# Docker

Step 1: Create the index.html File

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Welcome to Nginx</title>

    <style>

        body {

            font-family: Arial, sans-serif;

            text-align: center;

            margin: 0;

            padding: 0;

            background-color: #f4f4f4;

        }

        h1 {

            color: #333;

        }

    </style>

</head>

<body>

    <h1>Welcome to Devops Final Exams!</h1>

    <p>This Ayushi Dhamane Roll no: A012.</p>

</body>

</html>

Step 2: Create the Dockerfile

# Use the official Nginx image as the base image

FROM nginx:latest

# Copy the index.html to the default Nginx HTML directory

COPY index.html /usr/share/nginx/html/index.html

# Expose port 80

EXPOSE 80

# Start Nginx server

CMD ["nginx", "-g", "daemon off;"]

**Step 3: Build and Run the Docker Container**

1. Save both files (index.html and Dockerfile) in the same directory.
2. Open a terminal in that directory.
3. Build the Docker image:

bash

Copy code

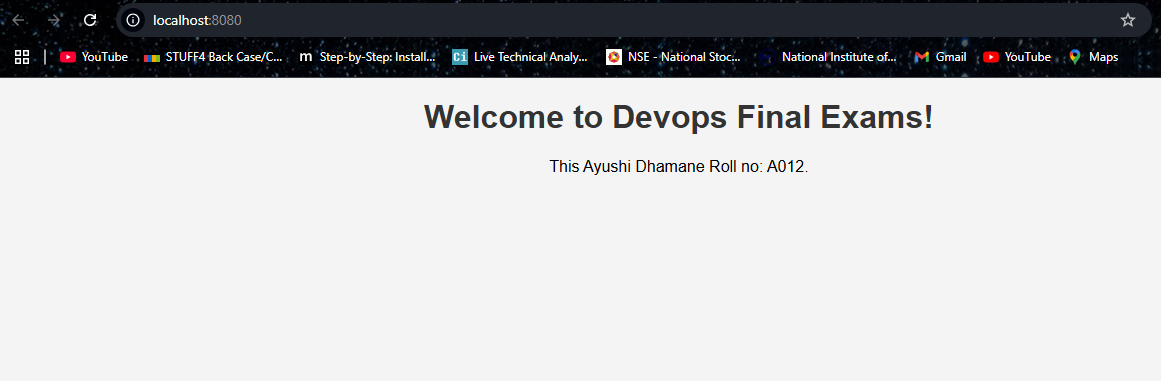
docker build -t my-nginx .

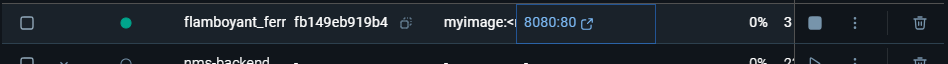
1. Run the container:

bash

Copy code

docker run -d -p 8080:80 my-nginx





# Kubernetes

## Using docker

**1. Build and Push the Custom Nginx Image**

First, create your custom Nginx image.

**Dockerfile**

dockerfile

Copy code

# Use the official Nginx image as the base image

FROM nginx:latest

# Copy a custom HTML file to the default Nginx directory

COPY index.html /usr/share/nginx/html/index.html

# Expose port 80

EXPOSE 80

**index.html**

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Custom Nginx</title>

</head>

<body>

<h1>Hello from Custom Nginx!</h1>

</body>

</html>

**Build and Push the Image**

1. Build the Docker image:

bash

Copy code

docker build -t <your-dockerhub-username>/custom-nginx:latest .

1. Log in to Docker Hub:

bash

Copy code

docker login

1. Push the image:

bash

Copy code

docker push <your-dockerhub-username>/custom-nginx:latest

**2. Create a Kubernetes Deployment**

**deployment.yaml**

1. apiVersion: apps/v1
2. kind: Deployment
3. metadata:
4. name: custom-nginx-deployment
5. spec:
6. replicas: 2 # Start with 2 replicas
7. selector:
8. matchLabels:
9. app: custom-nginx
10. template:
11. metadata:
12. labels:
13. app: custom-nginx
14. spec:
15. containers:
16. - name: custom-nginx
17. image: ayushiii0308/custom-nginx:latest # Use your custom image
18. ports:
19. - containerPort: 80

Service.yaml

apiVersion: v1

kind: Service

metadata:

  name: nginx-service2

spec:

  selector:

    app: nginx

  ports:

    - protocol: TCP

      port: 80  # Exposing port 80

      targetPort: 80  # Inside the container

  type: NodePort    # Or you can use NodePort if LoadBalancer isn't available

1. Apply the deployment:

bash

Copy code

kubectl apply -f deployment.yaml

1. Verify that the pods are running:

bash

Copy code

kubectl get pods

**3. Scale the Deployment**

**Scale to 5 replicas:**

bash

Copy code

kubectl scale deployment custom-nginx-deployment --replicas=5

**Verify the scaling:**

bash

Copy code

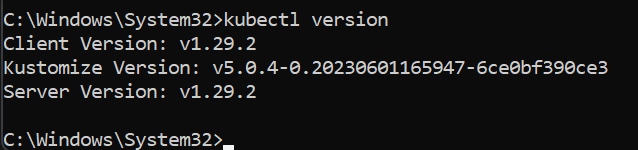
kubectl get deployment custom-nginx-deployment

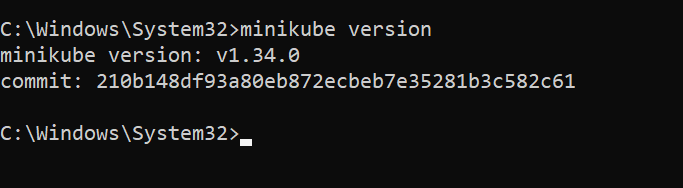
kubectl get pods -l app=custom-nginx

## Using minikube

1. Install Kubectl and Minikube along with Docker Desktop. Check the installation: kubectl version

minikube version

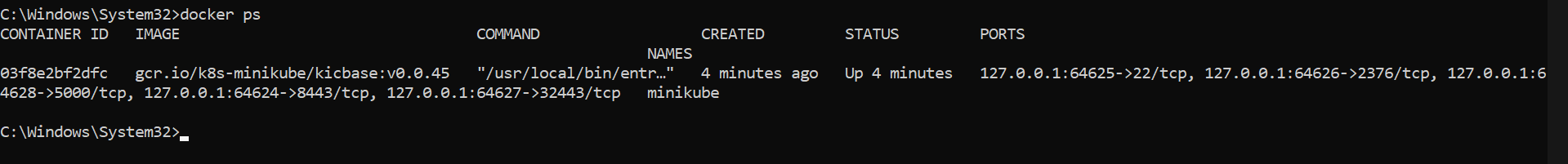




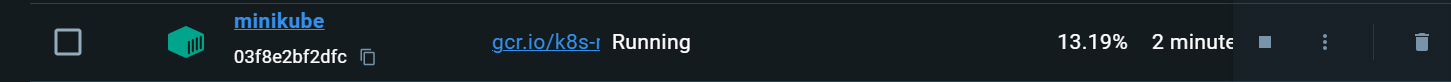
3. Minikube needs to be started with a driver. We use docker driver: minikube start --driver=docker



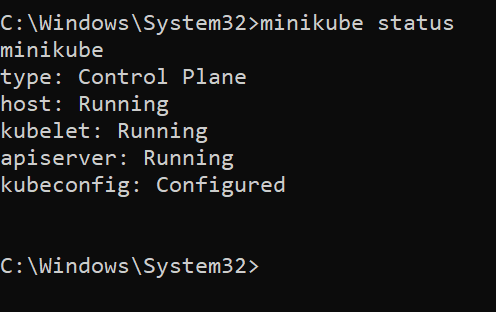
4. Check with docker ps to see if the minikube container is running



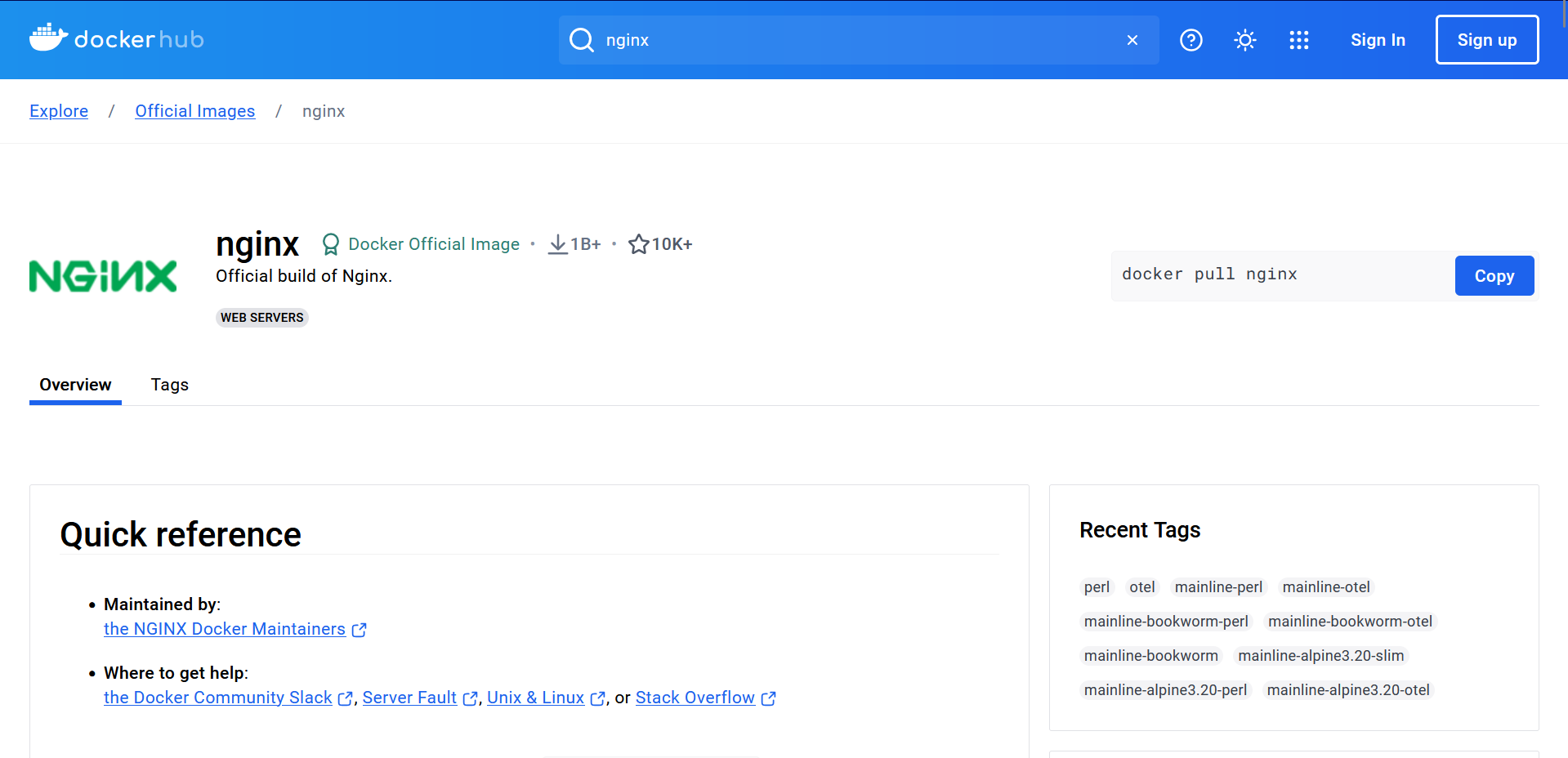
On Docker Desktop: we can see that the minikube container is up and running



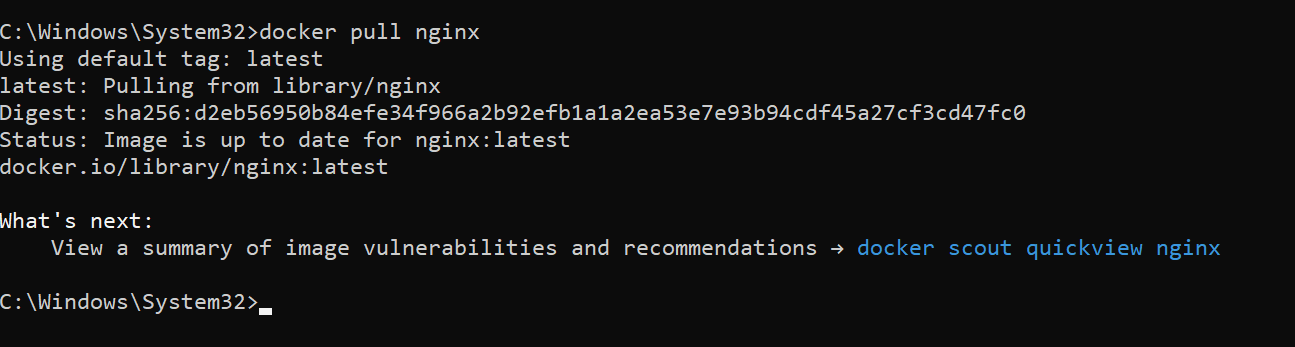
5. Check the status of minikube to ensure that it is properly configured



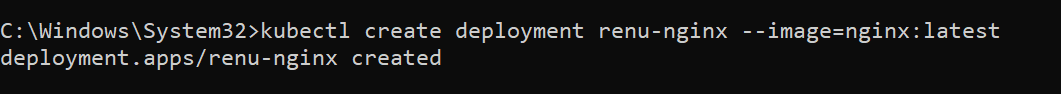
6. Now we need to pull the official image of Nginx from docker hub. In the browser, open docker hub and search for the official image for Nginx server.



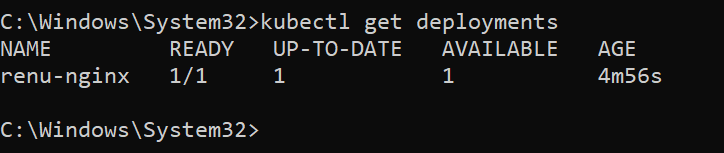
7. Pull the above image using docker pull nginx

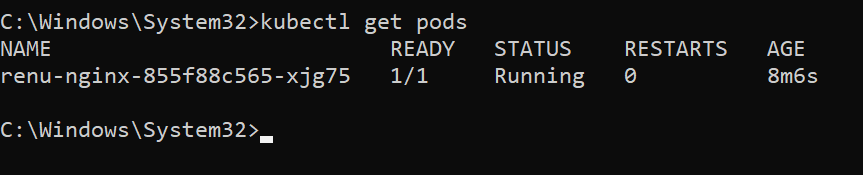


8. Create deployment

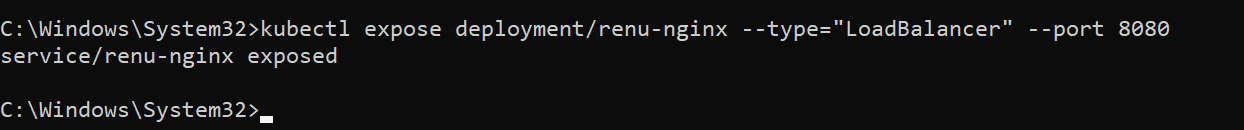


9. Check Deploment using kubectl get deployments and kubectl get pods to check the number od pods running

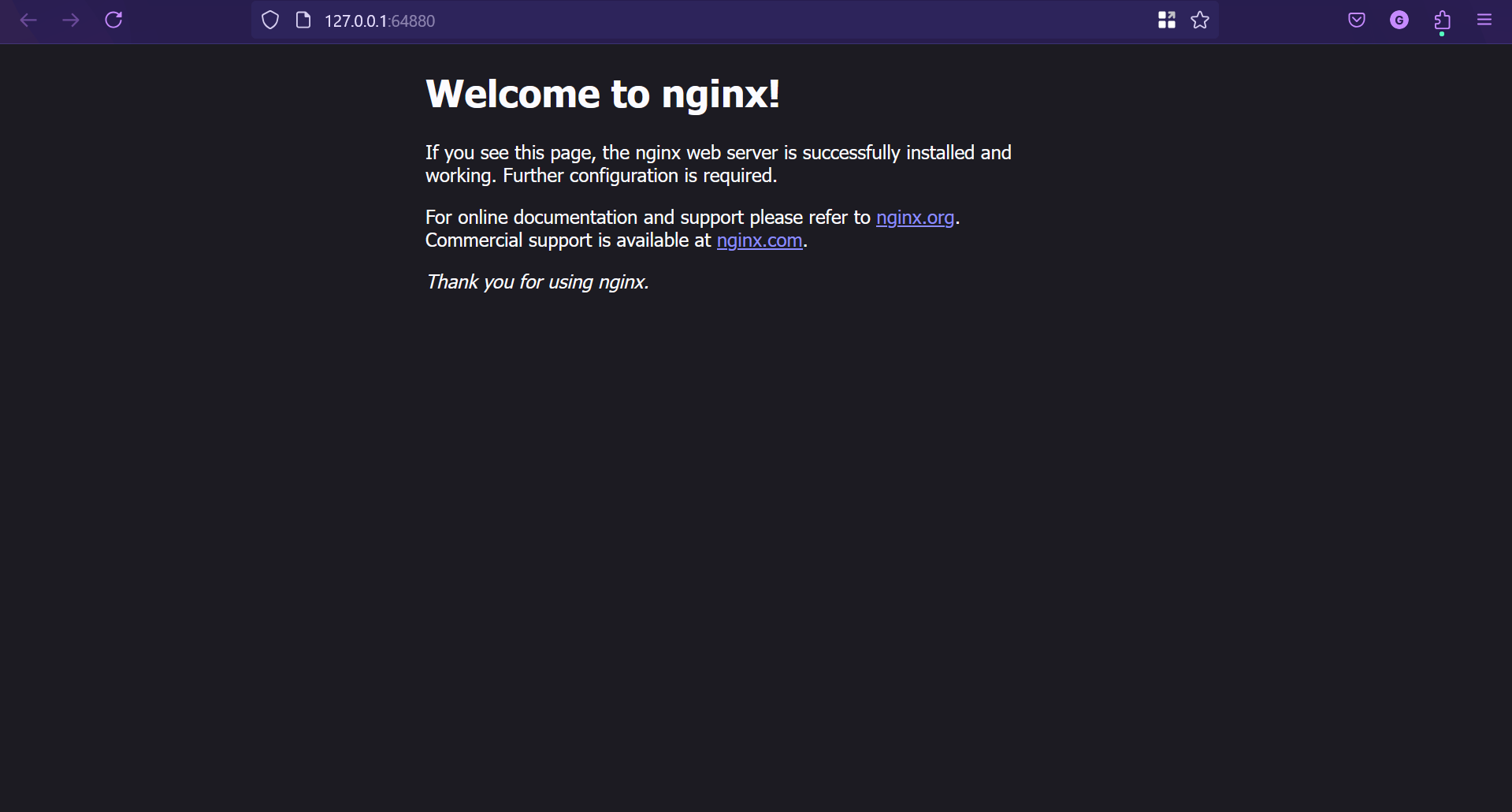




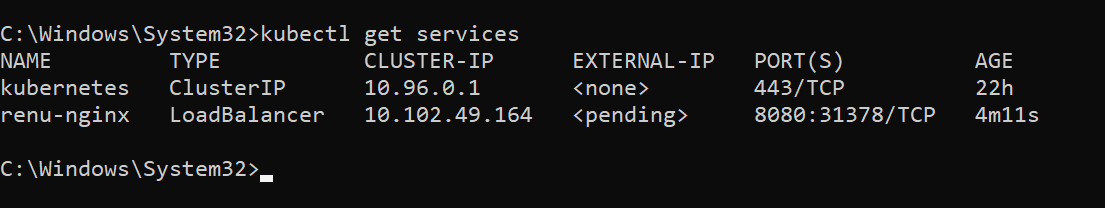
10. Expose deployment so that you can access the homepage of nginx on web browser



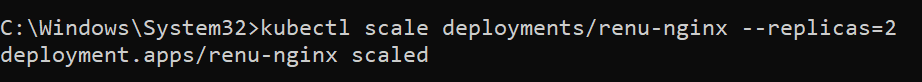
Then do minikube service renu-nginx to open it on browser



11. Check if nginx is showing in the Kubernetes services

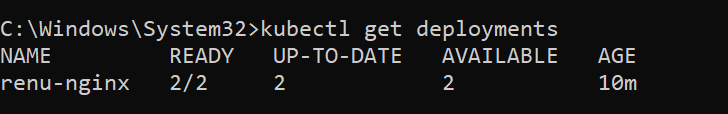


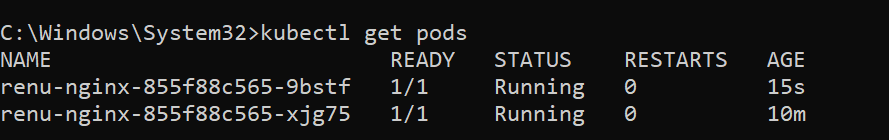
12. Now we can scale up or down the Kubernetes cluster. Scaling with 2 replicas



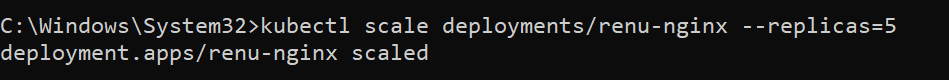
13. Check the number of deployments in Kubernetes using: kubectl get deployments

kubectl get pods





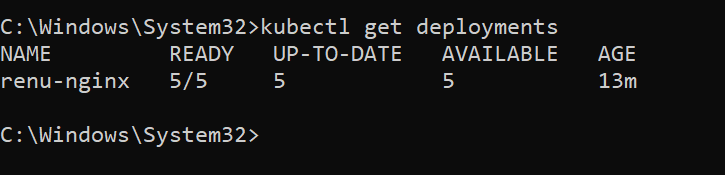
14. Scaling with 5 replicas

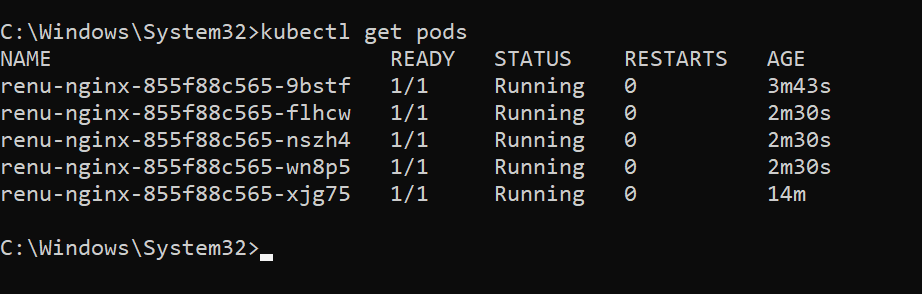


15. Again check the number of deployments

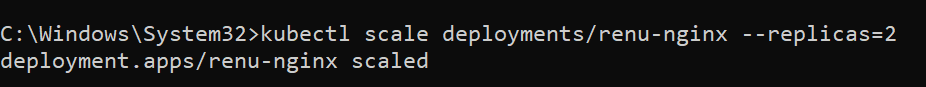
kubectl get deployments

kubectl get pods





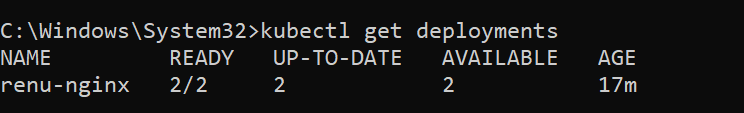
16. Scaling down: kubectl scale deployments/renu-nginx –replicas=2

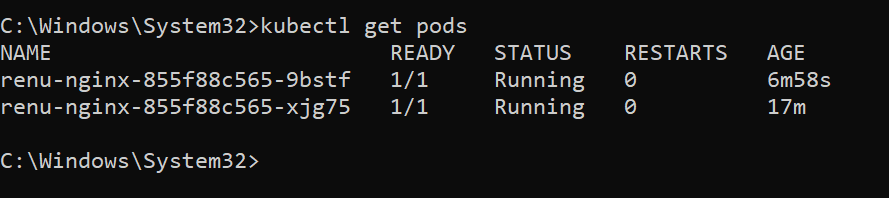


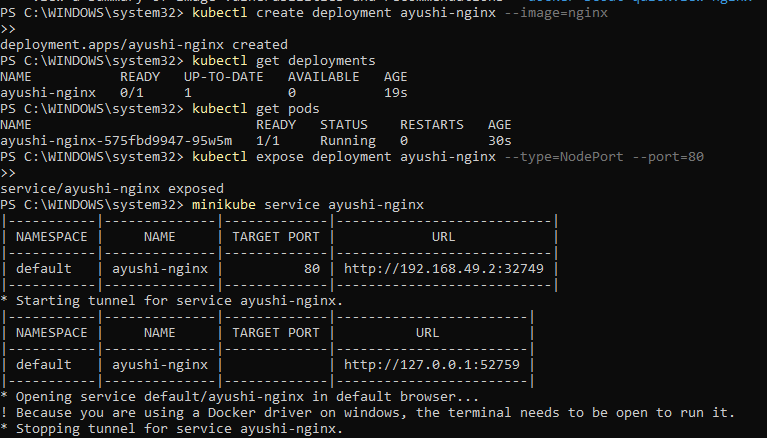
17. Check if the number of deployments and pods has changed to 2

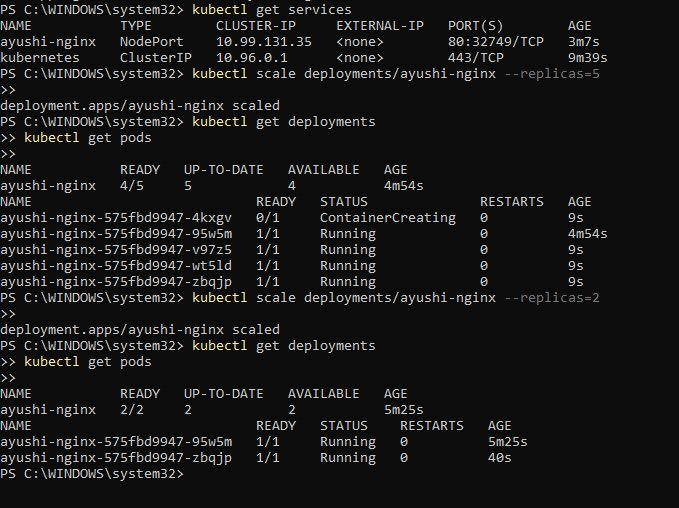
kubectl get deployments

kubectl get pods









# SonarCloud

Login to sonarcloud thru git

Make a project put it for analzes

**Step 1: Sign Up for SonarCloud**

1. Go to [SonarCloud](https://sonarcloud.io/).
2. Sign up using your GitHub account.
3. Authorize SonarCloud to access your GitHub repositories.

**Step 2: Create a Project on SonarCloud**

1. After logging into SonarCloud, go to the **Projects** section.
2. Click on **Analyze new project**.
3. Select the GitHub repository that contains your Python project.
4. Click on **Set Up** to start configuring the project for analysis.

**Step 3: Set Up SonarScanner in Your Project Repository**

**Option 1: Using SonarCloud GitHub Actions (Recommended for GitHub Projects)**

1. **Add SonarCloud Configuration:**
   * In your repository, create a directory .github/workflows/.
   * Inside that folder, create a file called sonarcloud.yml.
   * Add the following content to the file:
2. name: Build
3. on:
4. push:
5. branches:
6. - master
7. pull\_request:
8. types: [opened, synchronize, reopened]
9. jobs:
10. sonarcloud:
11. name: SonarQube Cloud
12. runs-on: ubuntu-latest
13. steps:
14. - uses: actions/checkout@v4
15. with:
16. fetch-depth: 0  # Shallow clones should be disabled for a better relevancy of analysis
17. - name: SonarQube Cloud Scan
18. uses: SonarSource/sonarcloud-github-action@master
19. env:
20. GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}  # Needed to get PR information, if any
21. SONAR\_TOKEN: ${{ secrets.SONAR\_TOKEN }}

This YAML configuration will automatically run SonarCloud analysis on every push or pull request to the main branch.

**Step 4: Generate a SonarCloud Token**

1. Go to SonarCloud > My Account > Security.
2. Click **Generate Token** and copy the token generated.
3. In your GitHub repository, go to **Settings > Secrets > Actions**.
4. Click **New repository secret** and name it SONAR\_TOKEN. Paste the token value you copied.

**Step 5: Push Your Changes to GitHub**

1. After setting up the .github/workflows/sonarcloud.yml file and adding the SonarCloud token as a secret, push the changes to your GitHub repository.
2. GitHub Actions will automatically trigger and begin the SonarCloud analysis.

**Step 6: Fix Issues and Re-Analyze**

1. Review the issues detected by SonarCloud, such as code smells, bugs, and security vulnerabilities.
2. Fix the identified issues in your Python code.
3. Push the changes back to GitHub, and SonarCloud will automatically re-run the analysis.

Terraform

### Main.tf

# Configure the Docker provider

terraform {

required\_providers {

docker = {

source = "kreuzwerker/docker"

version = "~> 2.0"

}

}

}

provider "docker" {

host = "unix:///var/run/docker.sock"

}

# Pull Nginx image from DockerHub

resource "docker\_image" "nginx" {

name = "nginx:latest"

keep\_locally = false

}

# Create an Nginx container

resource "docker\_container" "nginx" {

image = docker\_image.nginx.image\_id

name = "nginx\_terraform"

ports {

internal = 80

external = 8088

}

}

### Docker-compose.yaml

version: '3.8'

services:

  terraform:

    image: hashicorp/terraform:latest # Official Terraform Docker image

    working\_dir: /workspace           # Directory inside the container

    volumes:

      - C:/Users/Ayushi/Downloads/Final/terraform:/workspace                 # Mount current directory into the container

    entrypoint: ["terraform"]             # Default shell entrypoint

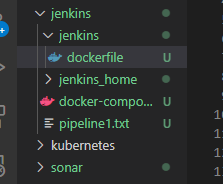
### Commands:

docker-compose run terraform init

docker-compose run terraform plan

docker-compose run terraform apply -auto-approve

## Jenkins



 **Create a Project Directory**:

* mkdir my-jenkins-project && cd my-jenkins-project

 **Create jenkins Folder**:

* mkdir jenkins

 **Create docker-compose.yml** in the root directory.

version: '3.8'

services:

  jenkins:

    build:

      context: ./jenkins  # This is where your Jenkins-specific Dockerfile is located

      dockerfile: Dockerfile  # This references the Dockerfile inside the jenkins folder

    privileged: true

    user: root

    ports:

      - "8083:8080"

      - "50000:50000"

    container\_name: jenkins

    volumes:

      - C:/Users/Ayushi/Downloads/Final/jenkins/jenkins\_home:/var/jenkins\_home

      - /var/run/docker.sock:/var/run/docker.sock

    networks:

      - jenkins-net

networks:

  jenkins-net:

    driver: bridge

volumes:

  jenkins\_home:

 **Create Dockerfile** in the jenkins folder.

FROM jenkins/jenkins:lts

# Install Docker

USER root

RUN apt-get update && apt-get install -y \

    ca-certificates \

    curl \

    gnupg \

    lsb-release \

    && curl -fsSL https://download.docker.com/linux/debian/gpg | tee /etc/apt/trusted.gpg.d/docker.asc \

    && echo "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb\_release -cs) stable" > /etc/apt/sources.list.d/docker.list \

    && apt-get update \

    && apt-get install -y docker-ce docker-ce-cli containerd.io \

    && apt-get clean

# Set permissions on the Docker socket

RUN usermod -aG docker jenkins

USER jenkins

 **Run Docker Compose**:

* docker-compose up --build -d

 **Access Jenkins** at <http://localhost:8083/>.

* Retrieve the initial password from jenkins\_home/secrets/initialAdminPassword.

 **Install Docker Plugin**:

* Manage Jenkins > Manage Plugins > Install Docker plugin.

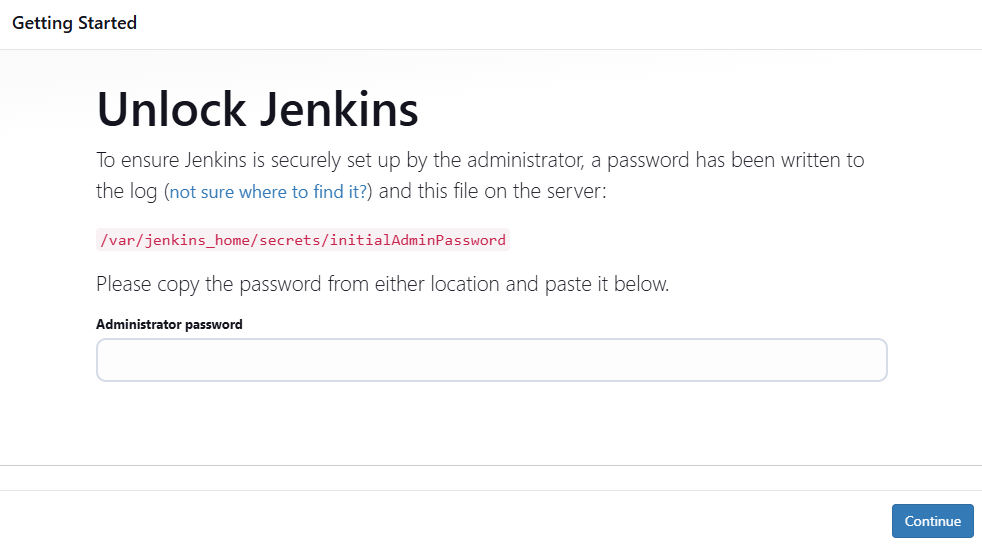
 **Create Pipeline**:

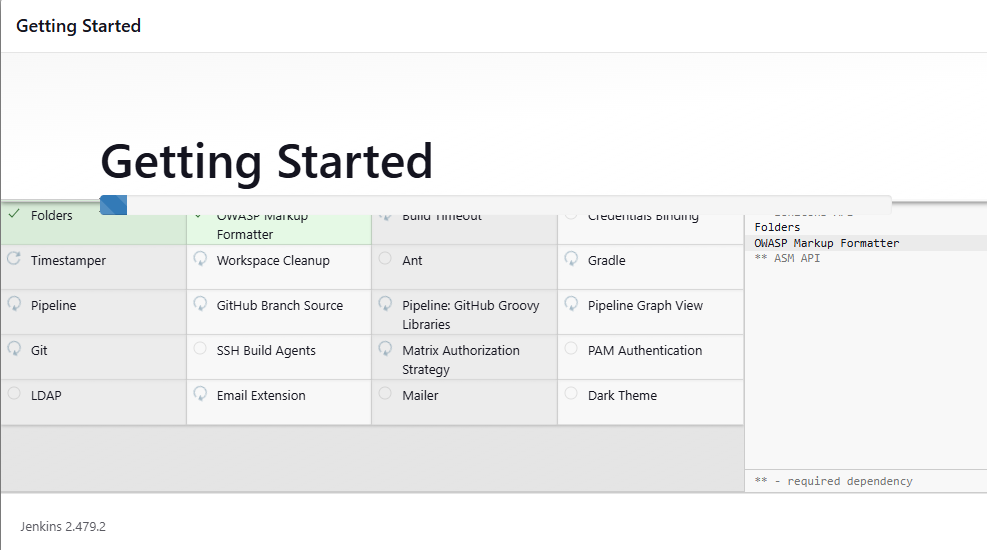
* New Item > Pipeline > Paste pipeline script > Save.

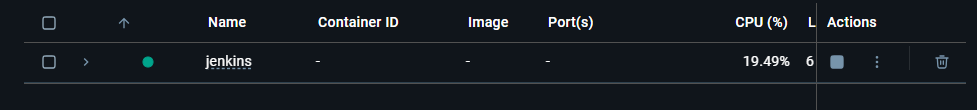
 **Run Pipeline**:

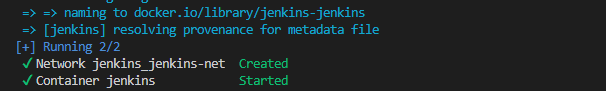
* Click **Build Now**.

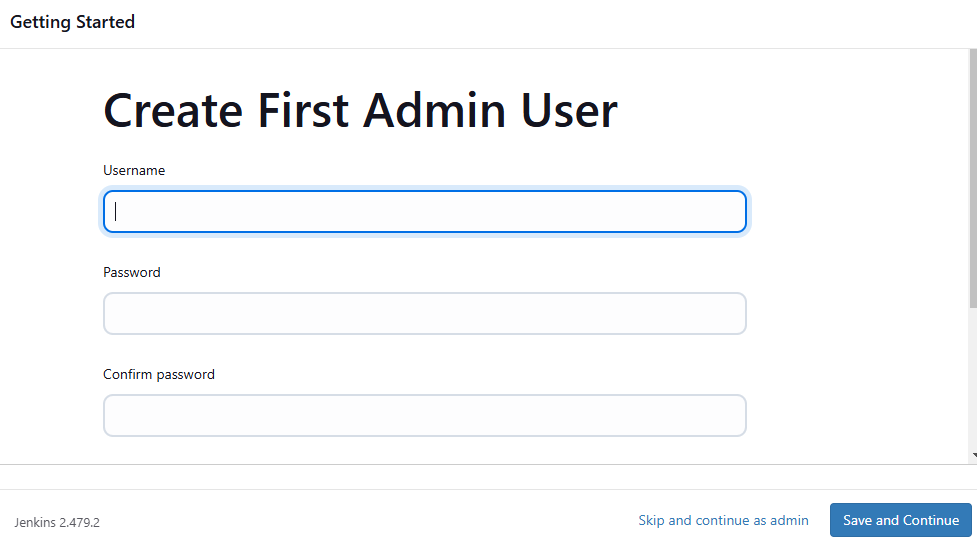
 **Check Docker Desktop** for the nginx-container.

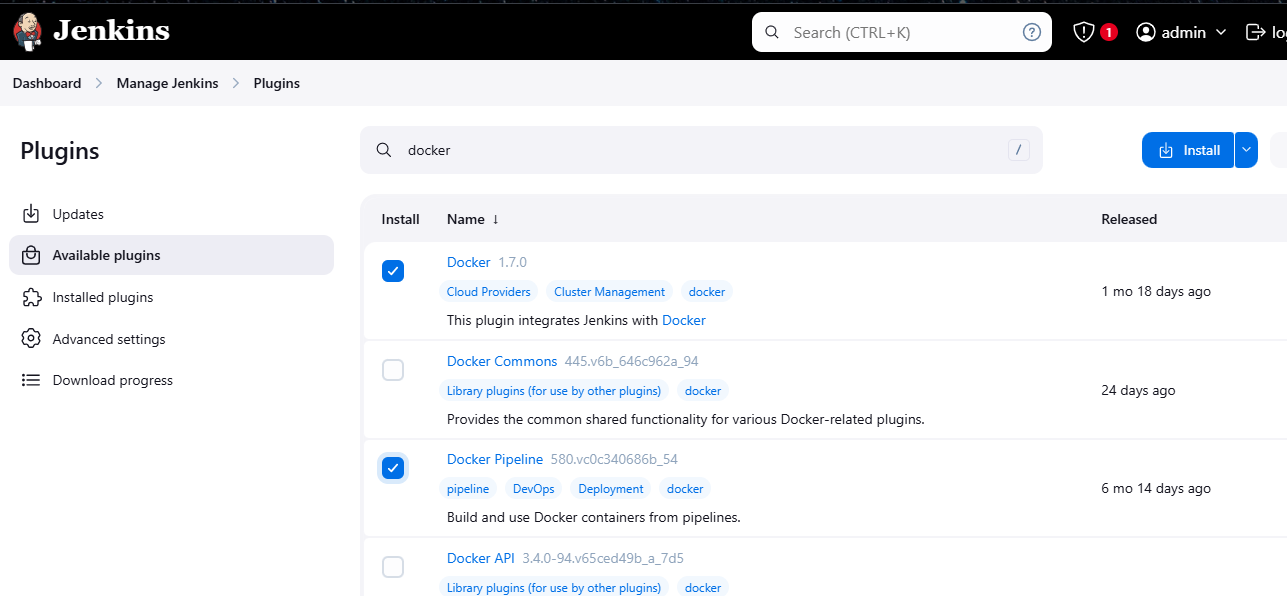


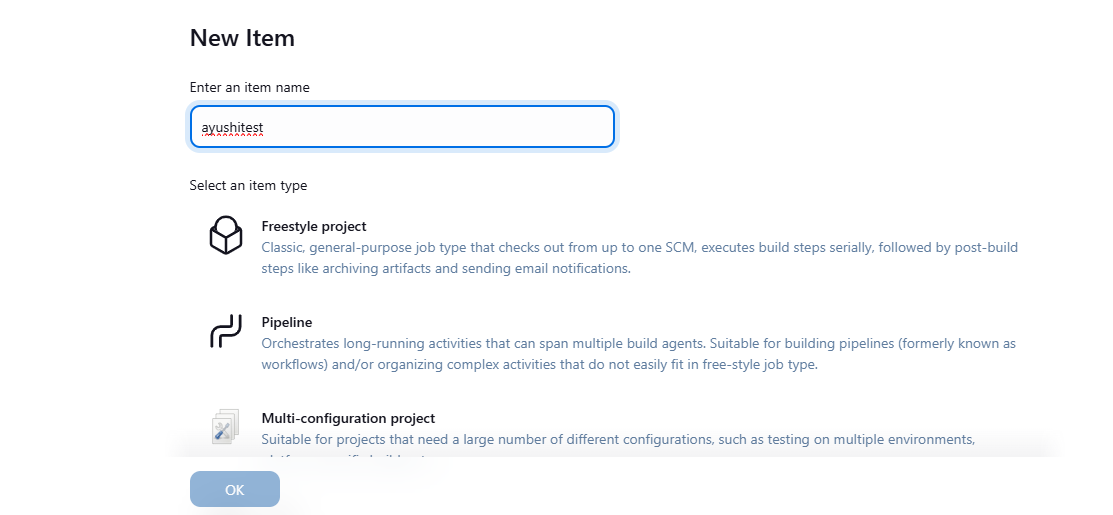


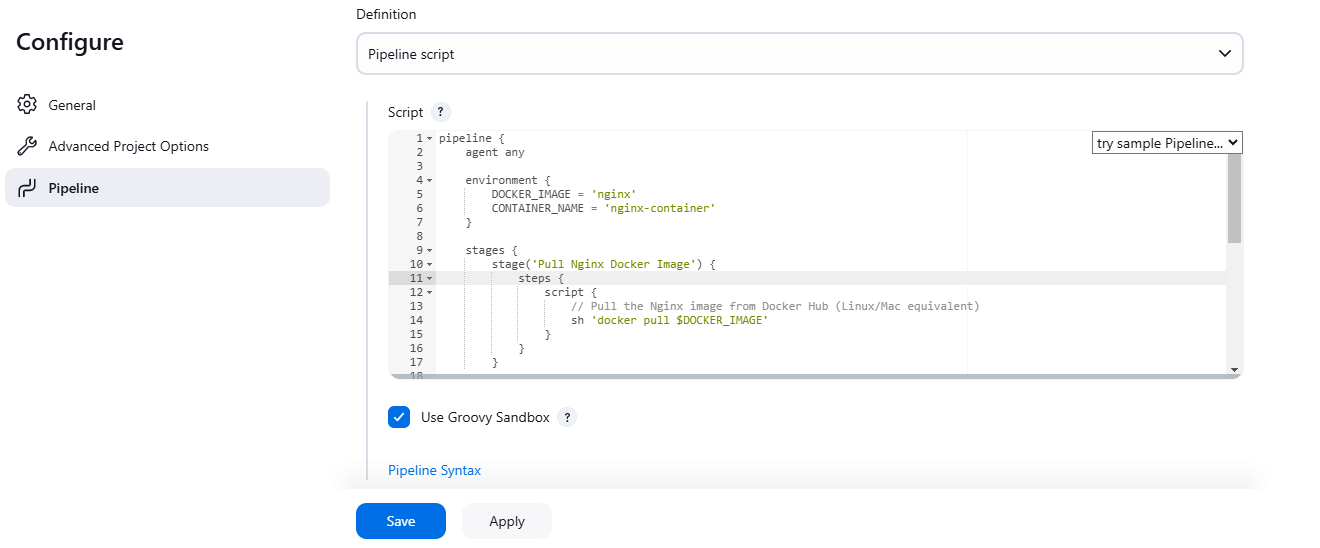


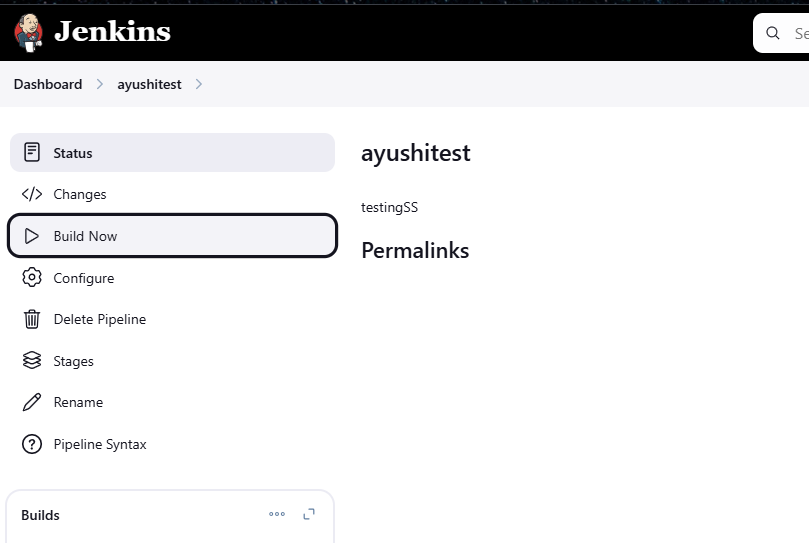


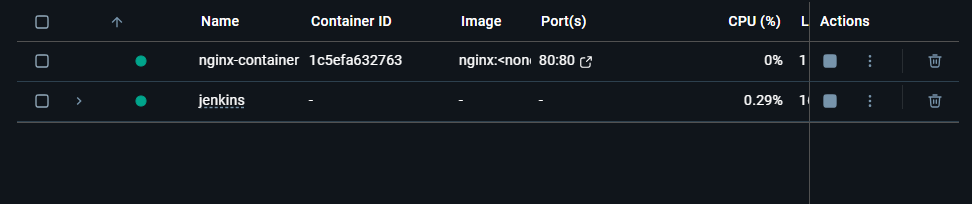


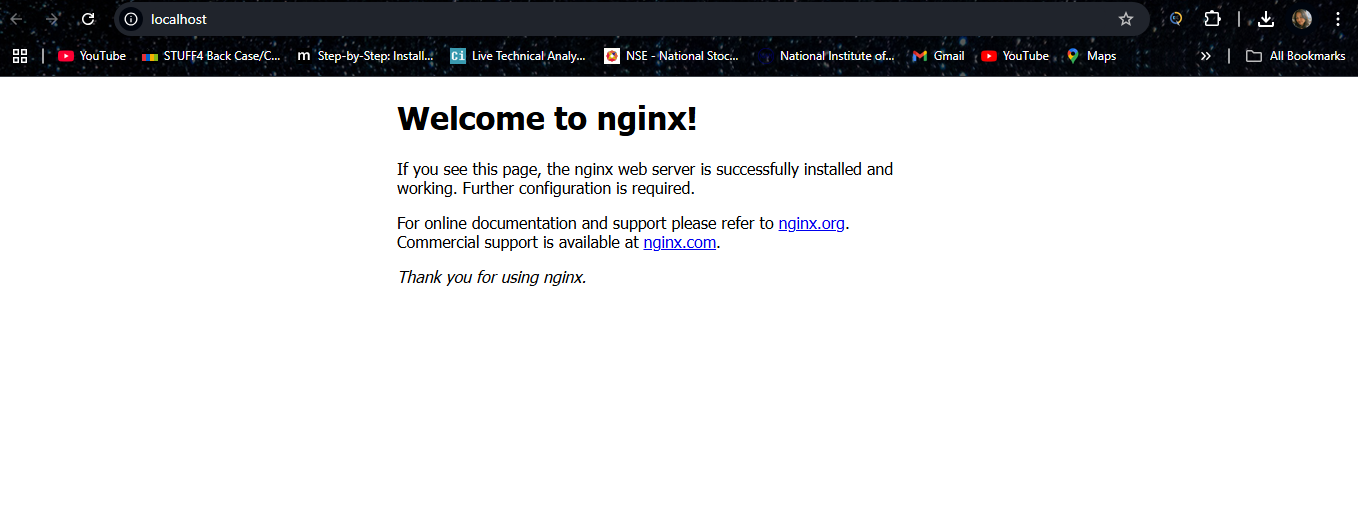


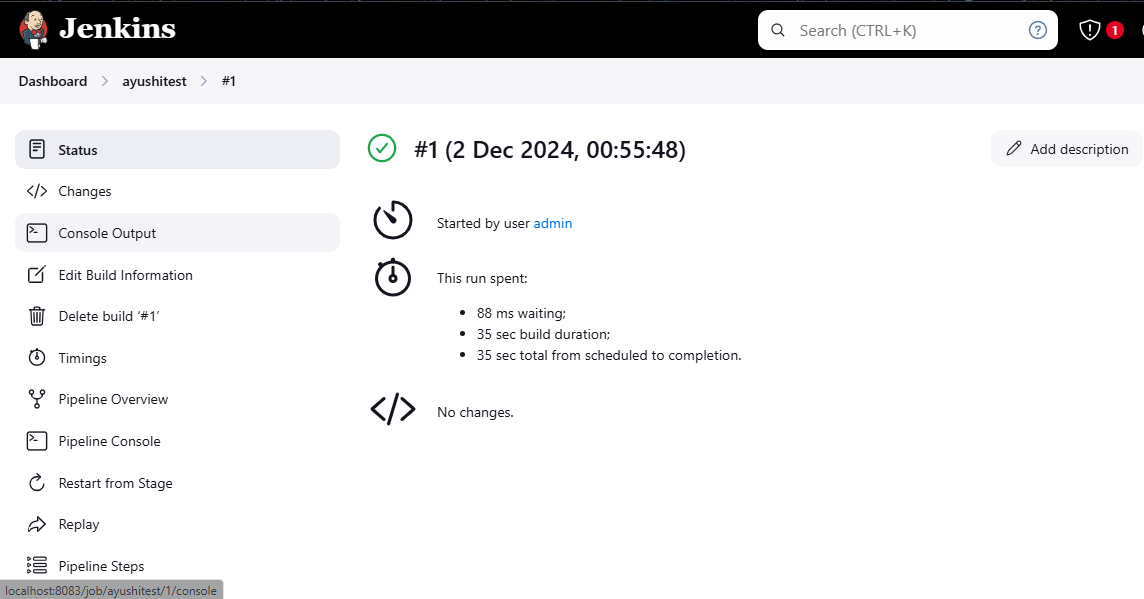












### Pipeline

pipeline {

agent any

environment {

DOCKER\_IMAGE = 'nginx'

CONTAINER\_NAME = 'nginx-container'

}

stages {

stage('Pull Nginx Docker Image') {

steps {

script {

// Pull the Nginx image from Docker Hub (Linux/Mac equivalent)

sh 'docker pull $DOCKER\_IMAGE'

}

}

}

stage('Run Docker Container') {

steps {

script {

// Run the Nginx container (Linux/Mac equivalent)

sh 'docker run -d --name $CONTAINER\_NAME -p 80:80 $DOCKER\_IMAGE'

}

}

}

}

post {

always {

echo 'Pipeline finished.'

}

success {

echo 'Deployment succeeded.'

}

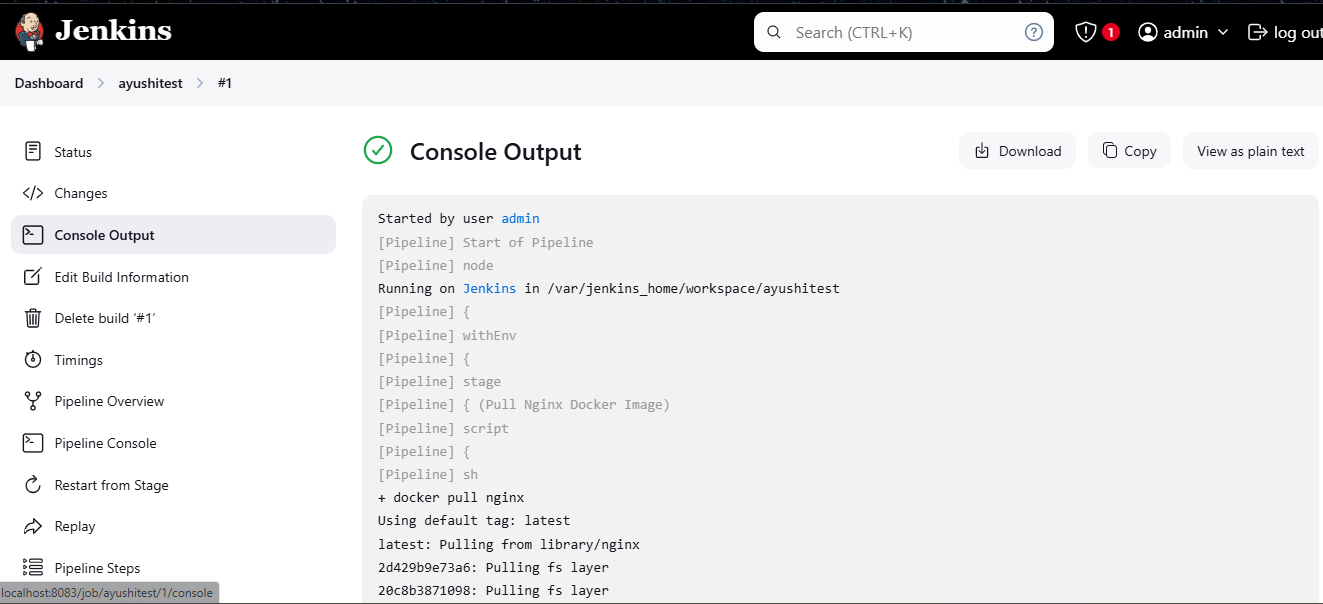
failure {

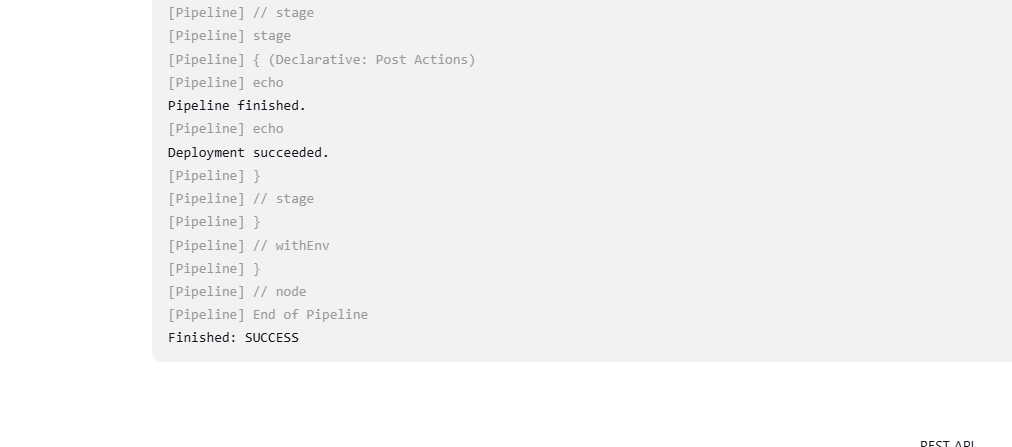
echo 'Deployment failed.'

}

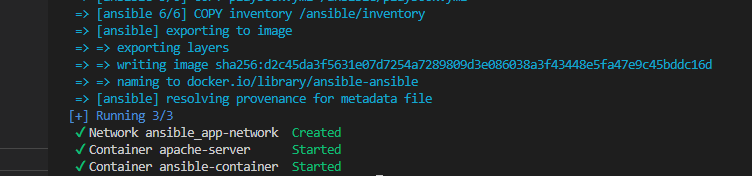
}

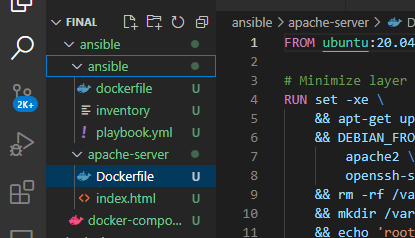
}

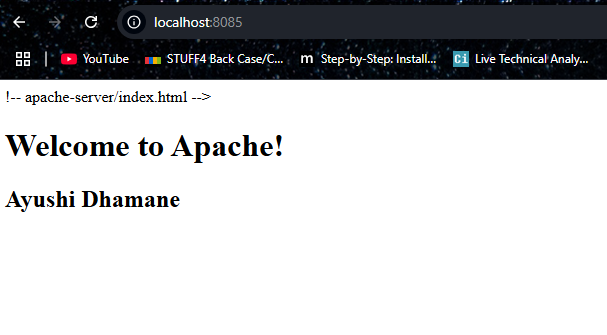


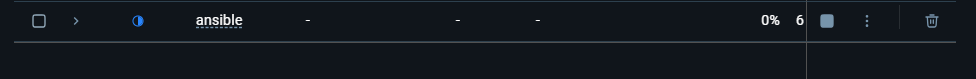


## Ansible









Ansible folder

### Dockerfile

# ansible/Dockerfile

FROM ubuntu:20.04

# Install Ansible and SSH client for remote communication

RUN apt-get update && \

    apt-get install -y ansible sshpass python3-pip && \

    apt-get clean

# Install necessary Ansible dependencies

RUN pip3 install ansible

# Set working directory

WORKDIR /ansible

# Copy the playbook and inventory into the container

COPY playbook.yml /ansible/playbook.yml

COPY inventory /ansible/inventory

# Entry point for Ansible to run the playbook

CMD ["ansible-playbook", "/ansible/playbook.yml", "-i", "inventory"]

### Inventory file

[apache-server]

apache-server ansible\_host=apache-server ansible\_port=22

[apache-server:vars]

ansible\_ssh\_user=root

ansible\_ssh\_pass=password

### playbook.yml

# ansible/playbook.yml

---

- name: Install Apache and ensure it is running

hosts: apache-server

become: true

tasks:

- name: Install Apache

apt:

name: apache2

state: present

update\_cache: yes

- name: Start Apache service

service:

name: apache2

state: started

enabled: yes

### anache server folder

### dockerfile

FROM ubuntu:20.04

# Minimize layer creation and reduce build time

RUN set -xe \

    && apt-get update \

    && DEBIAN\_FRONTEND=noninteractive apt-get install -y --no-install-recommends \

        apache2 \

        openssh-server \

    && rm -rf /var/lib/apt/lists/\* \

    && mkdir /var/run/sshd \

    && echo 'root:password' | chpasswd \

    && sed -i 's/^PermitRootLogin prohibit-password/PermitRootLogin yes/' /etc/ssh/sshd\_config

# Expose Apache and SSH ports

EXPOSE 80 22

# Create a basic index.html for Apache

COPY index.html /var/www/html/index.html

# Start SSH and Apache in the background and keep the container running

CMD service ssh start && apache2ctl -D FOREGROUND

### Index.html

!-- apache-server/index.html -->

<html>

<head>

<title>Apache Server</title>

</head>

<body>

<h1>Welcome to Apache!</h1>

<h2>Ayushi Dhamane</h2>

</body>

</html>

### Out of both folders

### Docker-compose.yaml

version: '3.8'

services:

  apache-server:

    build: ./apache-server

    container\_name: apache-server

    ports:

      - "8085:80"

    networks:

      - app-network

  ansible:

    build: ./ansible

    container\_name: ansible-container

    depends\_on:

      - apache-server

    networks:

      - app-network

networks:

  app-network:

    driver: bridge