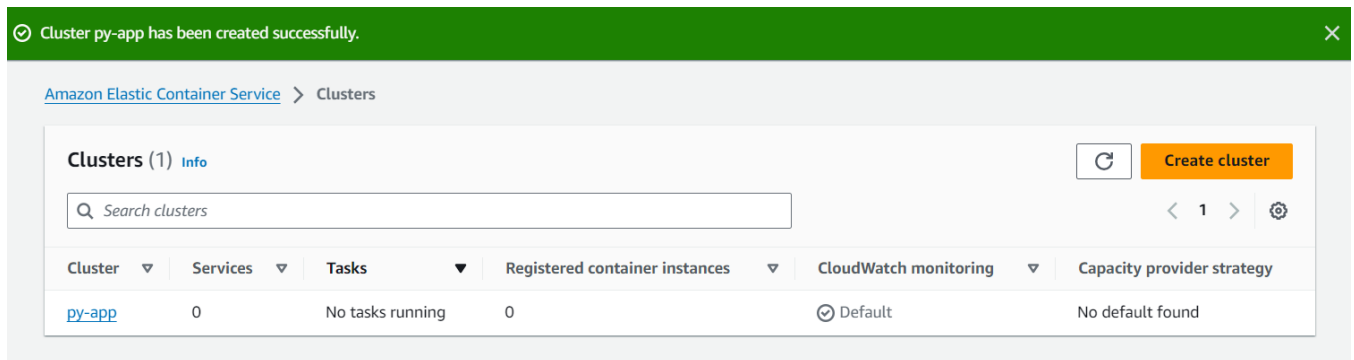
Cancel

Create

- The cluster creation has been initiated:

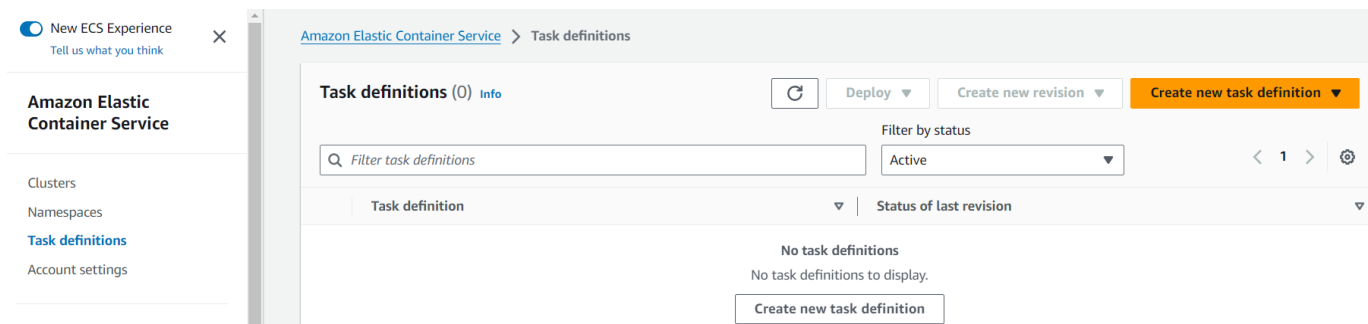


- The ECS Cluster has been created successfully:

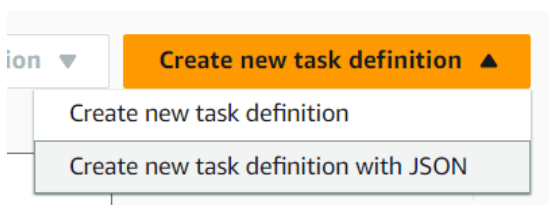


Creating task definition:

- On the left-hand side, we could able to see task definitions, click that one:
We could able to see option to **create new task definition**:



- After clicking create new task definition, we could able to see two options:
 ➔ **Create new task definition**
 ➔ **Create new task definition with JSON**, but here I am selecting first option:



- Then naming the task definition:

Create new task definition [Info](#)

Task definition configuration

Task definition family [Info](#)
Specify a unique task definition family name.

py-app

Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

- Then selecting **AWS Fargate** under the infrastructure requirements option:

▼ Infrastructure requirements

Specify the infrastructure requirements for the task definition.

Launch type [Info](#)
Selection of the launch type will change task definition parameters.

☒ **AWS Fargate**
Serverless compute for containers.

- Then **OS as Linux**, remaining options keeping as the default:

Operating system/Architecture [Info](#)

Linux/X86_64 ▼

Network mode [Info](#)

awsvpc ▼

Task size [Info](#)
Specify the amount of CPU and memory to reserve for your task.

CPU

1 vCPU ▼

Memory

3 GB ▼

▼ Task roles - conditional

Task role [Info](#)
A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the [IAM console](#) [🔗](#).

- ▼

Task execution role [Info](#)
A task execution IAM role is used by the container agent to make AWS API requests on your behalf. If you don't already have a task execution IAM

None ▼

- Then naming the container, **inserting the registry URL**:

Container - 1 [Info](#)

Container details

Specify a name, container image, and whether the container should be marked as essential. Each task definition must have at least one essential container.

Name	Image URI	Essential container
<input type="text" value="python-application"/>	<input type="text" value="docker.io/ravivarman46/python:app"/>	<input type="text" value="Yes"/>

- Then under port mappings, mapping the application to the port **5000**:

Port mappings [Info](#)

Add port mappings to allow the container to access ports on the host to send or receive traffic. Any changes to port mappings configuration impacts the associated service.

Container port	Protocol	Port name	App protocol	
<input type="text" value="5000"/>	<input type="text" value="TCP"/>	<input type="text" value="python-application-5000"/>	<input type="text" value="HTTP"/>	<input type="button" value="Remove"/>

- Keeping the remaining options as the default one, click create:

Monitoring - optional

Configure your application trace and metric collection settings using the AWS Distro for OpenTelemetry integration.

Tags - optional [Info](#)

Tags help you to identify and organize your task definitions.

- The task definitions have been created successfully:

Task definition successfully created

py-app:1 has been successfully created. You can use this task definition to deploy a service or run a task.

Deploy

[Amazon Elastic Container Service](#) > [Task definitions](#) > [py-app](#) > [Revision 1](#) > Containers

py-app:1

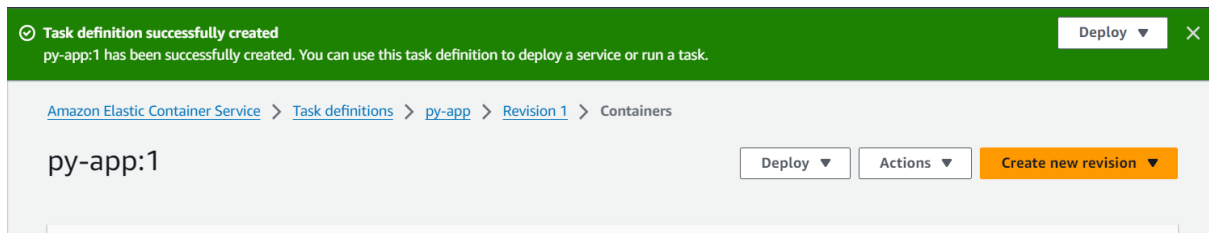
Deploy

Actions

Create new revision

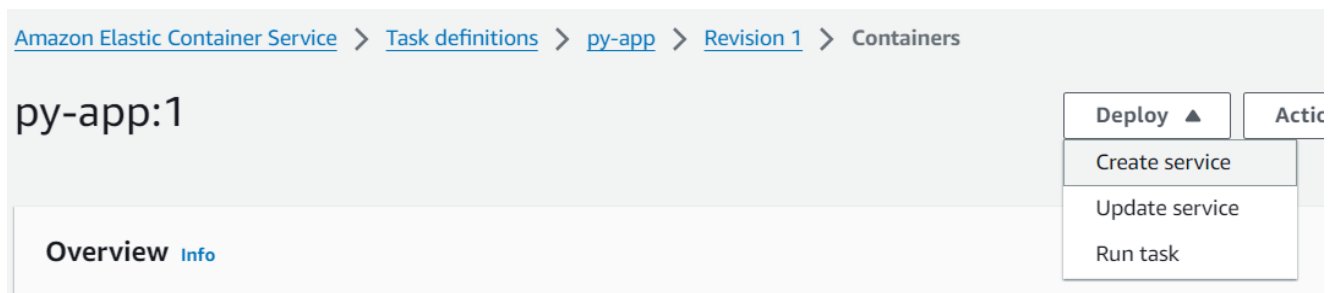
Deploying the task definitions as the service under the EC2 Cluster:

- Under the newly created task definitions, we could able to the **deploy option**, click that one:

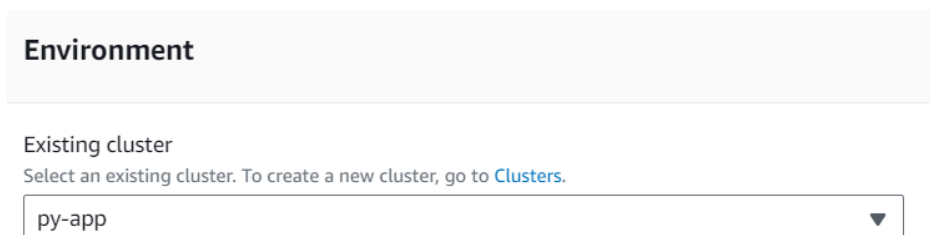


- Then we could able to see three options:
 - ➔ **Create service**
 - ➔ **Update service**
 - ➔ **Run task**

Here we need to create service to run a task: click **create service** option:



- Then choosing the **cluster**:



- Then selecting compute options: **launch type option**

▼ 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.

- Under launch type selecting: **AWS Fargate** option:

Launch type | [Info](#)

Select either managed capacity (Fargate), or c your cluster using the ECS Anywhere capabilit

FARGATE ▼

Platform version | [Info](#)

Specify the platform version on which to run :

LATEST ▼

- Then under deployment configuration: **selecting service option as the application type:**

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.

- Then naming the service:

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

py-app ▼

Revision

1

Service name

Assign a unique name for this service.

py-app

- Then under service type, keeping the default option:

Service type [Info](#)

Specify the service type that the service scheduler will follow.

☒ **Replica**

Place and maintain a desired number of tasks across your cluster.

☐ **Daemon**

Place and maintain one copy of your task on each container instance.

Desired tasks

Specify the number of tasks to launch.

1

► Deployment options

- Then creating load balancer for this application deployment:

▼ Load balancing - *optional*

Load balancer

Load balancer type [Info](#)

Configure a load balancer to distribute incoming traffic across the tasks running in your service.

None ▼

- Then selecting the **application load balancer**:

Load balancer

Load balancer type | [Info](#)

Configure a load balancer to distribute incoming traffic across the tasks running in your service.

Application Load Balancer ▼

- Then **naming** application load balancer:

Application Load Balancer

Specify whether to create a new load balancer or choose an existing one.

- ☒ Create a new load balancer
☐ Use an existing load balancer

Load balancer name

Assign a unique name for the load balancer.

project-6-application

- Changing the port number from **80** to **5000** where the application will **run**:

Listener | [Info](#)

Specify the port and protocol that the load balancer will listen for connection requests on.

- ☒ Create new listener
☐ Use an existing listener
You need to select an existing load balancer.

Port

5000

Protocol

HTTP ▼

- Then naming the target group:

Target group [Info](#)

Specify whether to create a new target group or choose an existing one that the load balancer will use to route requests to the tasks in your service.

☒ Create new target group

☐ Use an existing target group

You need to select an existing load balancer.

Target group name

project-6

Protocol

HTTP

Health check protocol

HTTP

Health check path [Info](#)

/

- Then keeping default option for the rest, click **create** option:

► Service auto scaling - *optional*

Automatically adjust your service's desired count up and down within a specified range in response to CloudWatch alarms. You can modify your service auto scaling configuration at any time to meet the needs of your application.

► Tags - *optional* [Info](#)

Tags help you to identify and organize your resources.

Cancel

Create

- The task definition has been updated successfully; deployment has been initiated. It would take some time for the deployment:

✓ Task definition successfully created

py-app:1 has been successfully created. You can use this task definition to deploy a service or run a task.

Deploy ▼

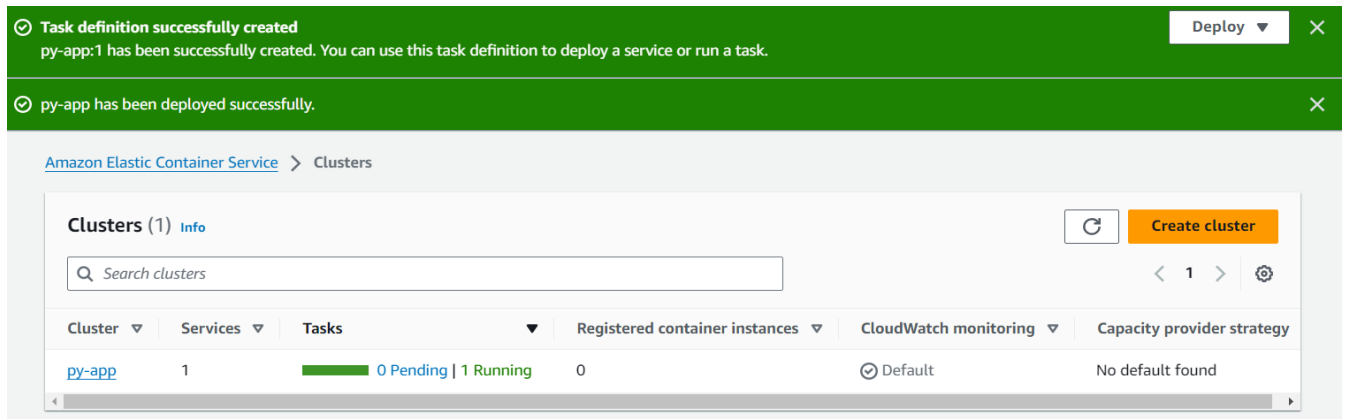
×

↻ py-app deployment is in progress. It takes a few minutes.

View in CloudFormation [↗](#)

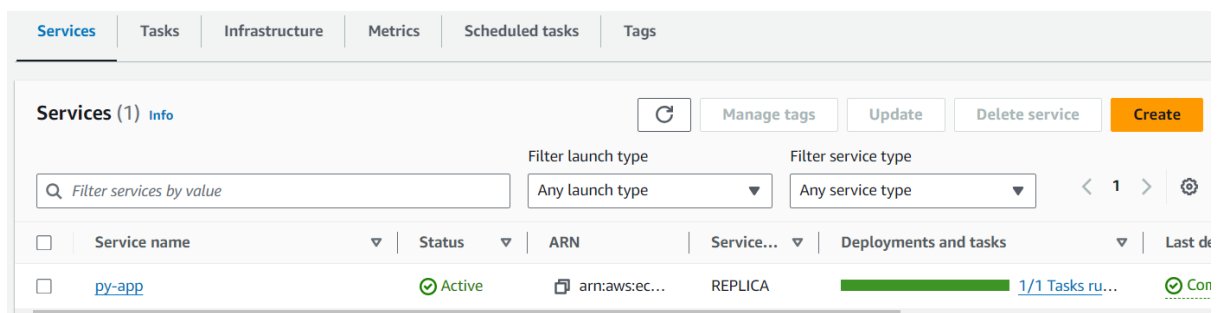
×

- After few minutes we could able to see the application is deployed successfully:

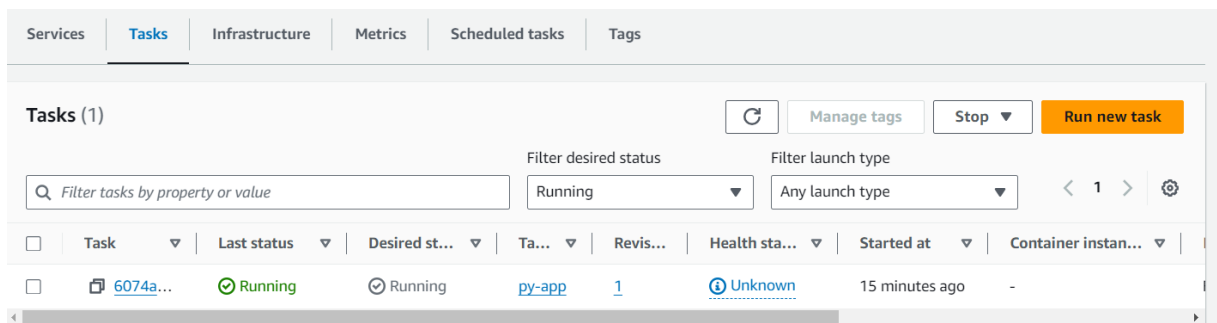


- Click the created cluster, to know about services & tasks:

Services:



Tasks:

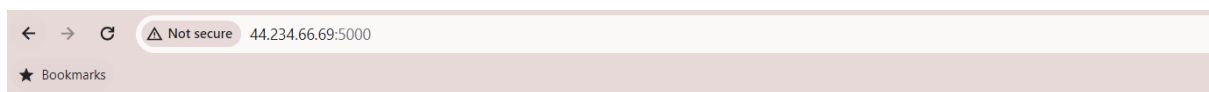


- Click the task to know about the ip address & etc:

Configuration			
Operating system/Architecture Linux/X86_64	Capacity provider -	ENI ID eni-09488bac7ab8a0761	Public IP 44.234.66.69 open address
CPU Memory 1 vCPU 3 GB	Launch type FARGATE	Network mode awsvpc	Private IP 172.31.50.4
Platform version 1.4.0	Task definition: revision py-app:1	Subnet ID subnet-0817a01dad7b0d7de	MAC address 0e:30:eb:c2:16:f3

Container details for python-application

- Copy the public ip address and paste it on the browser along with the port number:



- Copy and paste the **load balancer dns** on the browser along with port number:

EC2

>

Load balancers

Load balancers (1)

Actions

Create load balancer



Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Filter load balancers

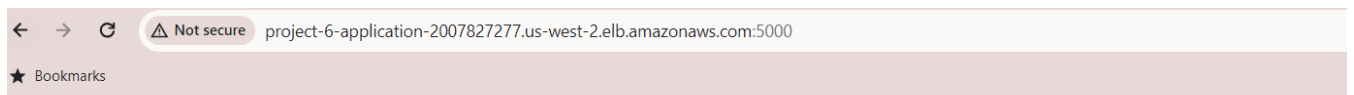
<

1

>

<input type="checkbox"/>	Name	DNS name	State	VPC ID	Availability Zones	Type	Date
<input type="checkbox"/>	project-6-application	 project-6-application-200...	 Active	vpc-07b957bb29075c...	4 Availability Zones	application	Nov

The load balancer output:



Hello,
this python application runs from Amazon ECS with the help of Jenkins CI/CD Pipeline!!!

Step:8 – Automating the deployment process by using Jenkins:

Installing Jenkins & AWS CLI:

- Script to install Jenkins:

```
sudo apt-get update
sudo apt-get install -y openjdk-11-jre
sudo apt-get update

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update
sudo apt-get install jenkins -y
```

```
root@ip-172-31-29-145:/home/ubuntu# ls
app.py dockerfile requirements.txt
root@ip-172-31-29-145:/home/ubuntu# vi jenkins.sh
root@ip-172-31-29-145:/home/ubuntu# ls
app.py dockerfile jenkins.sh requirements.txt
root@ip-172-31-29-145:/home/ubuntu# chmod 777 jenkins.sh
root@ip-172-31-29-145:/home/ubuntu# ./jenkins.sh
Hit:1 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal InRelease
Hit:2 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:3 http://us-west-2.ec2.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu focal-security InRelease
```

- Checking Jenkins:

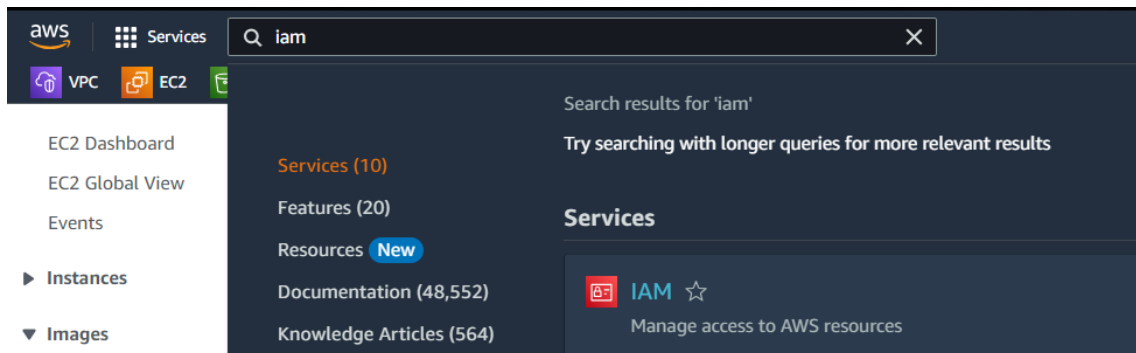
```
root@ip-172-31-29-145:/home/ubuntu# jenkins --version
2.414.3
root@ip-172-31-29-145:/home/ubuntu# systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2023-11-13 17:49:19 UTC; 27s ago
     Main PID: 7629 (java)
       Tasks: 50 (limit: 4671)
```

- Command to install aws cli: `apt-get install -y awscli`

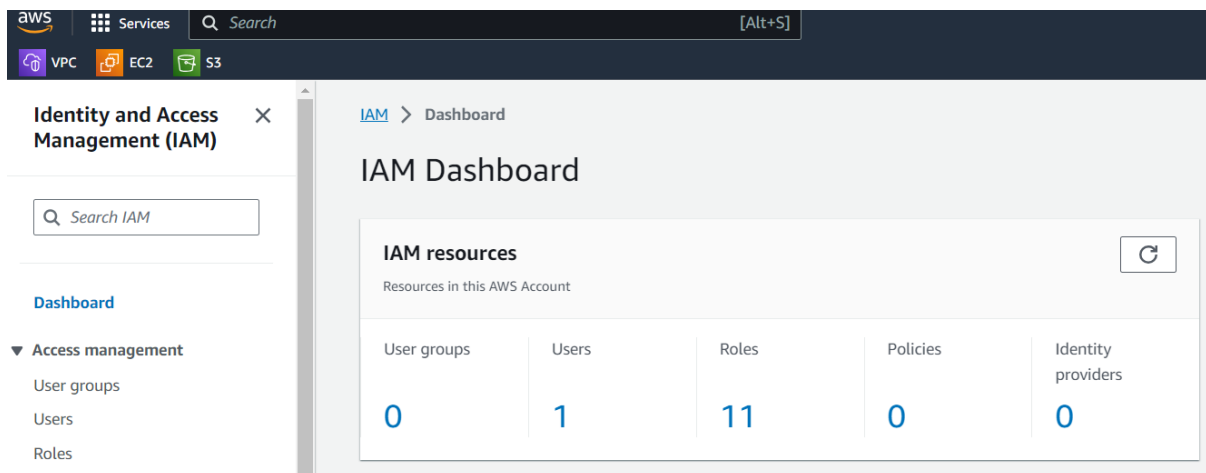
```
root@ip-172-31-29-145:/home/ubuntu# aws --version
aws-cli/1.18.69 Python/3.8.10 Linux/5.15.0-1048-aws botocore/1.16.19
root@ip-172-31-29-145:/home/ubuntu#
```

Creating an IAM User:

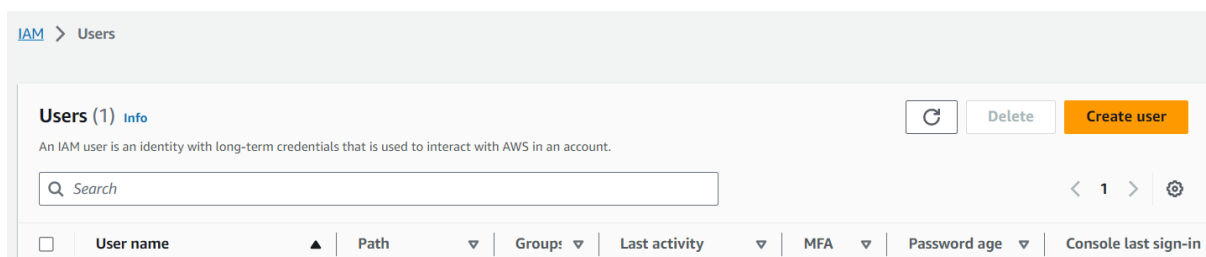
- On the search panel, search IAM, and click that one:



- Then click the user:



- Then click create user option on left side:



- Then name the IAM user, click next:

Specify user details

User details

User name

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a [best practice](#) to manage their access in IAM Identity Center.

📘 If you are creating programmatic access through access keys or service-specific credentials for AWS CodeCommit or Amazon Keyspaces, you can generate them after you create this IAM user. [Learn more](#)

Cancel Next

- Then select attach policies directly option:

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

☐ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

☐ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

☒ Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

- Then select policy **AmazonECS_FullAccess**, scroll down & click next:

Permissions policies (1/1142)

Choose one or more policies to attach to your new user.

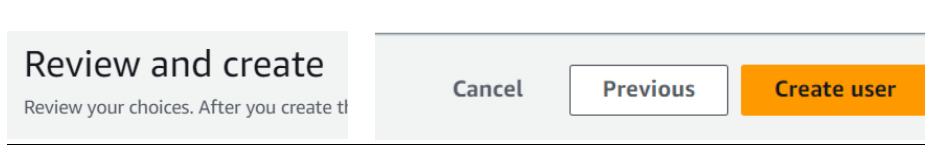
Filter All t

	Policy name	Type
<input checked="" type="checkbox"/>	AmazonECS_FullAccess	AWS managed

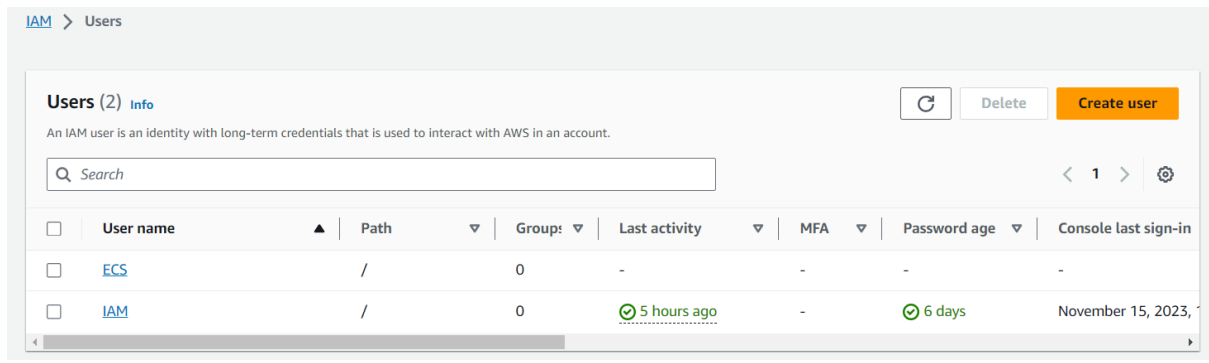
► Set permissions boundary - *optional*

Cancel Previous Next

- Then review & create page, just review the configurations click create user option:

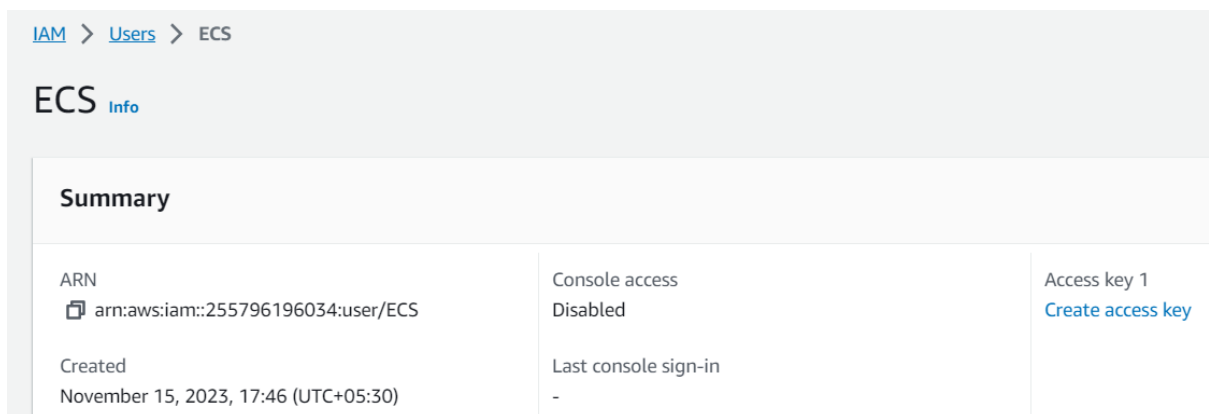


- The IAM user has been created successfully:



Creating access key & secret access key for the IAM user:

- Click the newly create IAM User, we could able to see create access key option. Click that one:



- Select the use-case option for creating access key, then acknowledge & click next:

Access key best practices & alternatives [Info](#)

Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Use case

☒ Command Line Interface (CLI)

You plan to use this access key to enable the AWS CLI to access your AWS account.

Confirmation

☒ I understand the above recommendation and want to proceed to create an access key.

[Cancel](#)

[Next](#)

- Then give the description, click create access key option:

Set description tag - *optional* [Info](#)

The description for this access key will be attached to this user as a tag and shown alongside the access key.

Description tag value

Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.

ECS

Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ . : / = + - @

[Cancel](#)

[Previous](#)

[Create access key](#)

- The access key & secret key has been created for the **IAM user: ECS**

Retrieve access keys [Info](#)

Access key

If you lose or forget your secret access key, you cannot retrieve it. Instead, create a new access key.

Access key

 AKIATXDVEGLBCYCPLD5H

Secret access key

 ***** [Show](#)

Configuring AWS Cli credentials:

- Configuring the aws credentials with the help of the command as the **Jenkins user**, with the help of the command: **aws configure**:

```
root@ip-172-31-29-145:/home/ubuntu# su jenkins
jenkins@ip-172-31-29-145:/home/ubuntu$ aws configure
AWS Access Key ID [None]: ASIAUIXM6DFDJZL3ETHX
AWS Secret Access Key [None]: CtwFRhcJAAM8YwTk+tteM6PkNfM9D2V7hWd5RIub
Default region name [None]: us-west-2
Default output format [None]: json
jenkins@ip-172-31-29-145:/home/ubuntu$ █
```

Configuring Jenkins GUI:

- Getting the initial admin password by using the command and paste on the browser & click continue:

```
root@ip-172-31-29-145:/home/ubuntu# cat /var/lib/jenkins/secrets/initialAdminPassword
c6c0ffa23d9849efa6b9cb2192d47bd3
root@ip-172-31-29-145:/home/ubuntu# █
```

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password the log ([not sure where to find it?](#)) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

.....

- Then select the option to install plugins, here I am selecting **install suggested plugins option**:

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

- Once selecting the option, the plugins will start to download and install:

Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding	** JUnit
✓ Timestampers	✓ Workspace Cleanup	✓ Ant	✓ Gradle	** Matrix Project
⌚ Pipeline	⌚ GitHub Branch Source	⌚ Pipeline: GitHub Groovy Libraries	⌚ Pipeline: Stage View	** Resource Disposer
⌚ Git	⌚ SSH Build Agents	⌚ Matrix Authorization Strategy	⌚ PAM Authentication	Workspace Cleanup
⌚ LDAP	⌚ Email Extension	✓ Mailer		Ant
				** Durable Task
				** Pipeline: Nodes and Processes
				** Pipeline: SCM Step
				** Pipeline: Groovy
				** Pipeline: Job
				** Jakarta Activation API
				** Jakarta Mail API
				** Apache HttpComponents Client 4.x API
				Mailer
				** Pipeline: Basic Steps
				Gradle
				** Pipeline: Milestone Step
				** Pipeline: Build Step
				** Variant
				** Pipeline: Groovy Libraries
				** Pipeline: Stage Step

- Then setting up the user credentials on the Jenkins for login purpose:

Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

3

[Skip and continue as admin](#)

[Save and Continue](#)

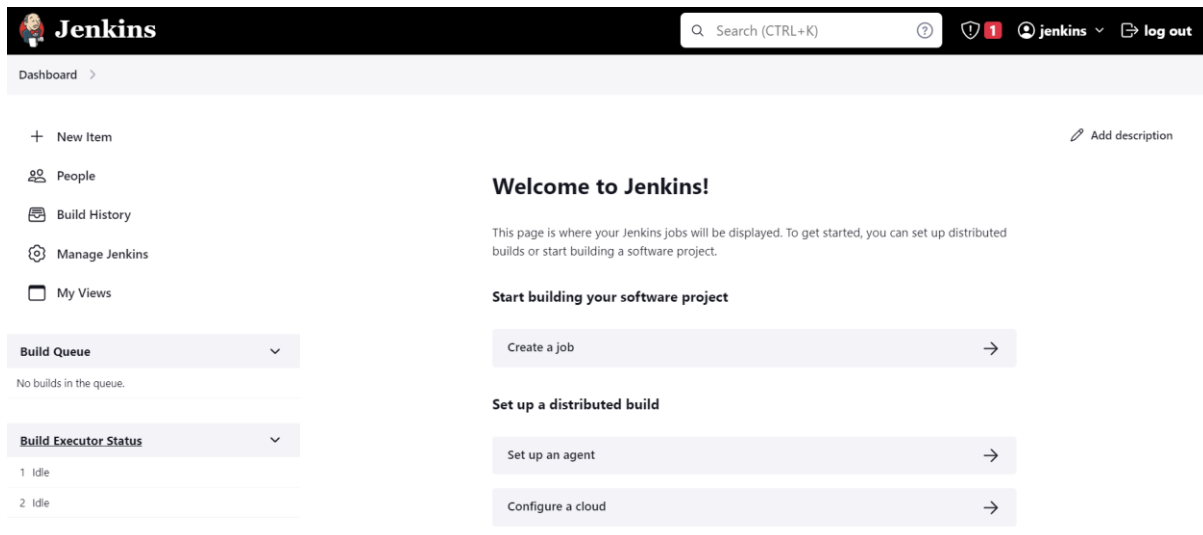
- Then click save and continue option: we could able to see the Jenkins is ready, **click the start using Jenkins:**

Jenkins is ready!

Your Jenkins setup is complete.

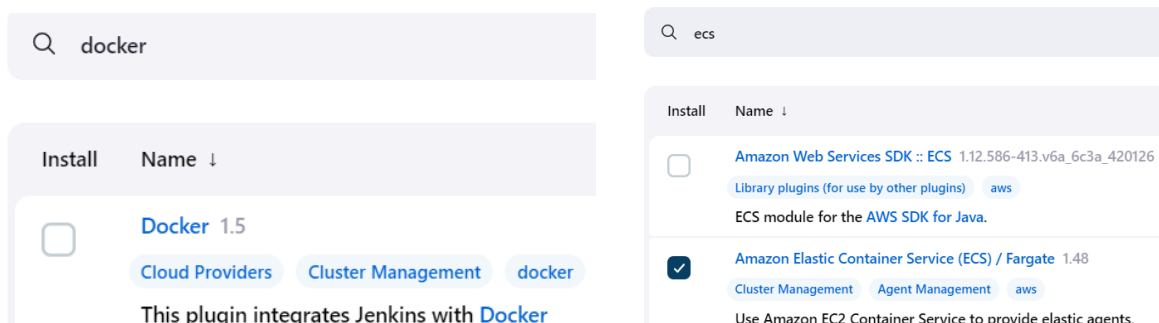
[Start using Jenkins](#)

- The Jenkins dashboard is ready:



Installing the necessary plugins:

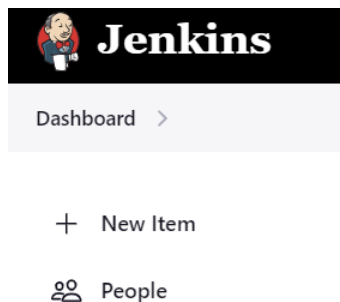
- On the dashboard we could able to see manage Jenkins option, under that we could able to see plugins click that one:
- For the project we need two plugins, they are available on available plugins option:
 - ➔ **Docker plugin**
 - ➔ **Amazon elastic container service**



Then we need to create a Jenkins job for automating purpose.

Creating a Jenkins job:

- On Jenkins dashboard, we could able to **new item** click that one:



- Then **name the job and select the pipeline type**, click okay:

The screenshot shows the 'Enter an item name' dialog box. It has a text input field containing 'JENKINS-DEPLOYMENT'. Below the field is a message '» Required field'. There are four options for project types: 'Freestyle project' (with a box icon), 'Pipeline' (with a pipe icon), 'Multi-configuration project' (with a document icon), and 'Folder' (with a folder icon). Each option has a brief description. At the bottom, there is a blue 'OK' button.

- Then give the description according the project:

General

Description

CONTINUOUS INTEGRATION & CONTINUOUS DEPLOYMENT

- Then select the option Github hook trigger under build triggers for automated trigger purpose:

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ GitHub hook trigger for GITScm polling ?
- ☐ Poll SCM ?
- ☐ Quiet period ?
- ☐ Trigger builds remotely (e.g., from scripts) ?

- Then under pipeline, just we need to enter the script: after entering the script click apply & save option:

Pipeline

Definition

Pipeline script

Script ?

```

1 pipeline {
2   agent any
3   stages {
4     stage ('cloning the GitHub repo:') {
5       steps {
6         git branch: 'master',
7         url: 'https://github.com/Ravivarman16/amazon_ecs-python.git'
8       }
9     }
10
11    stage ('Building a Docker-image:') {
12      steps {
13        sh 'docker build -t ravivarman46/python:app .'
14      }
15    }
16
17    stage ('Pushing it to the Docker-Hub:') {

```

- ☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

Save

Apply

- **Pipeline script contains: 4 stages**
 - ➔ **1st stage:** cloning the code from GitHub.
 - ➔ **2nd stage:** building the docker image from dockerfile.
 - ➔ **3rd stage:** pushing the created docker image to docker hub.
 - ➔ **4th stage:** deploying the newly created docker image to Amazon ECS Cluster.

```
pipeline {
  agent any
  stages {
    stage ('cloning the GitHub repo:') {
      steps {
        git branch: 'master',
        url:
'https://github.com/Ravivarman16/amazon_ecs-python.git'
      }
    }

    stage ('Building a Docker-image:') {
      steps {
        sh 'docker build -t ravivarman46/python:app
.'
      }
    }

    stage ('Pushing it to the Docker-Hub:') {
      steps {
        sh ' docker login -u ravivarman46 -p
dckr_pat_4RpB6x_mUNVKrFCLk2W5pqBnywE '
        sh ' docker push ravivarman46/python:app '
      }
    }

    stage ('deploying to ecs') {
      steps {
        sh 'aws ecs update-service --cluster py-app
--service py-app --force-new-deployment'
      }
    }
  }
}
```

➔ We could able to see the Jenkins job is ready:

Dashboard > jENKINS-DEPLOYMENT >

Status

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline

🔍 Full Stage View

✎ Rename

❓ Pipeline Syntax

📄 GitHub Hook Log

Pipeline jENKINS-DEPLOYMENT

CONTINUOUS INTEGRATION & CONTINUOUS DEPLOYMENT

Stage View

No data available. This Pipeline has not yet run.

Permalinks

- Before executing the pipeline, on the command line use this command for Jenkins to execute the docker daemon:

```
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6# chmod 777 /var/run/docker.sock
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6#
```

- Now click build now option on the Jenkins pipeline, we could able to see the pipeline execute successfully:

Dashboard > ecs-deployment >

Status

</> Changes

▶ Build Now

⚙️ Configure

🗑️ Delete Pipeline

🔍 Full Stage View

✎ Rename

❓ Pipeline Syntax

📄 GitHub Hook Log

Pipeline ecs-deployment

Stage View

Average stage times:
(Average full run time: ~31s)

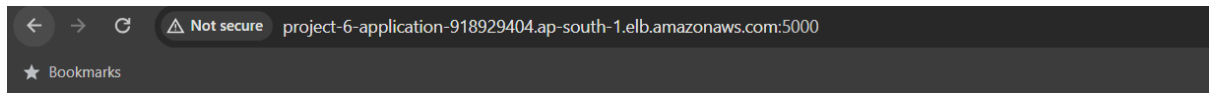
	cloning the GitHub repo:	Building a Docker-image:	Pushing it to the Docker-Hub:	deploying to ecs
	2s	7s	7s	1s
#2 Nov 15 18:06 No Changes	596ms	14s	14s	1s

Console Output

```
Started by user jenkins
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/ecs-deployment
```

```
}
}
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

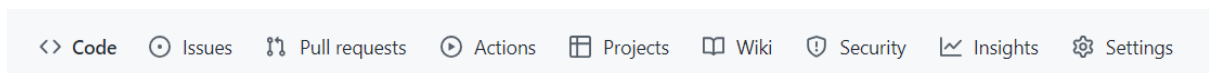
- Let us check the output from the browser:



**Hello,
this python application runs from Amazon ECS with the help of Jenkins CI/CD Pipeline!!!**

Setting GitHub hook trigger for automated deployment:

- On the GitHub repository, we could able to see the **settings** option, click that one:



- On the left side, we could able to see **webhook** option click that one:



- Then **click add webhook option** on the right side:

Webhooks

Add webhook

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our [Webhooks Guide](#).

- Then we need to enter the **Jenkins URL along with github-webhook** option on the **payload URL**:

Payload URL *

http://13.234.117.106:8080/github-webhook/

Content type

application/x-www-form-urlencoded

- Then select the events according to your preferences, here I am selecting **send me everything** option. Then click add webhook option:

Which events would you like to trigger this webhook?

- ☐ Just the push event.
- ☒ Send me everything.
- ☐ Let me select individual events.

☒ Active

We will deliver event details when this hook is triggered.

Update webhook

Delete webhook

- Then on the Jenkins side, click **manage Jenkins**, under that click **system**:

Manage Jenkins

Building on the built-in node can be a security issue. You

System Configuration



System

Configure global settings and paths.

- Then on the middle of the screen we could able to find out GitHub option: on that we could able to advanced option, click that one:

GitHub

GitHub Servers ?

Add GitHub Server ▾

Advanced ▾

 Edited

- On the advanced option, we could able to see **override hook URL** option click that one: then enter **Jenkins URL along with github-webhook &** click apply save:

Advanced ^

 Edited

Override Hook URL

☒ Specify another hook URL for GitHub configuration

- Then make a change in the application code, and save it.

```
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6# vi app.py
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6#
```

```
# app.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return '<b>Hello,</b><br><b>Project 6: Setting up a Continuous Delivery Pipeline with Git, Jenkins, Docker, and AWS ECS!!!</b>'

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0')
```

- Checking the **git status**:

```
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6# git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   app.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        jenkins.sh
```

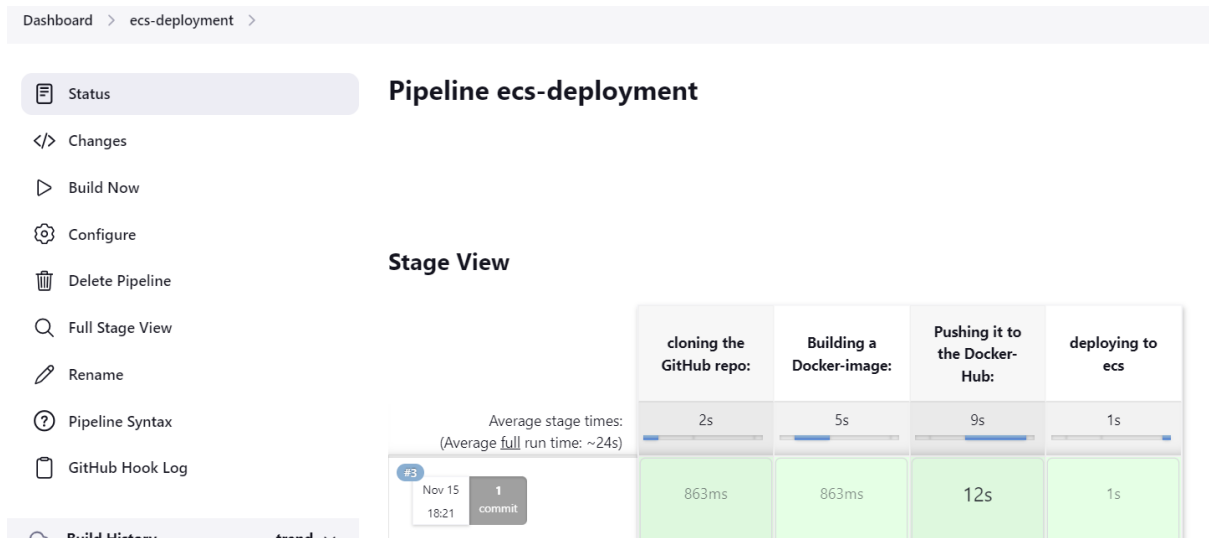
- Then staging it and committing it.

```
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6# git add .
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6# git commit -m "updated app.py"
[master b814550] updated app.py
Committer: root <root@ip-172-31-47-53.ap-south-1.compute.internal>
```

- Then pushing it to the remote repo:

```
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6# git push origin master
Username for 'https://github.com': Ravivarman46
Password for 'https://Ravivarman46@github.com':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 717 bytes | 717.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Ravivarman16/use-case-project-6.git
   699d1c3..b814550  master -> master
root@ip-172-31-47-53:/home/ubuntu/use-case-project-6#
```


- The Jenkins job got triggered automatically, when there is a code change in GitHub:



- On the AWS ECS Cluster we could see its updating the service:

Amazon Elastic Container Service > Clusters

Clusters (1) [Info](#)

Cluster	Services	Tasks	Registered container instances
py-app	1	1 Pending 1 Running	0

- Checking the output on the browser:



Hello,
Project 6: Setting up a Continuous Delivery Pipeline with Git, Jenkins, Docker, and AWS ECS!!!

← → ↻ ⚠ Not secure project-6-application-918929404.ap-south-1.elb.amazonaws.com:5000

★ Bookmarks

Hello,

Project 6: Setting up a Continuous Delivery Pipeline with Git, Jenkins, Docker, and AWS ECS!!!

**The application has deployed successfully to the AWS ECS
Cluster**

Reference: <https://github.com/Ravivarman16/use-case-project-6.git>