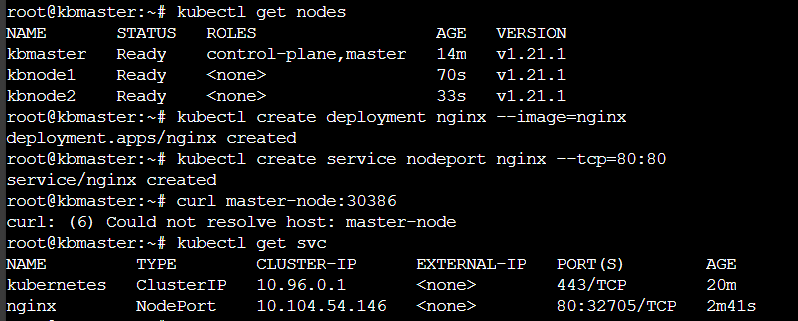
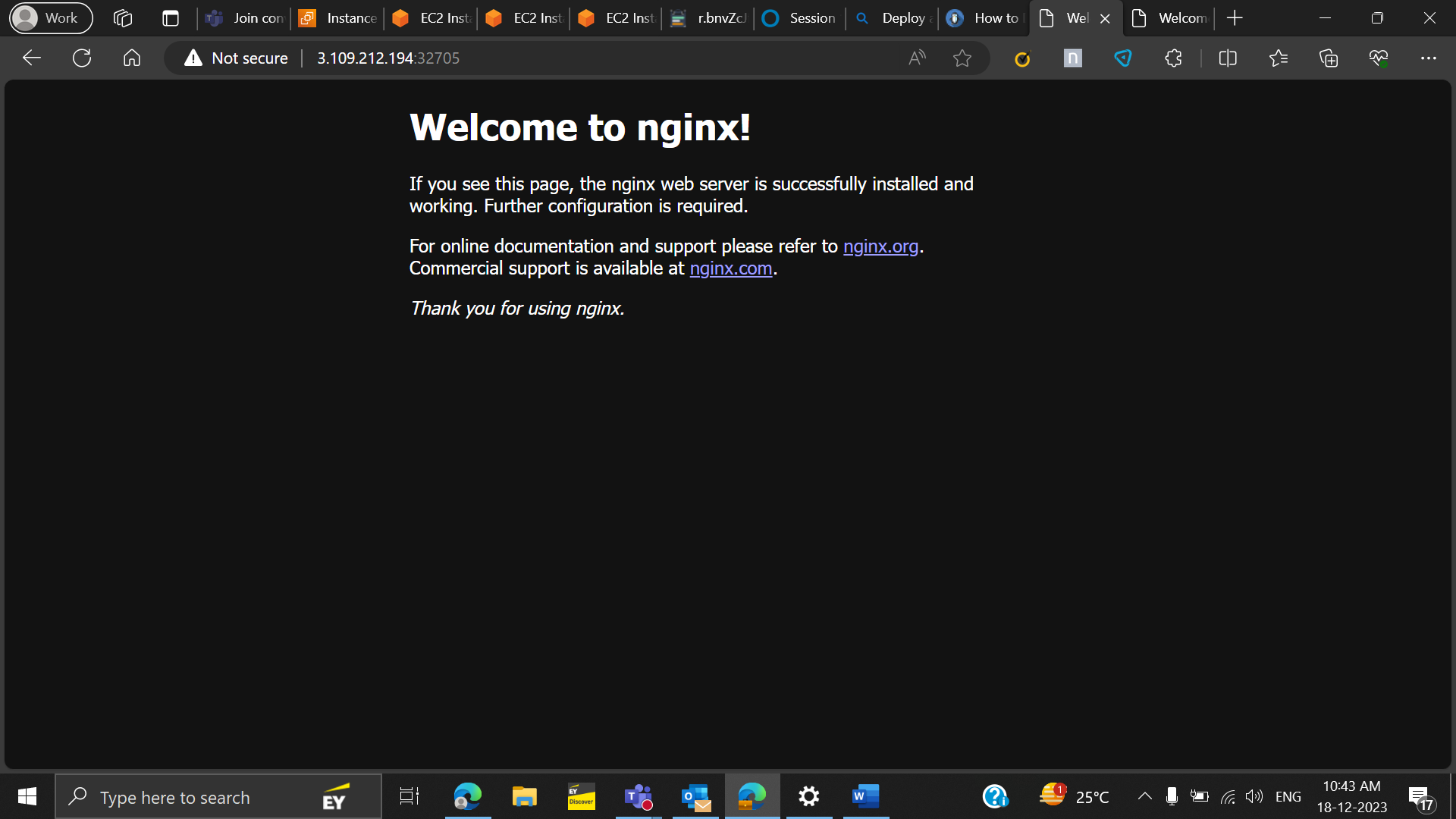
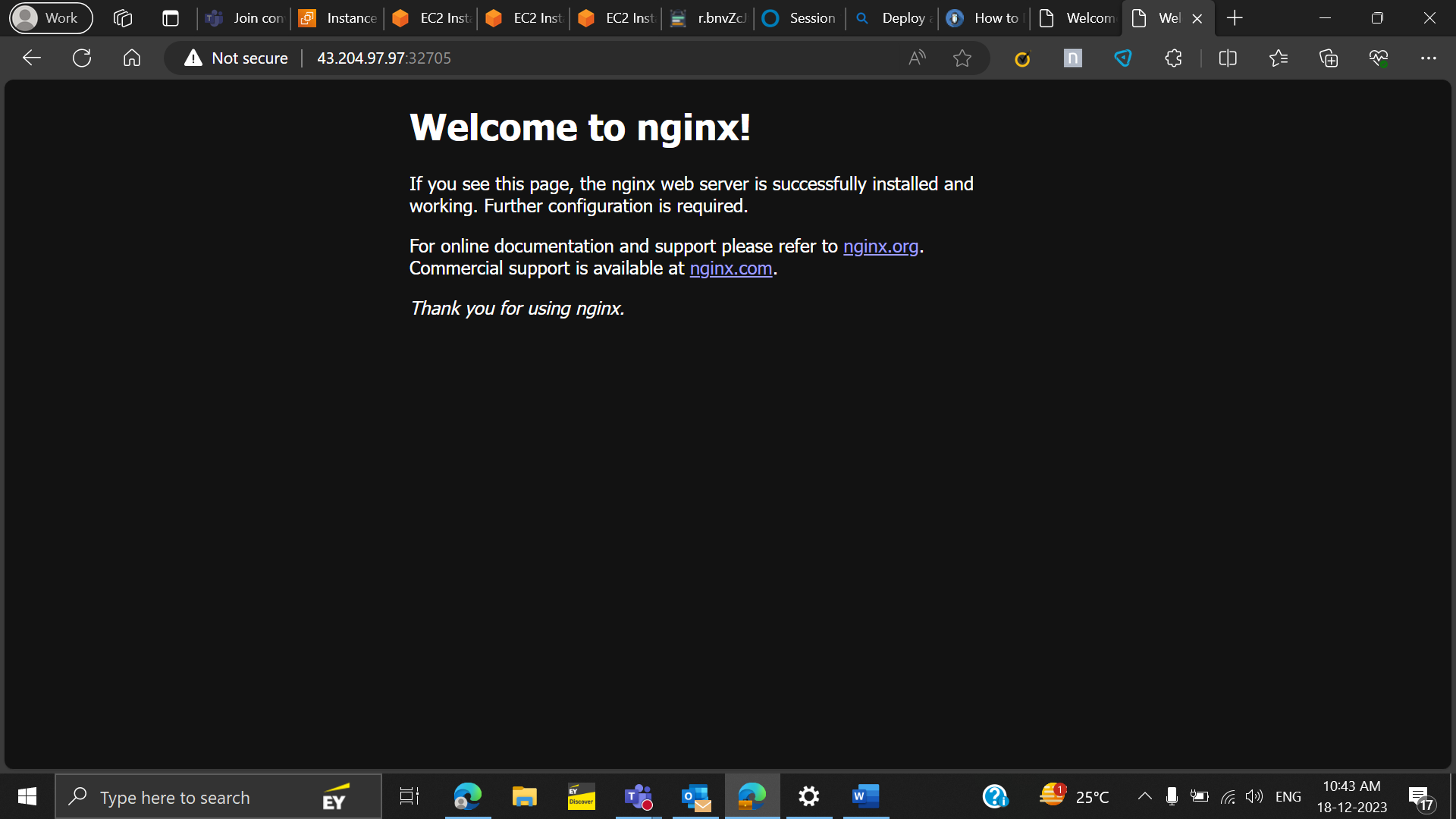
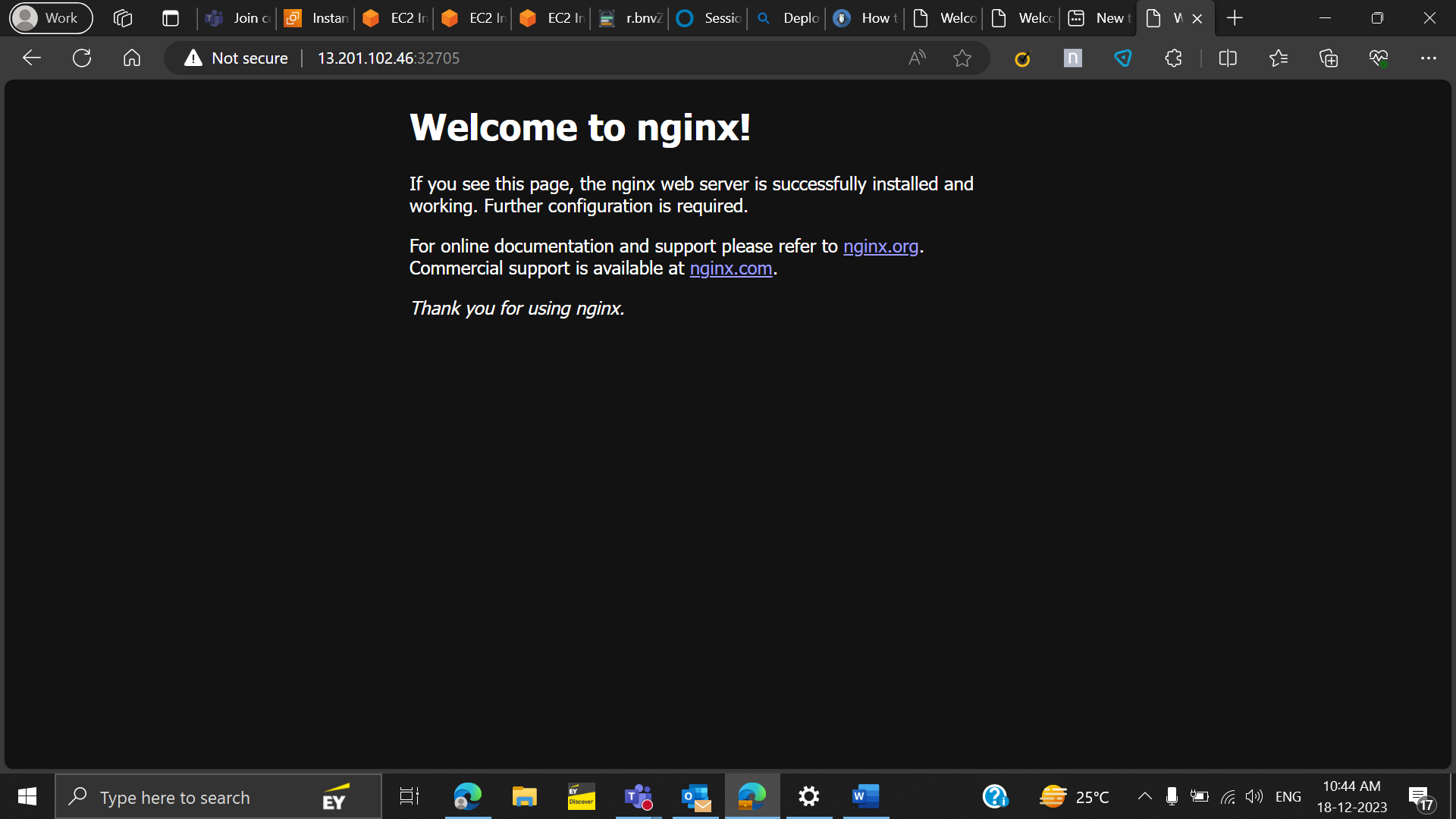
**Task-1: Deploy a simple Nginx web server in a Kubernetes cluster.**



Result:







**Task 2: Create a Kubernetes Deployment for a multi-replica Nginx**

Yaml File:

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

labels:

app: nginx

spec:

replicas: 3

selector:

matchLabels:

app: nginx

template:

metadata:

labels:

app: nginx

spec:

containers:

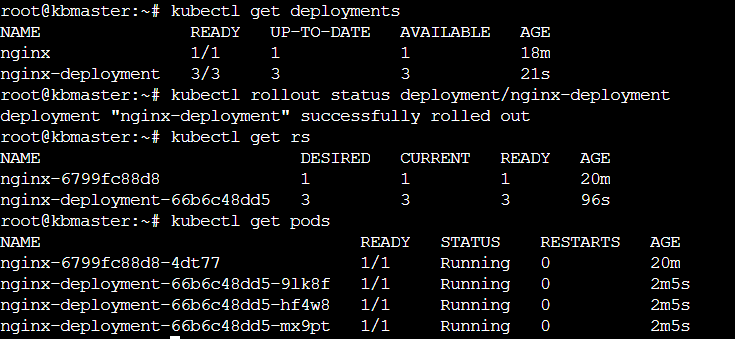
- name: nginx

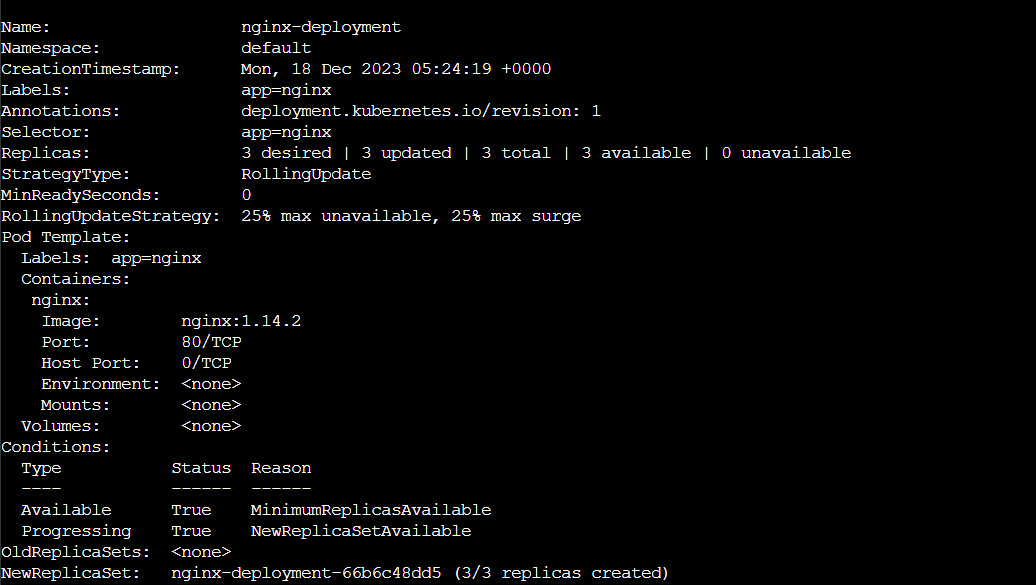
image: nginx:1.14.2

ports:

- containerPort: 80

Result:





**Task 3: Expose the Nginx Deployment using a Kubernetes Service.**

Yaml file:

apiVersion: v1

kind: Service

metadata:

name: nginx

namespace: default

labels:

app: nginx

spec:

externalTrafficPolicy: Local

ports:

- name: http

port: 80

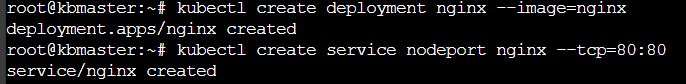
protocol: TCP

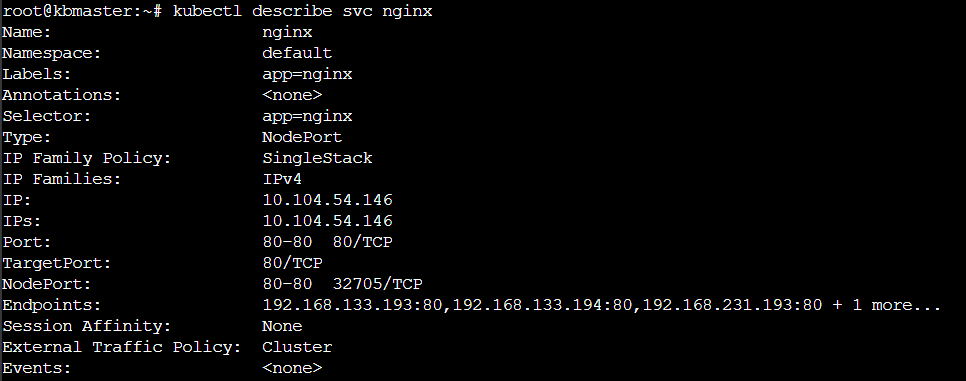
targetPort: 80

selector:

app: nginx

type: NodePort





**Task 4: Create a Config-Map for storing configuration data and mount it into a Pod.**

Config.yaml:

apiVersion: v1

kind: ConfigMap

metadata:

name: my-configmap

data:

app\_config.properties: |

key1=value1

key2=value2

key3=value3

pod.yaml:

apiVersion: v1

kind: Pod

metadata:

name: mypod

spec:

containers:

- name: mycontainer

image: ubuntu

volumeMounts:

- name: config-volume

mountPath: /etc/config

volumes:

- name: config-volume

configMap:

name: my-configmap

**Task 5: Create a Docker file to run a WordPress application with a linked MySQL container.**

Docker-compose.yml

version: "3"

services:

db:

image: mysql:latest

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: your\_mysql\_root\_password

MYSQL\_DATABASE: your\_mysql\_database\_name

volumes:

- ./mysql:/var/lib/mysql

wordpress:

depends\_on:

- db

image: wordpress:latest

restart: always

ports:

- "8080:80"

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: root

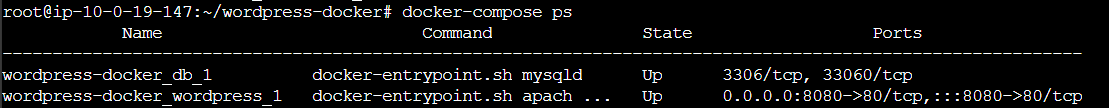
WORDPRESS\_DB\_PASSWORD: your\_mysql\_root\_password

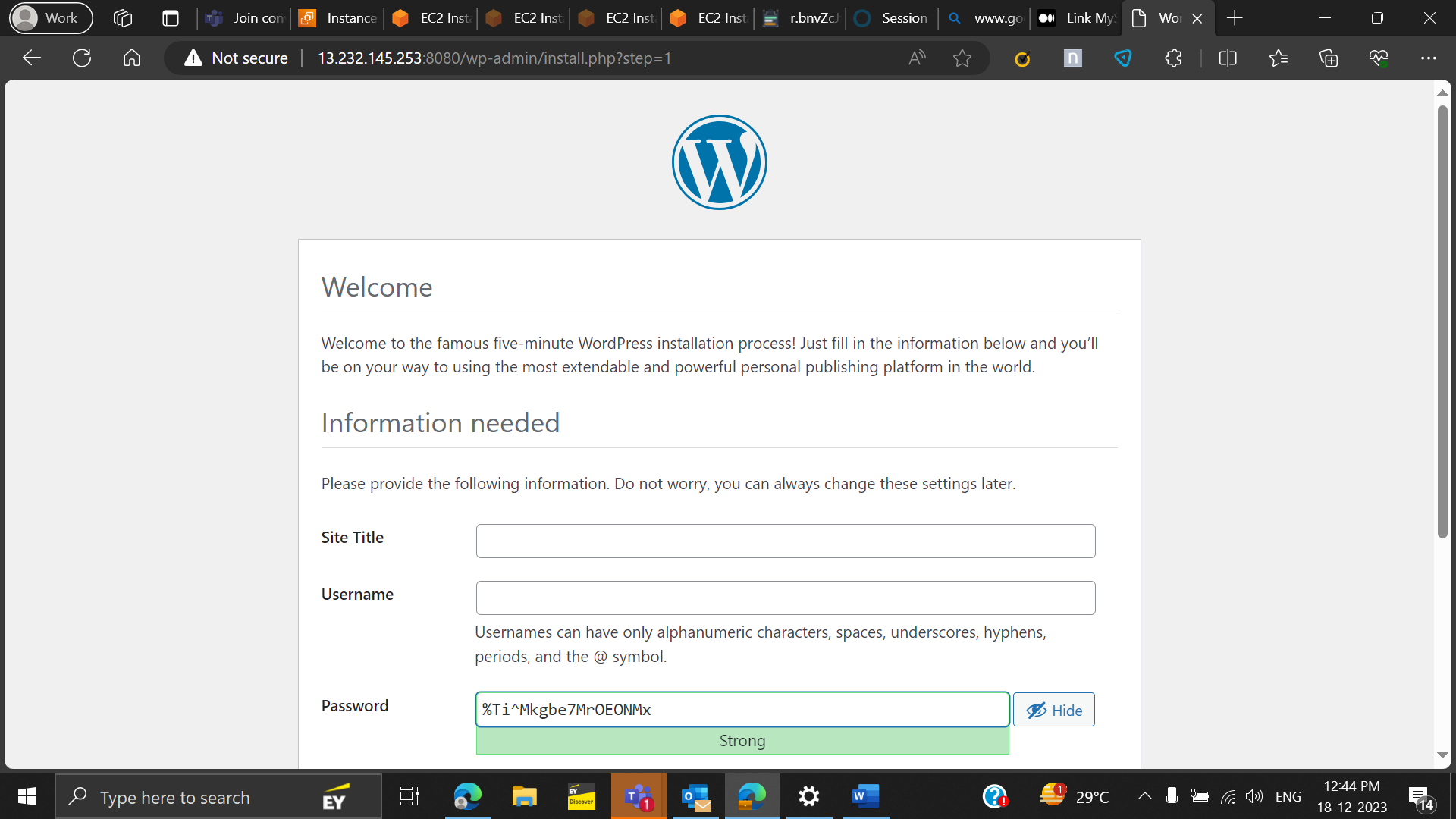
WORDPRESS\_DB\_NAME: your\_mysql\_database\_name

volumes:

- ./wordpress:/var/www/html

Result:





Task 6 : Dockerize a Node.js application and expose it on port 3000

**Dockerfile**

FROM node:latest

WORKDIR /usr/src/app

COPY package\*.json ./

RUN npm install

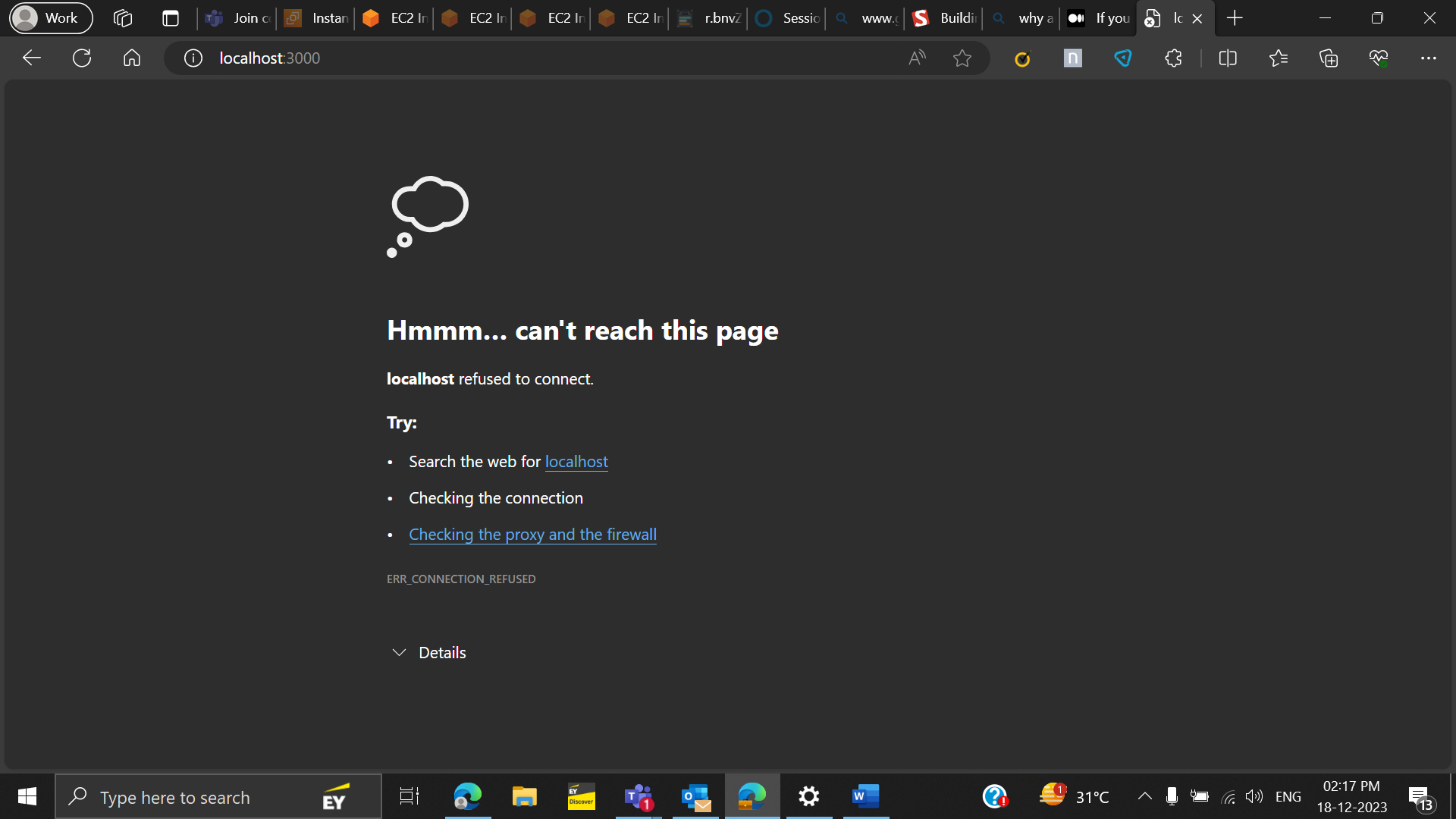
COPY . .

ENV PORT=3000

CMD [ 'npm', 'app.js' ]

Result:

Node.js app not reachable



**Task 7: Create a Dockerfile for a Python web application using Flask. The application should expose port 5000, and when accessed, it should display "Hello, Docker!" in the browser.**

**Dockerfile:**

FROM python:3.9.12-buster

WORKDIR /flask-docker

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

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

**App.py:**

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def hello():

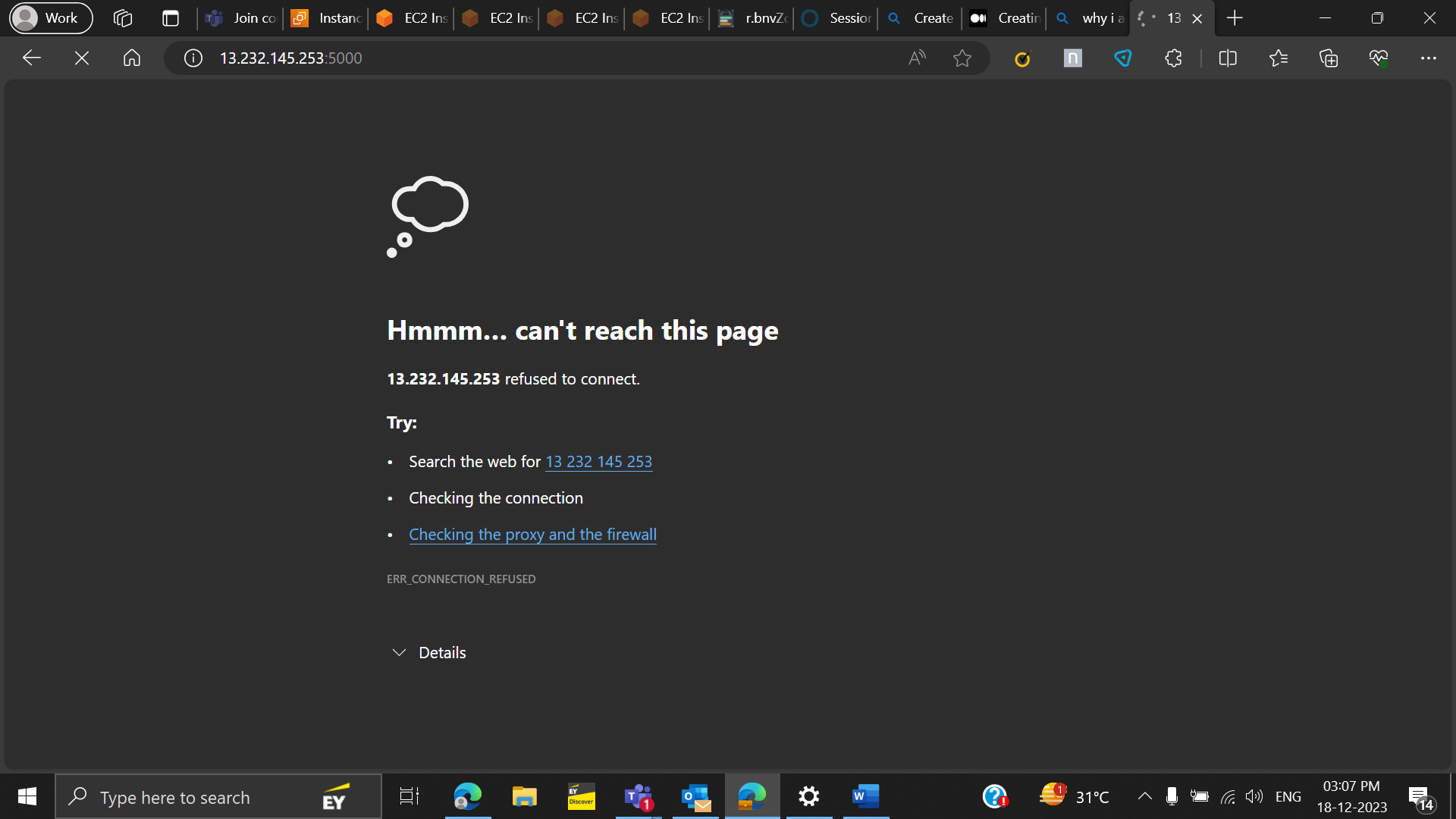
return 'Hello,Docker!'

if \_\_name\_\_ == '\_\_main\_\_':

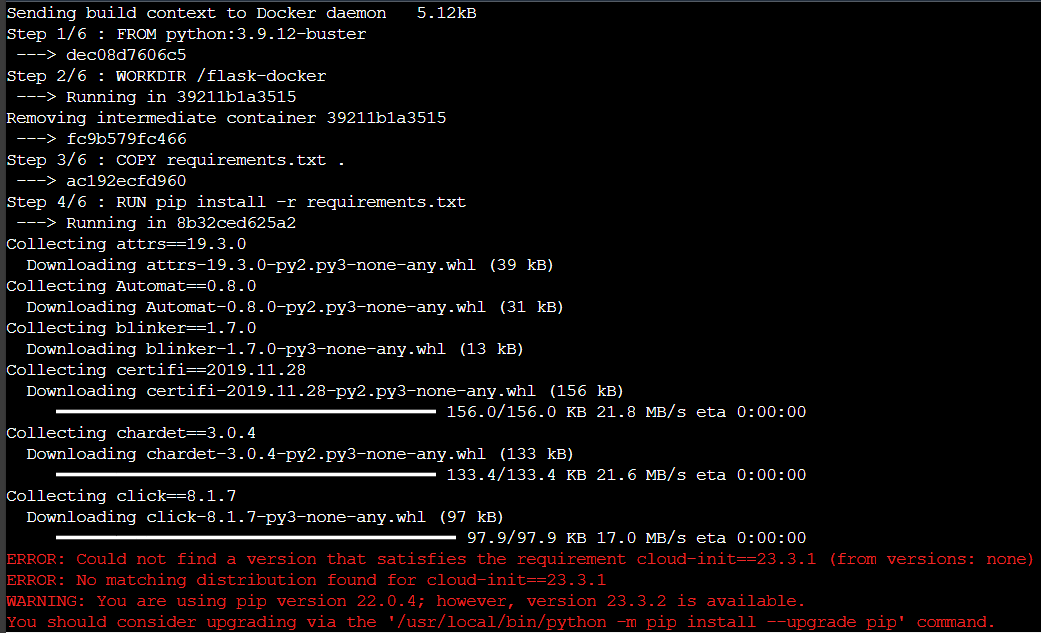
app.run(debug=True)

docker run -d -p 5000:5000 7176fe81a234

result:



Error:



**Task 8: Create a Terraform script to provision an AWS EC2 instance with an associated security group allowing incoming traffic on port 22 for SSH.**

vpc.tf

provider "aws" {

region = var.region

access\_key = var.access\_key

secret\_key = var.secret\_key

}

resource "aws\_vpc" "vpc" {

cidr\_block = "10.0.0.0/16"

tags = {

Name = "vpc"

}

}

resource "aws\_subnet" "subnet" {

vpc\_id = aws\_vpc.vpc.id

cidr\_block = "10.0.1.0/24"

availability\_zone = "ap-south-1b"

map\_public\_ip\_on\_launch = true

tags = {

Name = "subnet"

}

}

resource "aws\_internet\_gateway" "igw" {

vpc\_id = aws\_vpc.vpc.id

tags = {

Name = "igw"

}

}

resource "aws\_route\_table" "route" {

vpc\_id = aws\_vpc.vpc.id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.igw.id

}

tags = {

Name = "route"

}

}

resource "aws\_route\_table\_association" "devrouteassoc" {

subnet\_id = aws\_subnet.subnet.id

route\_table\_id = aws\_route\_table.route.id

}

resource "aws\_security\_group" "sg" {

vpc\_id = aws\_vpc.vpc.id

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "sg"

}

}

Ec2.tf

provider "aws" {

region = var.region

access\_key = var.access\_key

secret\_key = var.secret\_key

}

variable "vpc\_id" {

default = "vpc-0db47fd19fbc32358"

}

variable "subnet\_id" {

default = "subnet-02c0ef68f549858a0"

}

resource "aws\_instance" "instance" {

ami = "ami-0287a05f0ef0e9d9a"

instance\_type = "t2.medium"

subnet\_id = var.subnet\_id

tags = {

Name = "instance"

}

}

**Task 9 : Write a Terraform script to provision an AWS RDS instance (MySQL)**

#main.tf

#defining the provider as aws

provider "aws" {

region = "${var.region}"

access\_key = "${var.access\_key}"

secret\_key = "${var.secret\_key}"

}

#create a security group for RDS Database Instance

resource "aws\_security\_group" "rds\_sg" {

name = "rds\_sg"

ingress {

from\_port = 3306

to\_port = 3306

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

#create a RDS Database Instance

resource "aws\_db\_instance" "myinstance" {

engine = "mysql"

identifier = "myrdsinstance"

allocated\_storage = 20

engine\_version = "5.7"

instance\_class = "db.t2.micro"

username = "myrdsuser"

password = "myrdspassword"

parameter\_group\_name = "default.mysql5.7"

vpc\_security\_group\_ids = ["${aws\_security\_group.rds\_sg.id}"]

skip\_final\_snapshot = true

publicly\_accessible = true

}

Result :

