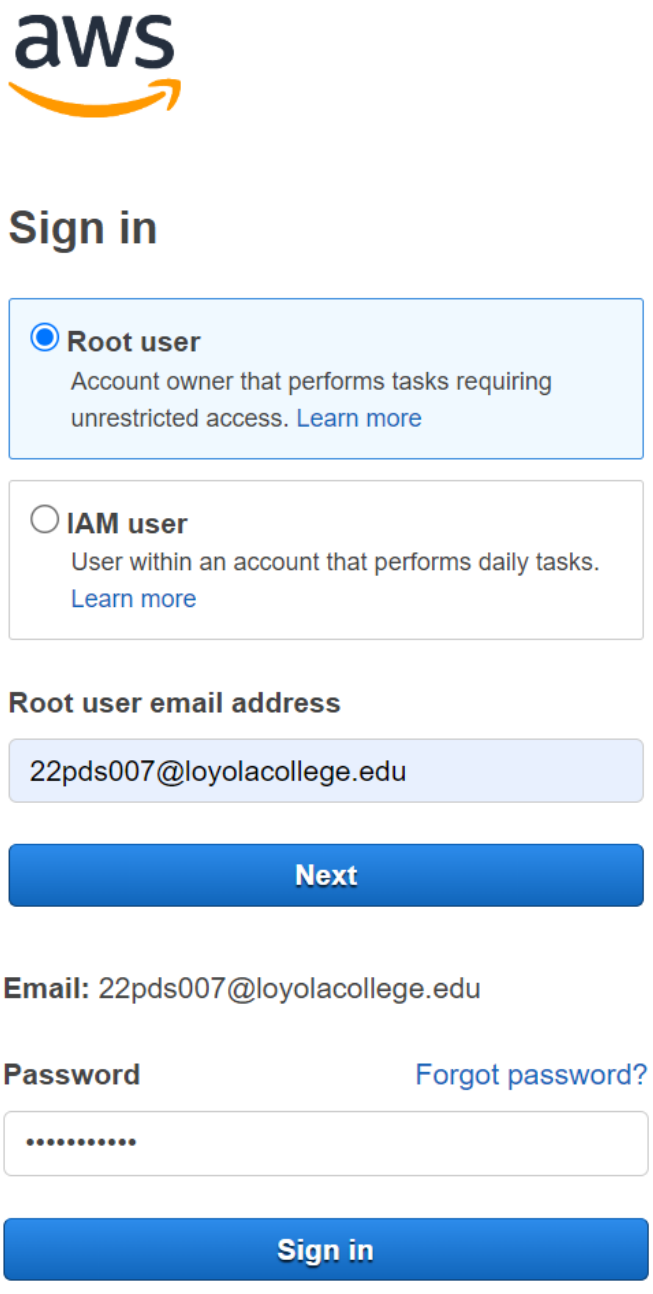
Creating an EC2 Instance and Deploying a Sample Web Application

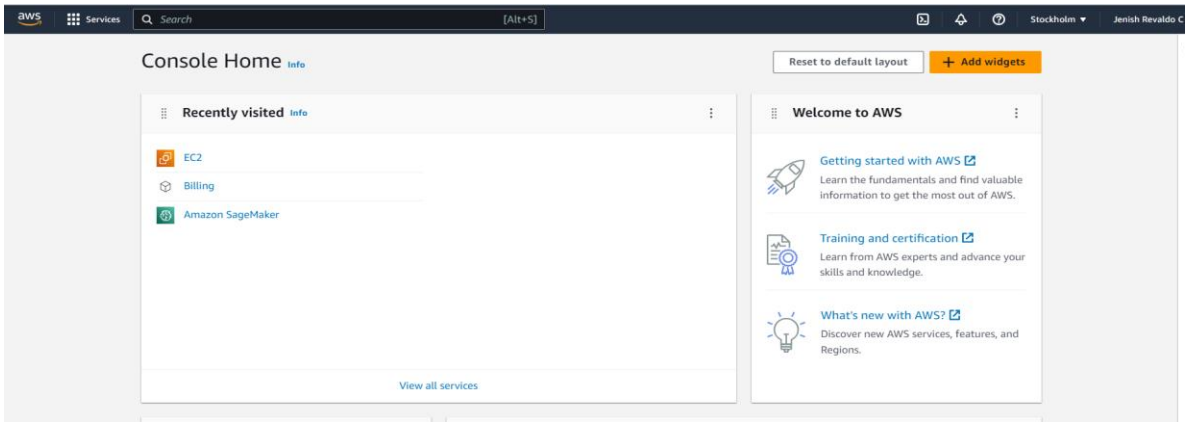
Aim:

The aim of this exercise is to demonstrate the process of creating an EC2 instance on Amazon Web Services (AWS) and deploying a sample web application on the instance.

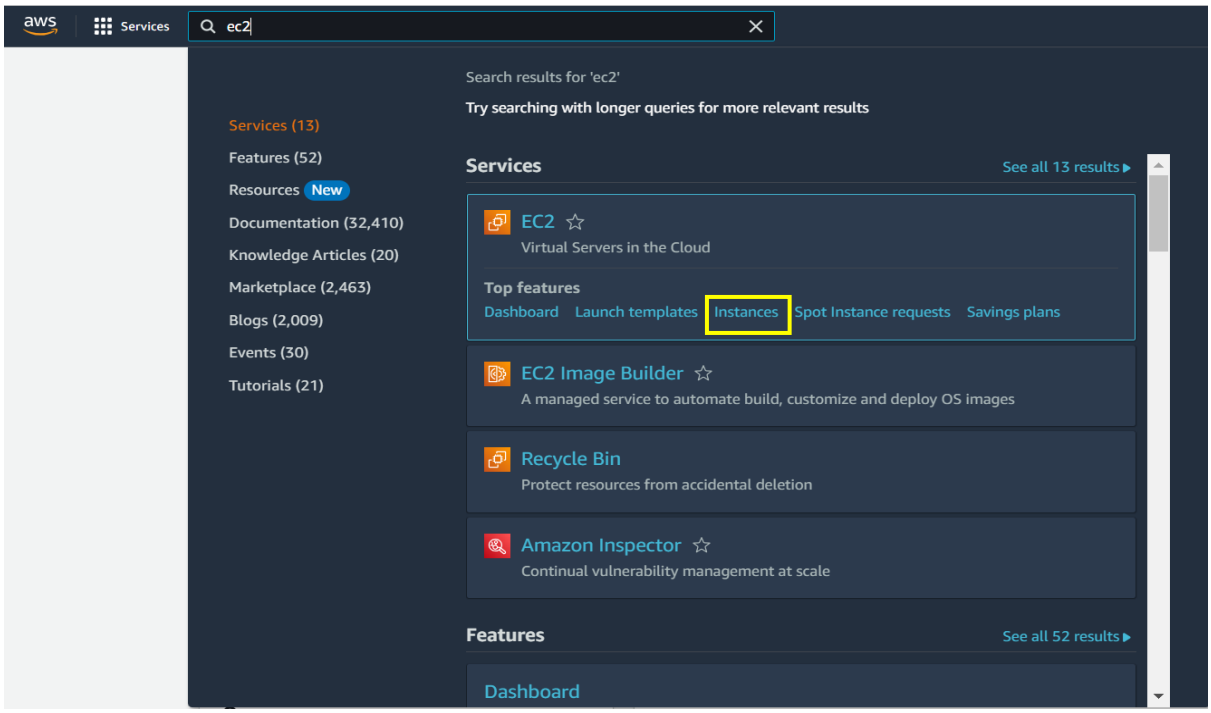
Procedure:

Step 1: Log in to your AWS Management Console.

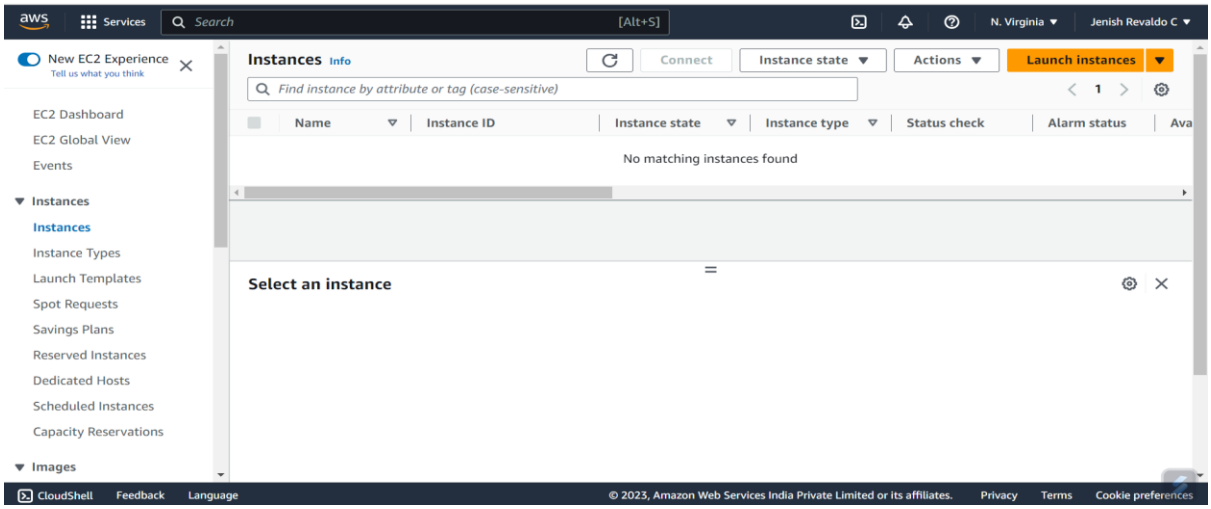




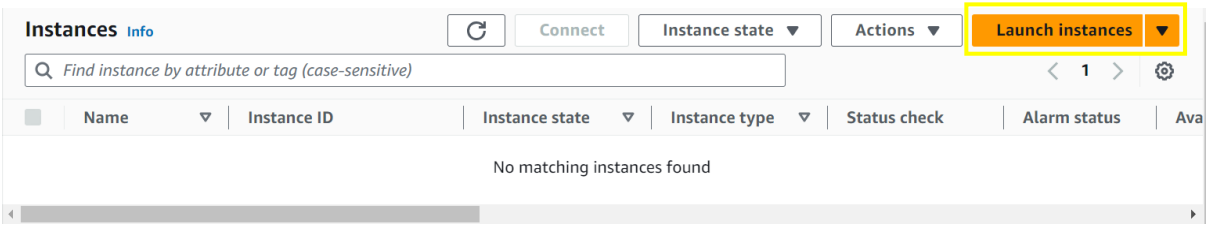
Step 2: Navigate to the EC2 Dashboard.



#EC2 Dashboard

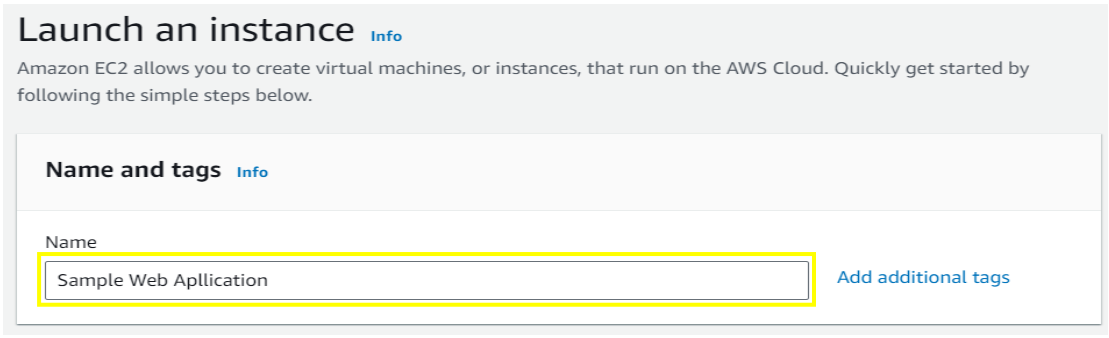


Step 3: Click on "Launch Instance" to start the EC2 instance creation wizard.

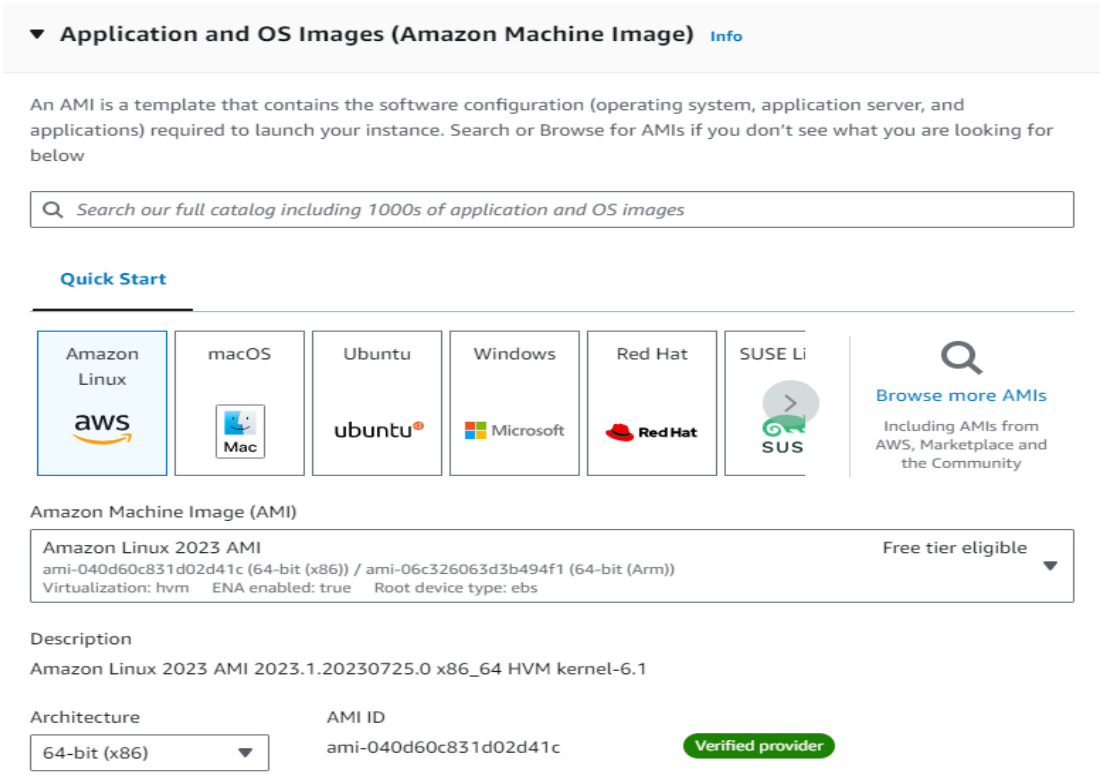


Step 4:

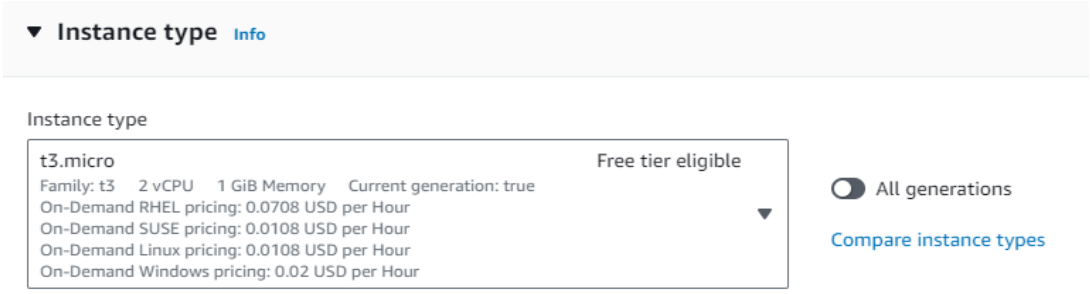
- Add Name to the Instance



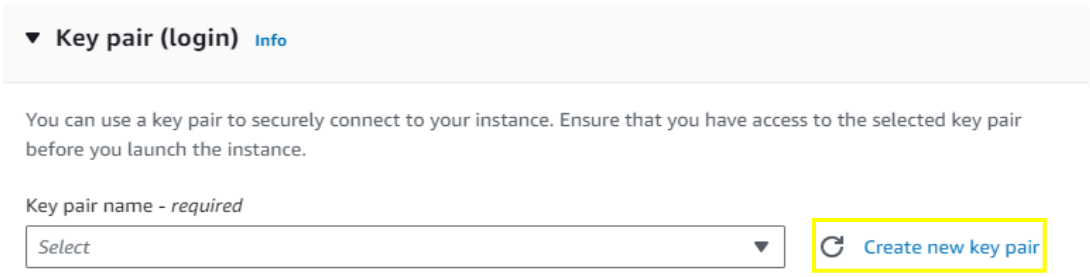
- Choose an Amazon Machine Image (AMI) for your instance (e.g., Amazon Linux 2).



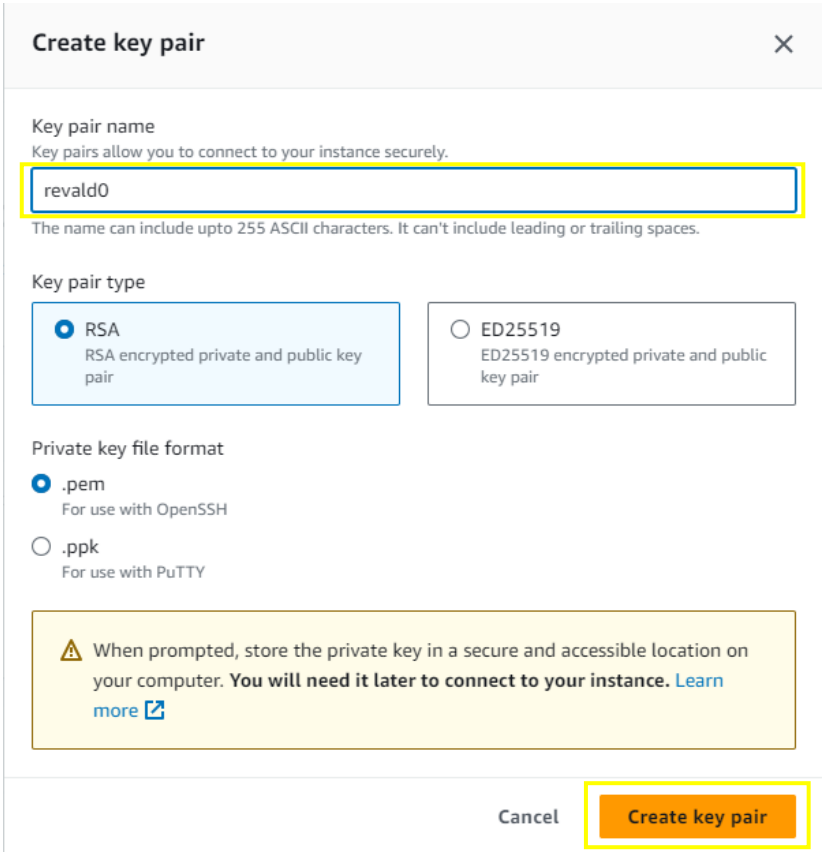
- Select an instance type (e.g., t2.micro) based on your requirements.

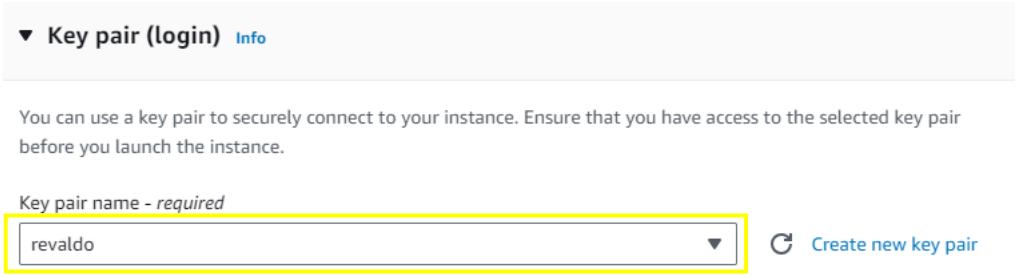


- Create a new key pair or use an existing one to securely connect to the instance later.

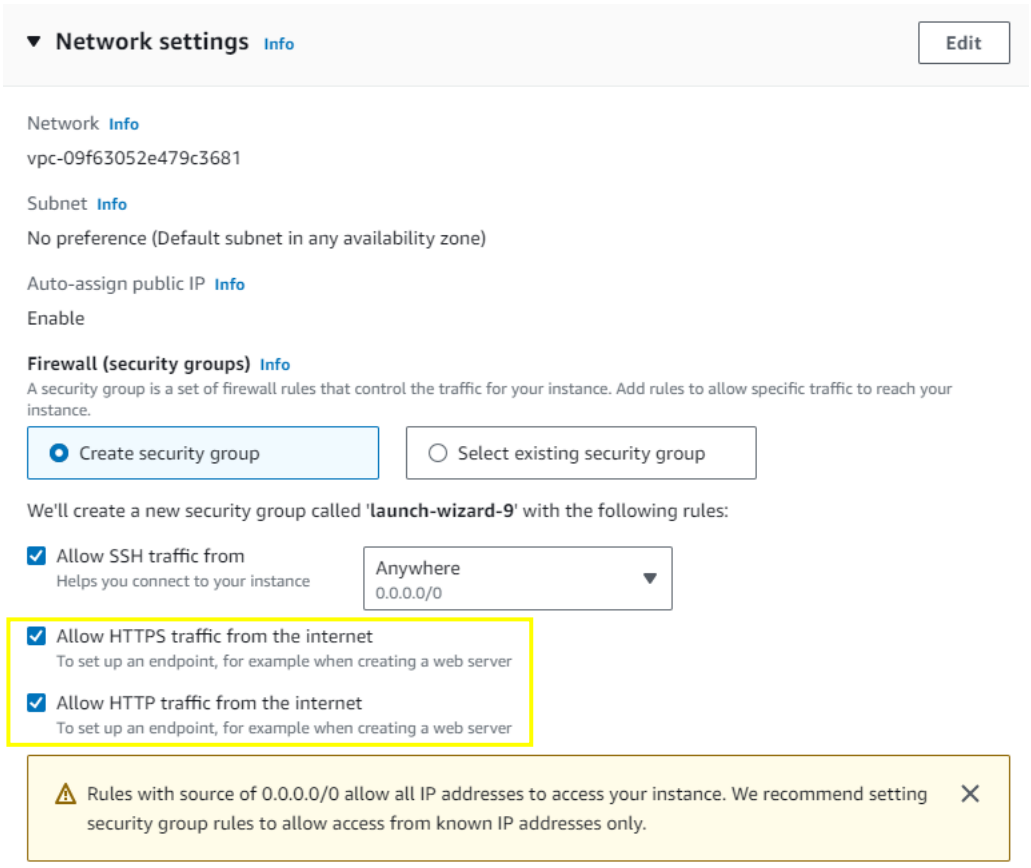


#Give name and create key pair

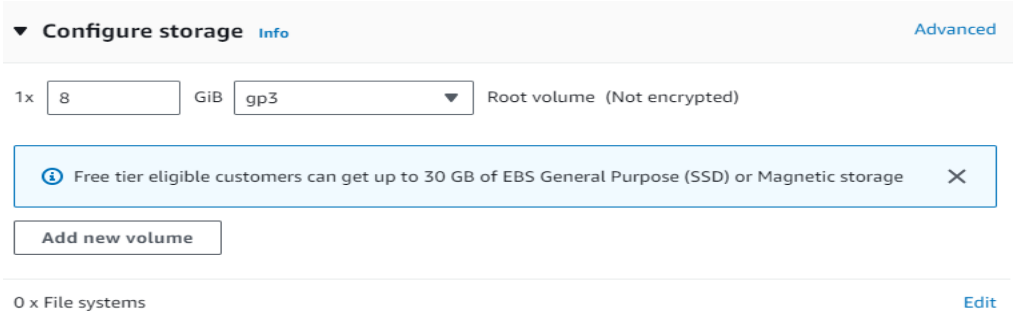




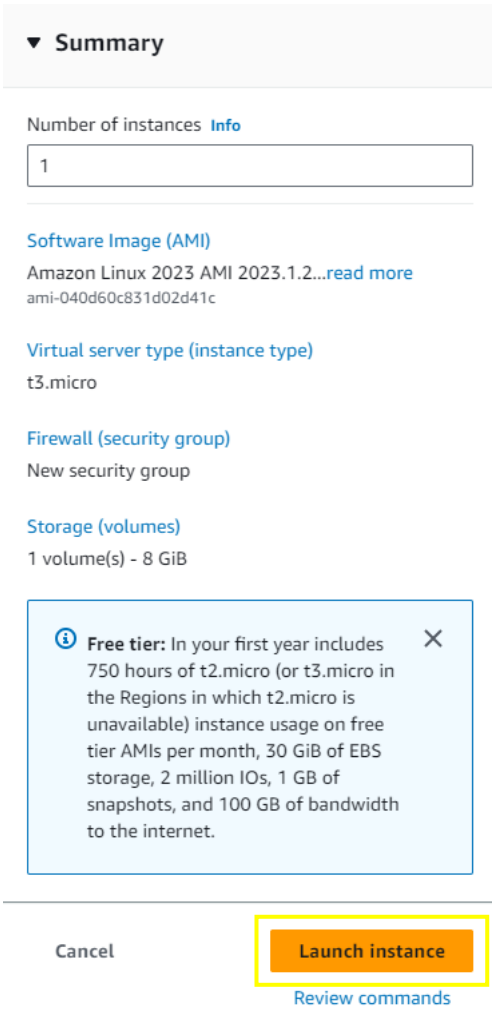
- Configure the instance details (e.g., network settings, security groups).



- Add storage if needed.



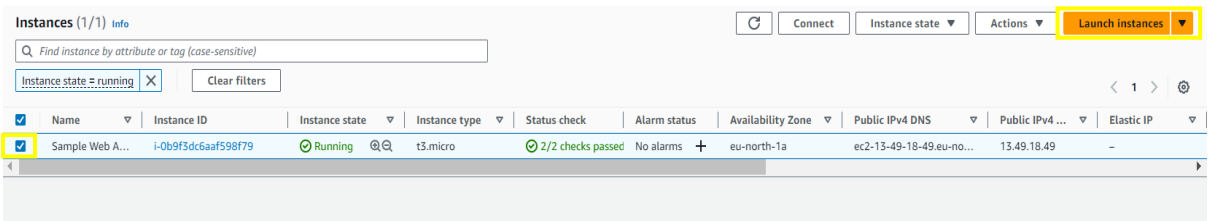
- Review your settings and click on "Launch."



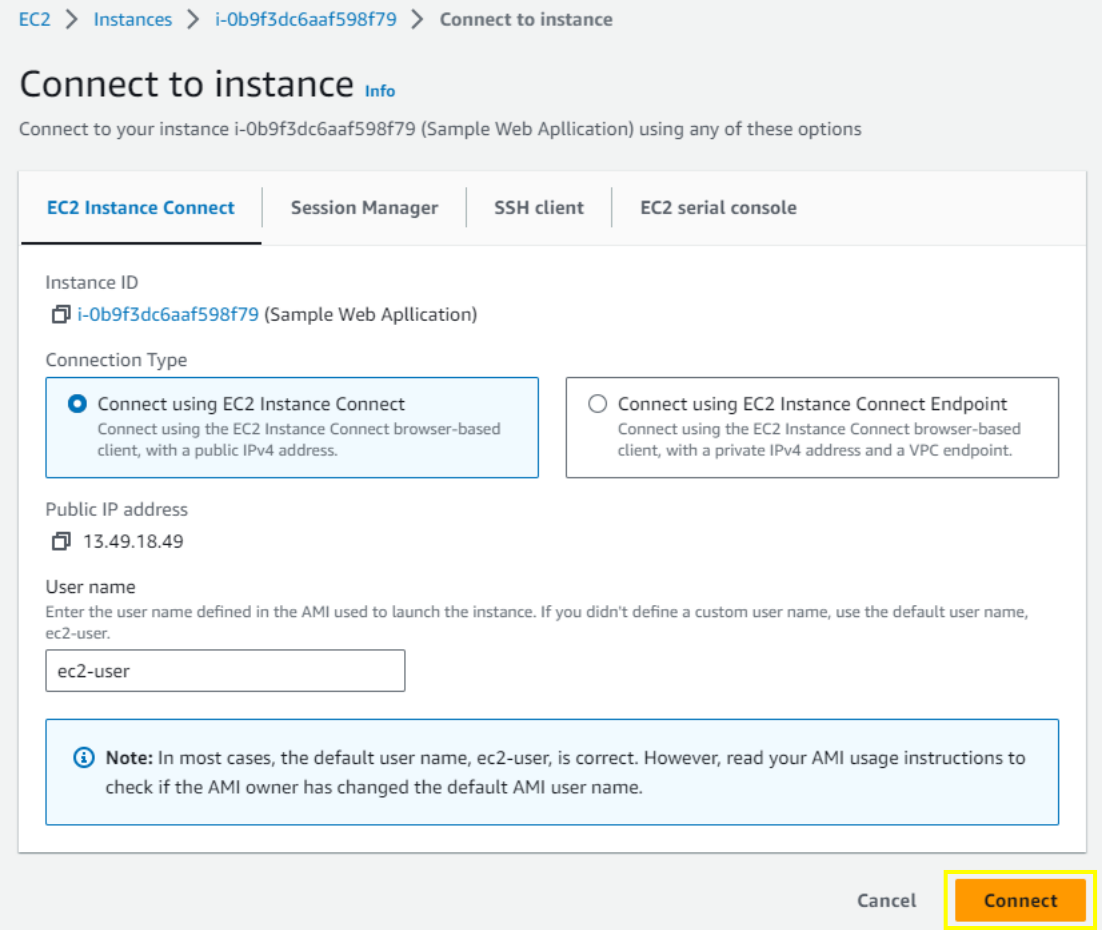
#After Launching

Step 5: Connecting the Instance

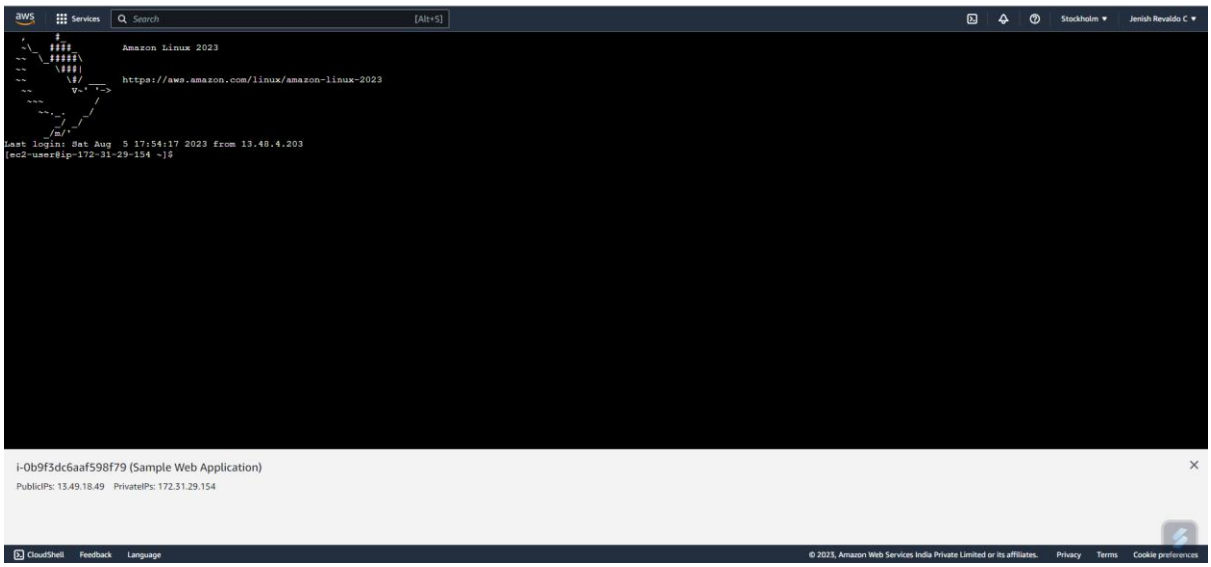
#Move to EC2 Dashboard, Select the newly created Instance and click Connect.



#Connect to instance using EC2 Instance Connect



#After establishing connection



Step 6: Switch to the superuser (root) account in Linux systems

```
[ec2-user@ip-172-31-29-154 ~]$ sudo su –
```

Step 7: Update the package manager and install Apache HTTP server.

```
[root@ip-172-31-29-154 ~]# yum update -y
```

```
[root@ip-172-31-29-154 ~]# yum install -y httpd
```

Step 8: Check the status of the Apache HTTP server

```
[root@ip-172-31-29-154 ~]# systemctl status httpd
```

- *httpd.service - The Apache HTTP Server*
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; preset: disabled)
  Active: inactive (dead)
  Docs: man:httpd.service(8)

Step 9: Make a directory temp1 and change to that directory

```
[root@ip-172-31-29-154 ~]# mkdir temp1
```

```
[root@ip-172-31-29-154 ~]# cd temp1
```

Step 10: Download the template from free-css.com using wget command

```
[root@ip-172-31-38-20 temp1]# wget https://www.free-css.com/assets/files/free-css-templates/download/page293/fitapp.zip
```

Step 11: Unzip the file

```
[root@ip-172-31-38-20 temp1]# unzip fitapp.zip
```

Step 12: List files and directories in the directory in long format.

```
[root@ip-172-31-29-154 temp1]# ls -ltr
```

total 552

-rw-r--r--. 1 root root 564102 Jan 7 2022 fitapp.zip

drwxr-xr-x. 7 root root   153 Aug 5 18:00 mobile-app-html-template

Step 13: Change directory to mobile-app-html-template

```
[root@ip-172-31-29-154 temp1]# cd mobile-app-html-template
```

Step 14: Move all the files to the Apache web server's document root.

```
[root@ip-172-31-29-154 mobile-app-html-template]# mv * /var/www/html
```

Step 15: Start the Apache web server

```
[root@ip-172-31-29-154 mobile-app-html-template]# service httpd start
```

Step 16: Check the status of the Apache HTTP server

```
[root@ip-172-31-29-154 mobile-app-html-template]# systemctl status httpd
```

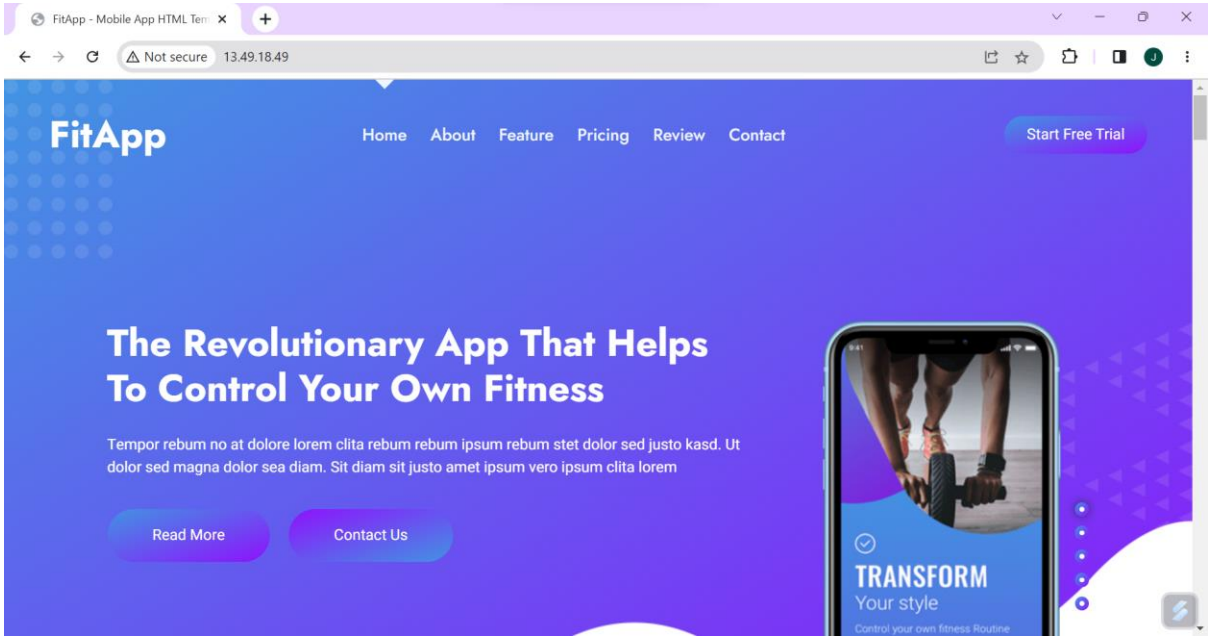
- *httpd.service - The Apache HTTP Server*

  Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; preset: disabled)

  Active: active (running) since Sat 2023-08-05 18:02:26 UTC; 16s ago

  Docs: man:httpd.service(8)

Output:



Result:

The steps for creating an Amazon EC2 instance, installing the Apache HTTP Server, and deploying a sample web application (FitApp) have been successfully performed, resulting in a fully operational web server hosting the web application on the EC2 instance.

Deployment of a basic Python web application using the Flask framework

Aim:

To demonstrate the deployment of a basic Python web application using the Flask framework on an Ubuntu EC2 instance.

Procedure:

Step 1: Create an EC2 instance with Ubuntu in Amazon Machine Image allowing HTTP and HTTPS traffic.

Step 2: Update package information on the Ubuntu EC2 instance.

```
ubuntu@ip-172-31-46-36:~$ sudo apt update
```

Step 3: Install Python3 package manager (pip) on the instance.

```
ubuntu@ip-172-31-46-36:~$ sudo apt install python3-pip
```

Step 4: Check the version of pip to verify installation.

```
ubuntu@ip-172-31-46-36:~$ pip --version
```

pip 22.0.2 from /usr/lib/python3/dist-packages/pip (python 3.10)

Step 5: Check the version of Git to verify installation.

```
ubuntu@ip-172-31-46-36:~$ git --version
```

git version 2.34.1

Step 6: Clone the GitHub repository containing the Flask web application code.

```
ubuntu@ip-172-31-46-36:~$ git clone https://github.com/yeshwanthlm/docker-flask-demo
```

Step 7: List files and directories in the directory in long format.

```
ubuntu@ip-172-31-46-36:~$ ls -ltr
```

total 4

drwxrwxr-x 4 ubuntu ubuntu 4096 Jul 15 16:13 docker-flask-demo

Step 8: Navigate to the cloned directory.

```
ubuntu@ip-172-31-46-36:~$ cd docker-flask-demo
```

Step 9: List the contents of the directory to see the files.

```
ubuntu@ip-172-31-46-36:~/docker-flask-demo$ ls -ltr
```

total 20

drwxrwxr-x 2 ubuntu ubuntu 4096 Jul 15 16:13 templates

-rw-rw-r-- 1 ubuntu ubuntu 5 Jul 15 16:13 requirements.txt

-rw-rw-r-- 1 ubuntu ubuntu 2258 Jul 15 16:13 app.py

-rw-rw-r-- 1 ubuntu ubuntu 687 Jul 15 16:13 Jenkinsfile

-rw-rw-r-- 1 ubuntu ubuntu 112 Jul 15 16:13 Dockerfile

Step 10: Install the Flask package using pip3.

```
ubuntu@ip-172-31-46-36:~/docker-flask-demo$ pip3 install flask
```

Step 11: Run the Flask web application.

```
ubuntu@ip-172-31-46-36:~/docker-flask-demo$ python3 app.py
```

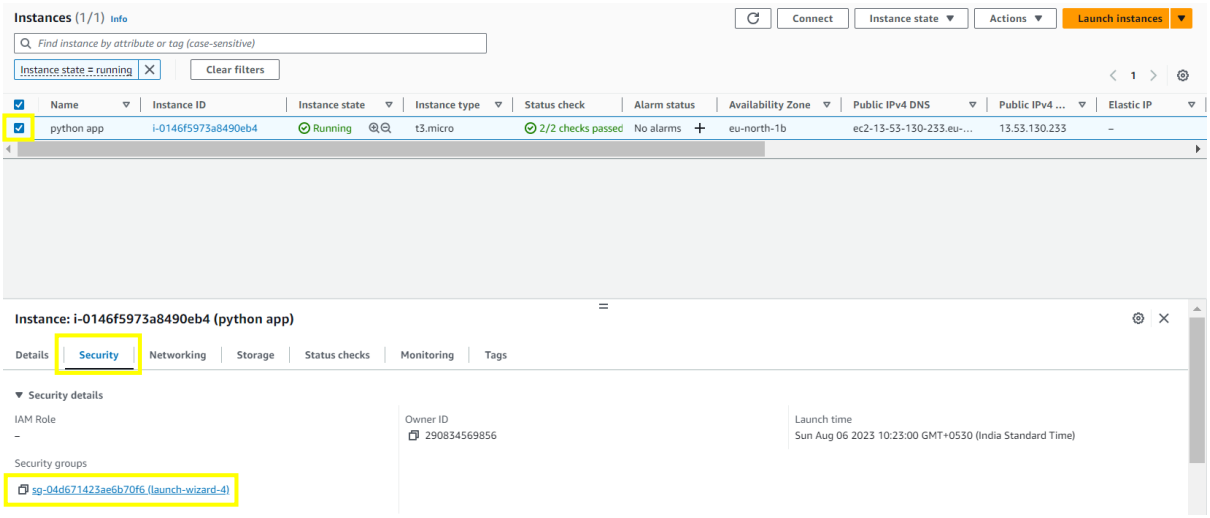
This is a sample web application that displays a colored background.
A color can be specified in two ways.

1. As a command line argument with --color as the argument. Accepts one of red,green,blue,blue2,pink,darkblue
2. As an Environment variable APP_COLOR. Accepts one of red,green,blue,blue2,pink,darkblue
3. If none of the above then a random color is picked from the above list.
Note: Command line argument precedes over environment variable.

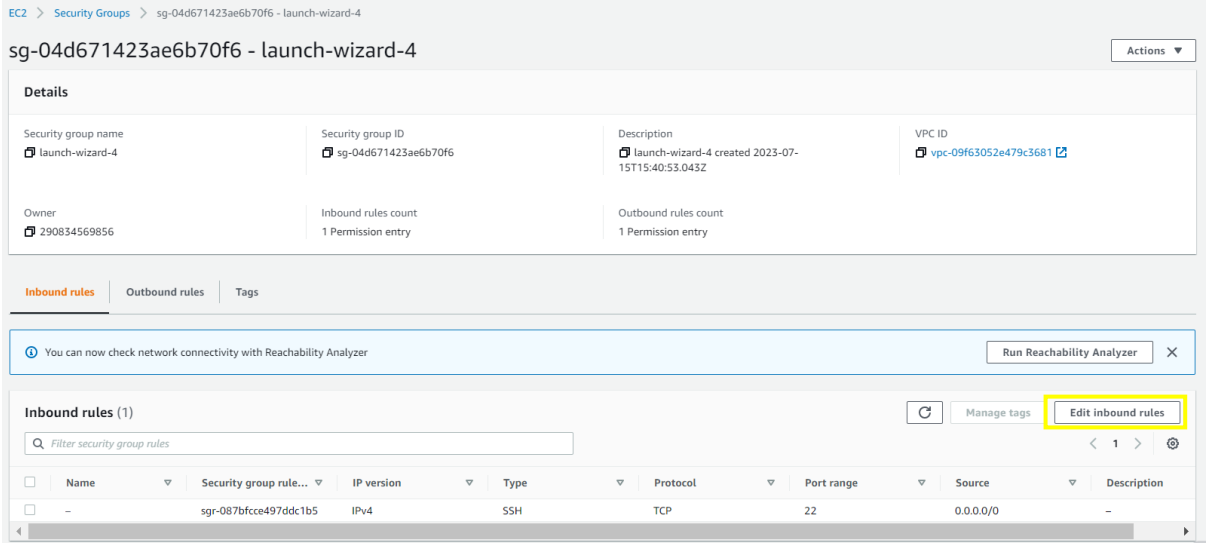
No command line argument or environment variable. Picking a Random Color =blue2
** Serving Flask app 'app'*
** Debug mode: off*
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
** Running on all addresses (0.0.0.0)*
** Running on http://127.0.0.1:8080*
** Running on http://172.31.46.36:8080*
Press CTRL+C to quit

Step 12: Allow Custom TCP port 8080 in Inbound rules.

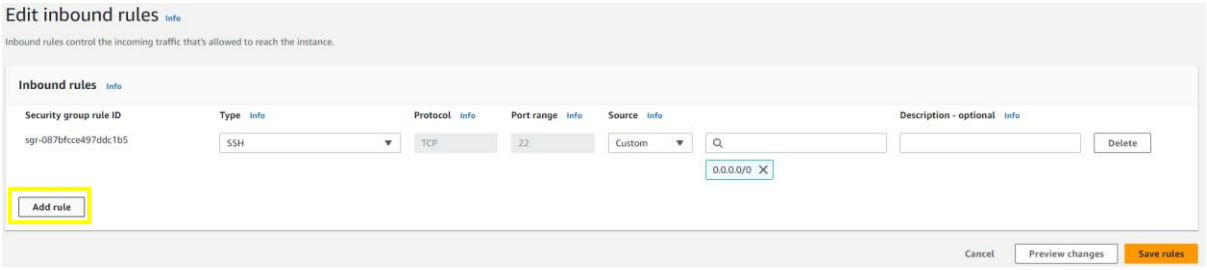
#Go to EC2 Dashboard, Select Instance – Security – Security groups



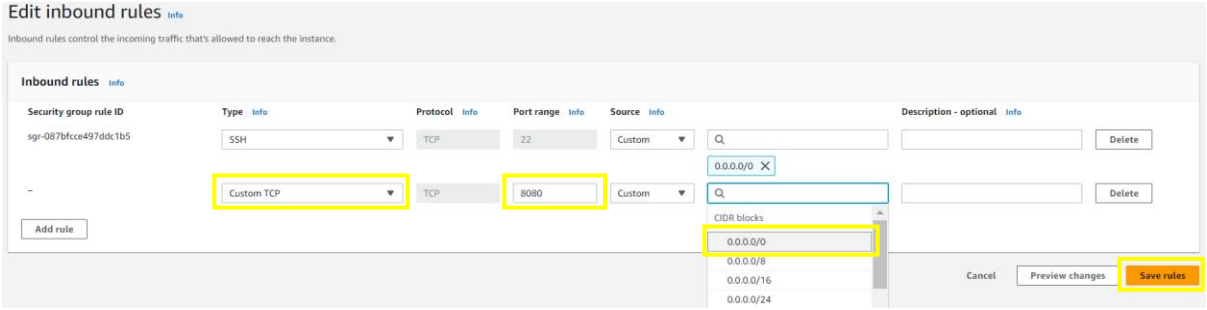
#Select Edit inbound rules



#Select Add rule



#Set port range 8080, choose Anywhere-IPv4 and Click save rules



Step 13: Now open <Public_IP_Address_of_EC2_Instance>:8080 in Browser

Output:

Result:

Thus, successfully deployed the basic Python web application using Flask on the Ubuntu EC2 instance, and the application was accessible through the EC2 instance's public IP address.

Installing MySQL in Ubuntu Instance and Performing CRUD Operations

Aim:

To demonstrate how to install MySQL on an Ubuntu EC2 instance and perform basic CRUD (Create, Read, Update, Delete) operations on a database using the MySQL command-line interface.

Procedure:

Step 1: Create an EC2 instance with Ubuntu in Amazon Machine Image

Step 2: Update package information on the Ubuntu EC2 instance.

```
ubuntu@ip-172-31-45-247:~$ sudo apt update
```

Step 3: Install MySQL server on the instance.

```
ubuntu@ip-172-31-45-247:~$ sudo apt install mysql-server
```

Step 4: Check the status of the MySQL service to ensure it's running.

```
ubuntu@ip-172-31-45-247:~$ sudo systemctl status mysql
```

```
● mysql.service - MySQL Community Server
     Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset:
enabled)
     Active: active (running) since Wed 2023-06-28 09:42:45 UTC; 8min ago
    Process: 2884 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre
(code=exited, status=0/SUCCESS)
   Main PID: 2892 (mysqld)
     Status: "Server is operational"
      Tasks: 37 (limit: 1111)
     Memory: 354.1M
        CPU: 3.015s
     CGroup: /system.slice/mysql.service
             └─2892 /usr/sbin/mysqld
Jun 28 09:42:45 ip-172-31-45-247 systemd[1]: Starting MySQL Community Server...
Jun 28 09:42:45 ip-172-31-45-247 systemd[1]: Started MySQL Community Server.
```

Step 5: Access the MySQL command-line interface (CLI) as the root user.

```
ubuntu@ip-172-31-45-247:~$ sudo mysql
```

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.33-0ubuntu0.22.04.2 (Ubuntu)

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

Now you can run the SQL commands

Step 6: Create a new database named 'student'.

```
mysql> create database student;
```

Query OK, 1 row affected (0.02 sec)

Step 7: Show the list of databases to verify the creation of the 'student' database.

```
mysql> show databases;
```

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| student            |
| sys                |
+--------------------+
5 rows in set (0.20 sec)
```

Step 8: Switch to the 'student' database for further operations.

```
mysql> use student;
```

Database changed

Step 9: Create a table named 'name_details' with two columns 'id' (integer) and 'name' (varchar).

```
mysql> create table name_details(id int,name varchar(45));
```

Query OK, 0 rows affected (0.04 sec)

Step 10: Insert three rows of data into the 'name_details' table.

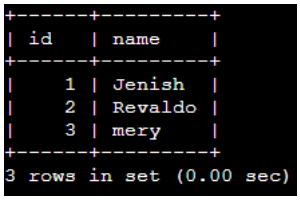
```
mysql> insert into name_details values(1,'Jenish'),(2,'Revaldo'),(3,'mery');
```

Query OK, 3 rows affected (0.01 sec)

Records: 3 Duplicates: 0 Warnings: 0

Step 11: Retrieve and display all records from the 'name_details' table.

```
mysql> select * from name_details;
```



```
+------+---------+
| id   | name    |
+------+---------+
|    1 | Jenish  |
|    2 | Revaldo |
|    3 | mery    |
+------+---------+
3 rows in set (0.00 sec)
```

Step 12: Update a record from the 'name_details' table.

```
mysql> UPDATE name_details SET name = 'Mahe' WHERE id = 3;
```

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

Step 13: Delete a record from the 'name_details' table.

```
mysql> DELETE FROM name_details WHERE id = 2;
```

Query OK, 1 row affected (0.00 sec)

Step 14: Retrieve and display all records from the 'name_details' table.

```
mysql> select * from name_details;
```

```
+------+--------+
| id   | name   |
+------+--------+
|    1 | Jenish |
|    3 | Mahe   |
+------+--------+
2 rows in set (0.00 sec)
```

Step 15: Exit the MySQL CLI.

```
mysql> exit
```

Bye

Result:

Thus, successfully installed MySQL, created a database, and performed CRUD operations to manage data efficiently on the Ubuntu instance.

Deployment of Flask Web Application with Docker

Aim:

To deploy a Flask web application using Docker on an Ubuntu instance with seamless containerization and port mapping.

Procedure:

Step 1: Create an EC2 instance with Ubuntu in Amazon Machine Image.

Step 2: Update package information on the Ubuntu EC2 instance.

```
ubuntu@ip-172-31-42-183:~$ sudo apt update
```

Step 3: Install Docker on the Ubuntu instance by downloading the installation script using 'curl' and then executing the script using the 'sh' shell to perform the installation

```
ubuntu@ip-172-31-42-183:~$ curl -fsSL https://get.docker.com -o get-docker.sh
```

```
ubuntu@ip-172-31-42-183:~$ sh get-docker.sh
```

Step 4: Check the version of docker to verify installation.

```
ubuntu@ip-172-31-42-183:~$ docker --version
```

Docker version 24.0.4, build 3713ee1

Step 5: Clone the GitHub repository containing the Flask web application and docker file.

```
ubuntu@ip-172-31-42-183:~$ git clone https://github.com/yeshwanthlm/docker-flask-demo
```

Step 6: List files and directories in the directory in long format.

```
ubuntu@ip-172-31-42-183:~$ ls -ltr
```

total 28

-rw-rw-r-- 1 ubuntu ubuntu 21926 Jul 17 08:22 get-docker.sh

drwxrwxr-x 4 ubuntu ubuntu 4096 Jul 17 08:26 docker-flask-demo

Step 7: Navigate to the cloned directory.

```
ubuntu@ip-172-31-42-183:~$ cd docker-flask-demo
```

Step 8: List the contents of the directory to see the files.

```
ubuntu@ip-172-31-42-183:~docker-flask-demo$ ls -ltr
```

total 20

drwxrwxr-x 2 ubuntu ubuntu 4096 Jul 15 16:13 templates

-rw-rw-r-- 1 ubuntu ubuntu 5 Jul 15 16:13 requirements.txt

-rw-rw-r-- 1 ubuntu ubuntu 2258 Jul 15 16:13 app.py

-rw-rw-r-- 1 ubuntu ubuntu 687 Jul 15 16:13 Jenkinsfile

-rw-rw-r-- 1 ubuntu ubuntu 112 Jul 15 16:13 Dockerfile

Step 9: Display the contents of the Dockerfile in the terminal.

```
ubuntu@ip-172-31-42-183:~/docker-flask-demo$ cat Dockerfile
```

FROM python:3.6

RUN pip install flask

COPY . /opt/

EXPOSE 8080

WORKDIR /opt

Step 10: Build the Docker image.

```
ubuntu@ip-172-31-42-183:~/docker-flask-demo$ sudo docker build -t amc/flaskapp .
```

Step 11: List all the Docker images available on the system.

```
ubuntu@ip-172-31-42-183:~/docker-flask-demo$ sudo docker images
```

REPOSITORY TAG IMAGE ID CREATED SIZE

amc/flaskapp latest e85ea9eeabdb 5 minutes ago 913MB

Step 12: Run the Flask web application as a Docker container and map port 8080 to port 80 on the host.

```
ubuntu@ip-172-31-42-183:~/docker-flask-demo$ sudo docker run -d -p 80:8080 e85ea9eeabdb
```

Step 13: List all the running containers on the system.

```
ubuntu@ip-172-31-42-183:~/docker-flask-demo$ sudo docker ps
```

CONTAINER ID IMAGE COMMAND CREATED STATUS

77904b80ac1d e85ea9eeabdb "python app.py" 19 seconds ago Up 18 seconds

PORTS NAMES

0.0.0.0:80->8080/tcp, :::80->8080/tcp reverent_williamson

Step 14: Allow Custom TCP port 80 in Inbound rules.

Step 15: Now open <Public_IP_Address_of_EC2_Instance>:80 in Browser

Output:

Result:

The Flask web application is deployed using Docker, and the container is running, accessible on port 80 of the host machine.

Setting up simple pipeline using Git, Jenkins, and Docker in local mode on an Ubuntu instance

Aim:

To set up a simple pipeline using Git, Jenkins, and Docker in local mode on an Ubuntu instance.

Procedure:

Step 1: Create an EC2 instance with Ubuntu in Amazon Machine Image.

Step 2: Update package information on the Ubuntu EC2 instance.

```
ubuntu@ip-172-31-47-227:~$ sudo apt-get update
```

```
ubuntu@ip-172-31-47-227:~$ sudo apt update
```

Step 3: Install Docker on the Ubuntu instance by downloading the installation script using 'curl' and then executing the script using the 'sh' shell to perform the installation

```
ubuntu@ip-172-31-47-227:~$ curl -fsSL https://get.docker.com -o get-docker.sh
```

```
ubuntu@ip-172-31-47-227:~$ sh get-docker.sh
```

Step 4: Check the version of docker to verify installation.

```
ubuntu@ip-172-31-47-227:~$ docker –version
```

Docker version 24.0.4, build 3713ee1

Step 5: Create a docker hub account in DockerHub website

Step 6: Login docker

```
ubuntu@ip-172-31-47-227:~$ sudo docker login -u <username> -p <password>
```

Replace <username> and <password> with DockerHub username and password.

Step 7: Install OpenJDK 17 (Java Runtime Environment).

```
ubuntu@ip-172-31-47-227:~$ sudo apt install openjdk-17-jre
```

Step 8: Check the Java version to ensure it is installed correctly.

```
ubuntu@ip-172-31-47-227:~$ java -version
```

openjdk version "17.0.7" 2023-04-18

OpenJDK Runtime Environment (build 17.0.7+7-Ubuntu-0ubuntu122.04.2)

OpenJDK 64-Bit Server VM (build 17.0.7+7-Ubuntu-0ubuntu122.04.2, mixed mode, sharing)

Step 9: Import Jenkins Repository Key.

```
ubuntu@ip-172-31-47-227:~$ curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

Step 10: Add Jenkins Repository to Sources List.

```
ubuntu@ip-172-31-47-227:~$echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
```

Step 11: Update the package lists from all repositories, including the newly added Jenkins repository.

```
ubuntu@ip-172-31-47-227:~$ sudo apt-get update
```

Step 12: Install Jenkins.

```
ubuntu@ip-172-31-47-227:~$ sudo apt-get install jenkins
```

Step 13: Start Jenkins service and check its status.

```
ubuntu@ip-172-31-47-227:~$ sudo systemctl start jenkins.service
```

```
ubuntu@ip-172-31-47-227:~$ sudo systemctl status jenkins
```

● *jenkins.service - Jenkins Continuous Integration Server*

Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)

Active: active (running) since Fri 2023-07-21 12:50:18 UTC; 1min 57s ago

Main PID: 6680 (java)

Tasks: 38 (limit: 1111)

Memory: 292.6M

CPU: 1min 17.623s

CGroup: /system.slice/jenkins.service

└─6680 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Jul 21 12:49:30 ip-172-31-47-227 jenkins[6680]: c84c43b9adf14d60873f4185427328ba

Step 14: Configure firewall to allow traffic on port 8080 for Jenkins, SSH connections for remote access, port 8000 for container communication.

```
ubuntu@ip-172-31-47-227:~$ sudo ufw allow 8080
```

```
ubuntu@ip-172-31-47-227:~$ sudo ufw allow OpenSSH
```

```
ubuntu@ip-172-31-47-227:~$ sudo ufw allow 8000
```

Step 15: Enable the firewall.

```
ubuntu@ip-172-31-47-227:~$ sudo ufw enable
```

Step 16: Check the firewall status.

```
ubuntu@ip-172-31-47-227:~$ sudo ufw status
```

Status: active

To	*Action*	*From*
8080	*ALLOW*	*Anywhere*
OpenSSH	*ALLOW*	*Anywhere*
8000	*ALLOW*	*Anywhere*
8080 (v6)	*ALLOW*	*Anywhere (v6)*
OpenSSH (v6)	*ALLOW*	*Anywhere (v6)*
8000 (v6)	*ALLOW*	*Anywhere (v6)*

Step 17: Retrieve the initialAdminPassword for Jenkins setup.

```
ubuntu@ip-172-31-47-227:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

c84c43b9adf14d60873f4185427328ba

Step 18: Add the Jenkins user to the docker group for Docker access.

```
ubuntu@ip-172-31-47-227:~$ sudo usermod -aG docker jenkins
```

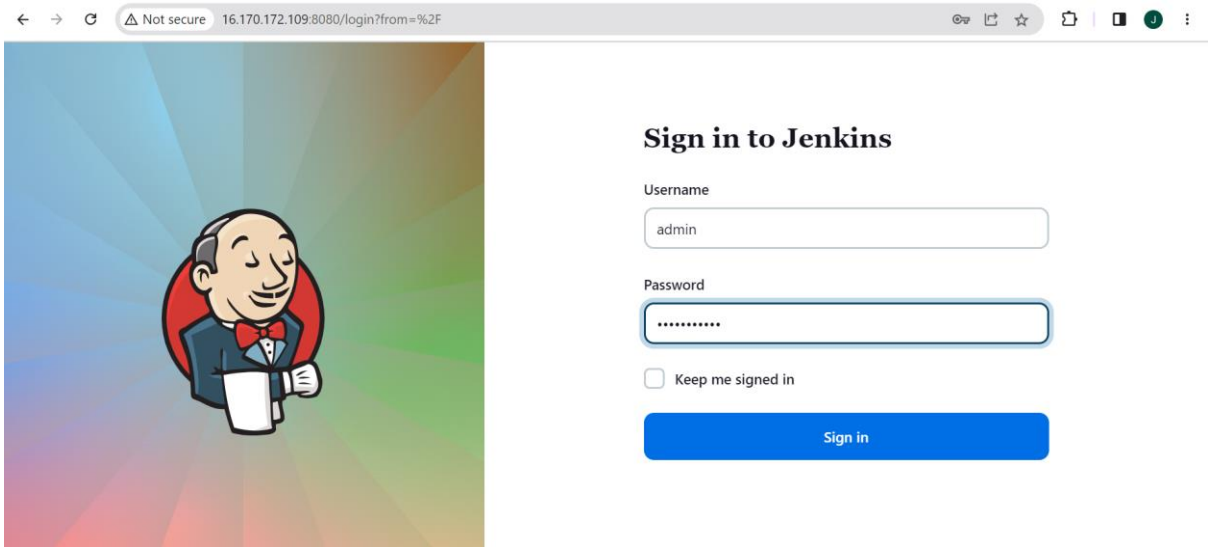
Step 19: Restart Jenkins service to apply the changes.

```
ubuntu@ip-172-31-47-227:~$ sudo systemctl restart jenkins
```

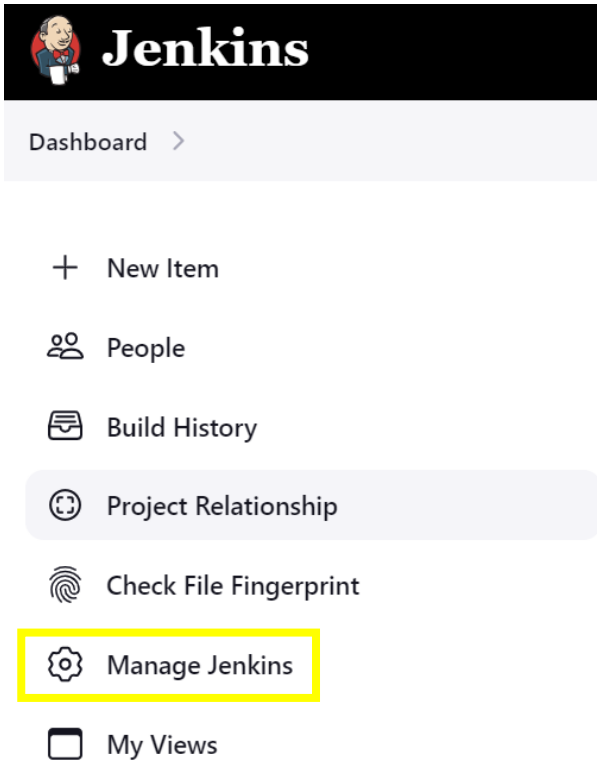
Step 20: Allow Custom TCP port 8080, 8000 in Inbound rules.

Step 21: Now open <Public_IP_Address_of_EC2_Instance>:8080 in Browser to open Jenkins.

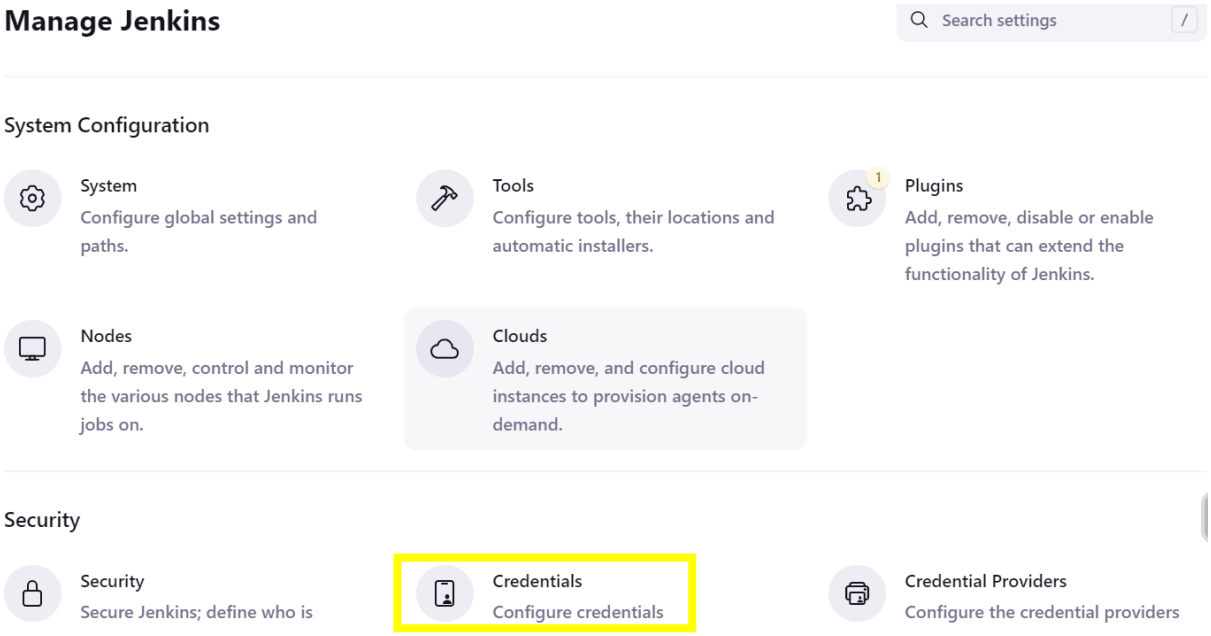
Step 22: Login with initialAdminPassword that we retrieved – Install suggested plugins – Setup new username and password.



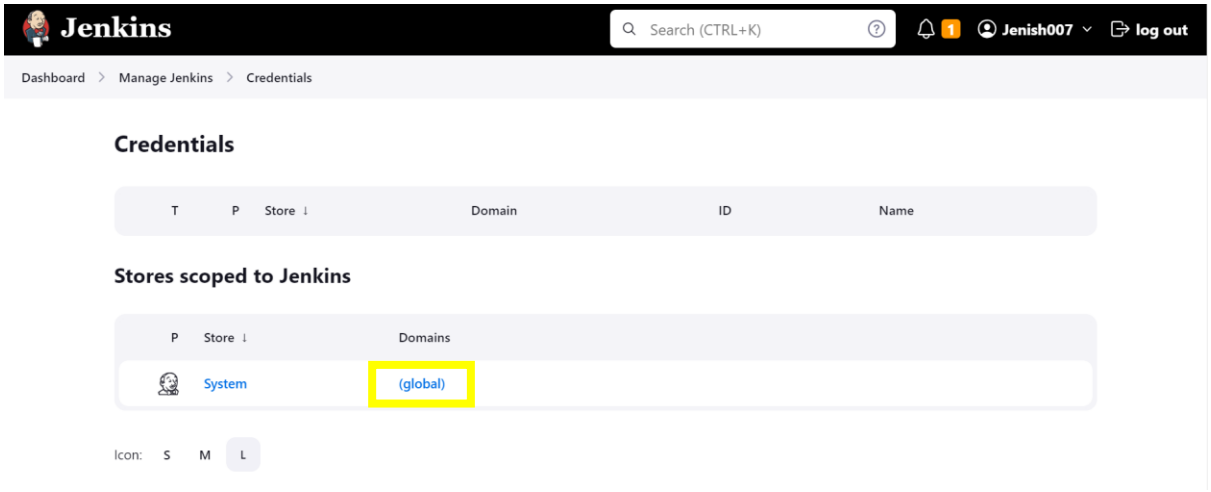
Step 23: Setup Credentials in Jenkins.



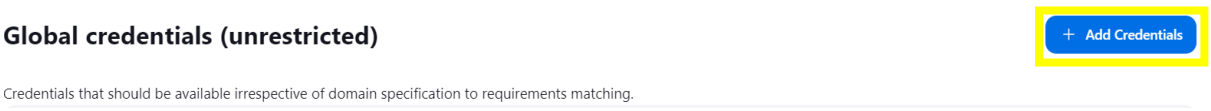
<next>



<next>



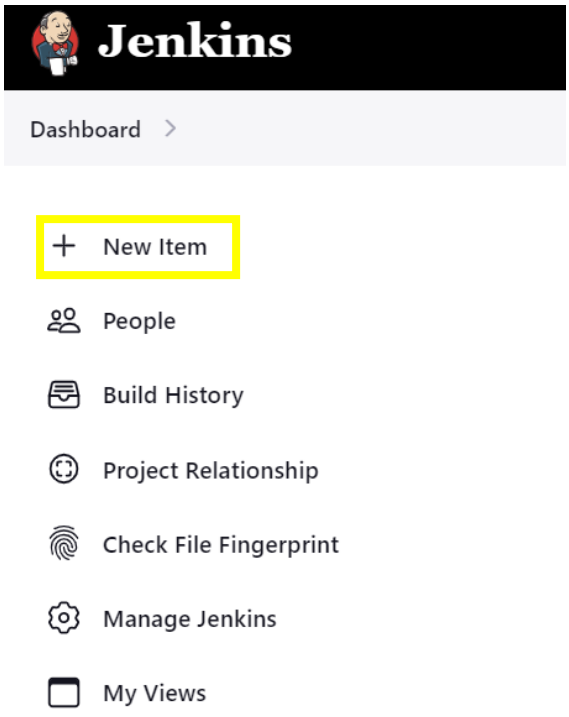
<next>



<next>

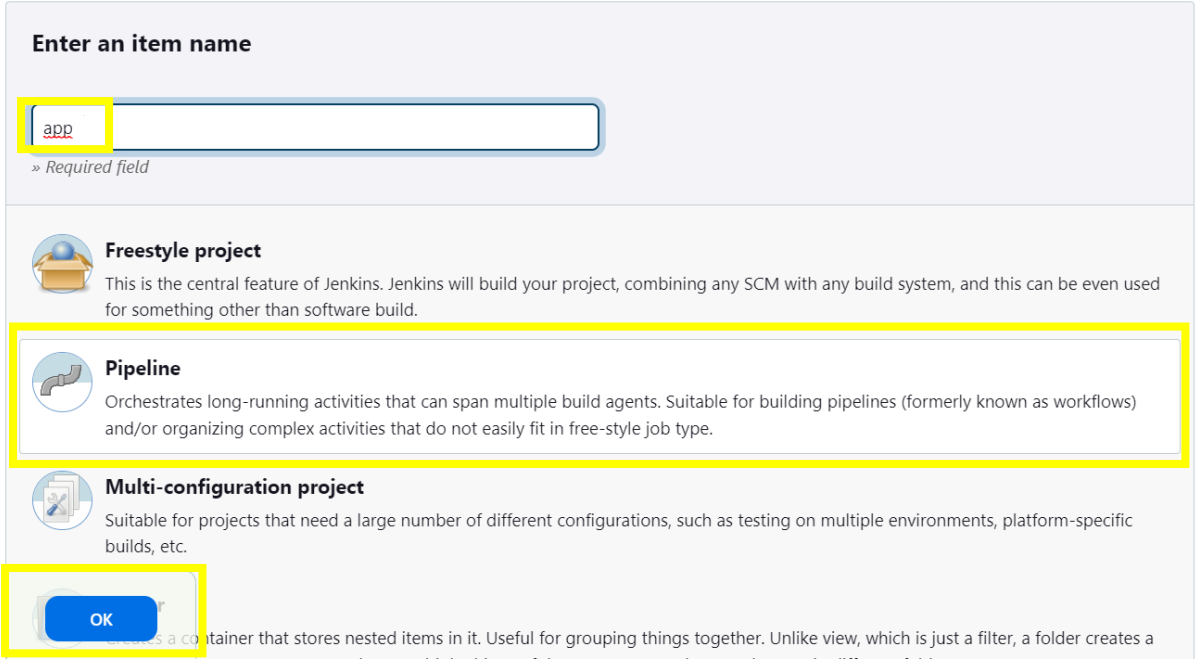
Now add your Docker hub User name and password and save it with Id (I have used test1). Also make necessary changes in the Jenkinsfile like setting **credentials id** and **username of docker**

Step 24: Create Pipeline in Jenkins.



<next>

Enter an item name of your wish.



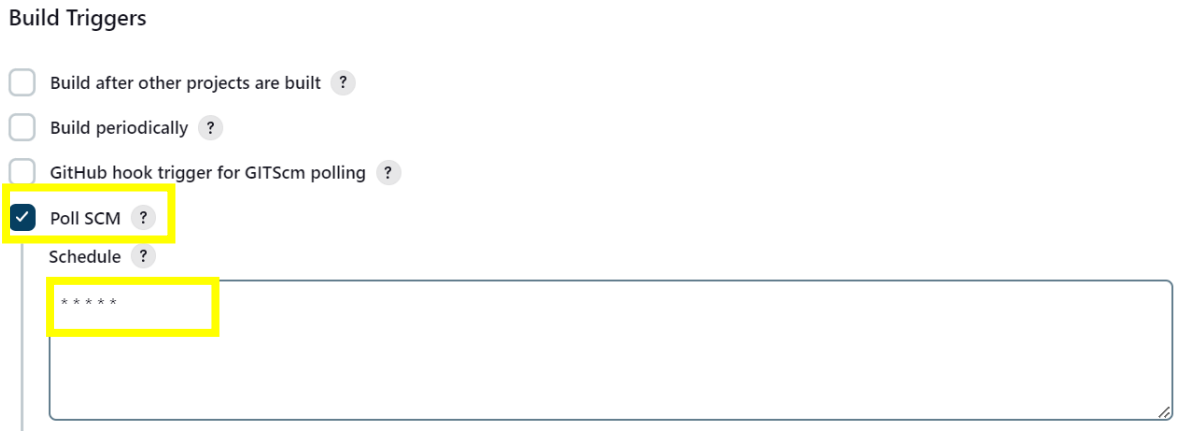
<next>

Add some description:

<next>

Build Triggers with Poll SCM.

This poll SCM with Schedule '* * * * *' will check the git repository every minute and if a commit is encounter it automatically build up…

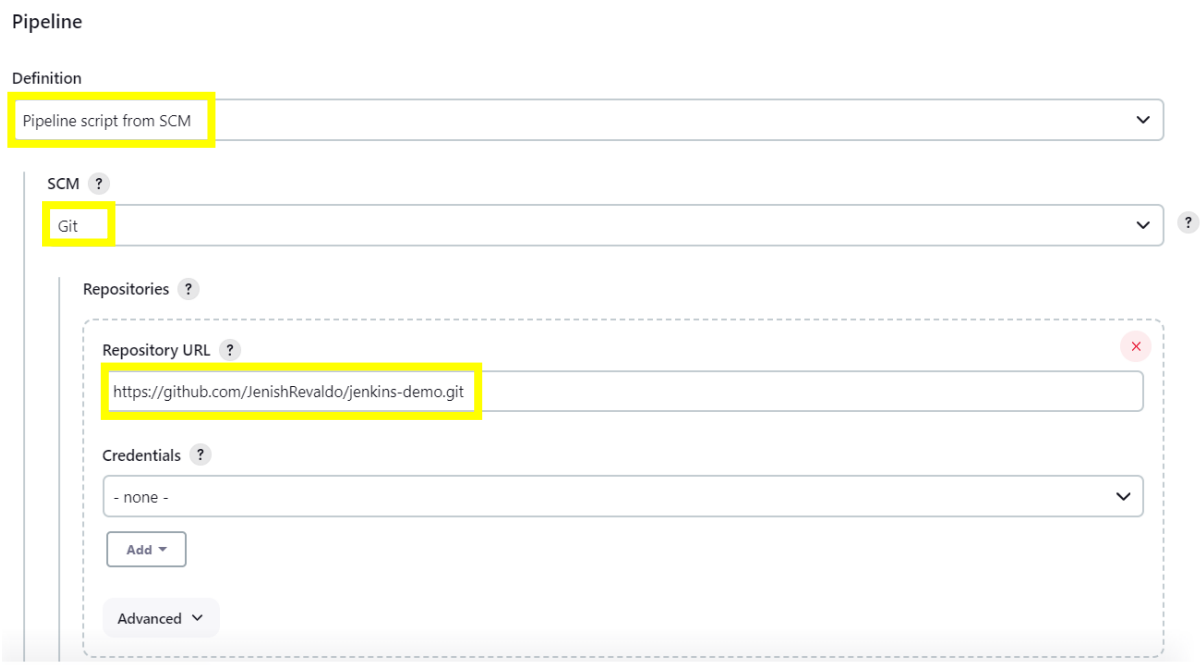


<next>

Display name – The same name given in item name

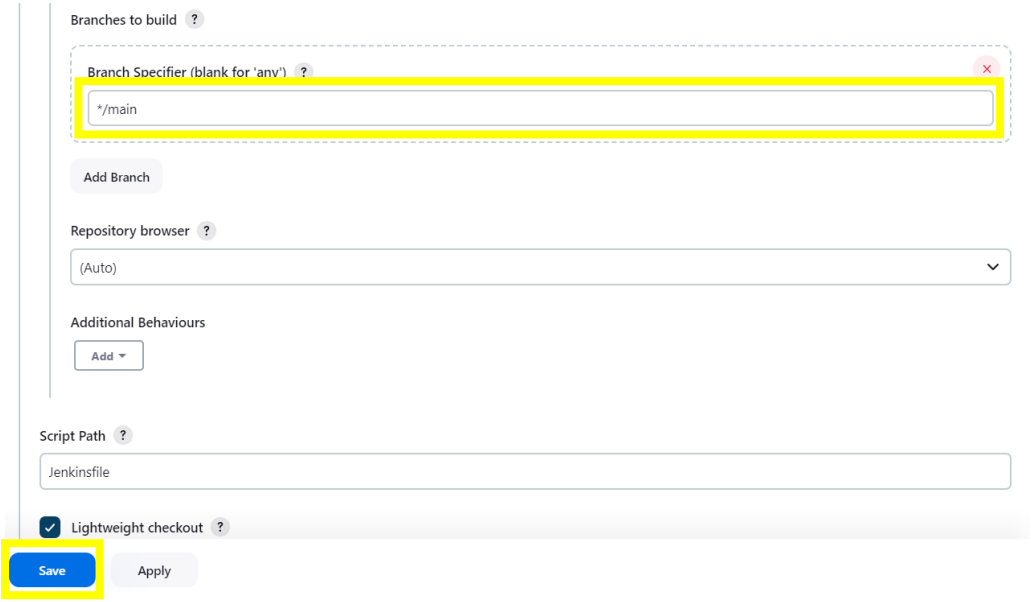
<next>

Setting up pipeline definition from my GitHub Repository.



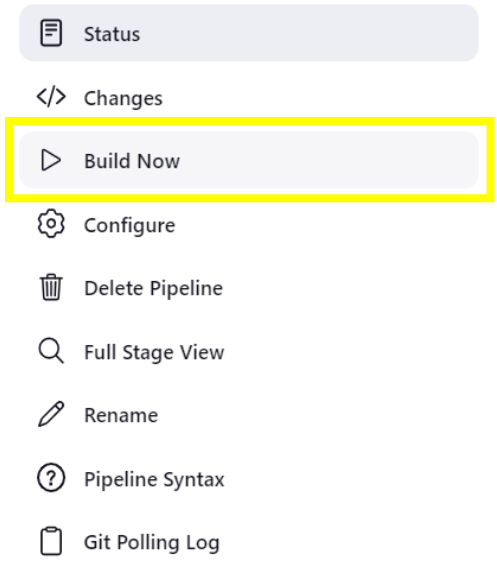
<next>

Change the branch specifier according to your repository and save the pipeline.



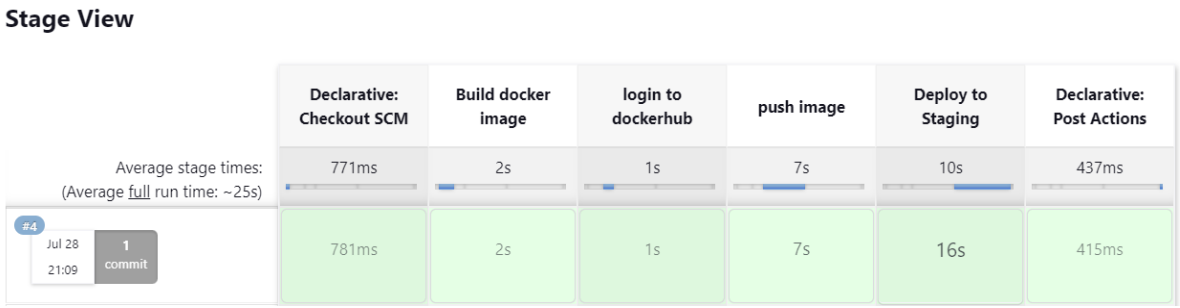
<next>

Now build the pipeline.



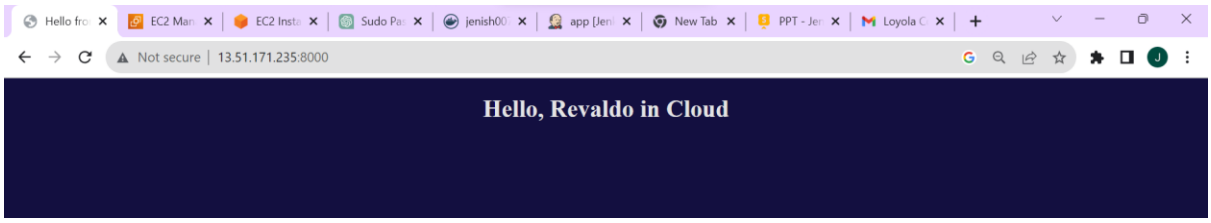
<next>

Stages of pipeline will get executed.



Step 25: Now open <Public_IP_Address_of_EC2_Instance>:8000 in Browser

Output:



Now you can make changes in the **hello.html** file located in the templates folder in the repository and commit it, within one minute it gets reflect in the end page.

Result:

The local pipeline setup for Git, Jenkins, and Docker has been successfully established on the Ubuntu instance, enabling efficient software development, testing, and deployment processes.

Setting Up Jenkins on AWS EC2 Instance and Creating an Image

Aim:

To install Jenkins on an AWS EC2 instance, create an Amazon Machine Image (AMI) of the instance, launch a new EC2 instance from the AMI, and verify that Jenkins is installed on the new instance.

Procedure:

Step 1: Create an EC2 instance with Ubuntu in Amazon Machine Image.

Step 2: Update package information on the Ubuntu EC2 instance.

```
ubuntu@ip-172-31-30-107:~$ sudo apt-get update
```

```
ubuntu@ip-172-31-30-107:~$ sudo apt update
```

Step 3: Install OpenJDK 17 (Java Runtime Environment).

```
ubuntu@ip-172-31-30-107:~$ sudo apt install openjdk-17-jre
```

Step 4: Check the Java version to ensure it is installed correctly.

```
ubuntu@ip-172-31-30-107:~$ java -version
```

openjdk version "17.0.7" 2023-04-18

Step 5: Import Jenkins Repository Key.

```
ubuntu@ip-172-31-30-107:~$ curl -fsSL https://pkg.jenkins.io/debian/jenkins.io-2023.key |
sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

Step 6: Add Jenkins Repository to Sources List.

```
ubuntu@ip-172-31-30-107:~$echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null
```

Step 7: Update the package lists from all repositories, including the newly added Jenkins repository.

```
ubuntu@ip-172-31-30-107:~$ sudo apt-get update
```

Step 8: Install Jenkins.

```
ubuntu@ip-172-31-30-107:~$ sudo apt-get install jenkins
```

Step 9: Start Jenkins service and check its status.

```
ubuntu@ip-172-31-30-107:~$ sudo systemctl start jenkins.service
```

```
ubuntu@ip-172-31-30-107:~$ sudo systemctl status jenkins
```

● *jenkins.service - Jenkins Continuous Integration Server*

Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)

Active: active (running) since Sun 2023-09-24 12:50:18 UTC; 1min 57s ago

Main PID: 6680 (java)

Tasks: 38 (limit: 1111)

Memory: 292.6M

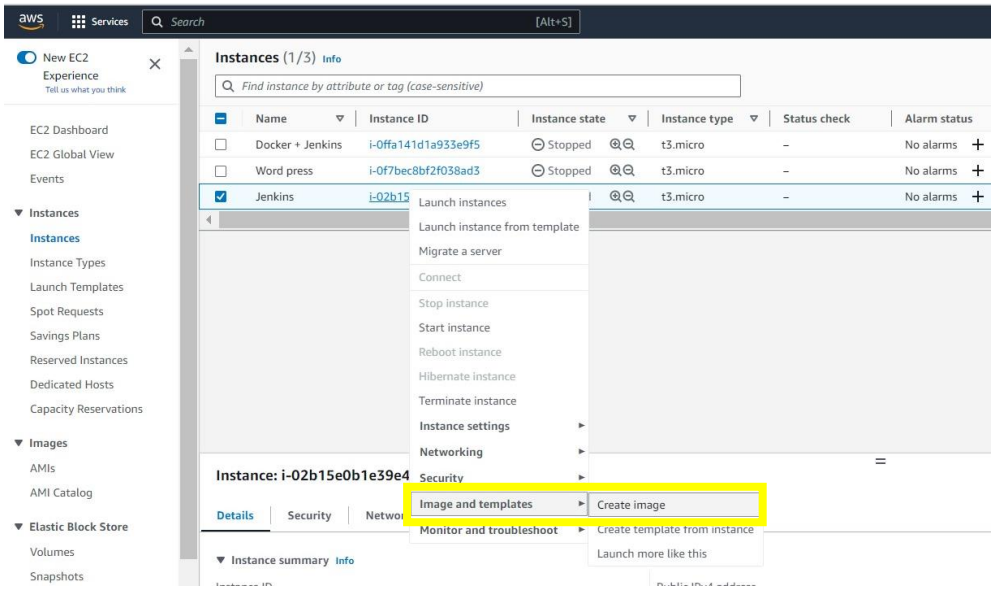
CPU: 1min 17.623s

CGroup: /system.slice/jenkins.service

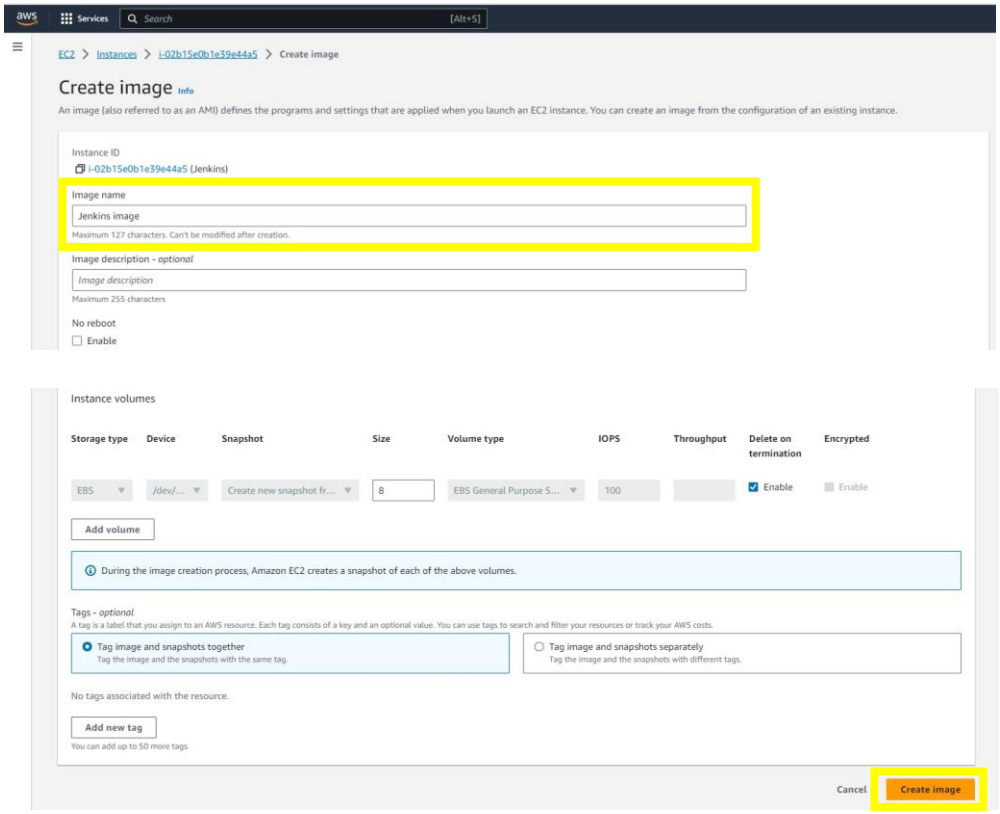
└─6680 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Step 10: Once Jenkins is installed and configured, navigate to the EC2 Management Console and stop the EC2 instance.

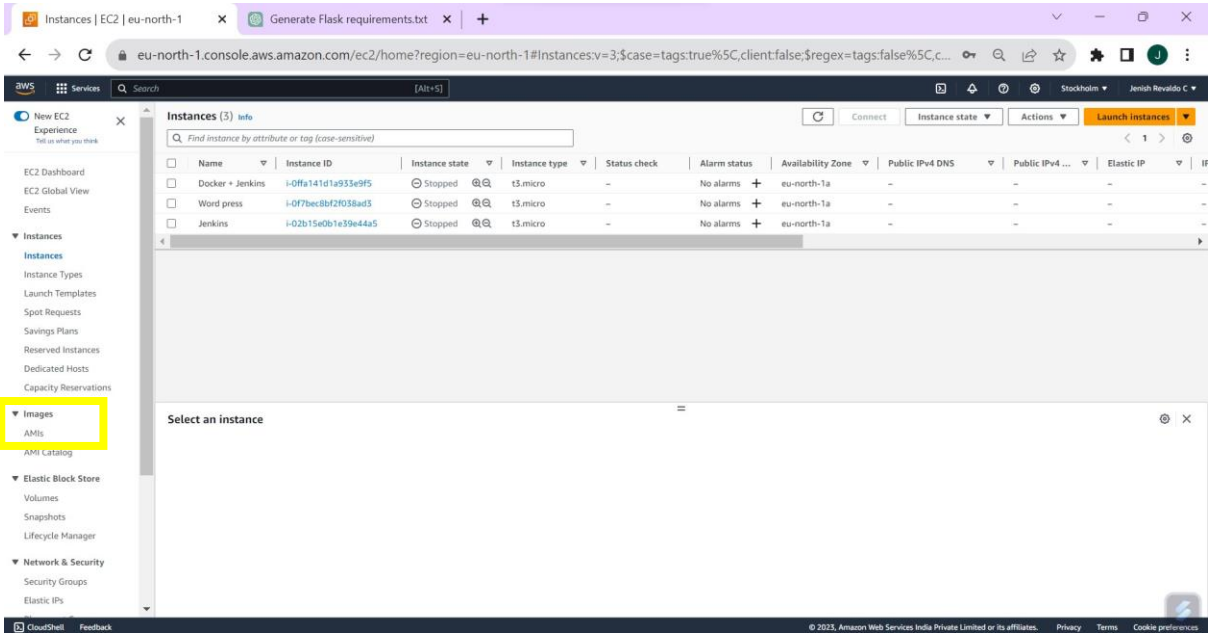
Step 11: Select your instance, right-click, and choose "Image" > "Create Image."



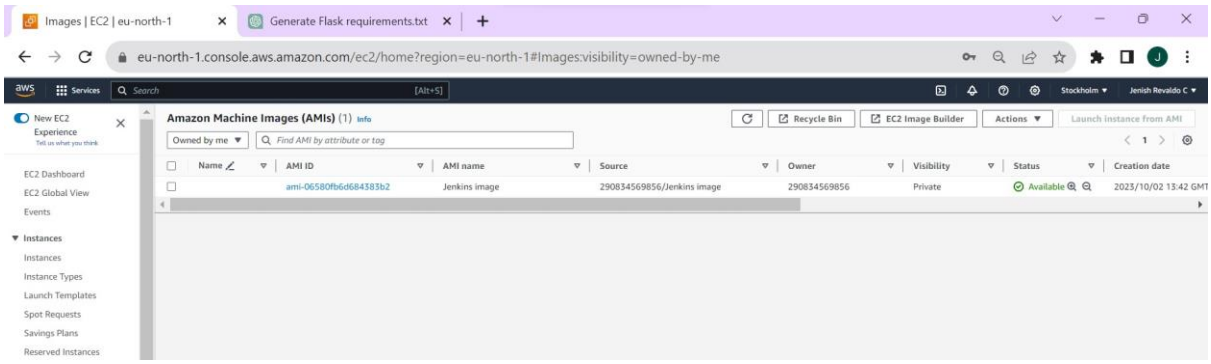
Step 12: Provide a name and description for the image and create it.



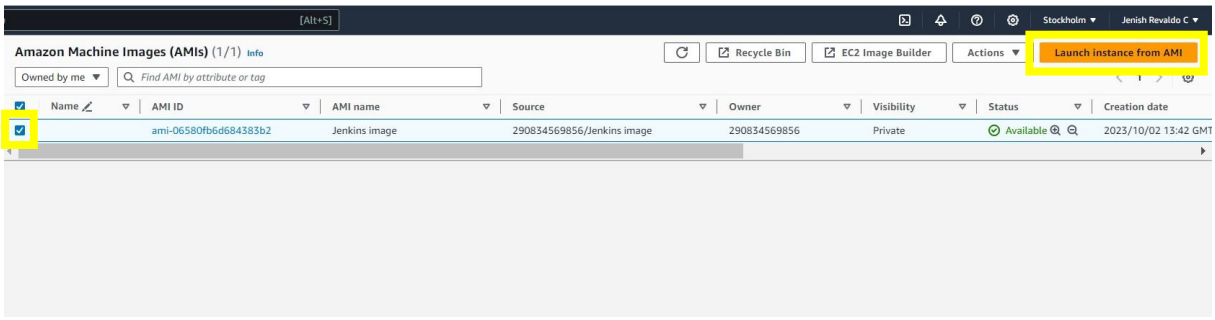
Step 13: Navigate to Images > AMIs in the EC2 dashboard to see the images



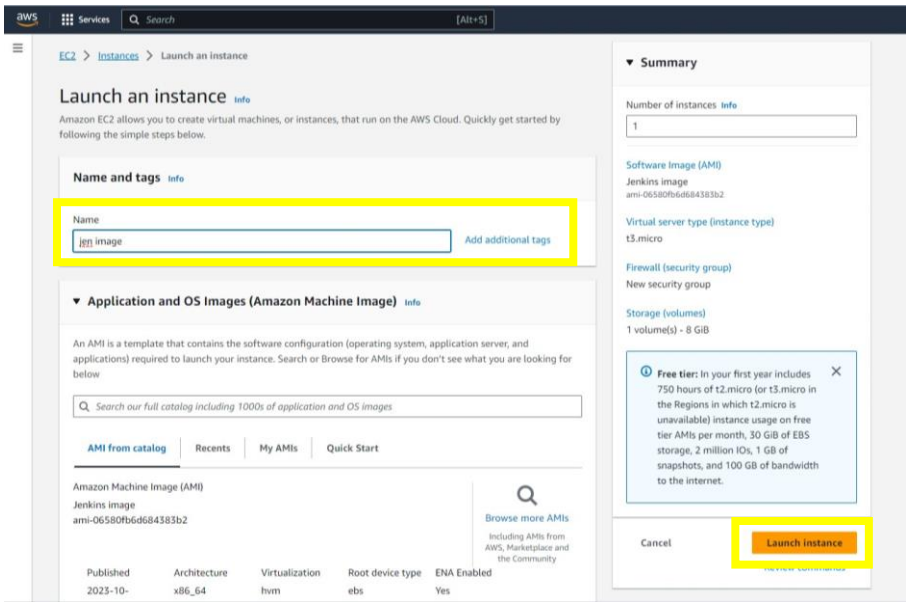
<after navigating>



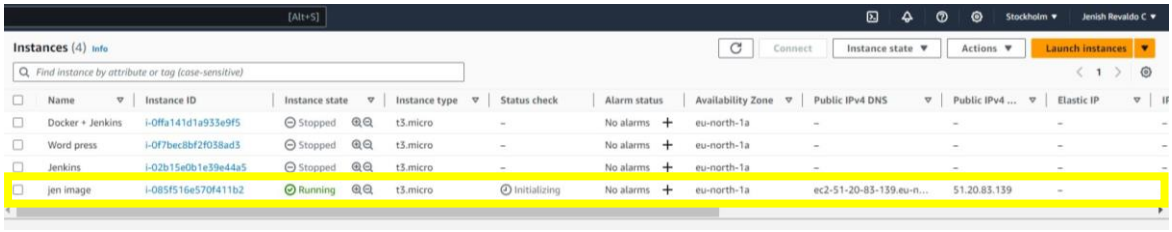
Step 14: To launch a new instance with the created image, select the image and click Launch instance from AMI



Step 15: Follow the instance launch wizard, giving name, choosing key pair, configuring security groups and other settings as needed and launch.



Step 16: Now navigate to the EC2 instances dashboard and you can see the newly created instance from image.



Step 17: Connect to the new EC2 instance using SSH. Check if Jenkins is installed and running.

```
root@ip-172-31-24-134:~# jenkins --version
```

2.424

```
root@ip-172-31-24-134:~# sudo 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 2023-10-02 08:22:25 UTC; 2min 50s ago

Main PID: 455 (java)

Tasks: 38 (limit: 1111)

Memory: 308.8M

CPU: 1min 6.098s

CGroup: /system.slice/jenkins.service

└─455 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Result:

Thus, set up Jenkins on an EC2 instance, created a reusable image, and launch new instances with Jenkins pre-installed, making it easier to scale and manage Jenkins-based infrastructure in the cloud.

Deploying MySQL on AWS, Creating Snapshots

Aim:

To install and configure MySQL on an AWS EC2 instance, create a snapshot of the database, perform various database operations, and then restore the database from the snapshot.

Procedure:

Step 1: Create an EC2 instance with Ubuntu in Amazon Machine Image

Step 2: Update package information on the Ubuntu EC2 instance.

```
ubuntu@ip-172-31-16-6:~$ sudo apt update
```

Step 3: Install MySQL server on the instance.

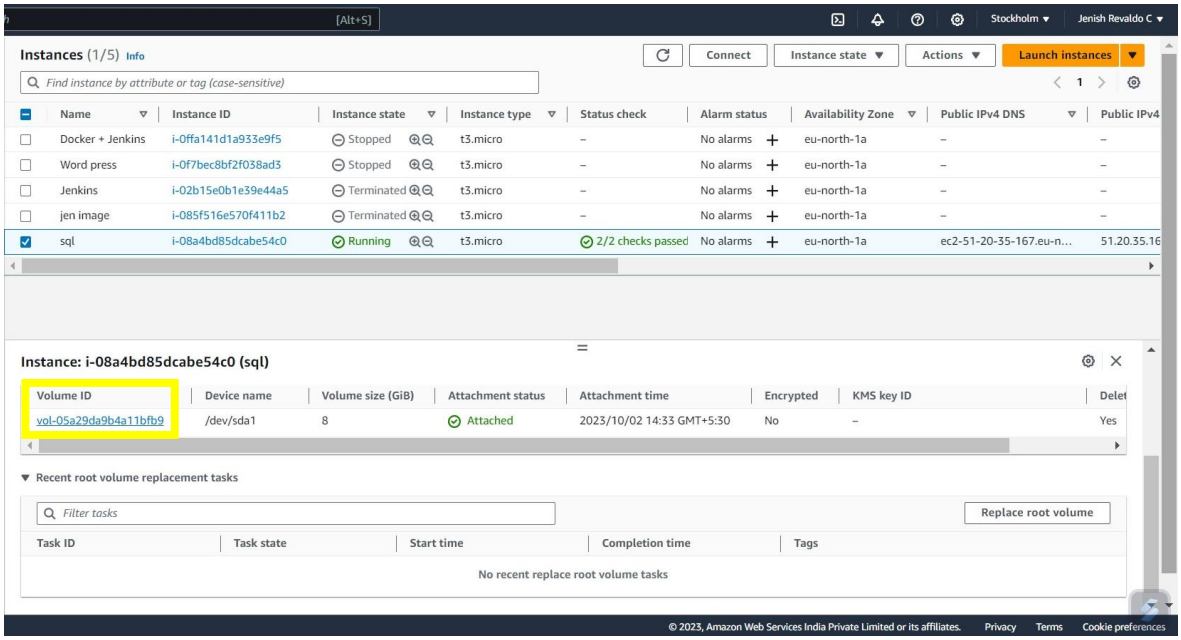
```
ubuntu@ip-172-31-16-6:~$ sudo apt install mysql-server
```

Step 4: Check the status of the MySQL service to ensure it's running.

```
ubuntu@ip-172-31-16-6:~$ sudo systemctl status mysql
```

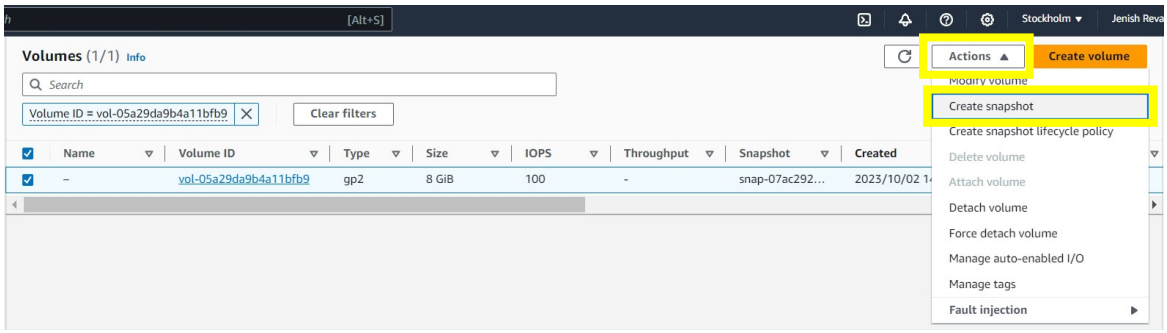
```
● mysql.service - MySQL Community Server
     Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset:
enabled)
     Active: active (running) since Mon 2023-10-02 09:05:35 UTC; 11s ago
    Process: 2901 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre
(code=exited, status=0/SUCCESS)
   Main PID: 2909 (mysqld)
     Status: "Server is operational"
      Tasks: 38 (limit: 1111)
     Memory: 356.9M
        CPU: 1.052s
     CGroup: /system.slice/mysql.service
             └─2909 /usr/sbin/mysqld
```

Step 5: Navigate to the EC2 Management Console and select your instance volume id.



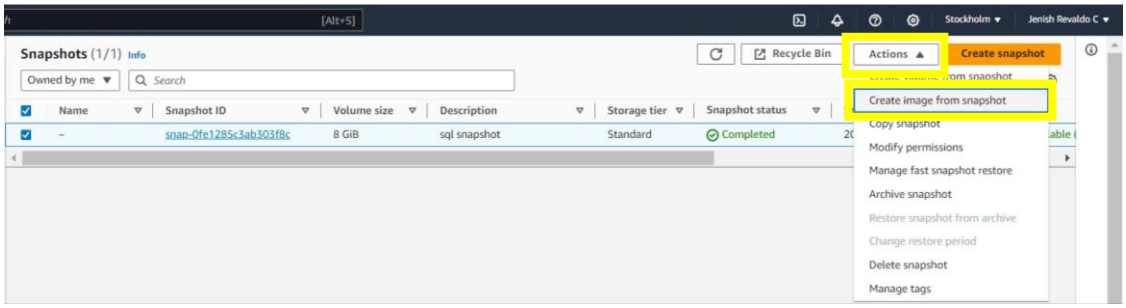
<navigates to volume dashboard>

Step 6: Select the volume, Actions > Create snapshot

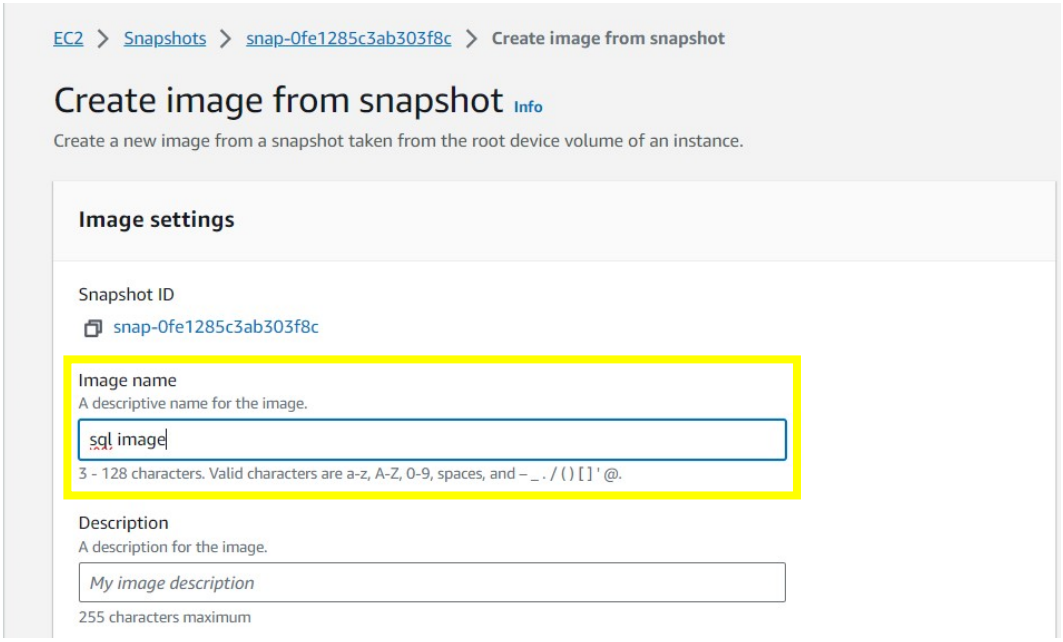


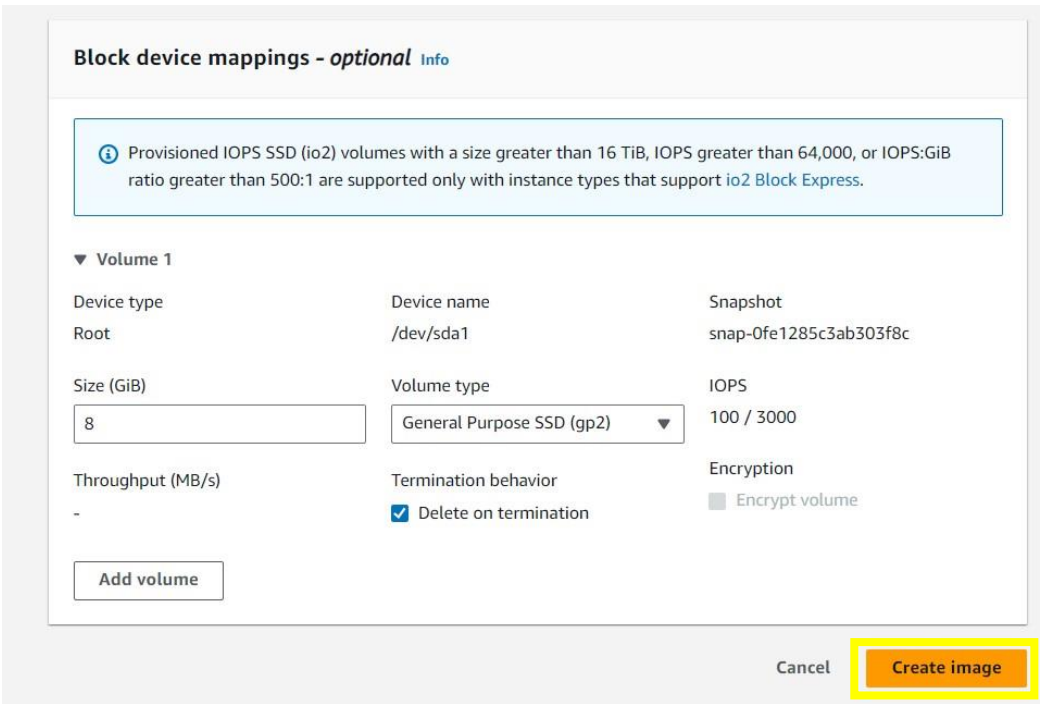
Step 7: Navigate to Elastic Block Store > Snapshots in the EC2 dashboard to see the snapshots.

Step 8: Select the snapshot, Actions > Create image from snapshot.

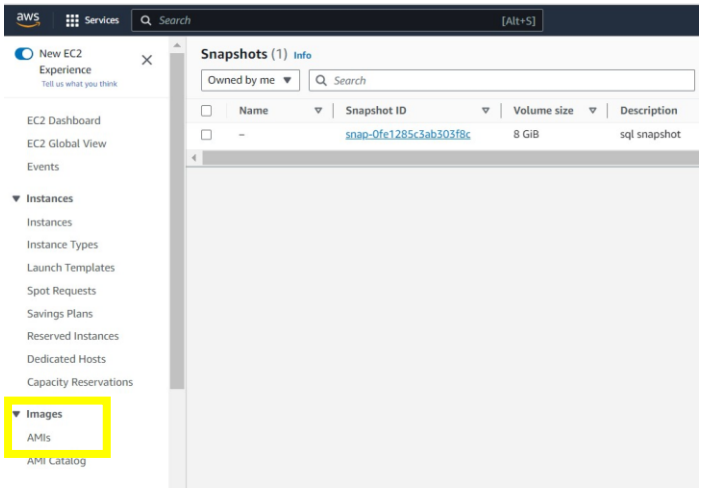


Step 9: Give image name and create image.



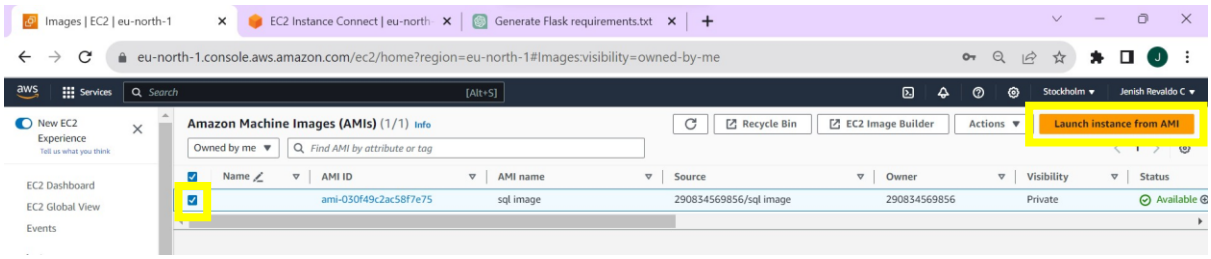


Step 10: Navigate to Images > AMIs in the EC2 dashboard to see the images

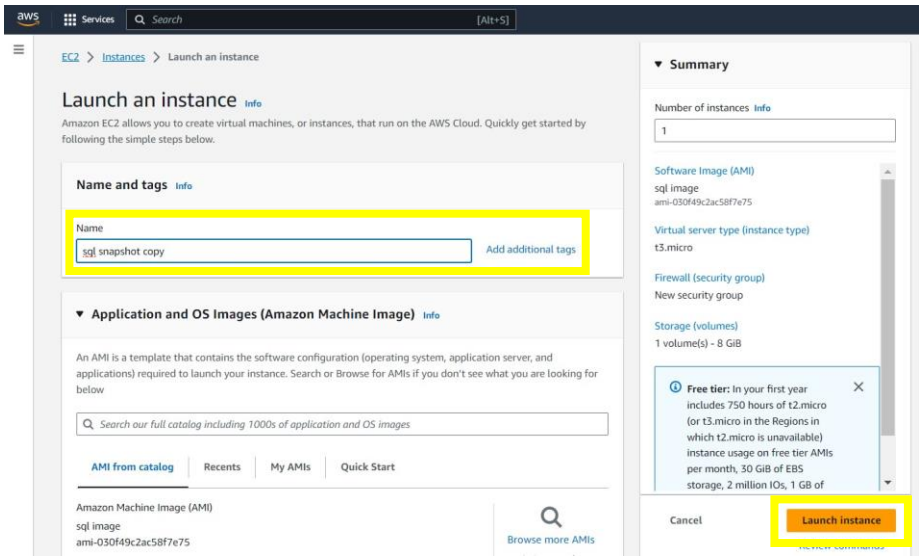


<After Navigating>

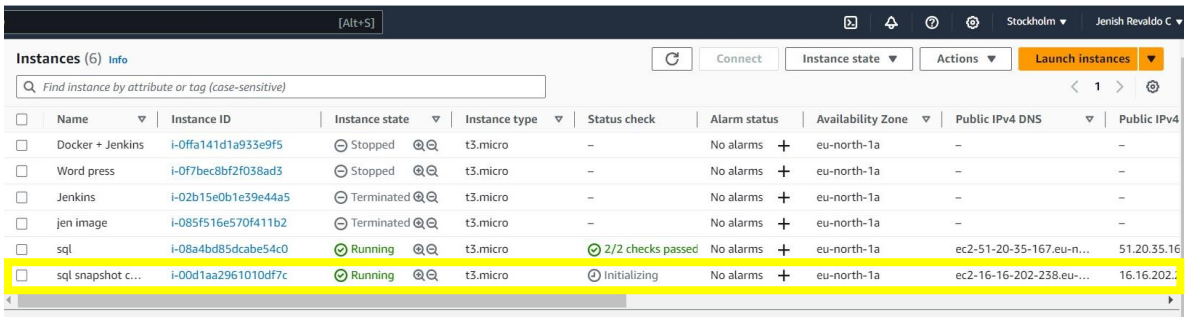
Step 11: To launch a new instance with the created image, select the image and click Launch instance from AMI



Step 12: Follow the instance launch wizard, giving name, choosing key pair, configuring security groups and other settings as needed and launch.



Step 13: Now navigate to the EC2 instances dashboard and you can see the newly created instance from image.



Step 14: Connect to the new EC2 instance using SSH. Check if MySQL is installed and running.

```
ubuntu@ip-172-31-16-6:~$ sudo systemctl status mysql
```

● *mysql.service - MySQL Community Server*
Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
Active: active (running) since Mon 2023-10-02 09:26:21 UTC; 53s ago
Process: 2901 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exited, status=0/SUCCESS)
Main PID: 2909 (mysqld)
Status: "Server is operational"
Tasks: 38 (limit: 1111)
Memory: 356.9M
CPU: 1.052s
CGroup: /system.slice/mysql.service
└─2909 /usr/sbin/mysqld

Result:

Thus, set up MySQL on an EC2 instance, created a snapshot for backup and restored the database to another instance, providing data resilience and disaster recovery capabilities in the cloud.

Deploying WordPress Using Bitnami's Amazon Marketplace AMI and Creating a User Profile

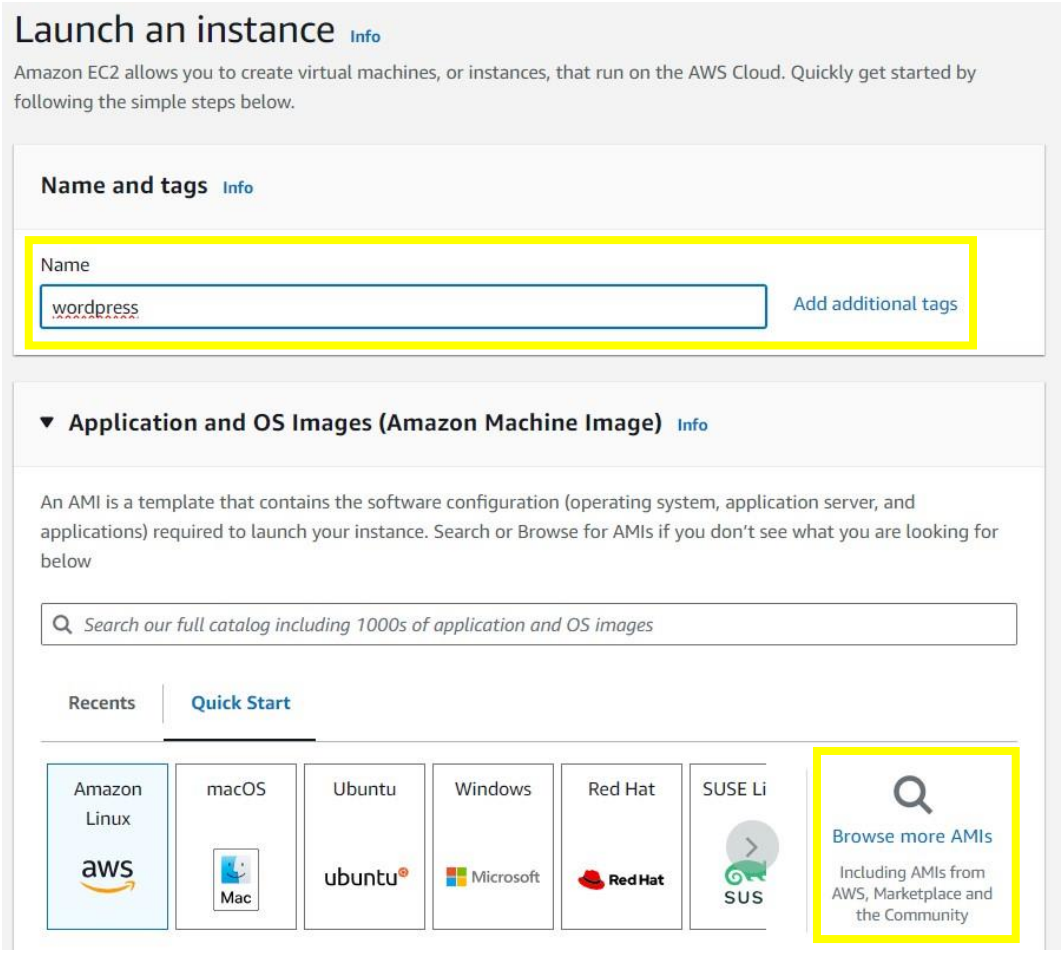
Aim:

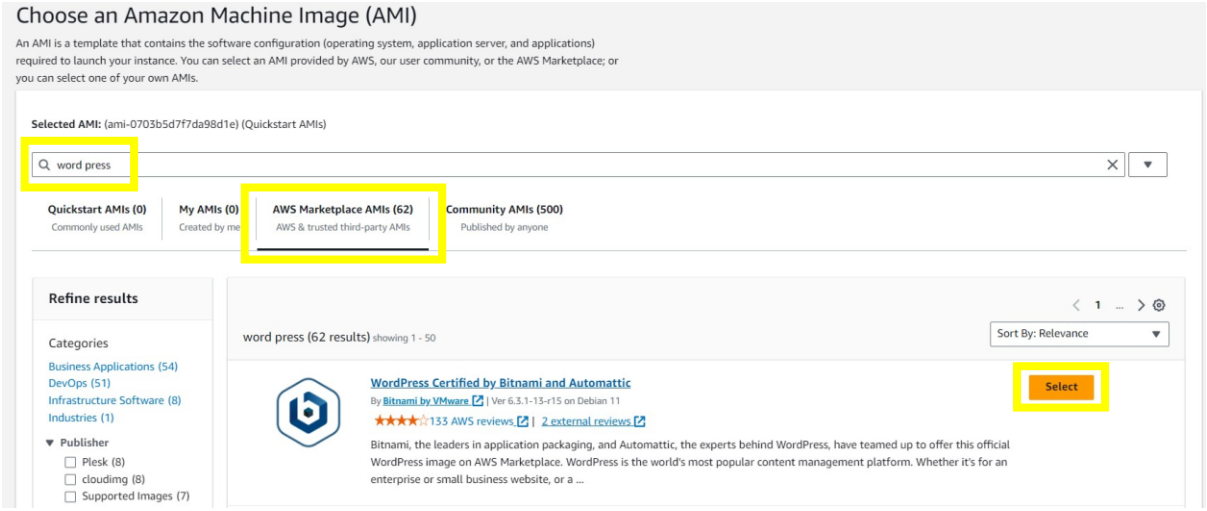
To deploy a WordPress instance on Amazon Web Services (AWS) using Bitnami's WordPress Amazon Machine Image (AMI) available on the AWS Marketplace and create a user profile on the WordPress website.

Procedure:

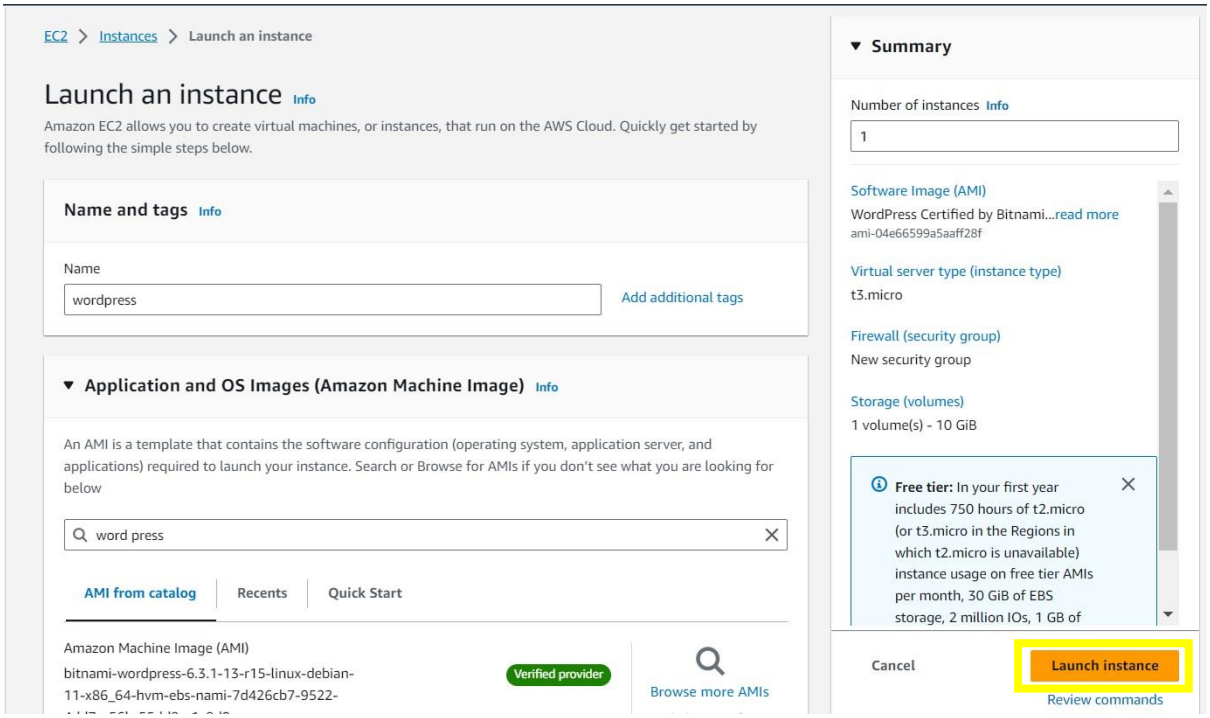
Step 1: In the AWS Management Console, navigate to the EC2 service, Click on "Launch Instance."

- In the "AWS Marketplace" section, search for "Bitnami WordPress."
- Select the Bitnami WordPress AMI from the list.

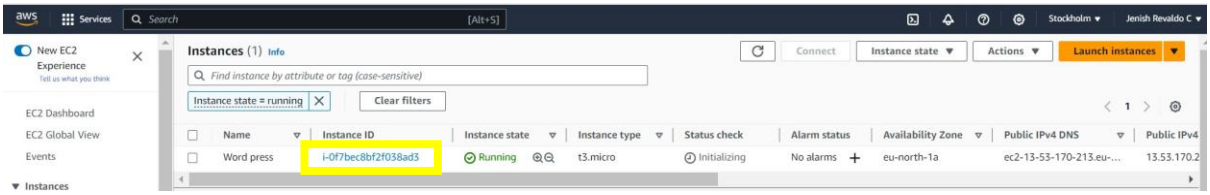




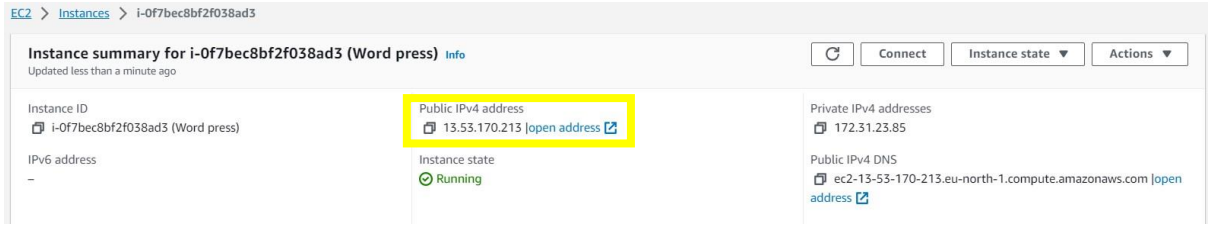
<now launch the instance>



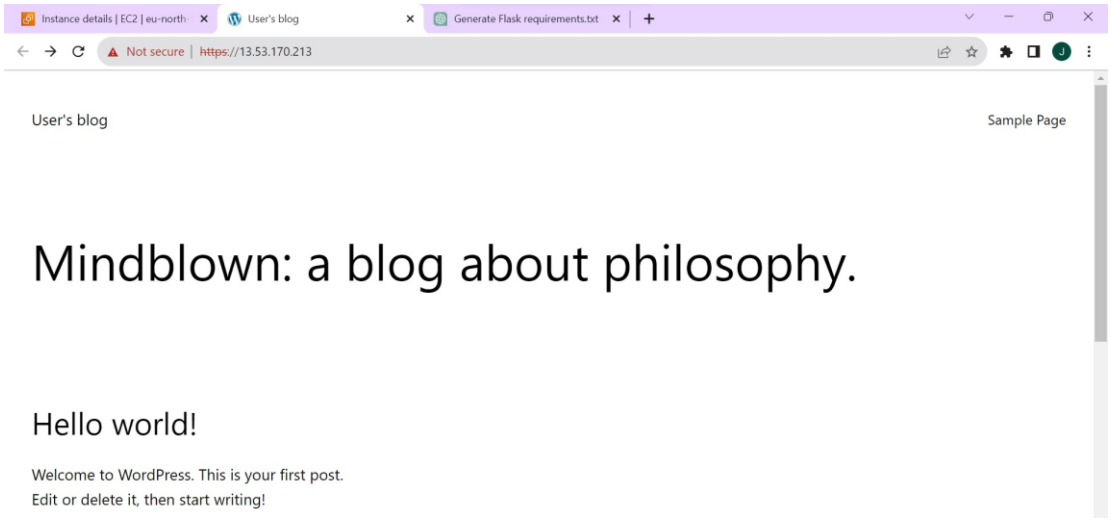
Step 2: Open the public Ip address of the created instance



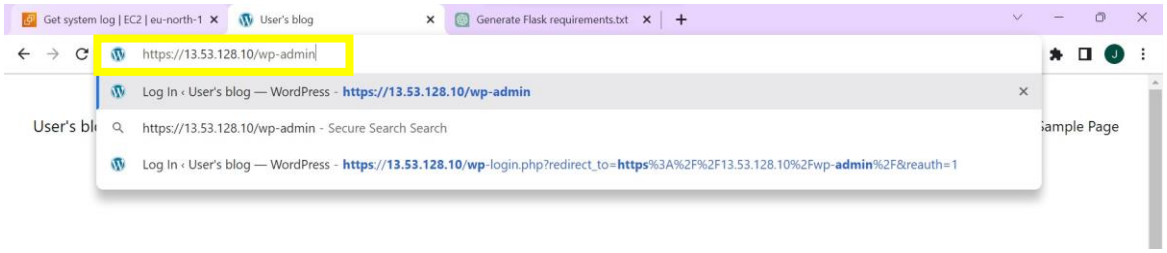
<next>



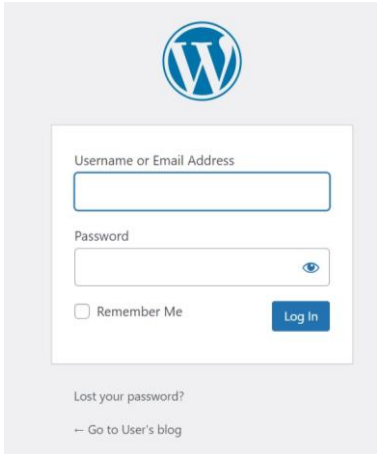
<after opening the public Ip address>



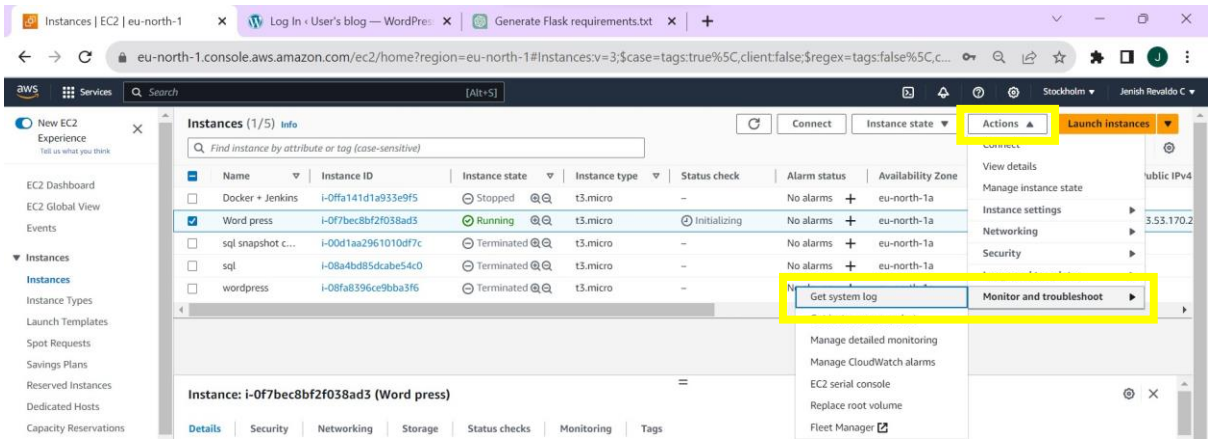
Step 3: Move to the admin page by adding /wp-admin after the Ip address.



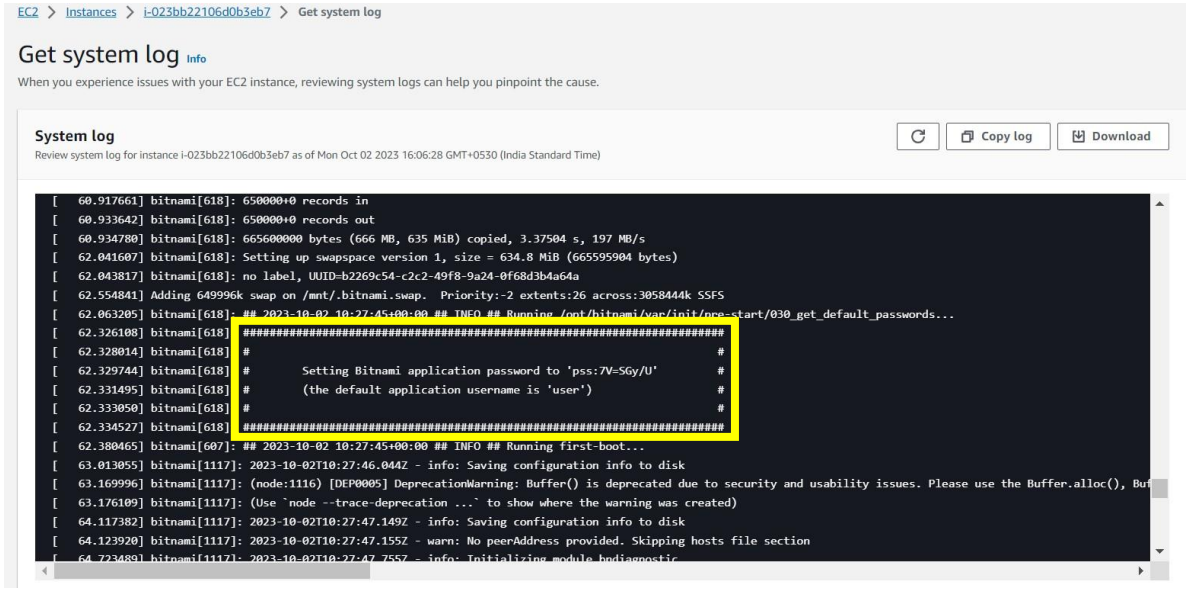
<after adding, the page would be>



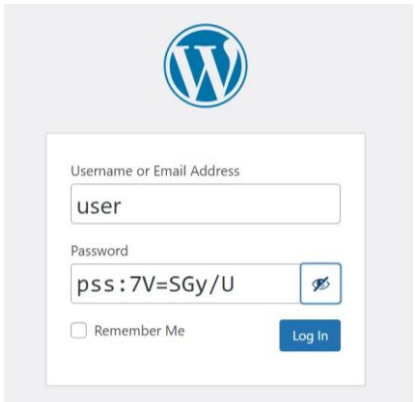
Step 4: Get username and password from System log (Actions > Monitor and troubleshoot > Get system log)



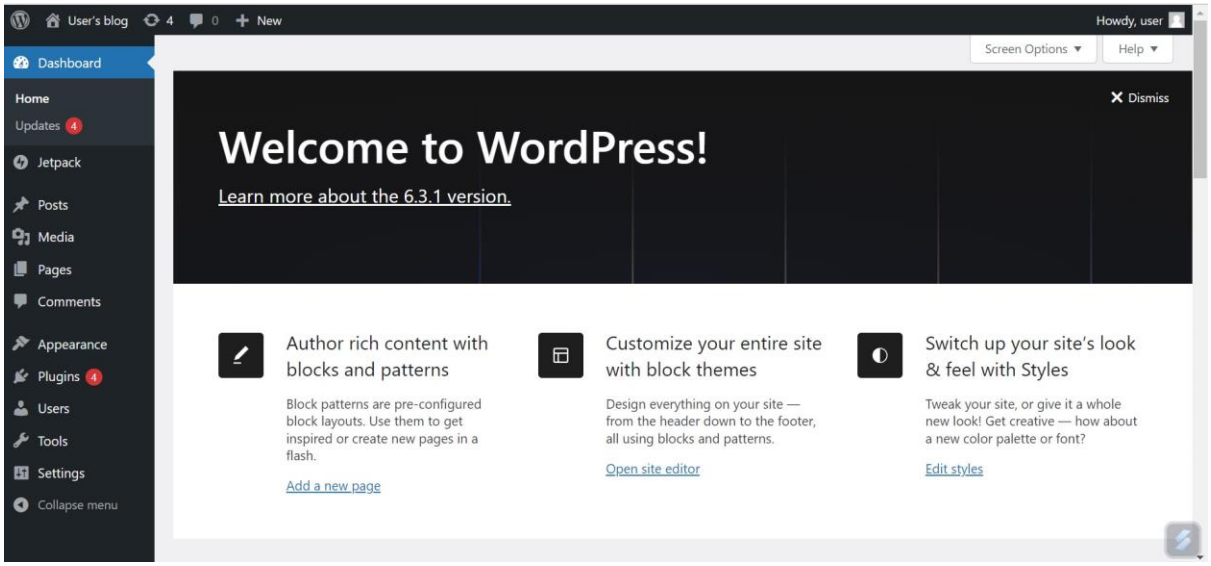
<you can see username and password here in system log>



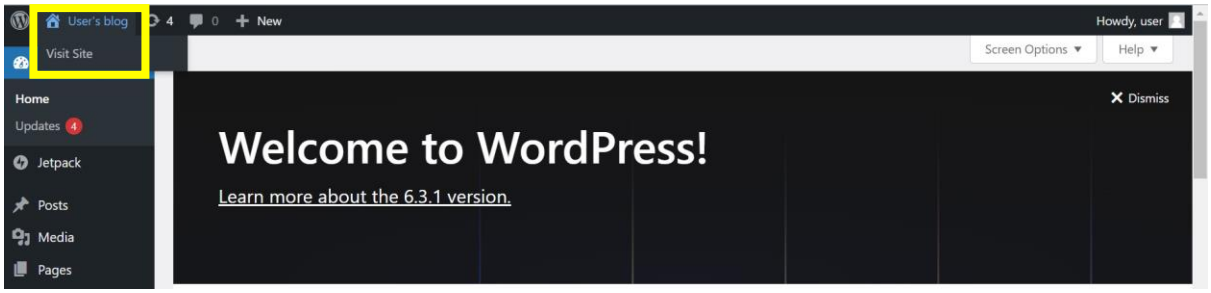
Step 5: Login using the credentials and it will navigate to the dashboard page.



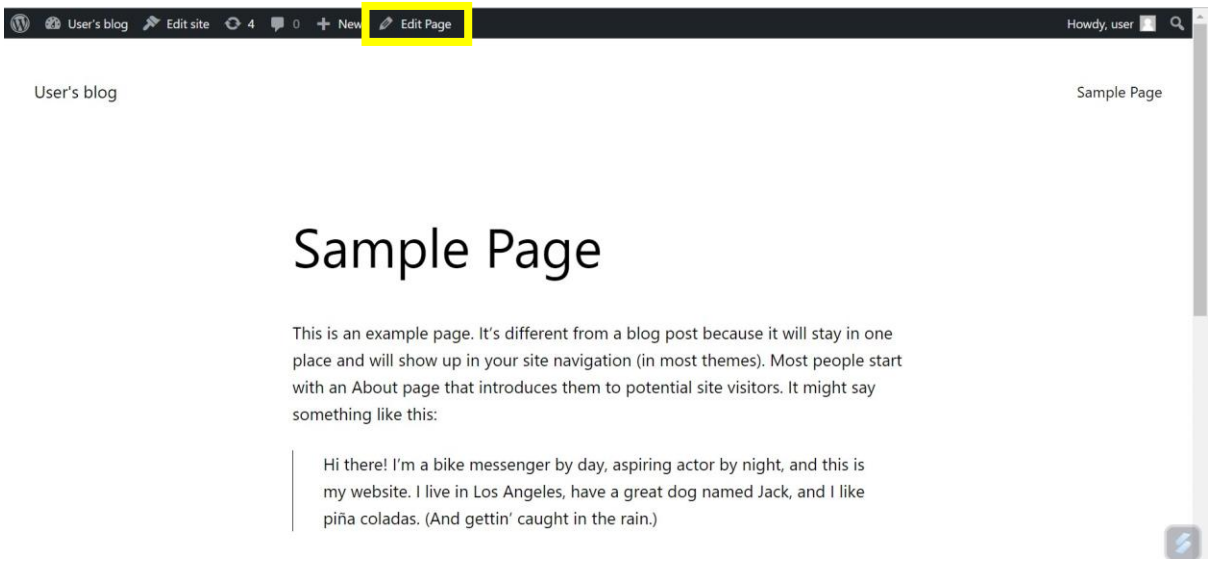
<Dashboard page>



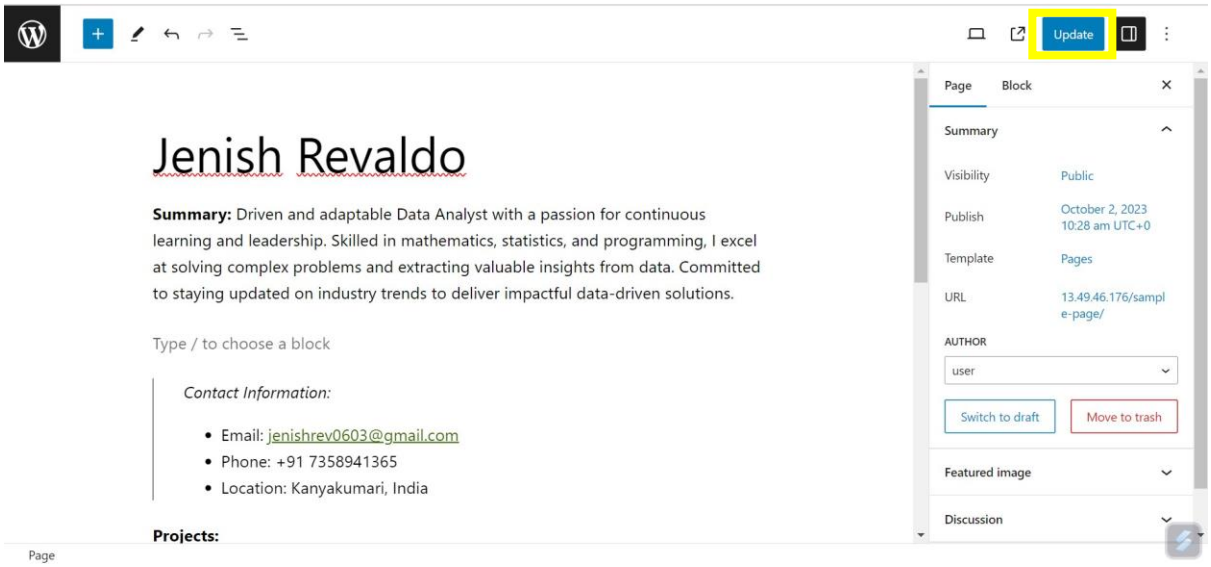
Step 6: Now you can edit the page by navigating to User's blob > Visit Site > Edit Page



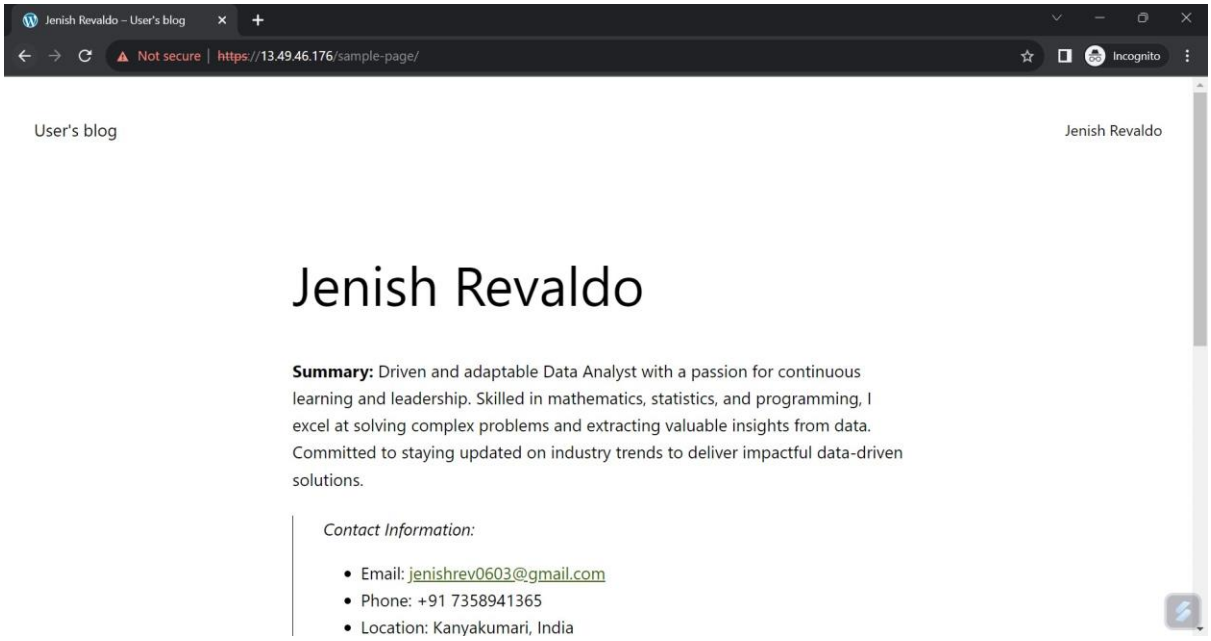
<next>



Step 7: Now you can start building your profile, after editing click on ‘Update’.



Step 8: Now navigate to the public Ip address, your contents will be updated.



Result:

Thus, deployed a WordPress website using Bitnami's pre-configured AMI from the AWS Marketplace, created and updated a user profile on the WordPress platform.

Deploying a Website in Azure Web App

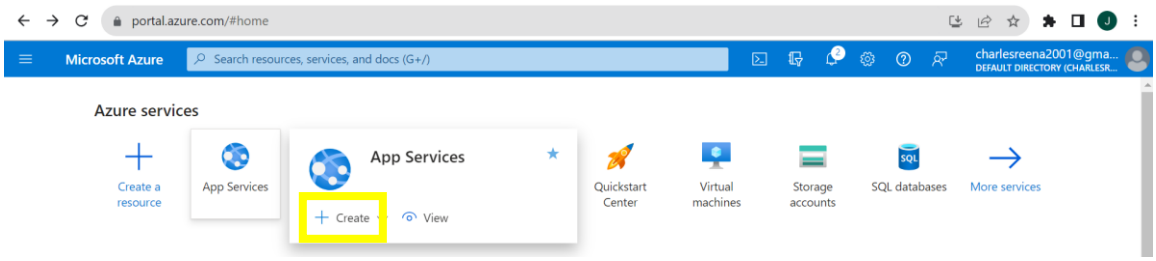
Aim:

To deploy a website to Azure Web App using the Azure CLI, ensuring that the website is accessible online.

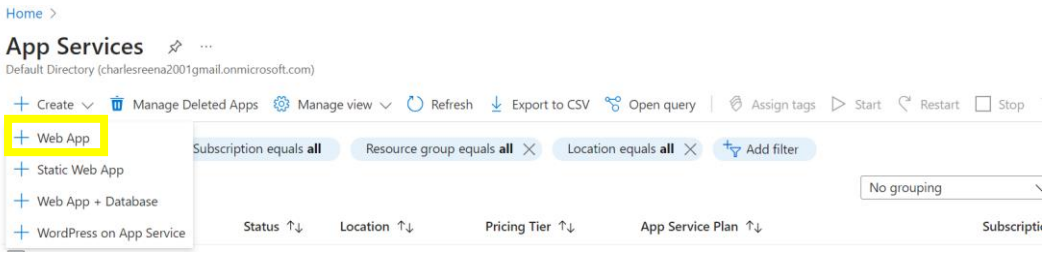
Procedure:

Step 1: Create a Web App in Azure App Service

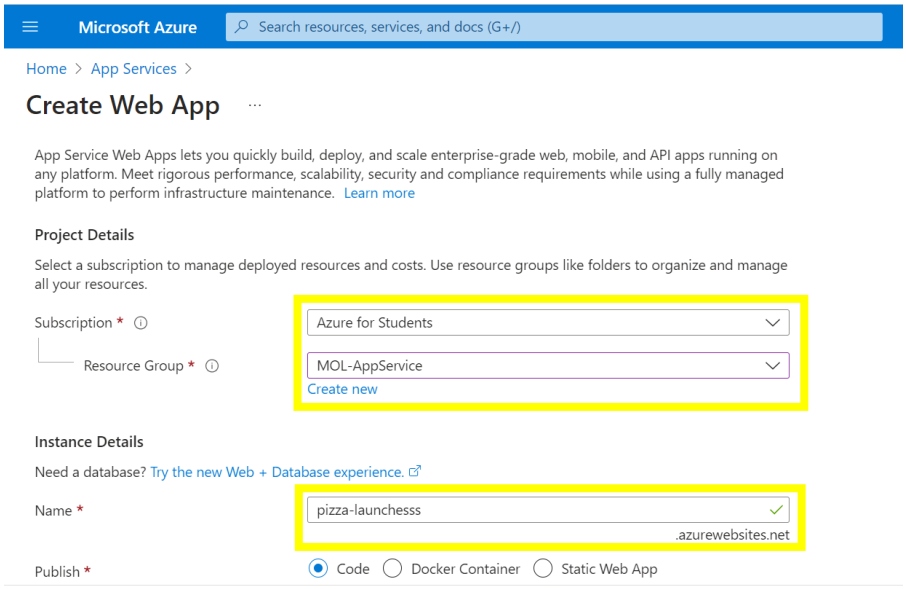
1. In the Azure Portal, navigate to the Azure App Service section.
2. Click on "Create" to create a new web app.



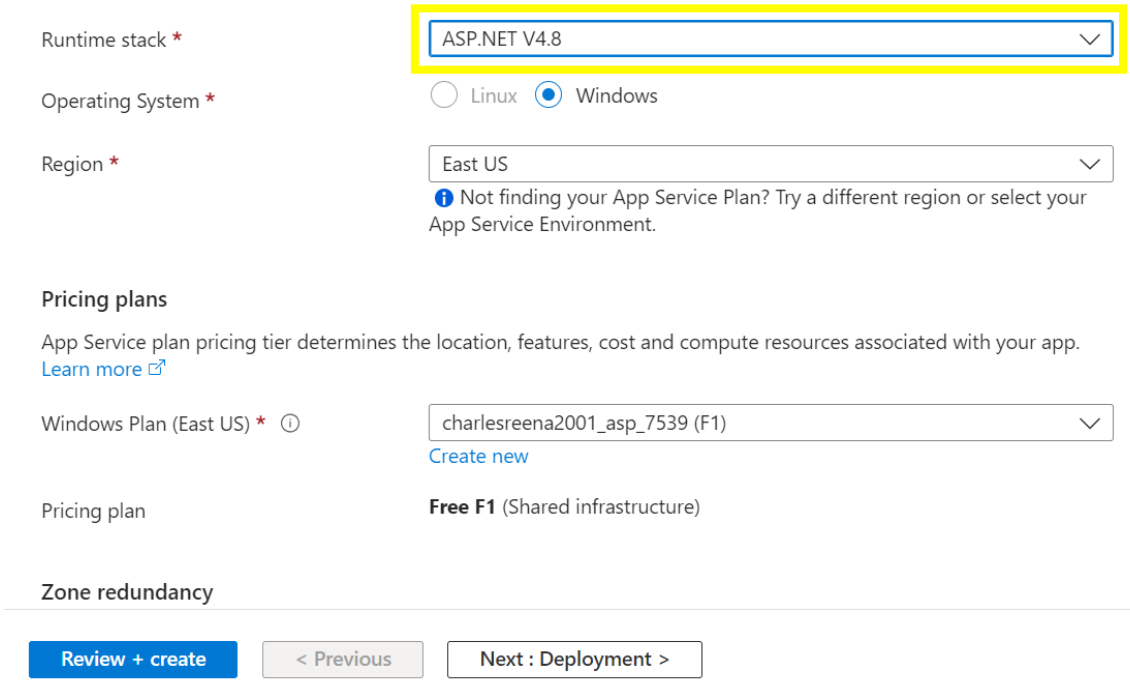
<next>



3. Provide a unique name for your web app. Choose your subscription, resource group, and App Service plan (or create a new one).
4. Select your runtime stack (e.g., .NET, Node.js, Python). Configure other settings like the operating system, region, and whether you want to enable or disable application insights.
5. Review your settings and click "Create" to provision the web app.

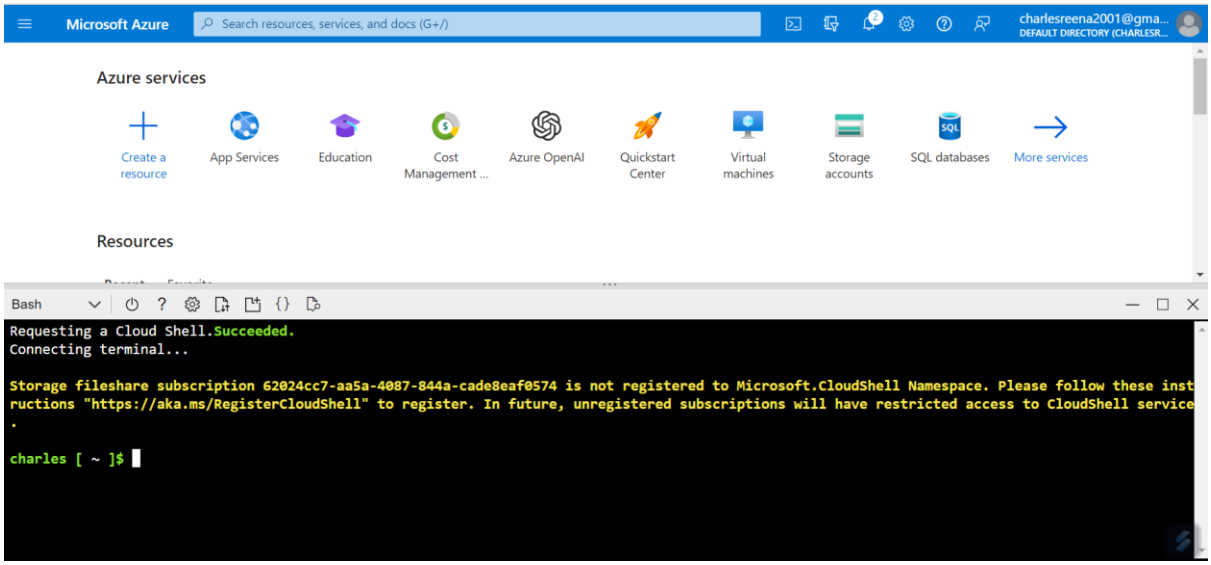


<next>



Step 2: Connect to the cloud shell using Bash

<then it shows the cloud shell>



Step 3: Use the **'wget'** command to download the website files from a source URL. In this example, we downloaded a zip file named "antique-cafe.zip."

```
charles [ ~ ]$ wget https://www.free-css.com/assets/files/free-css-templates/download/page295/antique-cafe.zip
```

Step 4: Use the '**unzip**' command to extract the contents of the downloaded zip file. This will create a directory containing the website files.

```
charles [ ~ ]$ ls
```

antique-cafe.zip azure-mol-samples-2nd-ed clouddrive

```
charles [ ~ ]$ unzip antique-cafe.zip
```

Step 5: Set your Git user email and name using the following commands

```
charles [ ~ ]$ git config --global user.email "your-email@example.com"
```

```
charles [ ~ ]$ git config --global user.name "your-username"
```

Step 6: Change your working directory to the directory containing your website files

```
charles [ ~ ]$ ls
```

2126_antique_cafe antique-cafe.zip azure-mol-samples-2nd-ed clouddrive

```
charles [ ~ ]$ cd 2126_antique_cafe
```

```
charles [ ~/2126_antique_cafe ]$ ls -ltr
```

total 36

-rw-r--r-- 1 charles charles   510 Jul 30 2019 'ABOUT THIS TEMPLATE.txt'

drwxr-xr-x 2 charles charles 4096 Sep 29 2021 webfonts

drwxr-xr-x 2 charles charles 4096 Sep 29 2021 js

drwxr-xr-x 2 charles charles 4096 Sep 29 2021 img

-rw-r--r-- 1 charles charles 15522 Sep 30 2021 index.html

drwxr-xr-x 2 charles charles 4096 Sep 30 2021 css

Step 7: Initialize a Git Repository

```
charles [ ~/2126_antique_cafe ]$ git init
```

Step 8: Add and Commit Website Files

```
charles [ ~/2126_antique_cafe ]$  git add .
```

```
charles [ ~/2126_antique_cafe ]$ git commit -m "css!"
```

Step 9: Use the Azure CLI to configure deployment credentials for your Azure Web App.

```
charles [ ~/2126_antique_cafe ]$ az webapp deployment source config-local-git --name pizza-launchess --resource-group MOL-AppService
```

Step 10: Retrieve the publishing credentials for your Azure Web App using the Azure CLI.

```
charles [ ~/2126_antique_cafe ]$ az webapp deployment list-publishing-credentials --name pizza-launchess --resource-group MOL-AppService
```


{

"id": "/subscriptions/62024cc7-aa5a-4087-844a-cade8eaf0574/resourceGroups/MOL-AppService/providers/Microsoft.Web/sites/pizza-launchess/publishingcredentials/$pizza-launchess",

"kind": null,

"location": "East US",

"name": "pizza-launchess",

"publishingPassword": "6bBoiQrt2WlNrzWlaXG3bKjDSsqmDlc6QetjJPdoynCckh1W6nKhkCljjcZ3",

"publishingPasswordHash": null,

"publishingPasswordHashSalt": null,

"publishingUserName": "$pizza-launchess",

"resourceGroup": "MOL-AppService",

"scmUri": "https://$pizza-launchess:6bBoiQrt2WlNrzWlaXG3bKjDSsqmDlc6QetjJPdoynCckh1W6nKhkCljjcZ3@pizza-launchess.scm.azurewebsites.net",

"type": "Microsoft.Web/sites/publishingcredentials"

}

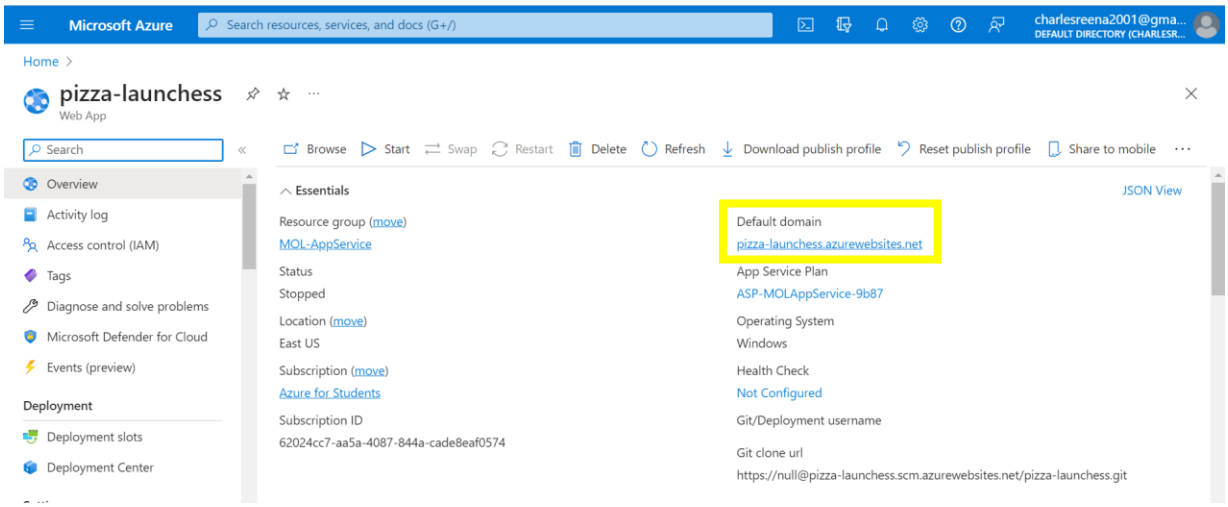
Step 11: Add the Azure Git repository as a remote to your local Git repository. Use the actual credentials you obtained in previous step.

```
charles [ ~/2126_antique_cafe ]$ git remote add azure 'https://$pizza-launchess:6bBoiQrt2WlNrzWlaXG3bKjDSsqmDlc6QetjJPdoynCckh1W6nKhkCljjcZ3@pizza-launchess.scm.azurewebsites.net'
```

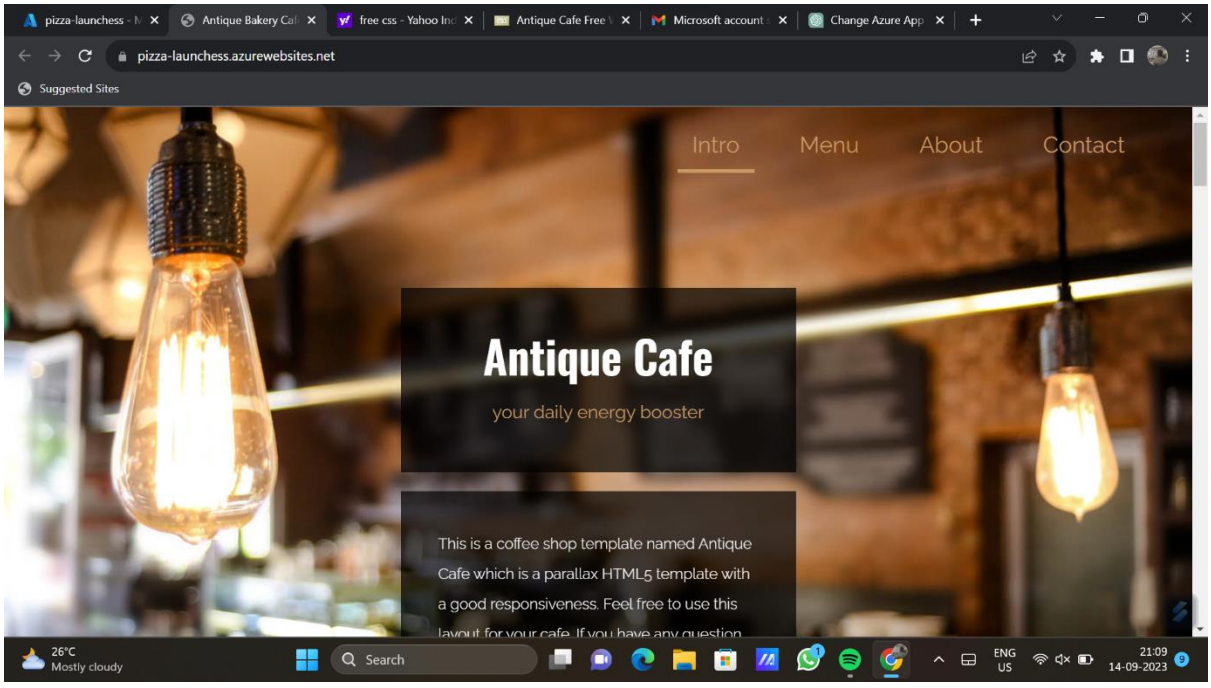
Step 12: Push Website to Azure

```
charles [ ~/2126_antique_cafe ]$ git push -f azure master
```

Step 13: Navigate to the default domain you obtained from Web App Overview



Output:



Result:

Thus, our website is successfully deployed to Azure Web App, and it is accessible online via the provided URL.

Azure Kubernetes Service (AKS)

Aim:

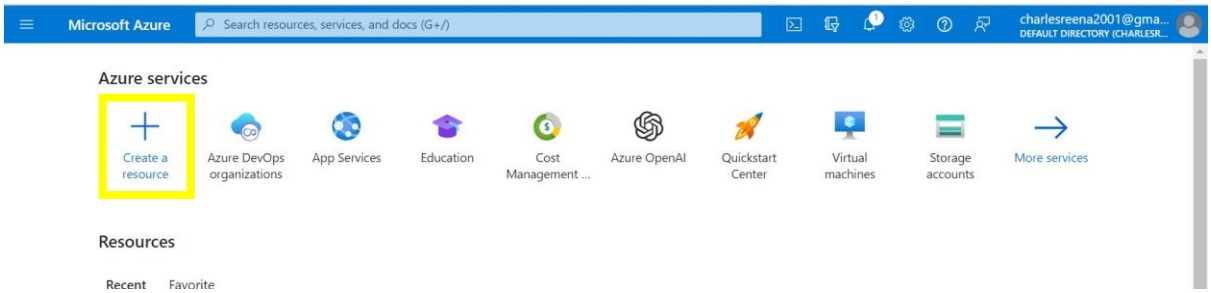
To Create an Azure Kubernetes Service (AKS) cluster, connect to it, and deploy a sample application to the cluster.

Procedure:

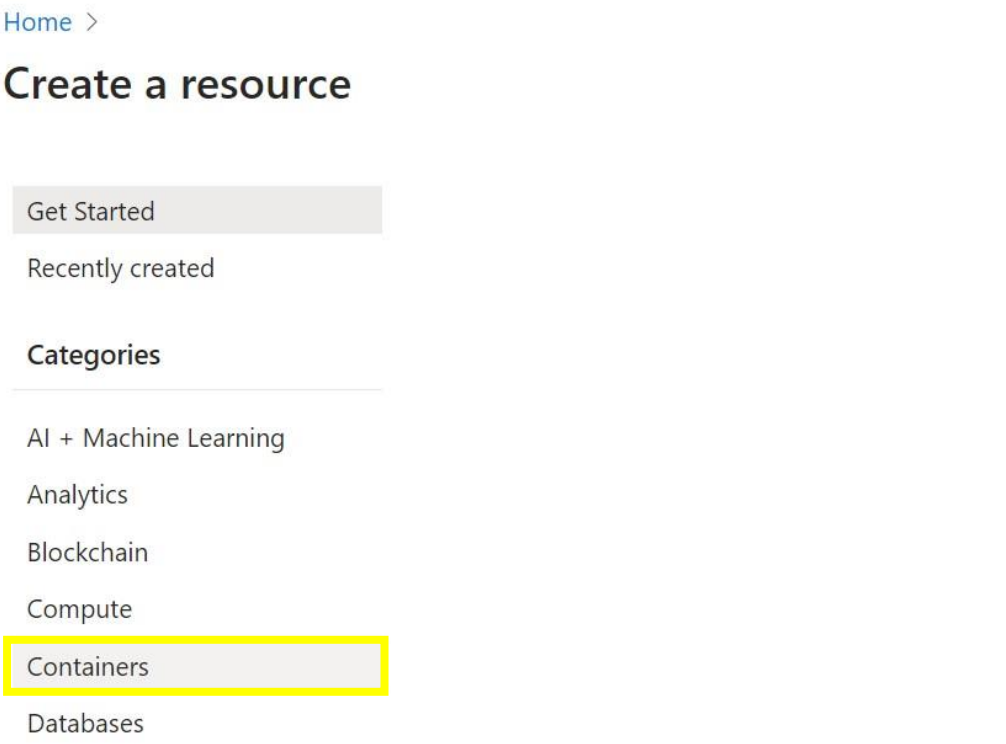
Step 1: Creating a Kubernetes cluster

Create an AKS cluster using Azure portal.

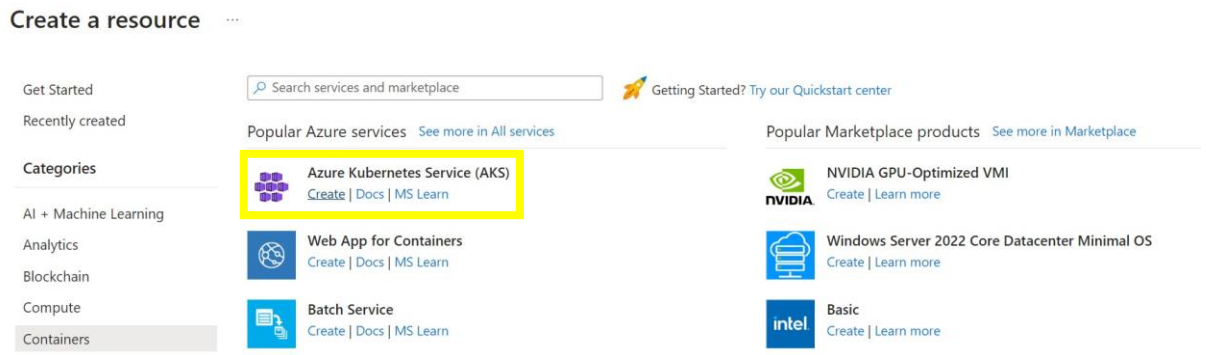
Create resource → Containers → Azure Kubernetes Services → Create.



<select containers>



<In Azure Kubernetes Services, Select Create>



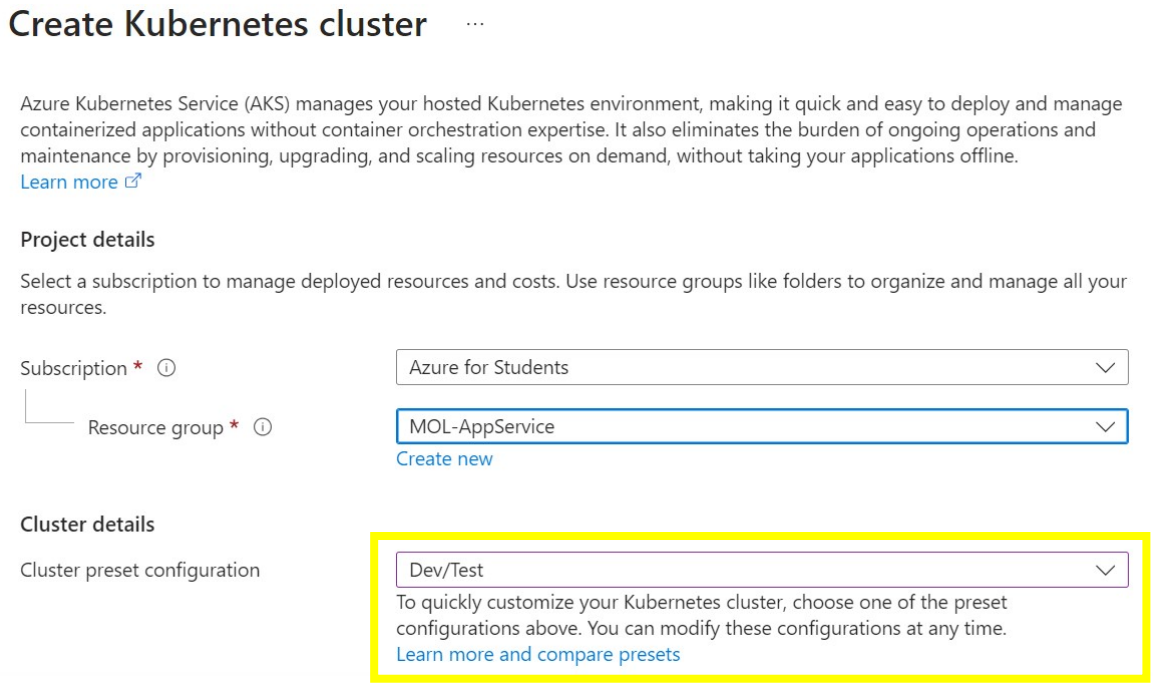
Step 2: Fill out the settings and configuration

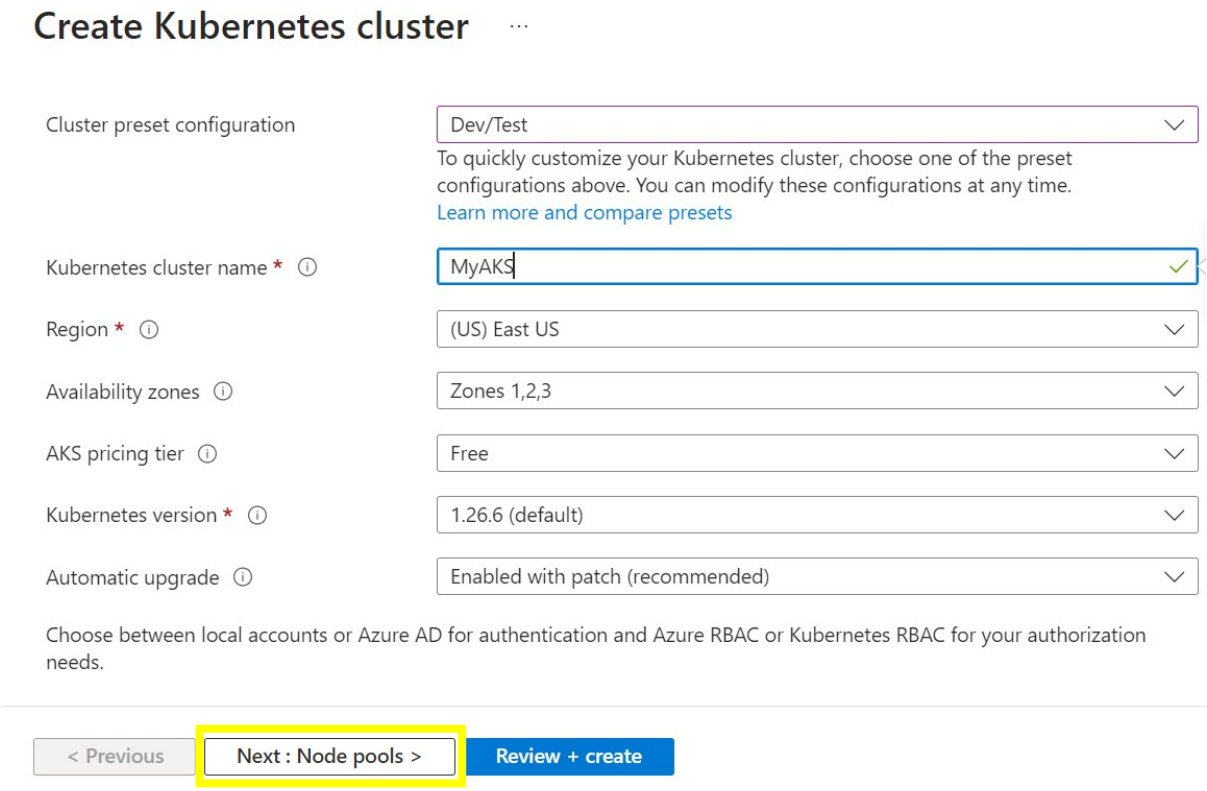
Basics tab:

1. Give a resource group and a cluster name

2. Select region as US East

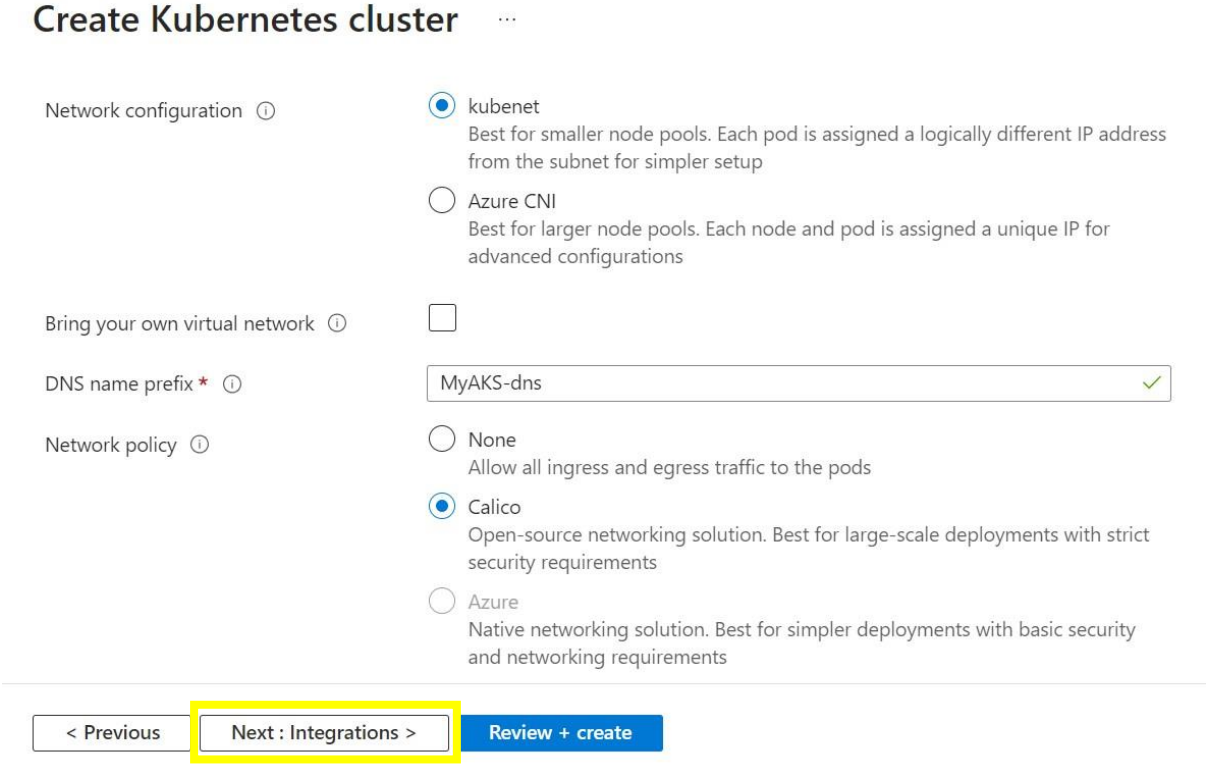
3. Cluster preset configuration as Dev/Test Nodes tab:





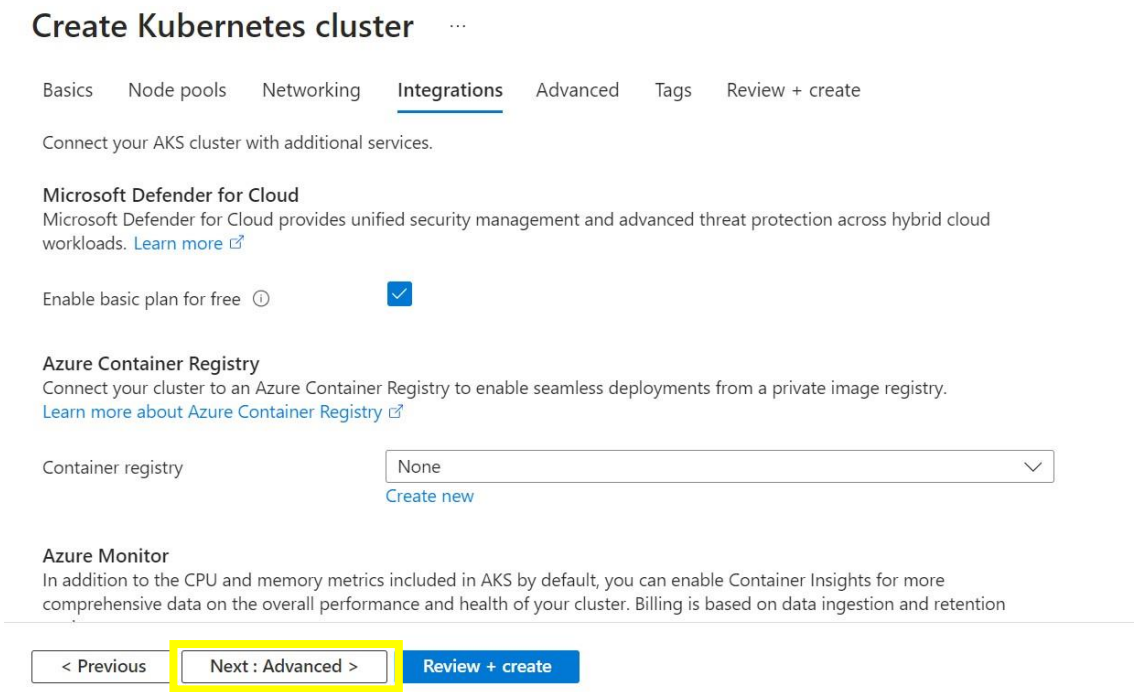
<Networking tab>

1. Select Network configuration as kubenet



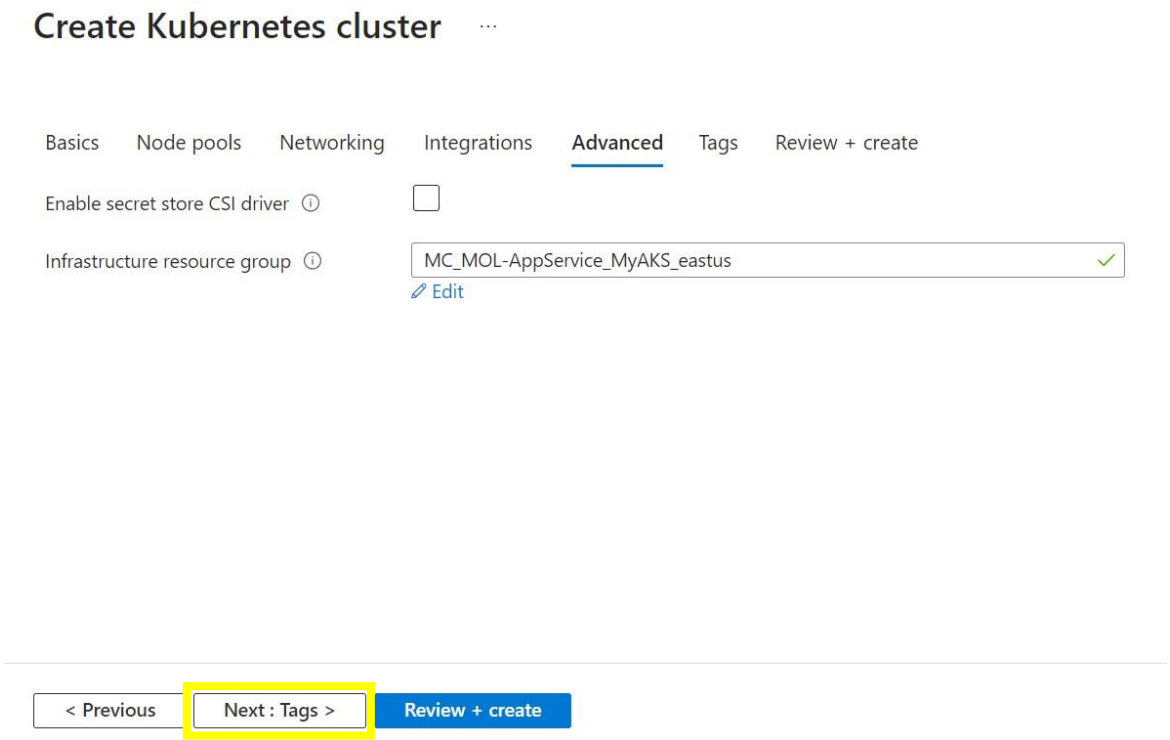
<Integrations tab>

Changes are not necessary. Proceed as it is.



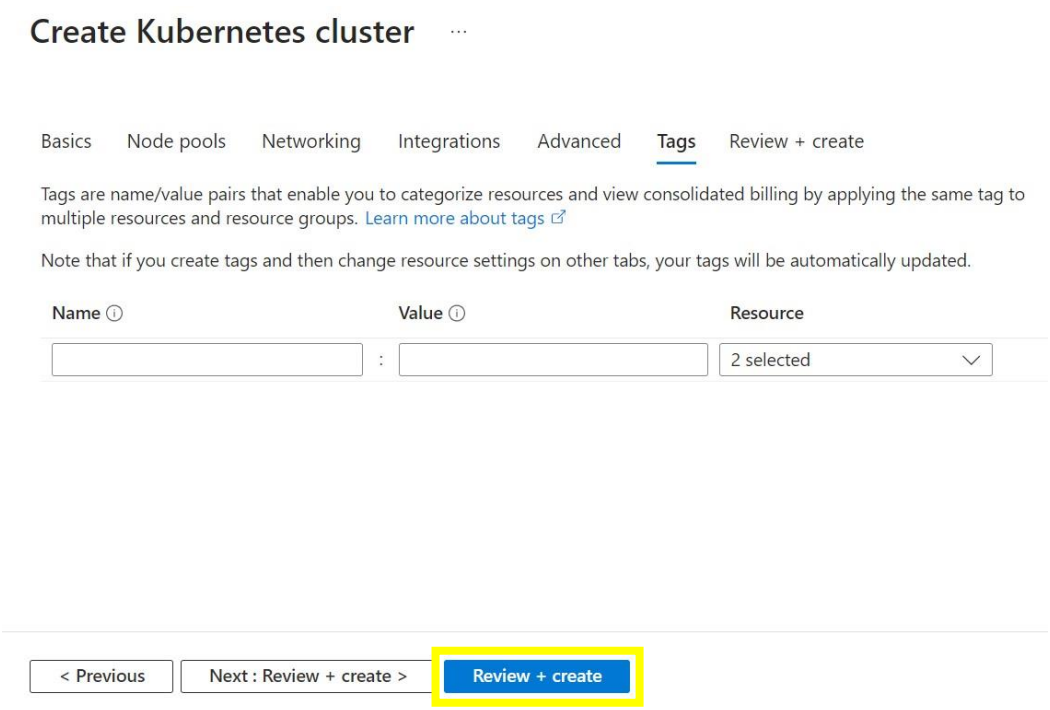
<Advanced tab>

Changes are not necessary. Proceed as it is.



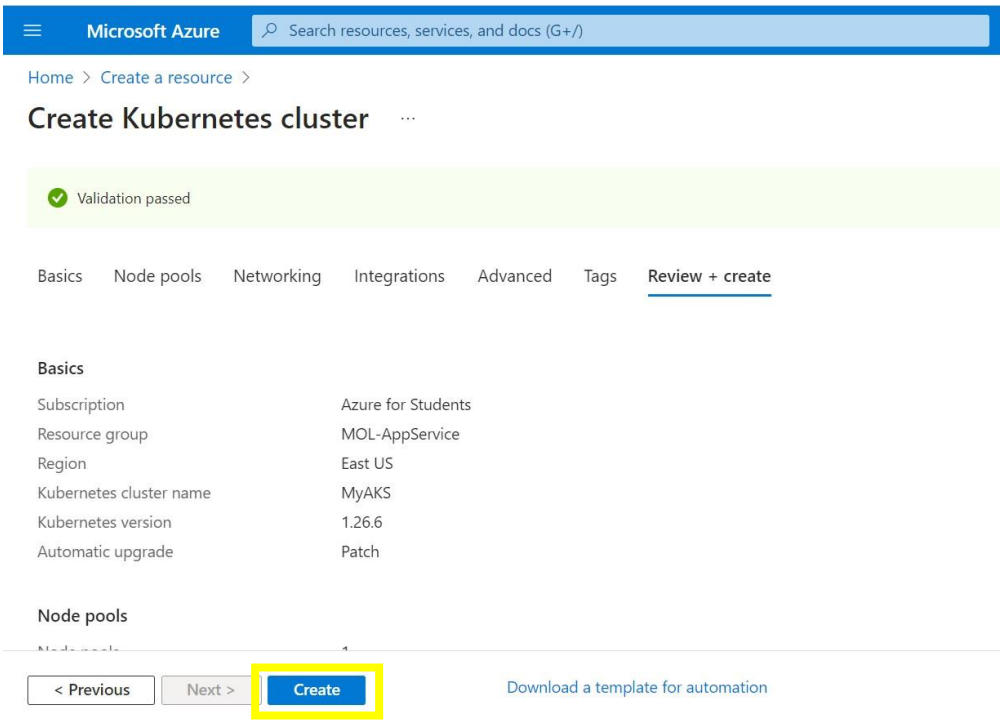
<Tags tab>

Changes are not necessary. Proceed as it is.

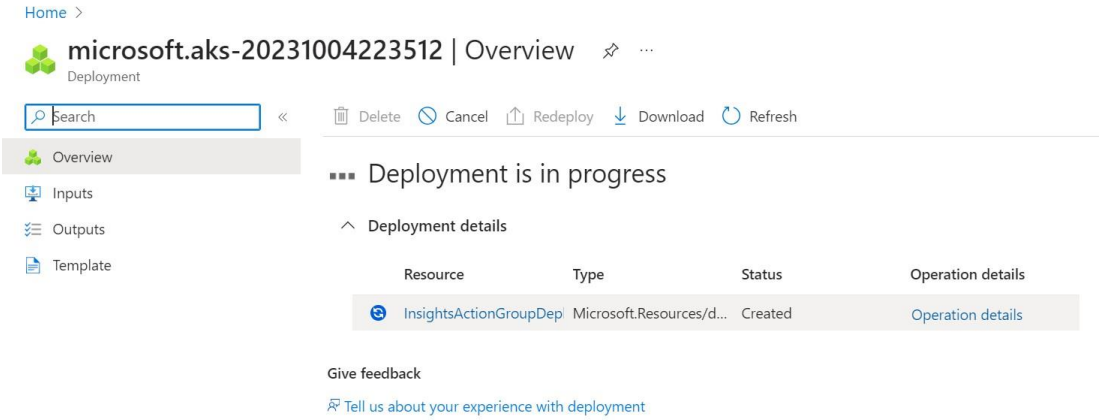


<Review and create tab>

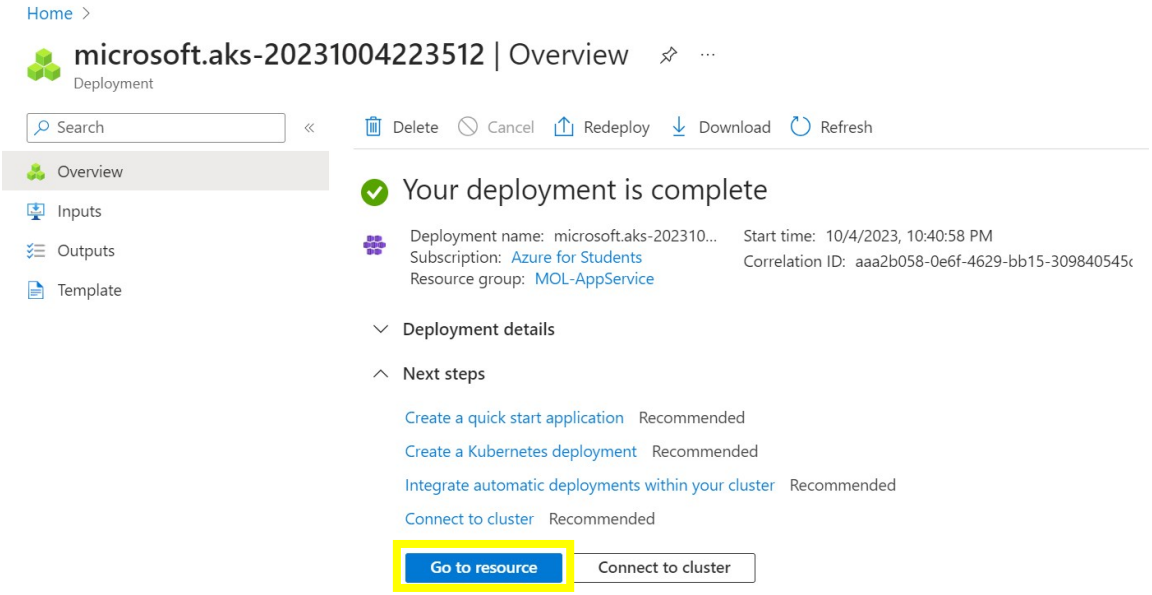
Click create after validation is passed



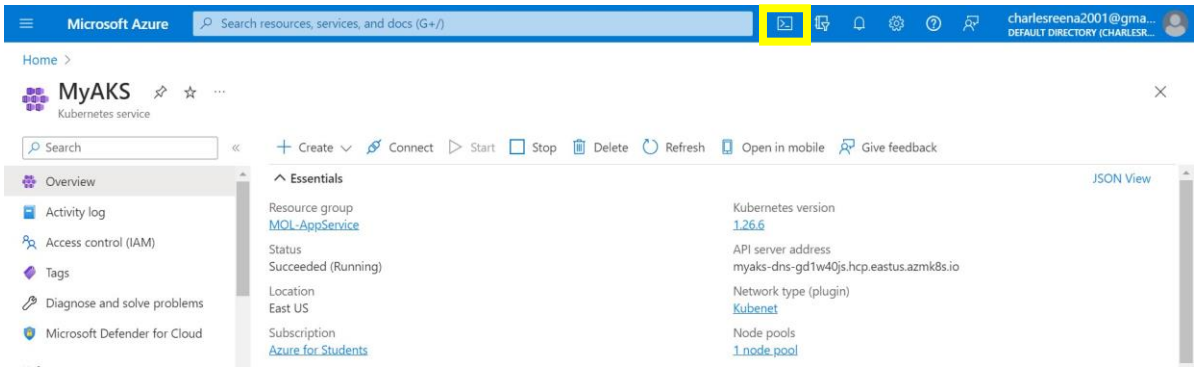
Step 3: Wait for the deployment to complete.



<after deployment, click on go to resources>



Step 4: Connect to Cloud shell using PowerShell



Step 5: Connect to the AKS Cluster

1. Run the following command to connect to your AKS cluster

PS /home/charles> az aks get-credentials --resource-group MOL-AppService --name MyAKS

Merged "MyAKS" as current context in /home/charles/.kube/config

This command configures your local **kubectl** to use the AKS cluster.

2. Verify that you are connected to the AKS cluster (display information about the nodes in your AKS cluster).

PS /home/charles> kubectl get nodes

NAME	*STATUS*	*ROLES*	*AGE*	*VERSION*
aks-agentpool-11990967-vmss000000	*Ready*	*agent*	*7m22s*	*v1.26.6*
aks-agentpool-11990967-vmss000001	*Ready*	*agent*	*7m28s*	*v1.26.6*

Step 6: Check for Existing Pods and Resources

1. Check for existing pods in the default namespace:

PS /home/charles> kubectl get pods

No resources found in default namespace.

2. Check for all resources (including services, deployments, and pods):

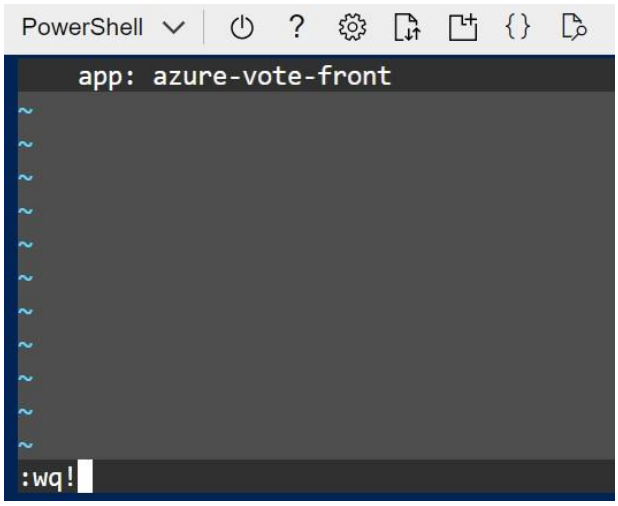
PS /home/charles> kubectl get all

NAME	*TYPE*	*CLUSTER-IP*	*EXTERNAL-IP*	*PORT(S)*	*AGE*
service/kubernetes	*ClusterIP*	*10.0.0.1*	*<none>*	*443/TCP*	*8m47s*

Step 7: Create and Apply a Kubernetes Manifest

1. Create or edit a Kubernetes manifest file (in this case, **askapp.yml**):

PS /home/charles> vi askapp.yml



Paste the yml contents and Finally add :wq! and click enter.

Contents are obtained from https://github.com/Azure-Samples/azure-voting-app-redis/blob/master/azure-vote-all-in-one-redis.yaml

2. List the files in your current directory to ensure **askapp.yml** is present:

PS /home/charles> ls

2126_antique_cafe antique-cafe.zip askapp.yml azure-mol-samples-2nd-ed clouddrive Microsoft

3. Apply the Kubernetes manifest to create the resources:

PS /home/charles> kubectl apply -f askapp.yml

deployment.apps/azure-vote-back created

service/azure-vote-back created

deployment.apps/azure-vote-front created

service/azure-vote-front created

Step 8: Check Deployments, Services, and Pods

1. Check the status of deployments:

```
PS /home/charles> kubectl get deployments
```

```
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
azure-vote-back   1/1     1            1           30s
azure-vote-front  1/1     1            1           30s
```

2. Check the status of services (This should display the services and their external IP addresses).

```
PS /home/charles> kubectl get svc
```

```
NAME              TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
azure-vote-back   ClusterIP      10.0.80.231    <none>          6379/TCP       75s
azure-vote-front  LoadBalancer   10.0.233.213   52.149.236.217  80:32717/TCP   74s
kubernetes        ClusterIP      10.0.0.1       <none>          443/TCP        16m
```

3. Retrieve information about ReplicaSets

```
PS /home/charles> kubectl get rs
```

```
NAME                        DESIRED   CURRENT   READY   AGE
azure-vote-back-66c88ccc8   1         1         1       2m18s
azure-vote-front-85dc674b97 1         1         1       2m18s
```

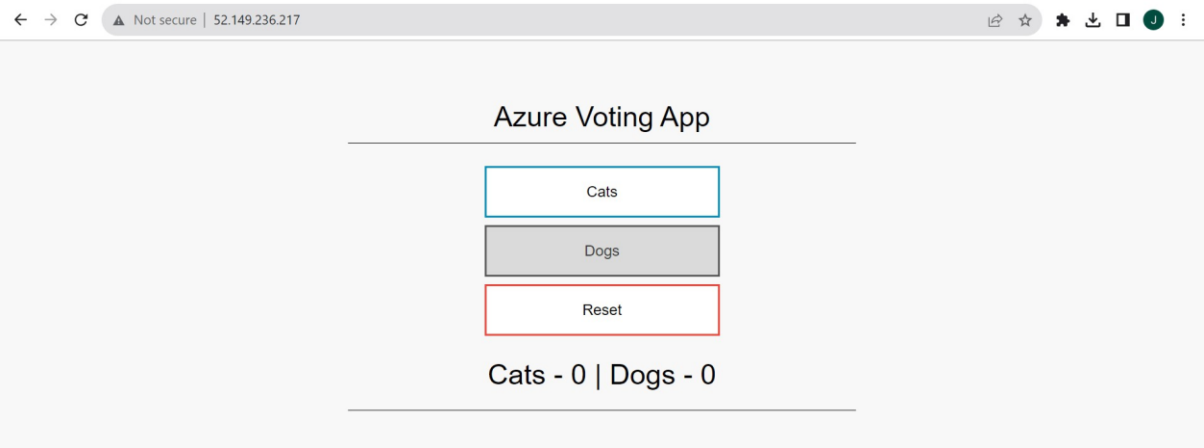
4. Verify that pods are running:

```
PS /home/charles> kubectl get pods
```

```
NAME                              READY   STATUS    RESTARTS   AGE
azure-vote-back-66c88ccc8-h572x   1/1     Running   0          2m33s
azure-vote-front-85dc674b97-c8zhh 1/1     Running   0          2m33s
```

Step 9: Paste the external IP address got in step 8.2

Output:



Result:

Thus, Created an Azure Kubernetes Service (AKS) cluster and deployed a sample application to the cluster.