DOCKER DEPLOYMENT ON AWS EC2

A PROJECT REPORT

Submitted by

SOUMYA RANJAN NAYAK

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF

SCHOOL OF ENGINEERING AND TECHNOLOGY BHUBANESWAR CAMPUS

CENTURION UNIVERSITY OF TECHNOLOGY AND MANAGEMENT ODISHA

DECEMBER 2023 / MAY 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF ENGINEERING AND TECHNOLOGY

BHUBANESWAR CAMPUS

BONAFIDE CERTIFICATE

Certified that this project report "**Docker Deployment on AWS EC2**" is the bonafide work of **Soumya Ranjan Nayak** who carried out the project work under my supervision. This is to further certify to the best of my knowledge, that this project has not been carried out earlier in this institute and the university.

SIGNATURE

Prof. Raj Kumar Mohanta

Assistant Professor of Computer Science and Engineering

Certified that the above -mentioned project has been duly carried out as per the norms of the college and statues of the university.

SIGNATURE

Dr. Sujata Chakravarty

DEAN OF SCHOOL OF ENGINEERING AND TECHNOLOGY

Professor of Computer Science and Engineering

DEPARTMENT SEAL

DECLARATION

I hereby declare that the project entitled “**Docker Deployment on AWS EC2”** submitted for the “Minor Project” of 8th semester B.Tech in Computer Science and Engineering is my original work and the project has not formed the basis for the award any Degree or any other similar titles in any other University / Institute.

Name of the Student: Soumya Ranjan Nayak

Signature of the Student:

Registration No.: 200301120175

Place: Centurion University of Technology and Management, Bhubaneswar

Date:

ACKNOWLEDGEMENTS

I wish to express my profound and sincere gratitude to Prof. Raj kumar Mohanta, Department of Computer Science and Engineering, SoET, Bhubaneswar Campus, who guided me into the intricacies of this project nonchalantly with matches magnanimity.

I thank Prof. Raj Kumar Mohanta, Head of the Dept. of Department of Computer Science and Engineering, SoET, Bhubaneswar Campus and Dr. Sujata Chakravarty, Dean, School of Engineering and Technology, Bhubaneswar Campus for extending their support during course of this investigation.

I would be failing in my duty if I do not acknowledge the cooperation rendered during various stages of image interpretation by.

I am highly grateful to Prof. Raj kumar Mohanta who evinced keen interest and invaluable support in the progress and successful completion of my project work.

I am indebted to Prof. Raj kumar Mohanta for their constant encouragement, cooperation, and help. Words of gratitude are not enough to describe the accommodation and fortitude which they have shown throughout my endeavor.

Name of the Student: Soumya Ranjan Nayak
Signature of the Student:
Registration No.: 200301120175
Place: Centurion University of Technology and Management, Bhubaneswar
Date:

ABSTRACT

This abstract explores the deployment of Docker containers on Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances, focusing on hosting a static website. Docker has emerged as a leading technology for containerization, offering portability, scalability, and efficiency in application deployment. AWS EC2 provides a scalable and flexible cloud computing environment ideal for hosting Dockerized applications. The abstract outlines the process of provisioning an EC2 instance, installing Docker, creating a Dockerfile, building a Docker image for the static website, and running a Docker container to host the website. It highlights the benefits of this approach, including simplified deployment, improved resource utilization, and enhanced scalability. Additionally, considerations such as security configurations, network settings, and domain name management are discussed to ensure the reliability and accessibility of the hosted website. Overall, deploying Docker containers on AWS EC2 offers a streamlined and efficient solution for hosting static websites, leveraging the strengths of both technologies to deliver reliable and scalable web hosting services.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE
1	INTRODUCTION	07
2	SYSTEM PROPOSAL	09
3	SYSTEM REQUIREMENT	10
4	SYSTEM DESIGN	11
5	SYSTEM DIAGRAM	13
6	IMPLEMENTATION	14
7	APPENDIX	15
8	CONCLUSION	21

CHAPTER 1

INTRODUCTION

Docker is a powerful tool for packaging, distributing, and running applications within containers, providing a consistent environment across different platforms. Deploying Docker on AWS EC2 instances leverages the scalability and reliability of AWS cloud infrastructure while harnessing the flexibility and efficiency of Docker containers. To begin, you'll first need to provision an EC2 instance on AWS, selecting an appropriate instance type based on your application's resource requirements. Once the EC2 instance is up and running, you'll need to install Docker on it, which can be achieved through simple commands or scripts provided by AWS documentation. With Docker installed, you can then build your application's Docker image, defining its dependencies and configuration within a Dockerfile. After building the image, you can push it to a Docker registry like Docker Hub or Amazon ECR (Elastic Container Registry). Finally, you can deploy your Dockerized application on the EC2 instance by pulling the Docker image from the registry and running it within a Docker container. By following this process, you can efficiently manage and scale your applications on AWS EC2 using Docker containers, benefiting from the flexibility, isolation, and portability they offer.

Deploying a static website on AWS EC2 using Docker provides a streamlined and efficient way to host and manage your web content. To get started, you'll first need to set up an EC2 instance on AWS, selecting an appropriate instance type based on your website's traffic and resource requirements. Once the EC2 instance is provisioned, you can install Docker on it, following AWS documentation or using standard Docker installation procedures. With Docker installed, you'll create a Dockerfile that specifies the configuration of your web server and includes your static website files. This Dockerfile might use a lightweight web server like Nginx or Apache to serve the static content.

After defining your Dockerfile, you'll build a Docker image for your static website, incorporating your website files and the necessary server configuration. Once the image is built, you can run a Docker container based on this image on your EC2 instance. This container will host your static website, making it accessible via the EC2 instance's public IP address or domain name.

CHAPTER 2

SYSTEM PROPOSAL

1. In today's dynamic and competitive digital landscape, efficient deployment and management of applications are paramount. This proposal outlines a comprehensive system for deploying Docker containers on Amazon Web Services (AWS) Elastic Compute Cloud (EC2), offering a scalable, reliable, and cost-effective solution for hosting various types of applications.

2. Objectives:
    - To streamline the deployment process of Dockerized applications on AWS EC2 instances.
    - To optimize resource utilization and scalability through containerization.
    - To ensure high availability and fault tolerance of deployed applications.
    - To enhance security measures and compliance standards for hosted applications.
    - To facilitate seamless integration with existing CI/CD pipelines and development workflows.

3. Proposed System Architecture:
    - Utilize AWS EC2 instances as the hosting infrastructure for Docker containers.
    - Implement Docker Engine on EC2 instances to manage containerized applications.
    - Utilize Docker Compose or Kubernetes for container orchestration and management.
    - Integrate with AWS services such as Elastic Container Registry (ECR), AWS Identity and Access Management (IAM), and Amazon Virtual Private Cloud (VPC) for enhanced security and networking capabilities.
    - Implement auto-scaling and load balancing mechanisms to ensure optimal performance and availability.

4. System Components: a. AWS EC2 Instances: Provisioned to host Docker containers, with customizable instance types based on workload requirements. b. Docker Engine: Installed on EC2 instances to manage containerized applications and their dependencies. c. Docker Compose or Kubernetes: Utilized for orchestrating and managing multi-container applications, enabling efficient scaling and resource allocation. d. AWS Services

Integration: Leveraged for enhanced security, scalability, and networking capabilities, including ECR for container image storage, IAM for access control, and VPC for network isolation. e. Monitoring and Logging: Implement monitoring tools like AWS CloudWatch and logging solutions such as AWS CloudTrail for real-time performance monitoring and auditing of deployed applications.

5. Implementation Plan:
    - Conduct an assessment of current infrastructure and application requirements.
    - Provision AWS EC2 instances and configure networking settings.
    - Install Docker Engine on EC2 instances and set up Docker Compose or Kubernetes for container orchestration.
    - Develop Dockerfiles and containerize applications following best practices.
    - Integrate with AWS services for enhanced security and scalability.
    - Implement monitoring and logging solutions for proactive management and troubleshooting.
    - Conduct thorough testing and validation of the deployed system.
    - Provide documentation and training for operations and development teams.

6. Benefits:
    - Simplified deployment process: Streamlines the process of deploying applications on AWS EC2 using Docker containers.
    - Enhanced scalability and resource utilization: Enables efficient scaling and allocation of resources based on application demand.
    - Improved security and compliance: Integrates with AWS security services to enforce access control and compliance standards.
    - Seamless integration: Facilitates integration with existing CI/CD pipelines and development workflows, enhancing productivity and collaboration.

CHAPTER 3

SYSTEM REQUIREMENT

HARDWARE REQUIREMENTS:

- Processor Name: Dual Core
- Processor Speed: 3.2 GHz
- RAM: 4 GB
- Hard Disk Capacity: 80 GB
- Display Device: 14’ to 19’ Inch Monitor
- Keyboard Type: PS2 or USB
- Mouse Type: PS2 or USB

SOFTWARE REQUIREMENTS:

- Technology Implemented: Apache Server, Dream weaver 18
- Language Used: PHP 5.2
- Database: My SQL 5.2
- User Interface Design: HTML, AJAX, JAVA SCRIPT
- Web Browser: Mozilla, IE8

CHAPTER 4

SYSTEM DESIGN

- System design is the process or art of defining the architecture, components, modules, interfaces and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. Design is the first phase in development phase for any engineer’s product system. Design is the creative process. It deals with the creative ability of the programmer. A good design is the key to effective system. The term “Design” is defined as “The process of applying various techniques and principles for the purpose of defining a process or a system in sufficient details to permit its physical realization”.

Input design: -

1. Overview:

The system design for Docker deployment on AWS EC2 aims to provide a scalable, reliable, and secure platform for hosting containerized applications. This design encompasses the architecture, components, and interactions necessary to achieve efficient deployment and management of Docker containers on AWS EC2 instances.

2. Architecture:

The system architecture consists of the following components:

- AWS EC2 Instances: Provisioned virtual servers running the Docker Engine to host containerized applications.
- Docker Engine: Installed on EC2 instances to manage containers, images, networks, and volumes.
- Docker Compose or Kubernetes: Utilized for orchestrating and managing multi-container applications, enabling scalability and resource optimization.
- AWS Services Integration: Integration with AWS services such as Elastic Container Registry (ECR), AWS Identity and Access Management (IAM), and Amazon Virtual Private Cloud (VPC) for enhanced security, scalability, and networking capabilities.
- Monitoring and Logging: Implementation of monitoring tools like AWS CloudWatch and logging solutions such as AWS CloudTrail for real-time performance monitoring and auditing.

3. Components:

a. AWS EC2 Instances:

- Selection of EC2 instance types based on workload requirements (e.g., CPU, memory, storage).
- Configuration of security groups, IAM roles, and key pairs for access control and security.

b. Docker Engine:

- Installation of Docker Engine on EC2 instances to manage containers.
- Configuration of Docker daemon settings for resource allocation and container networking.

c. Container Orchestration:

- Use of Docker Compose for managing single-host deployments or Kubernetes for multi-host deployments.
- Configuration of Docker Compose files or Kubernetes manifests to define application services, networks, and volumes.

d. AWS Services Integration:

- Utilization of Elastic Container Registry (ECR) for storing Docker container images securely.
- Configuration of IAM roles and policies to control access to AWS resources.
- Utilization of Amazon VPC for network isolation and security.

e. Monitoring and Logging:

- Implementation of AWS CloudWatch for monitoring EC2 instances, containers, and application metrics.
- Configuration of CloudWatch alarms for automated scaling and alerting.
- Integration with AWS CloudTrail for auditing and logging API calls and resource changes.

4. Interactions:

- EC2 instances interact with Docker Engine to manage containers and execute Docker commands.
- Docker Compose or Kubernetes orchestrates container deployments and manages application services.
- AWS services interact with EC2 instances and Docker containers for tasks such as image storage, access control, and network management.
- Monitoring and logging tools continuously monitor the health and performance of EC2 instances, Docker containers, and applications.

5. Security Considerations:

- Implementation of IAM roles and policies to restrict access to AWS resources.
- Configuration of security groups and network ACLs to control inbound and outbound traffic.
- Encryption of data in transit and at rest using AWS Key Management Service (KMS) and SSL/TLS.
- Regular security assessments and vulnerability scanning of EC2 instances and Docker containers.

6. Scalability and High Availability:

- Use of auto-scaling groups to automatically adjust the number of EC2 instances based on workload demand.
- Implementation of load balancers to distribute incoming traffic across multiple EC2 instances.
- Configuration of Docker Compose or Kubernetes for horizontal scaling of application services.

7. Conclusion:

The system design for Docker deployment on AWS EC2 provides a robust and scalable platform for hosting containerized applications. By leveraging AWS services and best practices in container orchestration and management, organizations can achieve enhanced agility, reliability, and security in deploying and managing their applications in the cloud.

CHAPTER 5

SYSTEM DIAGRAM

FLOW DIAGRAM

DATA FLOWDIAGRAM

CHAPTER 6

IMPLEMENTATION

Implementing Docker deployment on AWS EC2 involves several steps, from provisioning EC2 instances to deploying Docker containers. Here's a step-by-step guide to the implementation process:

1. **Provision AWS EC2 Instances:**
   - Log in to the AWS Management Console.
   - Navigate to the EC2 dashboard.
   - Launch a new EC2 instance by selecting an Amazon Machine Image (AMI), instance type, and other configuration details.
   - Configure security groups to allow inbound traffic on necessary ports (e.g., port 80 for HTTP).
   - Optionally, configure IAM roles for EC2 instances to access other AWS services.

2. **Install Docker on EC2 Instances:**
   - Connect to the newly provisioned EC2 instance via SSH.
   - Update the package repository: `sudo yum update` (for Amazon Linux) or `sudo apt update` (for Ubuntu).
   - Install Docker:
     - For Amazon Linux: `sudo yum install docker`
     - For Ubuntu: `sudo apt install docker.io`
   - Start the Docker service: `sudo service docker start`
   - Add the current user to the docker group to run Docker commands without sudo: `sudo usermod -aG docker $USER`
   - Log out and log back in to apply the group membership changes.

3. **Create Dockerfile for Your Application:**
   - Create a directory for your Docker project: `mkdir myapp && cd myapp`.
   - Create a Dockerfile in the project directory with instructions to build your application image.
   - Add necessary commands to install dependencies, copy application files, and define the runtime environment.
   - Here's a basic example for a Node.js application:

```Dockerfile
FROM node:14

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3000

CMD ["node", "app.js"]
```

4. **Build Docker Image:**
   - Build the Docker image from the Dockerfile: `docker build -t myapp .`.
   - Replace `myapp` with the desired image name.

5. **Run Docker Container:**
   - Start a Docker container from the built image: `docker run -d -p 80:3000 myapp`.
   - Replace `myapp` with the image name specified during the build process.
   - This command runs the container in detached mode (`-d`) and maps port 3000 of the container to port 80 of the host (`-p 80:3000`).

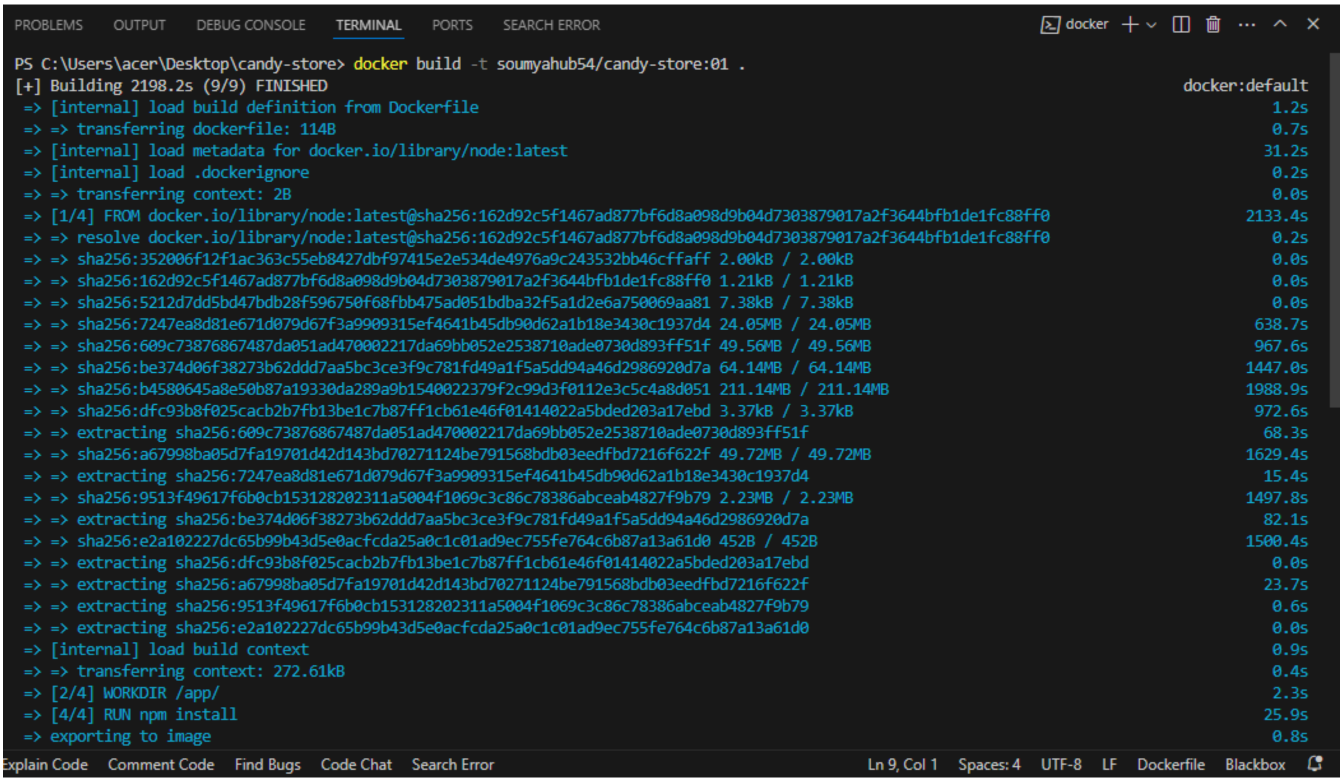
6. **Access Your Application:**
   - Access your application by navigating to the public IP address or DNS name of your EC2 instance in a web browser.
   - If using a custom domain, configure DNS settings accordingly.

7. **Additional Considerations:**
   - Implement security best practices, such as securing SSH access to EC2 instances, configuring network ACLs, and enabling HTTPS.
   - Set up monitoring and logging using AWS CloudWatch or third-party tools for performance monitoring and troubleshooting.
   - Explore container orchestration tools like Docker Compose or Kubernetes for managing multi-container applications.

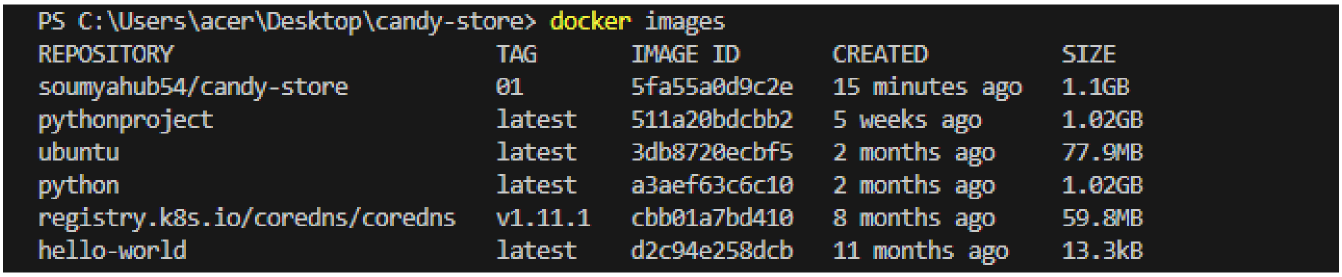
CHAPTER 7

APPENDIX

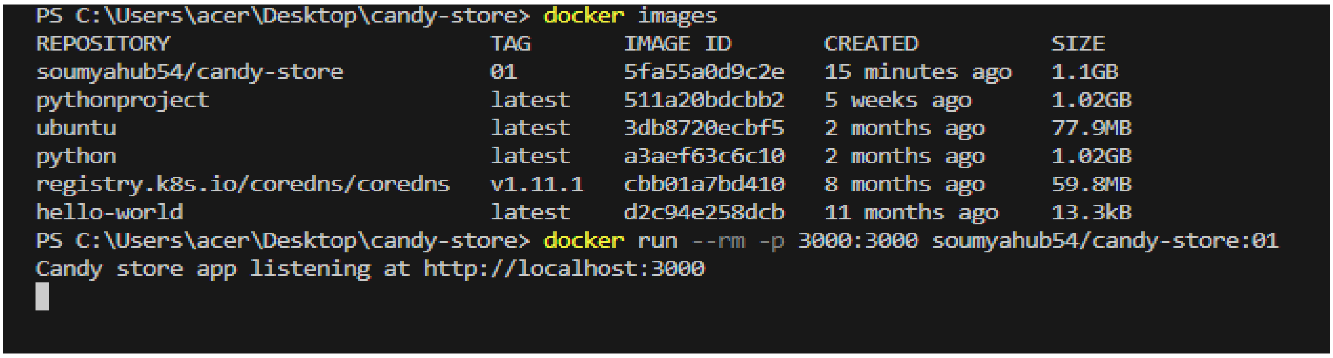
SNAPSHOTS



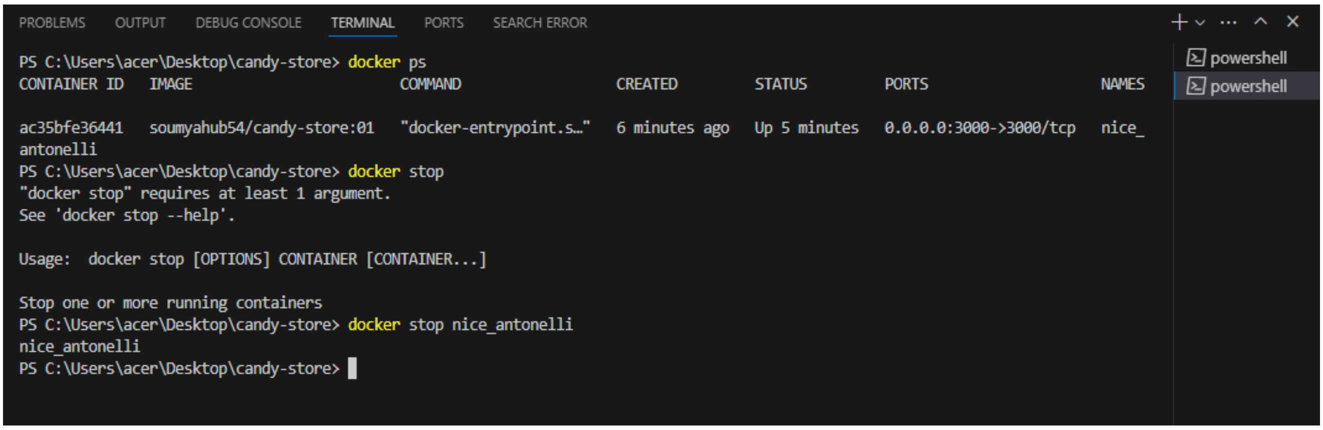
```
PS C:\Users\acer\Desktop\candy-store> docker build -t soumyahub54/candy-store:01 .
[+] Building 2198.2s (9/9) FINISHED                                                                                      docker:default
 => [internal] load build definition from Dockerfile                                                                               1.2s
 => => transferring dockerfile: 114B                                                                                               0.7s
 => [internal] load metadata for docker.io/library/node:latest                                                                    31.2s
 => [internal] load .dockerignore                                                                                                  0.2s
 => => transferring context: 2B                                                                                                    0.0s
 => [1/4] FROM docker.io/library/node:latest@sha256:162d92c5f1467ad877bf6d8a098d9b04d7303879017a2f3644bfb1de1fc88ff0            2133.4s
 => => resolve docker.io/library/node:latest@sha256:162d92c5f1467ad877bf6d8a098d9b04d7303879017a2f3644bfb1de1fc88ff0               0.2s
 => => sha256:352006f12f1ac363c55eb8427dbf97415e2e534de4976a9c243532bb46cffaff 2.00kB / 2.00kB                                     0.0s
 => => sha256:162d92c5f1467ad877bf6d8a098d9b04d7303879017a2f3644bfb1de1fc88ff0 1.21kB / 1.21kB                                     0.0s
 => => sha256:5212d7dd5bd47bdb28f596750f68fbb475ad051bdba32f5a1d2e6a750069aa81 7.38kB / 7.38kB                                     0.0s
 => => sha256:7247ea8d81e671d079d67f3a9909315ef4641b45db90d62a1b18e3430c1937d4 24.05MB / 24.05MB                                 638.7s
 => => sha256:609c73876867487da051ad470002217da69bb052e2538710ade0730d893ff51f 49.56MB / 49.56MB                                 967.6s
 => => sha256:be374d06f38273b62ddd7aa5bc3ce3f9c781fd49a1f5a5dd94a46d2986920d7a 64.14MB / 64.14MB                                1447.0s
 => => sha256:b4580645a8e50b87a19330da289a9b1540022379f2c99d3f0112e3c5c4a8d051 211.14MB / 211.14MB                              1988.9s
 => => sha256:dfc93b8f025cacb2b7fb13be1c7b87ff1cb61e46f01414022a5bded203a17ebd 3.37kB / 3.37kB                                   972.6s
 => => extracting sha256:609c73876867487da051ad470002217da69bb052e2538710ade0730d893ff51f                                         68.3s
 => => sha256:a67998ba05d7fa19701d42d143bd70271124be791568bdb03eedfbd7216f622f 49.72MB / 49.72MB                                1629.4s
 => => extracting sha256:7247ea8d81e671d079d67f3a9909315ef4641b45db90d62a1b18e3430c1937d4                                         15.4s
 => => sha256:9513f49617f6b0cb153128202311a5004f1069c3c86c78386abceab4827f9b79 2.23MB / 2.23MB                                  1497.8s
 => => extracting sha256:be374d06f38273b62ddd7aa5bc3ce3f9c781fd49a1f5a5dd94a46d2986920d7a                                         82.1s
 => => sha256:e2a102227dc65b99b43d5e0acfcda25a0c1c01ad9ec755fe764c6b87a13a61d0 452B / 452B                                      1500.4s
 => => extracting sha256:dfc93b8f025cacb2b7fb13be1c7b87ff1cb61e46f01414022a5bded203a17ebd                                          0.0s
 => => extracting sha256:a67998ba05d7fa19701d42d143bd70271124be791568bdb03eedfbd7216f622f                                         23.7s
 => => extracting sha256:9513f49617f6b0cb153128202311a5004f1069c3c86c78386abceab4827f9b79                                          0.6s
 => => extracting sha256:e2a102227dc65b99b43d5e0acfcda25a0c1c01ad9ec755fe764c6b87a13a61d0                                          0.0s
 => [internal] load build context                                                                                                  0.9s
 => => transferring context: 272.61kB                                                                                              0.4s
 => [2/4] WORKDIR /app/                                                                                                            2.3s
 => [4/4] RUN npm install                                                                                                         25.9s
 => exporting to image                                                                                                             0.8s
```



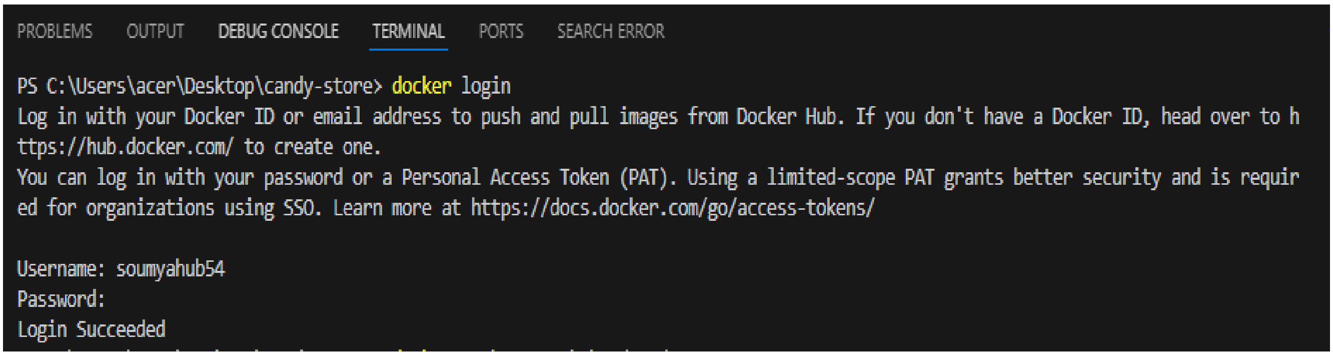
```
PS C:\Users\acer\Desktop\candy-store> docker images
REPOSITORY                       TAG       IMAGE ID       CREATED          SIZE
soumyahub54/candy-store          01        5fa55a0d9c2e   15 minutes ago   1.1GB
pythonproject                    latest    511a20bdcbb2   5 weeks ago      1.02GB
ubuntu                           latest    3db8720ecbf5   2 months ago     77.9MB
python                           latest    a3aef63c6c10   2 months ago     1.02GB
registry.k8s.io/coredns/coredns  v1.11.1   cbb01a7bd410   8 months ago     59.8MB
hello-world                      latest    d2c94e258dcb   11 months ago    13.3kB
```



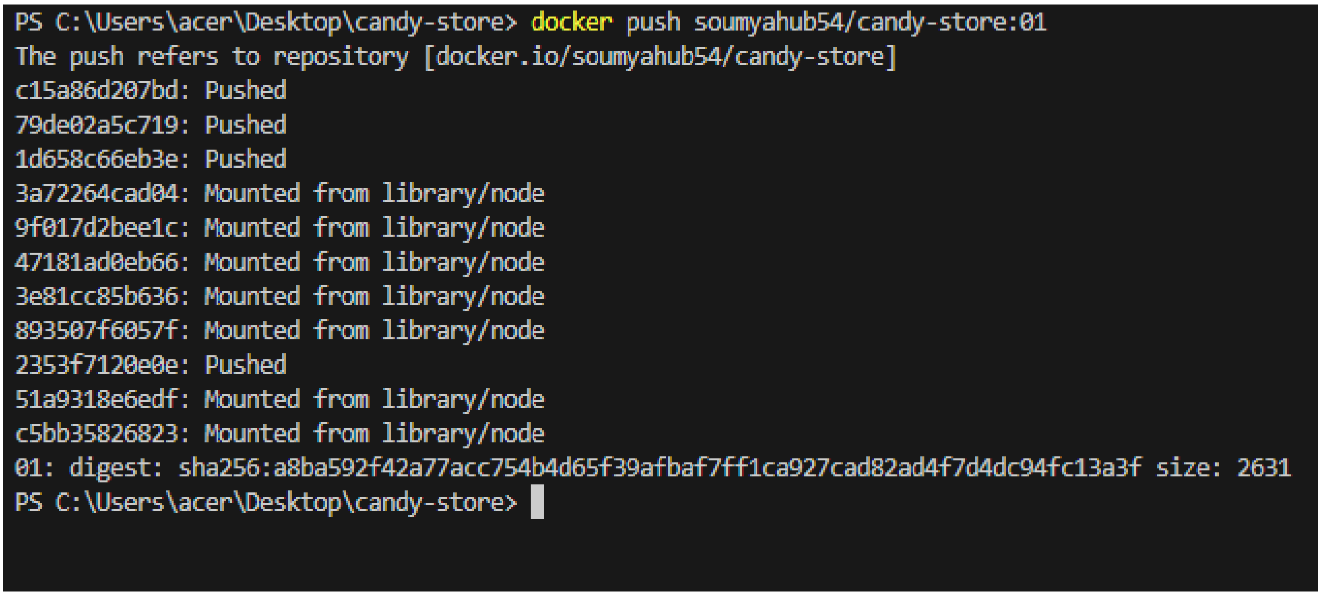
```
PS C:\Users\acer\Desktop\candy-store> docker images
REPOSITORY                       TAG       IMAGE ID       CREATED          SIZE
soumyahub54/candy-store          01        5fa55a0d9c2e   15 minutes ago   1.1GB
pythonproject                    latest    511a20bdcbb2   5 weeks ago      1.02GB
ubuntu                           latest    3db8720ecbf5   2 months ago     77.9MB
python                           latest    a3aef63c6c10   2 months ago     1.02GB
registry.k8s.io/coredns/coredns  v1.11.1   cbb01a7bd410   8 months ago     59.8MB
hello-world                      latest    d2c94e258dcb   11 months ago    13.3kB
PS C:\Users\acer\Desktop\candy-store> docker run --rm -p 3000:3000 soumyahub54/candy-store:01
Candy store app listening at http://localhost:3000
▌
```



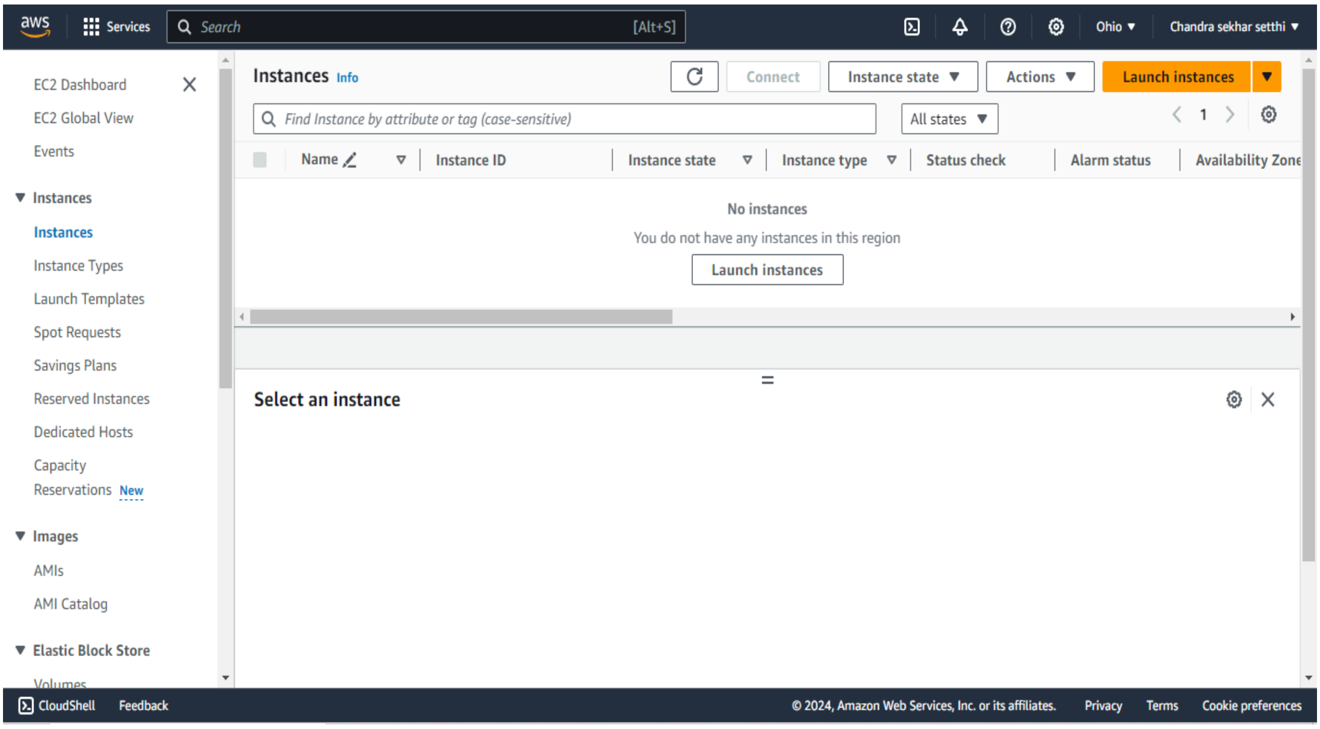
```
PS C:\Users\acer\Desktop\candy-store> docker ps
CONTAINER ID   IMAGE                             COMMAND                  CREATED         STATUS         PORTS                    NAMES
ac35bfe36441   soumyahub54/candy-store:01   "docker-entrypoint.s…"   6 minutes ago   Up 5 minutes   0.0.0.0:3000->3000/tcp   nice_
antonelli
PS C:\Users\acer\Desktop\candy-store> docker stop
"docker stop" requires at least 1 argument.
See 'docker stop --help'.

Usage:  docker stop [OPTIONS] CONTAINER [CONTAINER...]

Stop one or more running containers
PS C:\Users\acer\Desktop\candy-store> docker stop nice_antonelli
nice_antonelli
PS C:\Users\acer\Desktop\candy-store> 
```



```
PS C:\Users\acer\Desktop\candy-store> docker login
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to h
ttps://hub.docker.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is requir
ed for organizations using SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: soumyahub54
Password:
Login Succeeded
```



```
PS C:\Users\acer\Desktop\candy-store> docker push soumyahub54/candy-store:01
The push refers to repository [docker.io/soumyahub54/candy-store]
c15a86d207bd: Pushed
79de02a5c719: Pushed
1d658c66eb3e: Pushed
3a72264cad04: Mounted from library/node
9f017d2bee1c: Mounted from library/node
47181ad0eb66: Mounted from library/node
3e81cc85b636: Mounted from library/node
893507f6057f: Mounted from library/node
2353f7120e0e: Pushed
51a9318e6edf: Mounted from library/node
c5bb35826823: Mounted from library/node
01: digest: sha256:a8ba592f42a77acc754b4d65f39afbaf7ff1ca927cad82ad4f7d4dc94fc13a3f size: 2631
PS C:\Users\acer\Desktop\candy-store> 
```



Instances

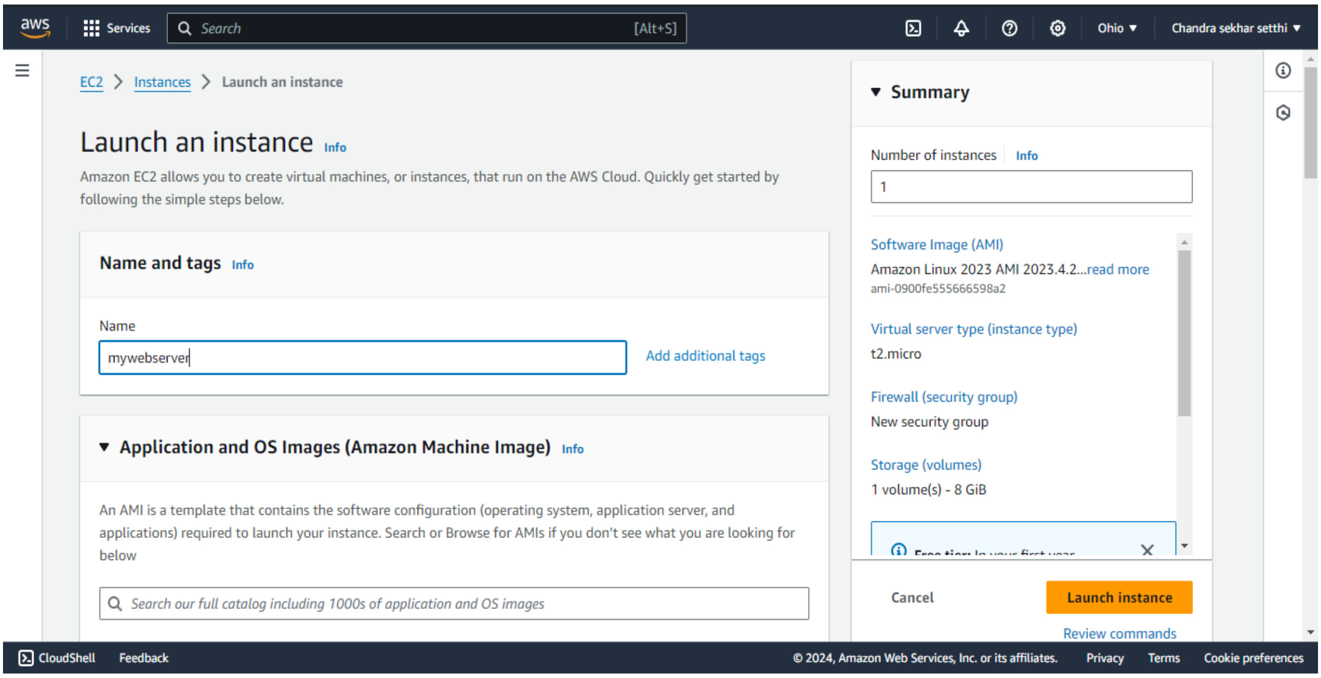
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone

No instances

You do not have any instances in this region

Select an instance

© 2024, Amazon Web Services, Inc. or its affiliates.



Launch an instance

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags

Name: mywebserver

Application and OS Images (Amazon Machine Image)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Summary

Number of instances: 1

Software Image (AMI)
Amazon Linux 2023 AMI 2023.4.2...read more
ami-0900fe555666598a2

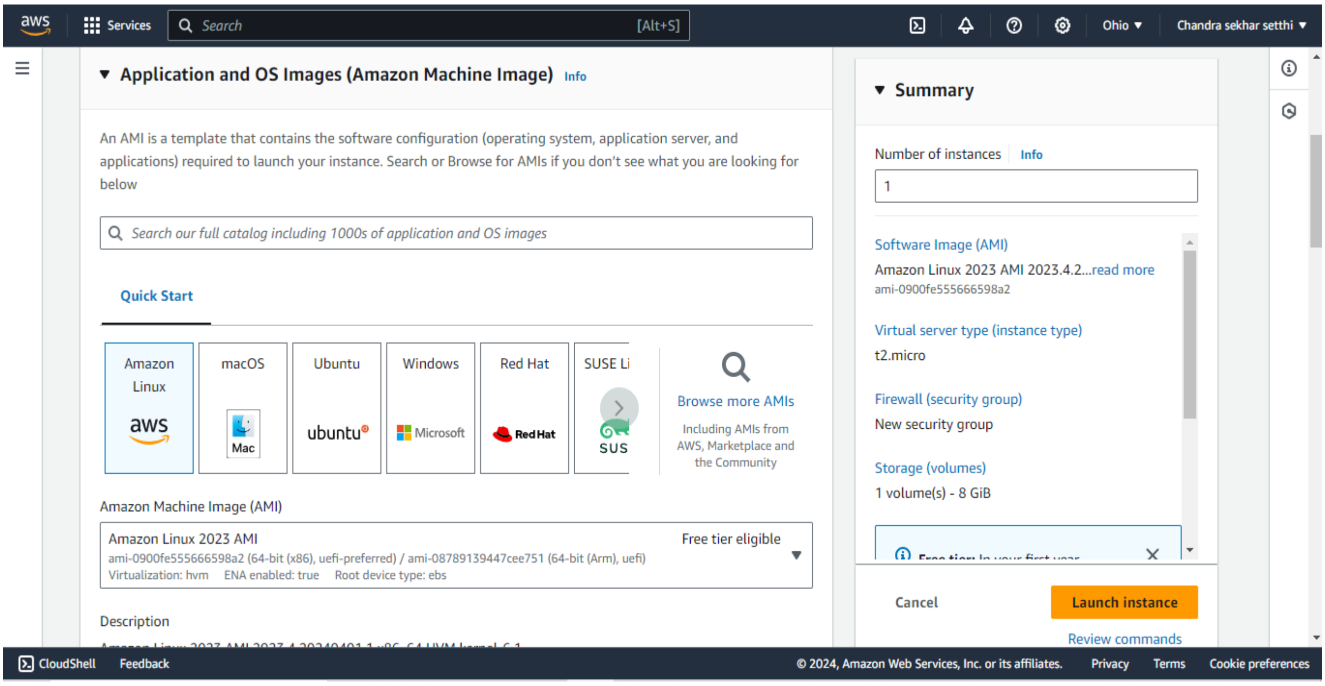
Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year

© 2024, Amazon Web Services, Inc. or its affiliates.



Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Amazon Linux | macOS | Ubuntu | Windows | Red Hat | SUSE Li

Amazon Machine Image (AMI)

Amazon Linux 2023 AMI — Free tier eligible
ami-0900fe555666598a2 (64-bit (x86), uefi-preferred) / ami-08789139447cee751 (64-bit (Arm), uefi)
Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Amazon Linux 2023 AMI 2023.4.20240401.1 x86_64 HVM kernel-6.1

Summary

Number of instances Info: 1

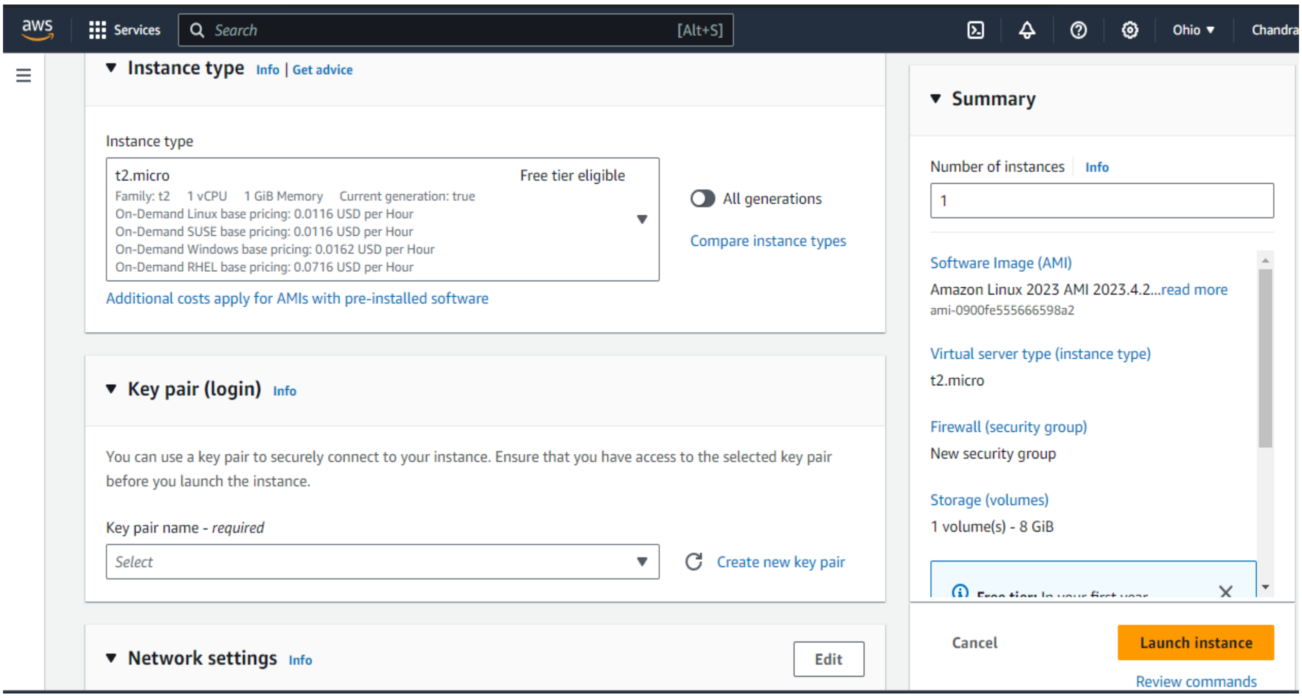
Software Image (AMI)
Amazon Linux 2023 AMI 2023.4.2...read more
ami-0900fe555666598a2

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year



Instance type Info | Get advice

Instance type

t2.micro — Free tier eligible
Family: t2 1 vCPU 1 GiB Memory Current generation: true
On-Demand Linux base pricing: 0.0116 USD per Hour
On-Demand SUSE base pricing: 0.0116 USD per Hour
On-Demand Windows base pricing: 0.0162 USD per Hour
On-Demand RHEL base pricing: 0.0716 USD per Hour

Additional costs apply for AMIs with pre-installed software

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*

Select

Network settings Info

Summary

Number of instances Info: 1

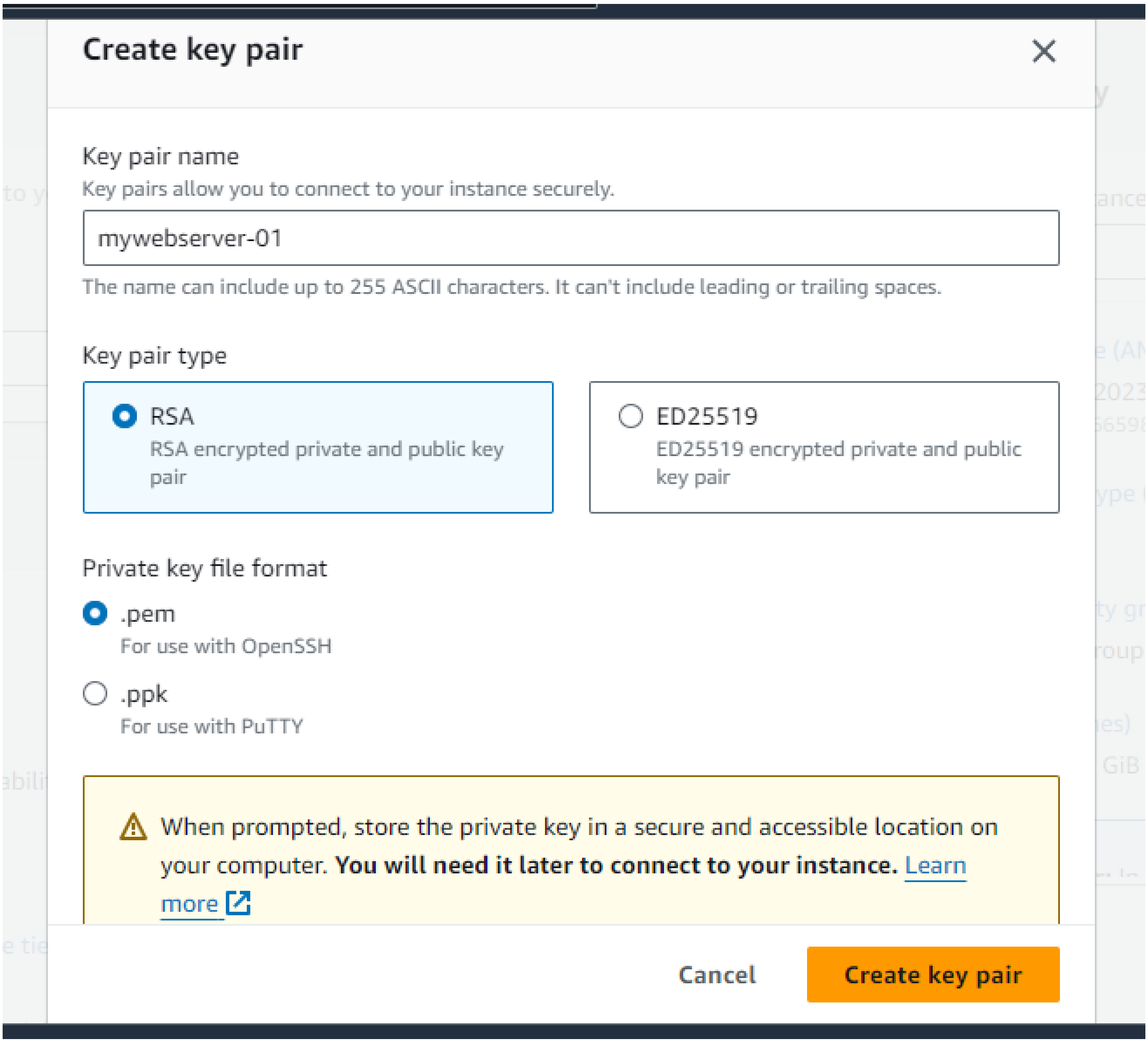
Software Image (AMI)
Amazon Linux 2023 AMI 2023.4.2...read more
ami-0900fe555666598a2

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year



Create key pair

Key pair name

Key pairs allow you to connect to your instance securely.

mywebserver-01

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

● RSA
RSA encrypted private and public key pair

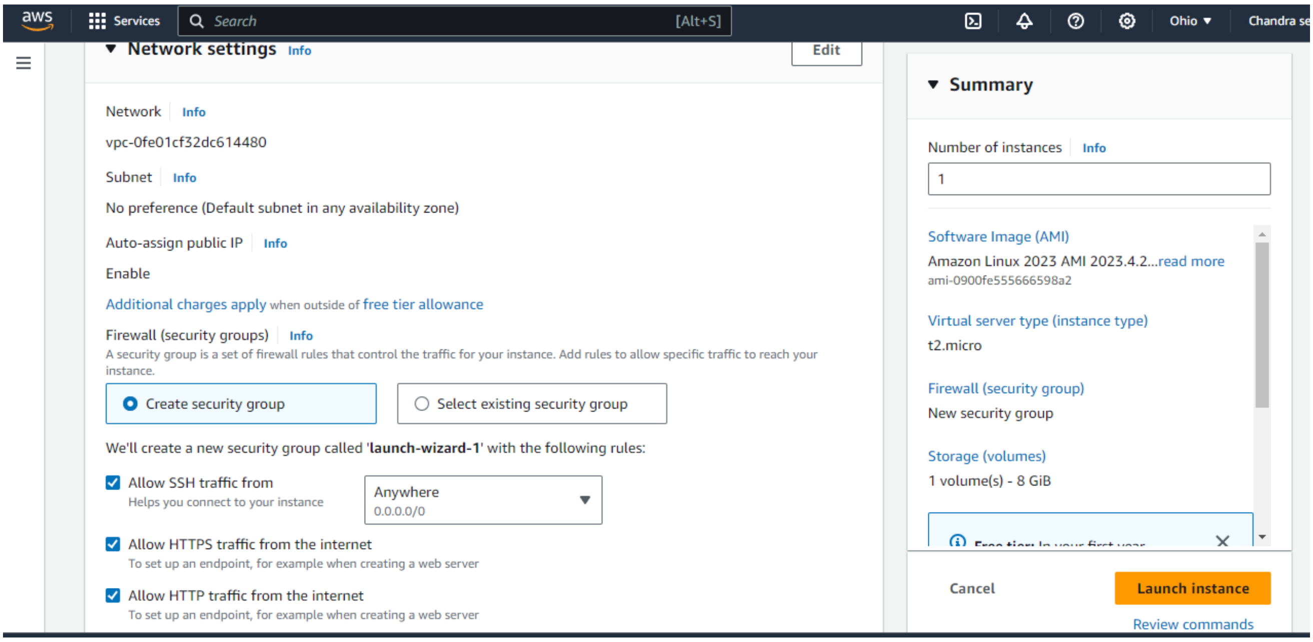
○ ED25519
ED25519 encrypted private and public key pair

Private key file format

● .pem
For use with OpenSSH

○ .ppk
For use with PuTTY

⚠ When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** Learn more



Network settings Info

Network Info
vpc-0fe01cf32dc614480

Subnet Info
No preference (Default subnet in any availability zone)

Auto-assign public IP Info
Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups) Info
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

● Create security group ○ Select existing security group

We'll create a new security group called '**launch-wizard-1**' with the following rules:

☑ Allow SSH traffic from — Anywhere 0.0.0.0/0
Helps you connect to your instance

☑ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

☑ Allow HTTP traffic from the internet
To set up an endpoint, for example when creating a web server

Summary

Number of instances Info
1

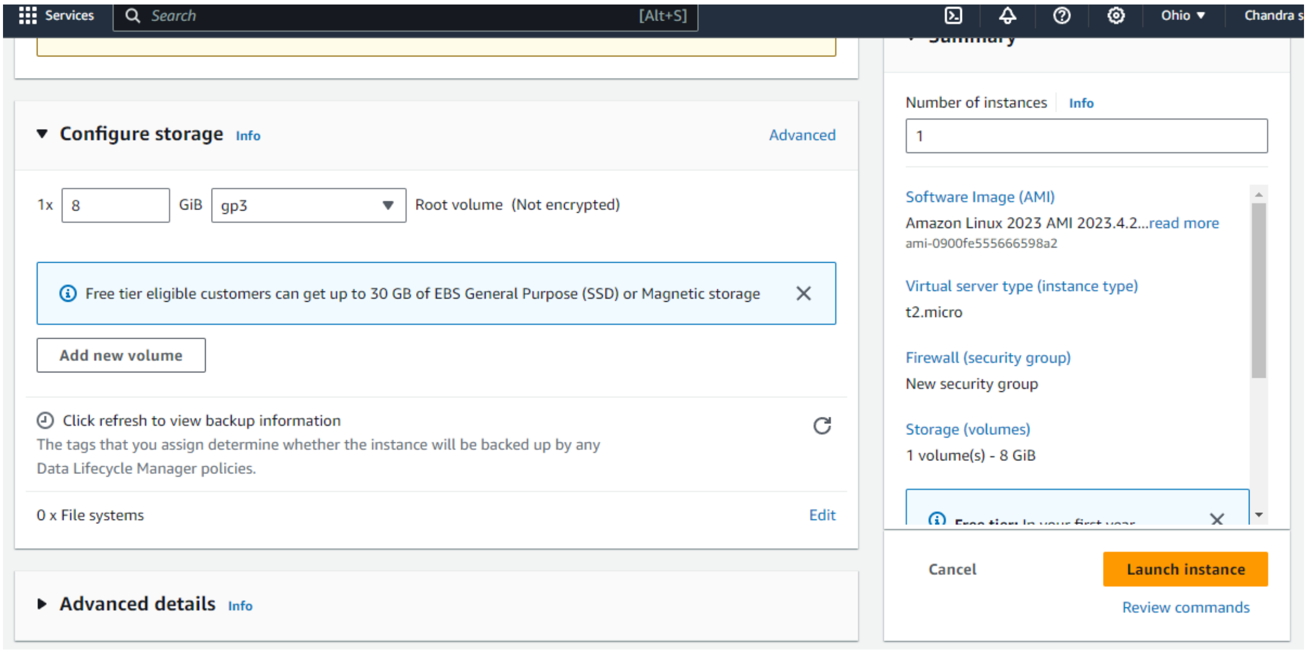
Software Image (AMI)
Amazon Linux 2023 AMI 2023.4.2...read more
ami-0900fe555666598a2

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

Free tier: In your first year



Configure storage Info

Advanced

1x 8 GiB gp3 Root volume (Not encrypted)

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

Click refresh to view backup information

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

0 x File systems

Advanced details Info

Summary

Number of instances Info

1

Software Image (AMI)

Amazon Linux 2023 AMI 2023.4.2...read more

ami-0900fe555666598a2

Virtual server type (instance type)

t2.micro

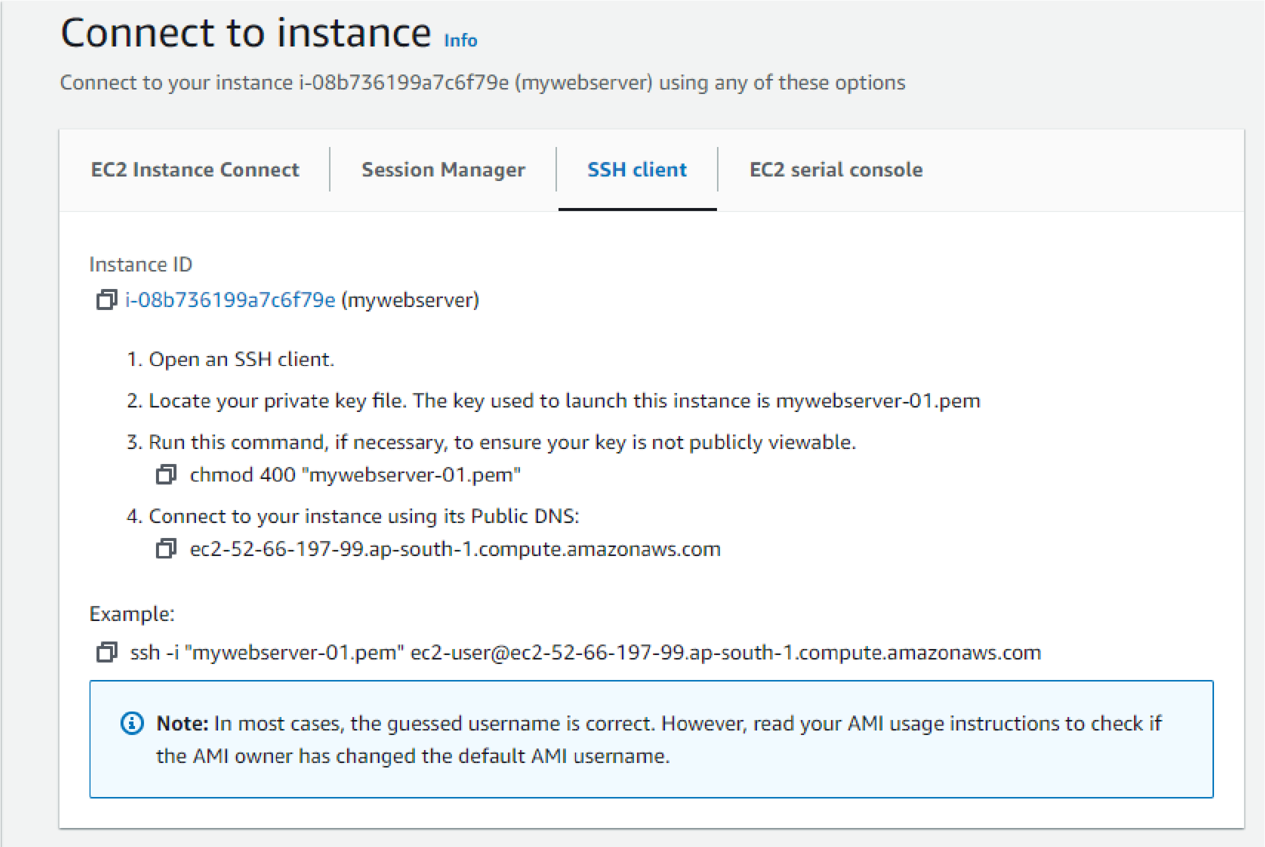
Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 8 GiB

Free tier: In your first year



Connect to instance Info

Connect to your instance i-08b736199a7c6f79e (mywebserver) using any of these options

Instance ID

i-08b736199a7c6f79e (mywebserver)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is mywebserver-01.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.

```
chmod 400 "mywebserver-01.pem"
```

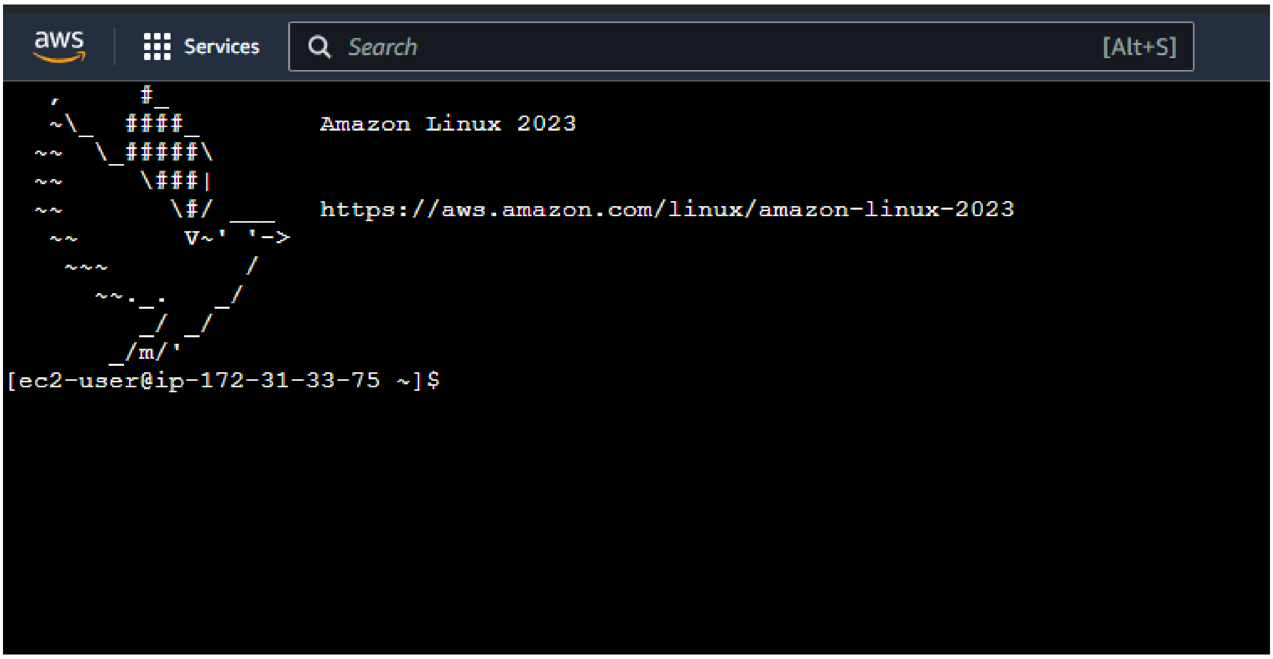
4. Connect to your instance using its Public DNS:

```
ec2-52-66-197-99.ap-south-1.compute.amazonaws.com
```

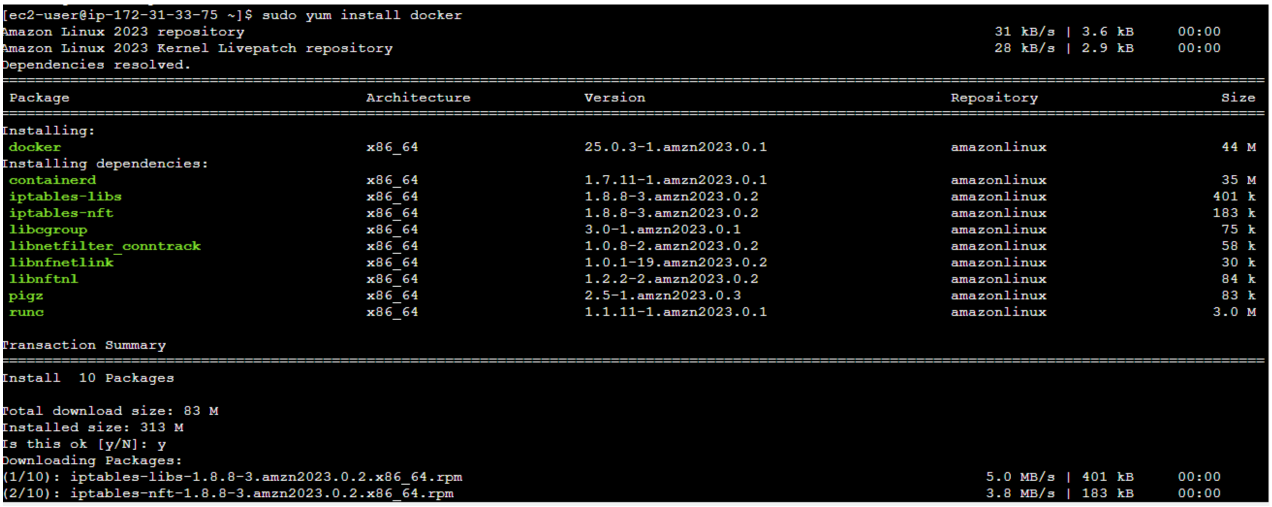
Example:

```
ssh -i "mywebserver-01.pem" ec2-user@ec2-52-66-197-99.ap-south-1.compute.amazonaws.com
```

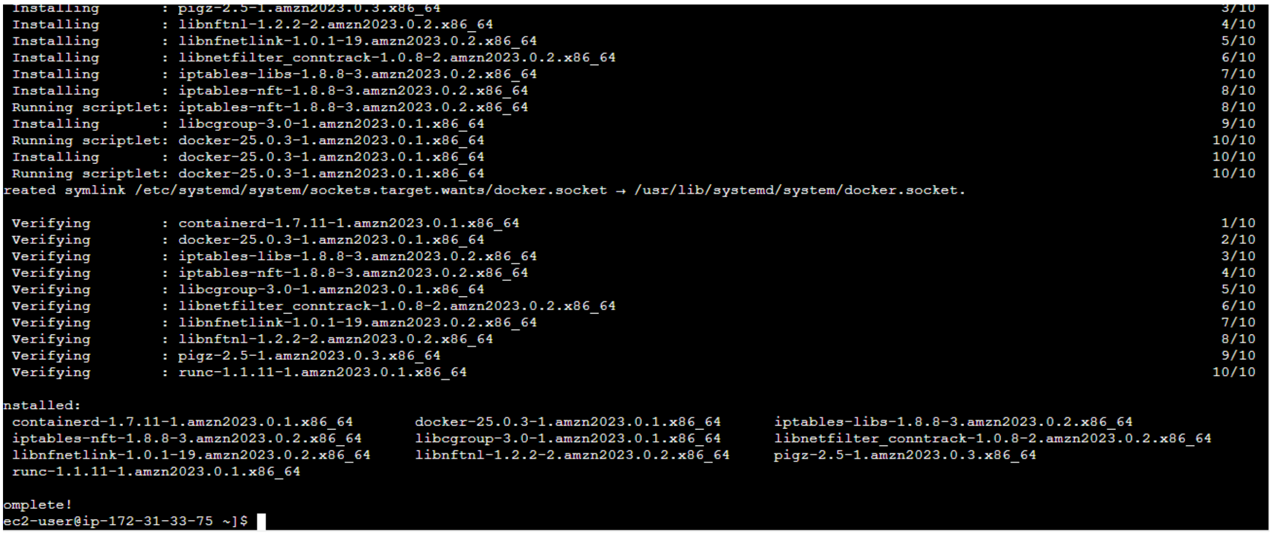
Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.



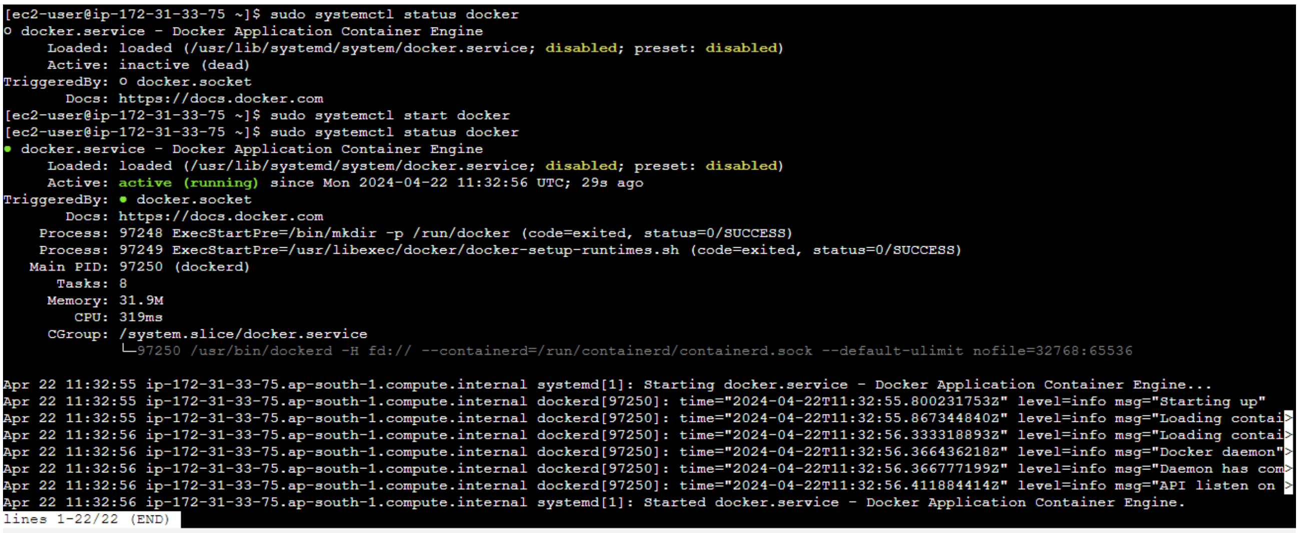
```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
[ec2-user@ip-172-31-33-75 ~]$
```



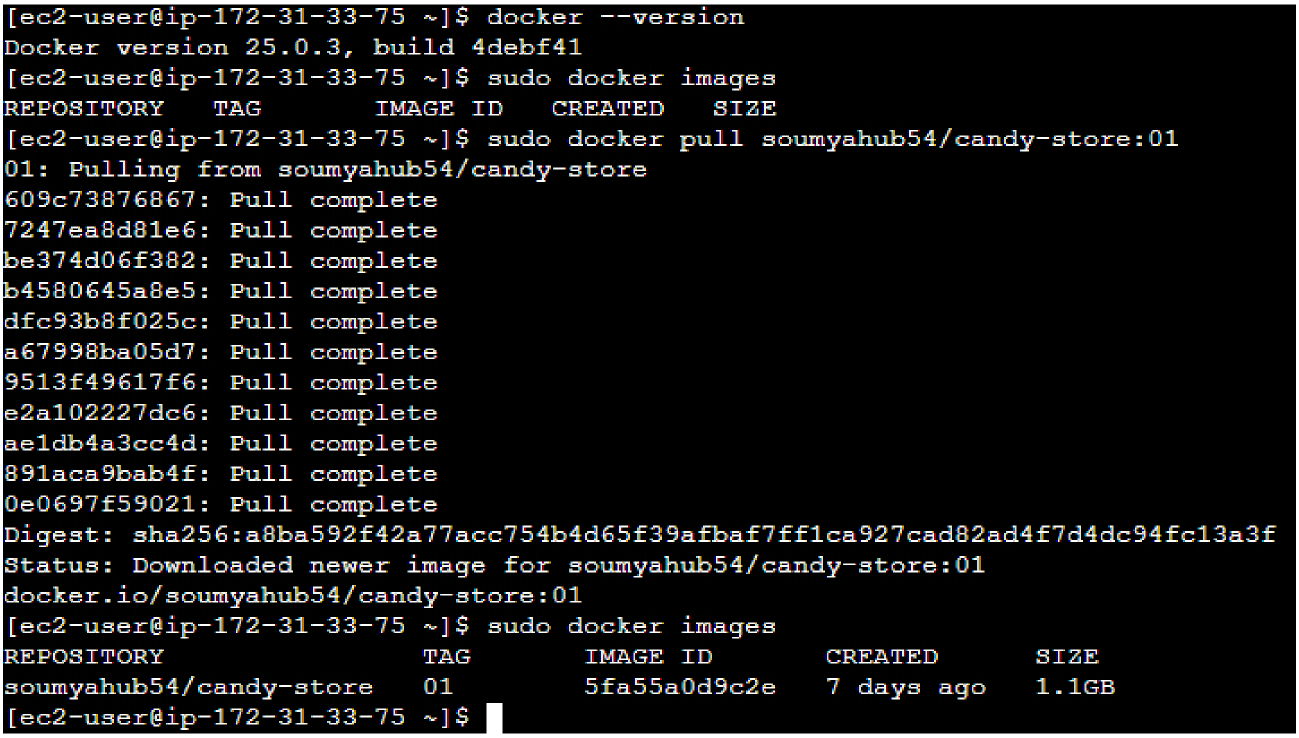
```
[ec2-user@ip-172-31-33-75 ~]$ sudo yum install docker
Amazon Linux 2023 repository                                                         31 kB/s | 3.6 kB     00:00
Amazon Linux 2023 Kernel Livepatch repository                                        28 kB/s | 2.9 kB     00:00
Dependencies resolved.
===================================================================================================================
 Package                        Architecture      Version                           Repository                Size
===================================================================================================================
Installing:
 docker                         x86_64            25.0.3-1.amzn2023.0.1             amazonlinux               44 M
Installing dependencies:
 containerd                     x86_64            1.7.11-1.amzn2023.0.1             amazonlinux               35 M
 iptables-libs                  x86_64            1.8.8-3.amzn2023.0.2              amazonlinux              401 k
 iptables-nft                   x86_64            1.8.8-3.amzn2023.0.2              amazonlinux              183 k
 libcgroup                      x86_64            3.0-1.amzn2023.0.1                amazonlinux               75 k
 libnetfilter_conntrack         x86_64            1.0.8-2.amzn2023.0.2              amazonlinux               58 k
 libnfnetlink                   x86_64            1.0.1-19.amzn2023.0.2             amazonlinux               30 k
 libnftnl                       x86_64            1.2.2-2.amzn2023.0.2              amazonlinux               84 k
 pigz                           x86_64            2.5-1.amzn2023.0.3                amazonlinux               83 k
 runc                           x86_64            1.1.11-1.amzn2023.0.1             amazonlinux              3.0 M

Transaction Summary
===================================================================================================================
Install  10 Packages

Total download size: 83 M
Installed size: 313 M
Is this ok [y/N]: y
Downloading Packages:
(1/10): iptables-libs-1.8.8-3.amzn2023.0.2.x86_64.rpm                               5.0 MB/s | 401 kB     00:00
(2/10): iptables-nft-1.8.8-3.amzn2023.0.2.x86_64.rpm                                3.8 MB/s | 183 kB     00:00
```



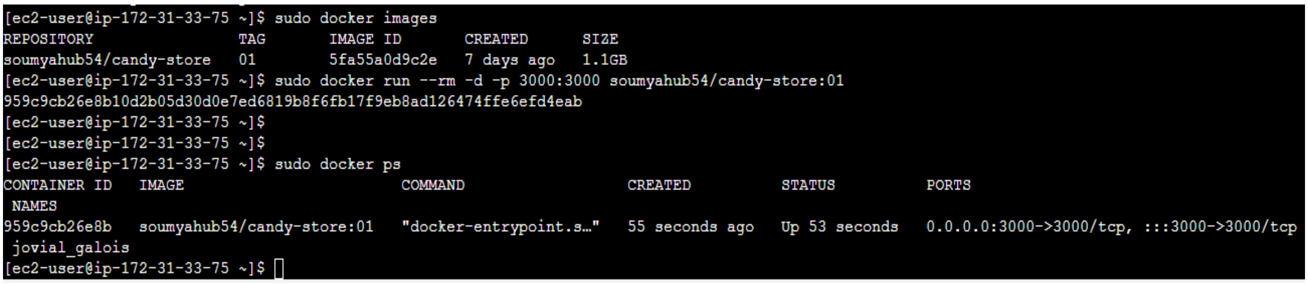
```
 Installing       : pigz-2.5-1.amzn2023.0.3.x86_64                                                           3/10
 Installing       : libnftnl-1.2.2-2.amzn2023.0.2.x86_64                                                     4/10
 Installing       : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64                                                5/10
 Installing       : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64                                       6/10
 Installing       : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64                                                7/10
 Installing       : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64                                                 8/10
 Running scriptlet: iptables-nft-1.8.8-3.amzn2023.0.2.x86_64                                                 8/10
 Installing       : libcgroup-3.0-1.amzn2023.0.1.x86_64                                                      9/10
 Running scriptlet: docker-25.0.3-1.amzn2023.0.1.x86_64                                                     10/10
 Installing       : docker-25.0.3-1.amzn2023.0.1.x86_64                                                     10/10
 Running scriptlet: docker-25.0.3-1.amzn2023.0.1.x86_64                                                     10/10
reated symlink /etc/systemd/system/sockets.target.wants/docker.socket → /usr/lib/systemd/system/docker.socket.

 Verifying        : containerd-1.7.11-1.amzn2023.0.1.x86_64                                                  1/10
 Verifying        : docker-25.0.3-1.amzn2023.0.1.x86_64                                                      2/10
 Verifying        : iptables-libs-1.8.8-3.amzn2023.0.2.x86_64                                                3/10
 Verifying        : iptables-nft-1.8.8-3.amzn2023.0.2.x86_64                                                 4/10
 Verifying        : libcgroup-3.0-1.amzn2023.0.1.x86_64                                                      5/10
 Verifying        : libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64                                       6/10
 Verifying        : libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64                                                7/10
 Verifying        : libnftnl-1.2.2-2.amzn2023.0.2.x86_64                                                     8/10
 Verifying        : pigz-2.5-1.amzn2023.0.3.x86_64                                                           9/10
 Verifying        : runc-1.1.11-1.amzn2023.0.1.x86_64                                                       10/10

nstalled:
 containerd-1.7.11-1.amzn2023.0.1.x86_64        docker-25.0.3-1.amzn2023.0.1.x86_64        iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
 iptables-nft-1.8.8-3.amzn2023.0.2.x86_64       libcgroup-3.0-1.amzn2023.0.1.x86_64        libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
 libnfnetlink-1.0.1-19.amzn2023.0.2.x86_64      libnftnl-1.2.2-2.amzn2023.0.2.x86_64       pigz-2.5-1.amzn2023.0.3.x86_64
 runc-1.1.11-1.amzn2023.0.1.x86_64

omplete!
ec2-user@ip-172-31-33-75 ~]$
```



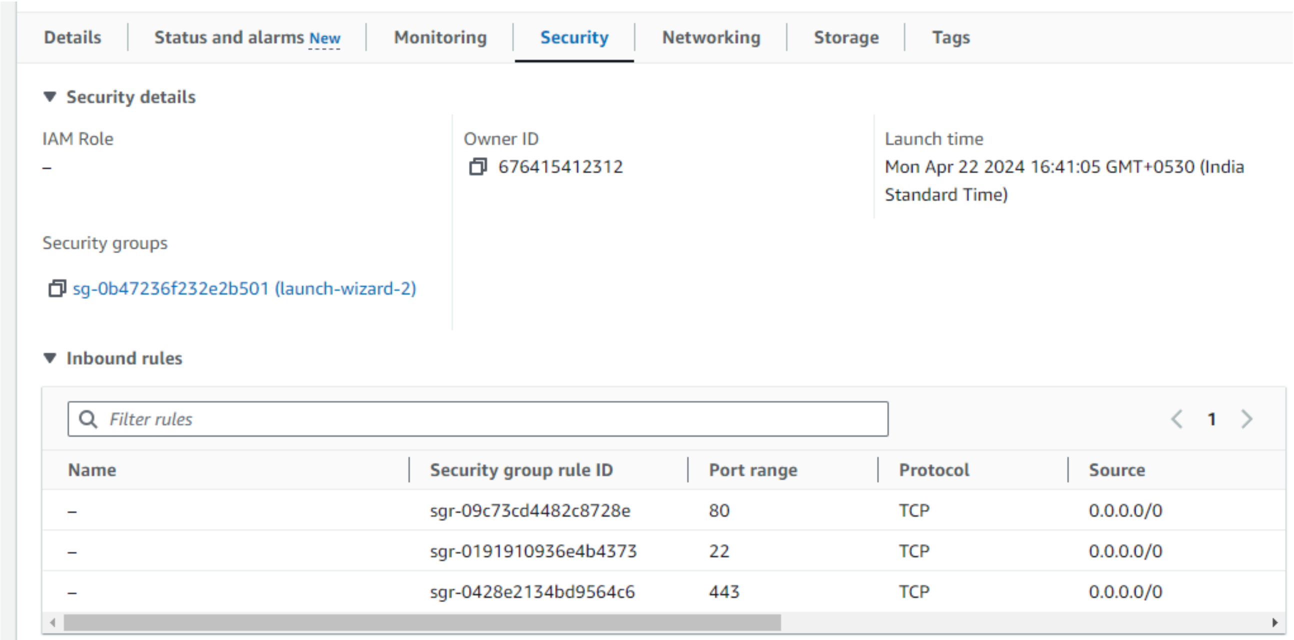
```
[ec2-user@ip-172-31-33-75 ~]$ sudo systemctl status docker
○ docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; preset: disabled)
     Active: inactive (dead)
TriggeredBy: ○ docker.socket
       Docs: https://docs.docker.com
[ec2-user@ip-172-31-33-75 ~]$ sudo systemctl start docker
[ec2-user@ip-172-31-33-75 ~]$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; preset: disabled)
     Active: active (running) since Mon 2024-04-22 11:32:56 UTC; 29s ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
    Process: 97248 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
    Process: 97249 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
   Main PID: 97250 (dockerd)
      Tasks: 8
     Memory: 31.9M
        CPU: 319ms
     CGroup: /system.slice/docker.service
             └─97250 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

Apr 22 11:32:55 ip-172-31-33-75.ap-south-1.compute.internal systemd[1]: Starting docker.service - Docker Application Container Engine...
Apr 22 11:32:55 ip-172-31-33-75.ap-south-1.compute.internal dockerd[97250]: time="2024-04-22T11:32:55.800231753Z" level=info msg="Starting up"
Apr 22 11:32:55 ip-172-31-33-75.ap-south-1.compute.internal dockerd[97250]: time="2024-04-22T11:32:55.867344840Z" level=info msg="Loading contai>
Apr 22 11:32:56 ip-172-31-33-75.ap-south-1.compute.internal dockerd[97250]: time="2024-04-22T11:32:56.333318893Z" level=info msg="Loading contai>
Apr 22 11:32:56 ip-172-31-33-75.ap-south-1.compute.internal dockerd[97250]: time="2024-04-22T11:32:56.366436218Z" level=info msg="Docker daemon">
Apr 22 11:32:56 ip-172-31-33-75.ap-south-1.compute.internal dockerd[97250]: time="2024-04-22T11:32:56.366777199Z" level=info msg="Daemon has com>
Apr 22 11:32:56 ip-172-31-33-75.ap-south-1.compute.internal dockerd[97250]: time="2024-04-22T11:32:56.411884414Z" level=info msg="API listen on >
Apr 22 11:32:56 ip-172-31-33-75.ap-south-1.compute.internal systemd[1]: Started docker.service - Docker Application Container Engine.
lines 1-22/22 (END)
```



```
[ec2-user@ip-172-31-33-75 ~]$ docker --version
Docker version 25.0.3, build 4debf41
[ec2-user@ip-172-31-33-75 ~]$ sudo docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
[ec2-user@ip-172-31-33-75 ~]$ sudo docker pull soumyahub54/candy-store:01
01: Pulling from soumyahub54/candy-store
609c73876867: Pull complete
7247ea8d81e6: Pull complete
be374d06f382: Pull complete
b4580645a8e5: Pull complete
dfc93b8f025c: Pull complete
a67998ba05d7: Pull complete
9513f49617f6: Pull complete
e2a102227dc6: Pull complete
ae1db4a3cc4d: Pull complete
891aca9bab4f: Pull complete
0e0697f59021: Pull complete
Digest: sha256:a8ba592f42a77acc754b4d65f39afbaf7ff1ca927cad82ad4f7d4dc94fc13a3f
Status: Downloaded newer image for soumyahub54/candy-store:01
docker.io/soumyahub54/candy-store:01
[ec2-user@ip-172-31-33-75 ~]$ sudo docker images
REPOSITORY                TAG       IMAGE ID       CREATED      SIZE
soumyahub54/candy-store   01        5fa55a0d9c2e   7 days ago   1.1GB
[ec2-user@ip-172-31-33-75 ~]$
```



```
[ec2-user@ip-172-31-33-75 ~]$ sudo docker images
REPOSITORY                TAG       IMAGE ID       CREATED      SIZE
soumyahub54/candy-store   01        5fa55a0d9c2e   7 days ago   1.1GB
[ec2-user@ip-172-31-33-75 ~]$ sudo docker run --rm -d -p 3000:3000 soumyahub54/candy-store:01
959c9cb26e8b10d2b05d30d0e7ed6819b8f6fb17f9eb8ad126474ffe6efd4eab
[ec2-user@ip-172-31-33-75 ~]$
[ec2-user@ip-172-31-33-75 ~]$
[ec2-user@ip-172-31-33-75 ~]$ sudo docker ps
CONTAINER ID   IMAGE                        COMMAND                    CREATED          STATUS          PORTS
 NAMES
959c9cb26e8b   soumyahub54/candy-store:01   "docker-entrypoint.s…"   55 seconds ago   Up 53 seconds   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
 jovial_galois
[ec2-user@ip-172-31-33-75 ~]$
```



▼ Security details

IAM Role
–

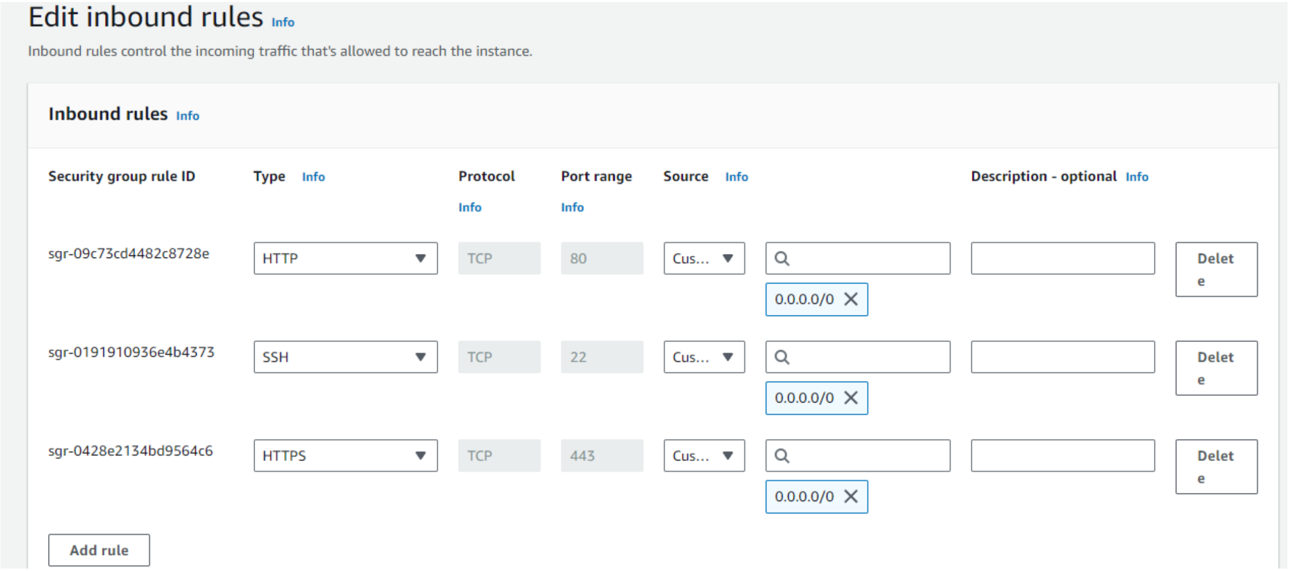
Owner ID
676415412312

Launch time
Mon Apr 22 2024 16:41:05 GMT+0530 (India Standard Time)

Security groups
sg-0b47236f232e2b501 (launch-wizard-2)

▼ Inbound rules

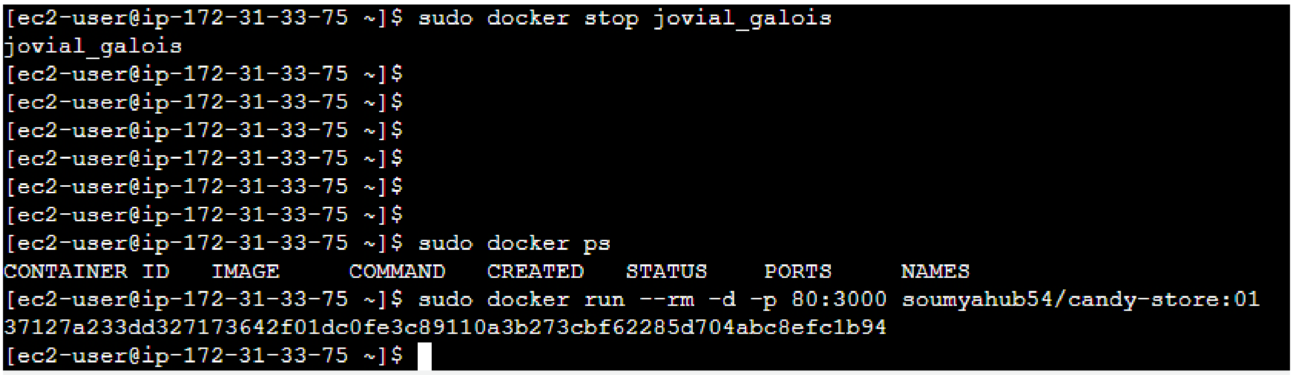
Name	Security group rule ID	Port range	Protocol	Source
–	sgr-09c73cd4482c8728e	80	TCP	0.0.0.0/0
–	sgr-0191910936e4b4373	22	TCP	0.0.0.0/0
–	sgr-0428e2134bd9564c6	443	TCP	0.0.0.0/0

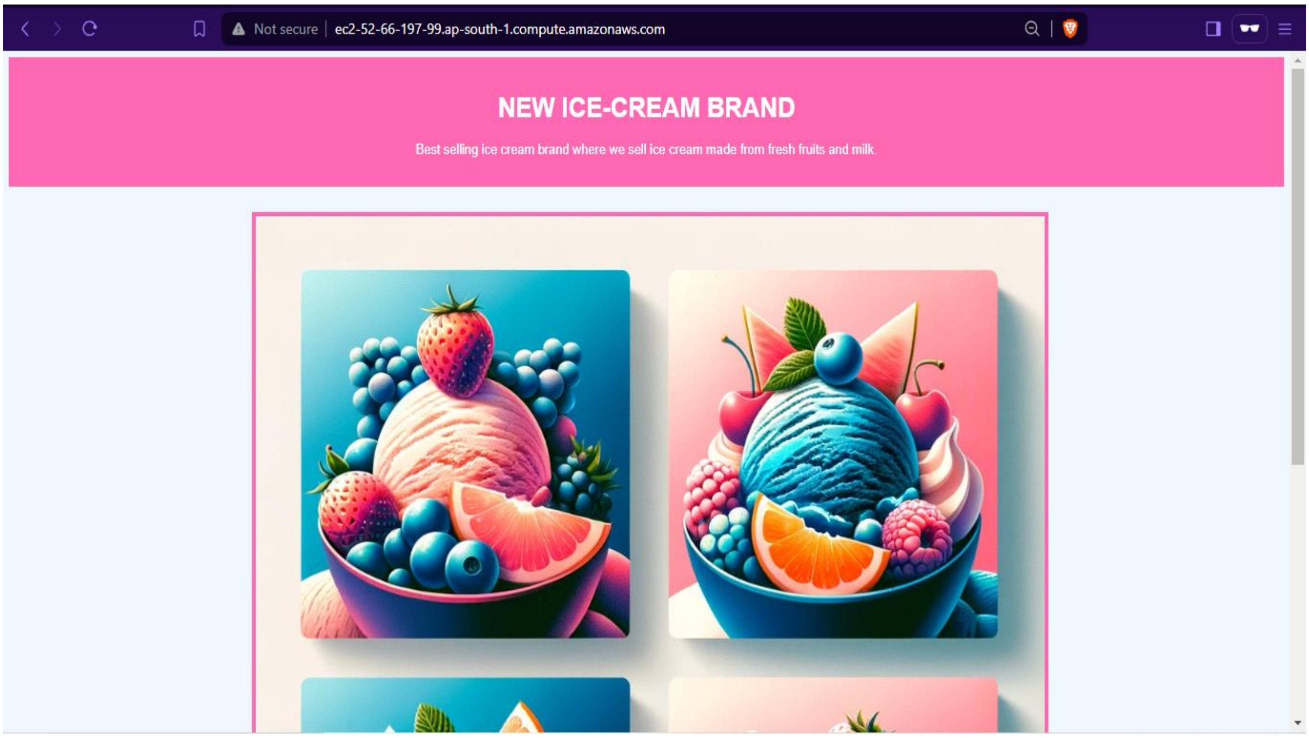

Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules Info

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info		Description - optional Info	
sgr-09c73cd4482c8728e	HTTP ▼	TCP	80	Cus... ▼	0.0.0.0/0 ✕		Delete
sgr-0191910936e4b4373	SSH ▼	TCP	22	Cus... ▼	0.0.0.0/0 ✕		Delete
sgr-0428e2134bd9564c6	HTTPS ▼	TCP	443	Cus... ▼	0.0.0.0/0 ✕		Delete

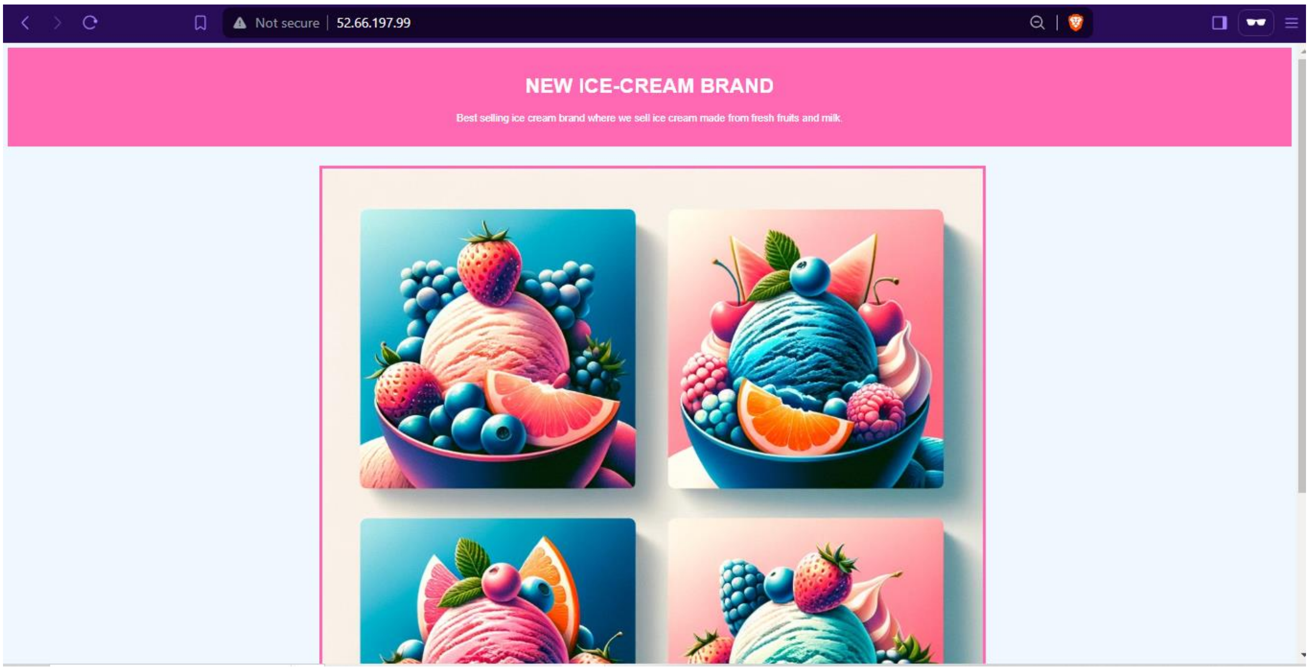

```
[ec2-user@ip-172-31-33-75 ~]$ sudo docker stop jovial_galois
jovial_galois
[ec2-user@ip-172-31-33-75 ~]$
[ec2-user@ip-172-31-33-75 ~]$
[ec2-user@ip-172-31-33-75 ~]$
[ec2-user@ip-172-31-33-75 ~]$
[ec2-user@ip-172-31-33-75 ~]$
[ec2-user@ip-172-31-33-75 ~]$
[ec2-user@ip-172-31-33-75 ~]$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
[ec2-user@ip-172-31-33-75 ~]$ sudo docker run --rm -d -p 80:3000 soumyahub54/candy-store:01
37127a233dd327173642f01dc0fe3c89110a3b273cbf62285d704abc8efc1b94
[ec2-user@ip-172-31-33-75 ~]$
```



ec2-52-66-197-99.ap-south-1.compute.amazonaws.com

NEW ICE-CREAM BRAND

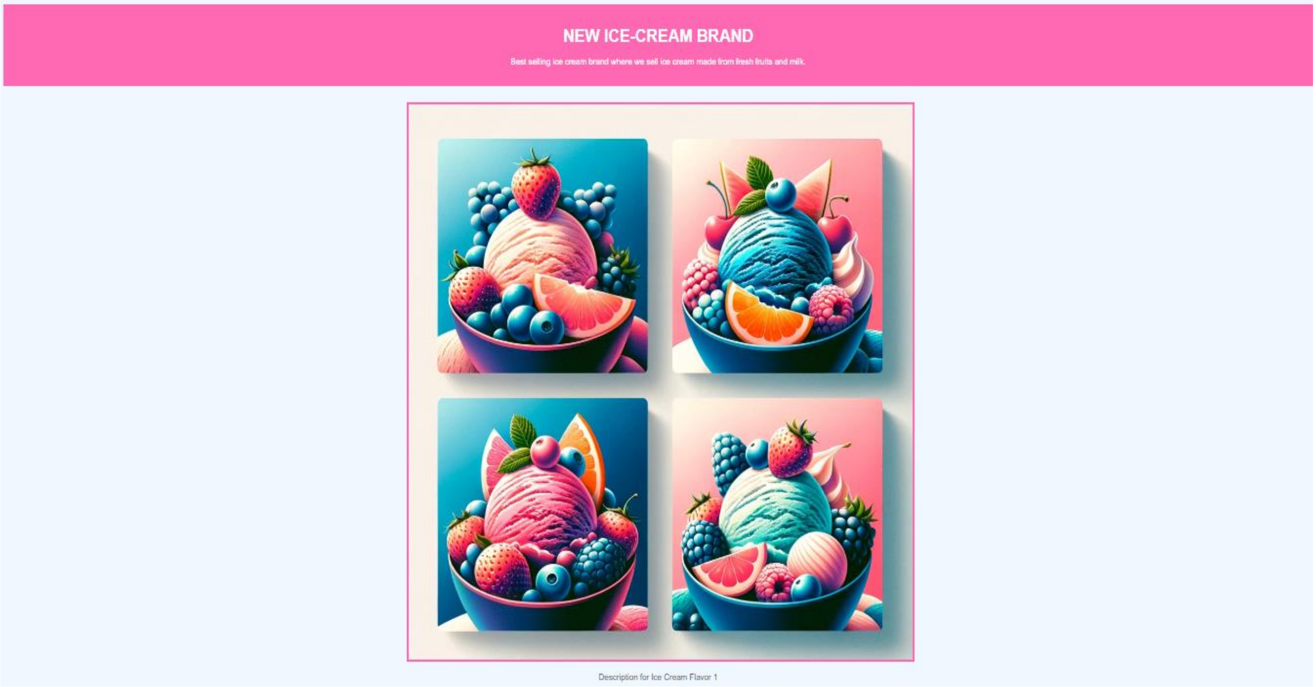
Best selling ice cream brand where we sell ice cream made from fresh fruits and milk.



52.66.197.99

NEW ICE-CREAM BRAND

Best selling ice cream brand where we sell ice cream made from fresh fruits and milk.



NEW ICE-CREAM BRAND

Best selling ice cream brand where we sell ice cream made from fresh fruits and milk.

Description for Ice Cream Flavor 1

CHAPTER 8

Conclusion

In conclusion, deploying Docker containers on Amazon Web Services (AWS) Elastic Compute Cloud (EC2) offers a powerful and flexible solution for hosting a wide range of applications. Throughout this process, we've explored the steps involved in provisioning EC2 instances, installing Docker, building Docker images, and running Docker containers to host applications.

By leveraging Docker's containerization technology on AWS EC2, organizations can achieve several key benefits:

1. Flexibility and Portability: Docker containers provide a consistent runtime environment across different platforms, enabling easy migration and deployment of applications.

2. Scalability: AWS EC2 offers scalable compute capacity, allowing organizations to dynamically adjust resources based on application demand. Docker containers further enhance scalability by enabling efficient resource utilization and horizontal scaling.

3. Efficiency: Docker's lightweight nature and efficient resource utilization minimize overhead and improve application performance on EC2 instances.

4. Isolation: Docker containers provide process isolation, ensuring that applications run independently without interference from other processes or dependencies.

5. Security: AWS provides robust security features, and Docker containers offer additional layers of isolation and security, such as sandboxing and resource constraints.

6. Cost-effectiveness: By optimizing resource utilization and scaling infrastructure based on demand, organizations can achieve cost savings compared to traditional deployment methods.

Overall, Docker deployment on AWS EC2 combines the scalability, reliability, and security of AWS infrastructure with the flexibility, efficiency, and portability of Docker containers. This approach empowers organizations to deploy, manage, and scale applications more effectively, ultimately enhancing agility, efficiency, and competitiveness in today's rapidly evolving digital landscape.

ASSESSMENT

Internal:

SL NO	**RUBRICS**	**FULL MARK**	**MARKS OBTAINED**	**REMARKS**
1	Understanding the relevance, scope and dimension of the project	10		
2	Methodology	10		
3	Quality of Analysis and Results	10		
4	Interpretations and Conclusions	10		
5	Report	10		
	Total	**50**		

Date: **Signature of the Faculty**

COURSE OUTCOME (COs) ATTAINMENT

- **Expected Course Outcomes (COs):**

  (Refer to COs Statement in the Syllabus)

- **Course Outcome Attained:**

  How would you rate your learning of the subject based on the specified COs?

□	□	□	□	□	□	□	□	□	□
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**

  LOW **HIGH**

- **Learning Gap (if any):**

- **Books / Manuals Referred:**

Date: **Signature of the Student**

- **Suggestions / Recommendations:**

  (By the Course Faculty)

Date: **Signature of the Faculty**