Name: Ali Shehzad

Project Title:

Deploying a Flask Application on Kubernetes Using KIND in AWS EC2

Introduction

Modern startups and tech teams are increasingly using containerization and Kubernetes to deploy scalable and reliable applications. In this project, we deploy a Flask web application on a Kubernetes cluster set up with KIND (Kubernetes IN Docker) on a single AWS EC2 instance, demonstrating lightweight and efficient orchestration

Objectives

The primary objectives of this project were:

- To provision a lightweight Kubernetes cluster on a single AWS EC2 instance using KIND.
- To containerize and deploy a Flask application inside the Kubernetes cluster.
- To demonstrate Kubernetes concepts including Namespaces, Deployments, Pods, Labels, and Services.
- To expose the Flask application externally for public browser access using Kubernetes networking.

Project Structure

K8s-project/

```
├── app.py                # Flask app
├── requirements.txt      # Dependencies
├── Dockerfile            # Docker image
│
├── deployment.yml        # Kubernetes Deployment
├── service.yml           # Kubernetes Service
├── namespace.yml         # Kubernetes Namespace
│
├── kind-config.yml       # KIND cluster config
│
└── README.md             # Documentation / setup steps
```

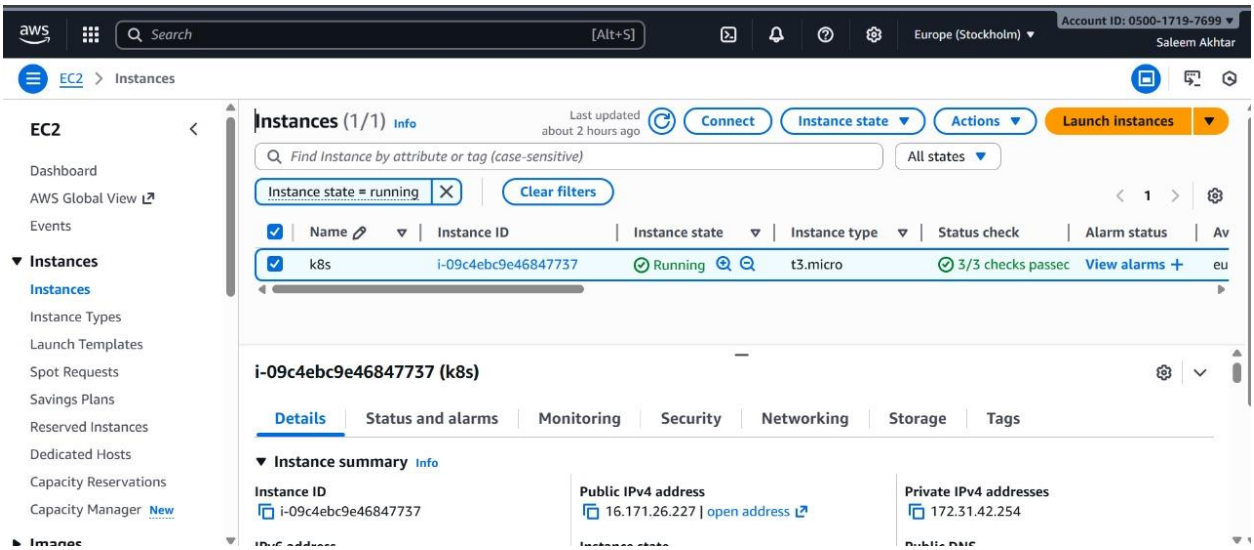
Github Repo Link: **https://github.com/alishehzad-943/k8s-project/tree/main**

Project Procedures

Below is a high-level breakdown of all procedures performed in the project. Each step mentions only actions and file names.

1. AWS EC2 Setup

- Launched an Ubuntu-based t3.micro EC2 instance on AWS.
- Installed essential dependencies including Docker and kubectl.
- Configured the server for Kubernetes operations.



2. KIND Cluster Setup

- Installed KIND on the EC2 instance.
- Created a configuration file: **kind-config.yaml**
- Launched a single-node Kubernetes cluster using the file.

3. Flask Application Preparation

- Created a simple Flask application file:app.py
- Listed required Python packages in: requirements.txt
- Added a Dockerfile: Dockerfile

GitHub repo here: **https://github.com/alishehzad-943/k8s-project/tree/main**

4. Build and Load Docker Image into KIND

- Built the Docker image for the Flask app using the Dockerfile.
- Loaded the built Docker image into the KIND cluster manually because KIND runs inside Docker and cannot pull images from the EC2 host automatically.
- Confirmed image availability inside the cluster.

5. Kubernetes Namespace Creation

- A dedicated Kubernetes namespace was created to logically separate the Flask application resources.
- **File Name: namespace.yml**
- The namespace defined in this file is: **flask-namespace**
- This ensures clean isolation of Pods, Services, and Deployments belonging to the project.

6. Kubernetes Deployment Setup

After preparing the Deployment manifest file (**flask-deployment.yml**), the Deployment configuration was applied to the Kubernetes cluster.

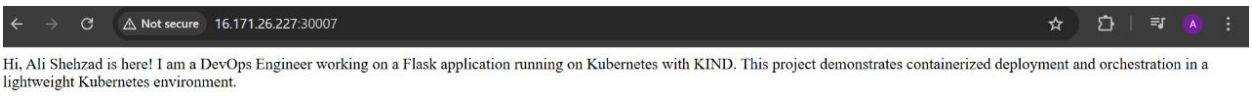
7. Kubernetes Service Setup

After preparing the Service manifest file (**flask-service.yml**) using the NodePort type for external access, it was applied to the Kubernetes cluster. Once applied, the Service successfully exposed the Flask application and assigned a NodePort for access from outside the cluster.

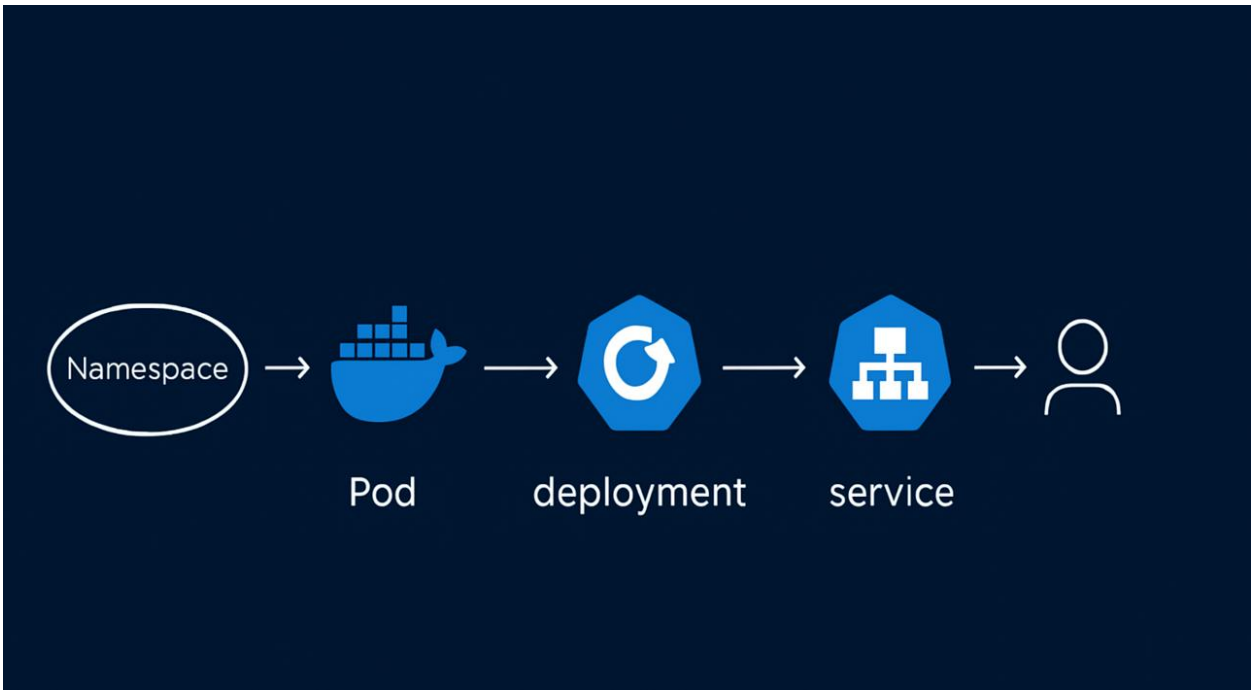
8. Application Access from Browser

- Retrieved the EC2 instance's public IP.
- Combined the IP with the NodePort exposed by the Service.
- Accessed the Flask application in the browser using: **http://16.171.26.227:30007**

Verified successful browser display.



App Working Flow:



Results

- Successfully created a fully functional Kubernetes cluster using KIND on AWS EC2.
- Deployed a Flask web app into the cluster with proper Kubernetes resources.
- Achieved external browser access via NodePort.
- Validated concepts including namespaces, deployments, pods, services, labels, and container orchestration.
- Completed end-to-end DevOps workflow from environment setup to application deployment.

Conclusion

This project demonstrates the complete lifecycle of deploying a containerized application on a Kubernetes cluster using a minimal cloud instance and lightweight cluster tool (KIND). It serves as a practical introduction to Kubernetes-based DevOps workflows, combining cloud provisioning, containerization, orchestration, and service exposure in a single exercise.

The project also highlights how Kubernetes abstracts complex infrastructure into manageable resources, making deployments consistent, repeatable, and scalable. This hands-on implementation is valuable for students and beginners aiming to strengthen their understanding of real-world DevOps practices