Vivekanand Education Society's Institute of Technology, Chembur, Mumbai,
Department of Technology,
Year: 2024-2025 (ODD SEM)

Advance DevOps Practical Examination

Name: Siddharth Jha **Div:** D15B
Roll No: 27 **Date of exam:** 24/10/2024

Case Study 7:

Problem Statement:

To provision a Kubernetes cluster using Terraform on AWS and deploy a sample application using AWS Cloud9.

Theory:

This case study delves into the process of creating and managing a Kubernetes cluster on AWS using Terraform, an Infrastructure as Code (IaC) tool. Kubernetes is a powerful open-source platform designed to automate the deployment, scaling, and management of containerized applications, providing a robust solution for orchestrating containers across clusters of hosts.

Terraform plays a crucial role in this setup by automating the provisioning of the infrastructure needed to run the Kubernetes cluster on AWS, specifically through Amazon Elastic Kubernetes Service (EKS). By defining the infrastructure as code in Terraform files, the setup becomes highly reproducible, scalable, and easy to manage. AWS services like IAM, EC2, and VPC are configured and deployed automatically using Terraform, ensuring consistency and reducing the possibility of manual errors.

Additionally, AWS Cloud9, a cloud-based integrated development environment (IDE), is leveraged for managing and interacting with the infrastructure. Cloud9 simplifies the development and management of applications by providing a fully configured environment to execute commands, develop applications, and interact with AWS services seamlessly.

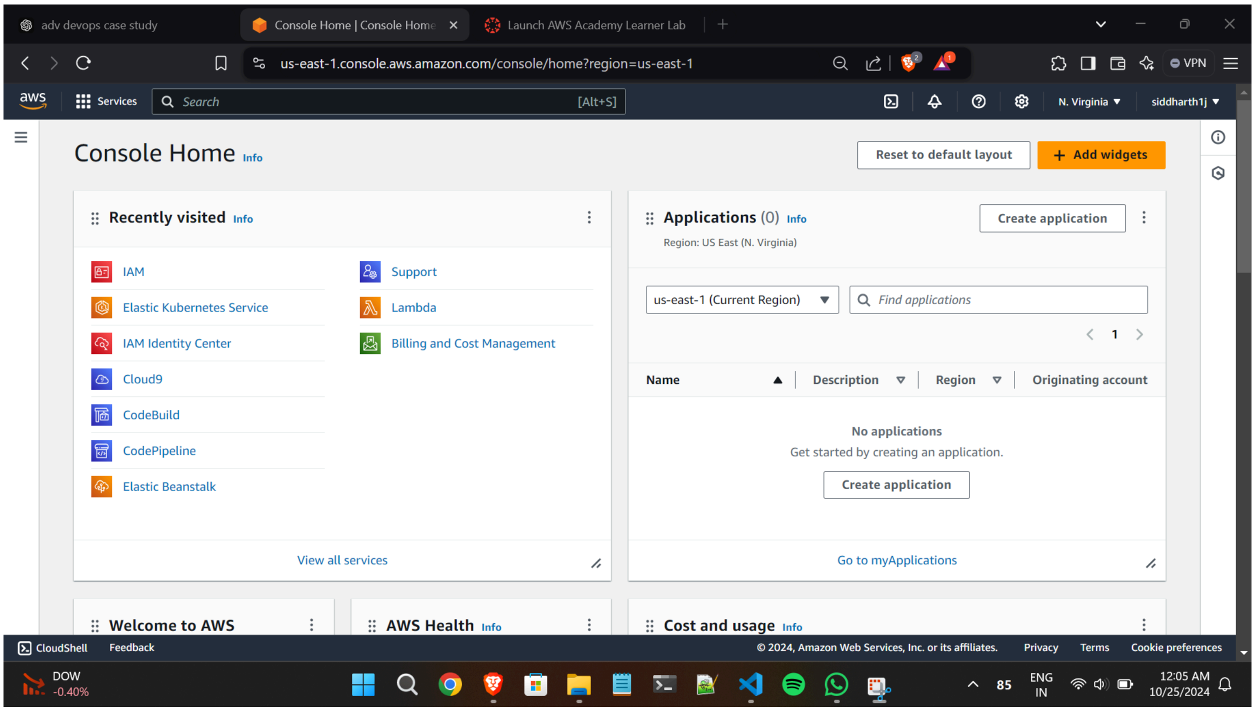
This study highlights how Terraform's integration with AWS and Kubernetes enables infrastructure scalability, container orchestration, and the implementation of modern DevOps practices such as continuous deployment, automation, and infrastructure management. By adopting such tools and practices, organizations can streamline their software development lifecycle, minimize operational overhead, and enhance infrastructure reliability.

Step-by-Step Implementation:

1. Setting Up AWS IAM User

1. **Login to AWS Management Console:**

Go to https://aws.amazon.com/console/.



2. **Navigate to IAM (Identity and Access Management):**

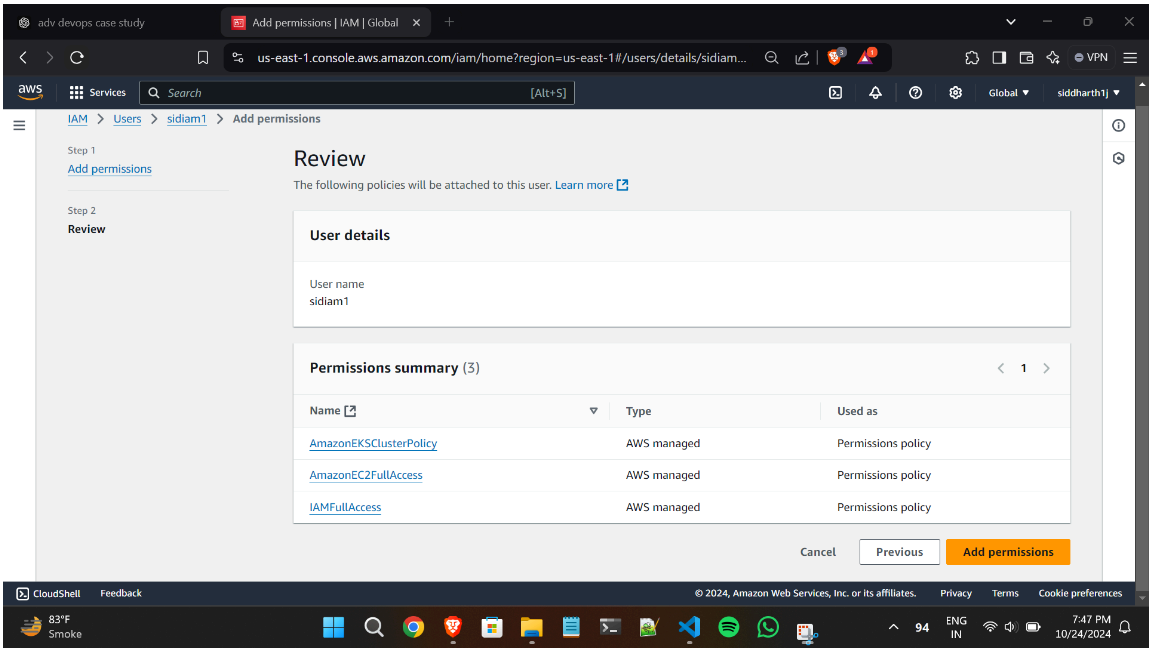
In the services menu, select **IAM**. Under **Users**, click **Add User**.

3. **Create a New IAM User:**
    - Name the user terraform-user.
    - Enable **Programmatic Access** to generate access keys.

4. **Attach Policies:**

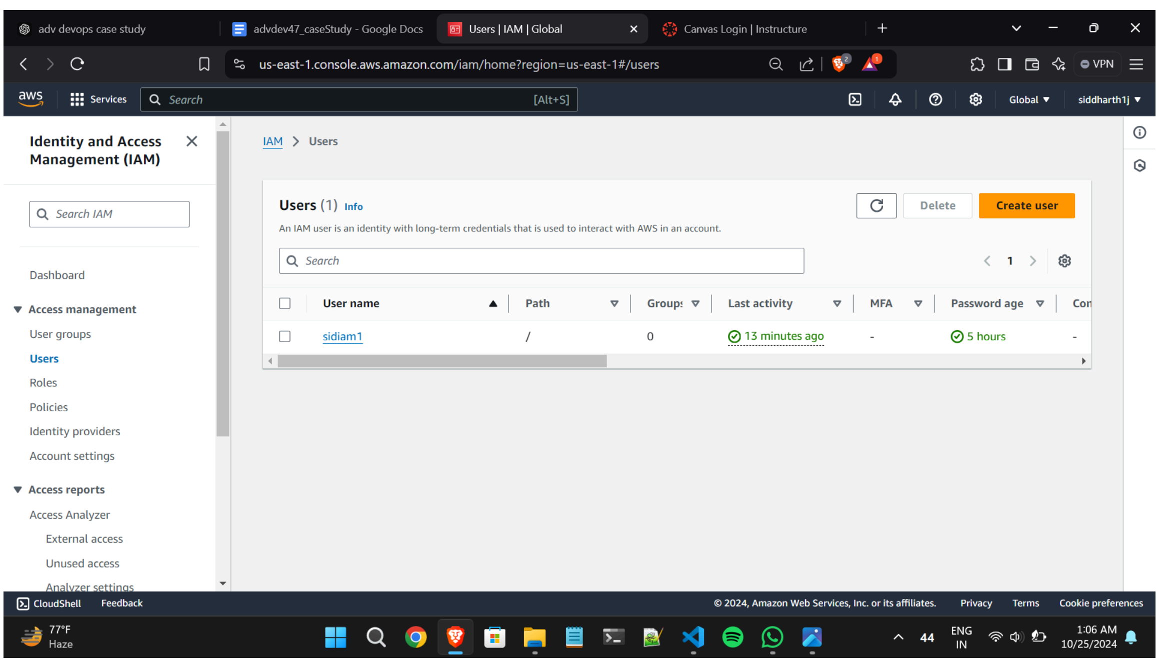
In the permissions section, select **Attach policies directly** and add the following policies:

- AmazonEKSClusterPolicy
- AmazonEKSServicePolicy
- AmazonEC2FullAccess



5. **Download Credentials:**

After creating the user, download the **Access Key ID** and **Secret Access Key**. This will be needed for configuring Terraform.



2. Creating the Terraform Script

1. **Create a Directory:**
    - Open a terminal and create a new directory for your Terraform project: mkdir eks-cluster
    - cd eks-cluster

```
C:\_home201\FOLDER J\_Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\adv-devops>mkdir eks-cluster

C:\_home201\FOLDER J\_Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\adv-devops>cd eks-cluster

C:\_home201\FOLDER J\_Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\adv-devops\eks-cluster>
```

2. **Create main.tf:**.

Add the Terraform configuration to set up the AWS provider, VPC, subnets, internet gateway, route tables, and EKS cluster:

```
provider "aws" {
  region = "us-east-1"  # Set your region
}

# VPC Configuration
resource "aws_vpc" "eks_vpc" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "eks-vpc"
  }
}

# Create 2 Public Subnets
resource "aws_subnet" "public_1" {
  vpc_id                  = aws_vpc.eks_vpc.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true

  tags = {
    Name                        = "public-us-east-1a"
    "kubernetes.io/cluster/eks" = "shared"
  }
}

resource "aws_subnet" "public_2" {
```

3. Initializing and Applying Terraform

Initialize Terraform:

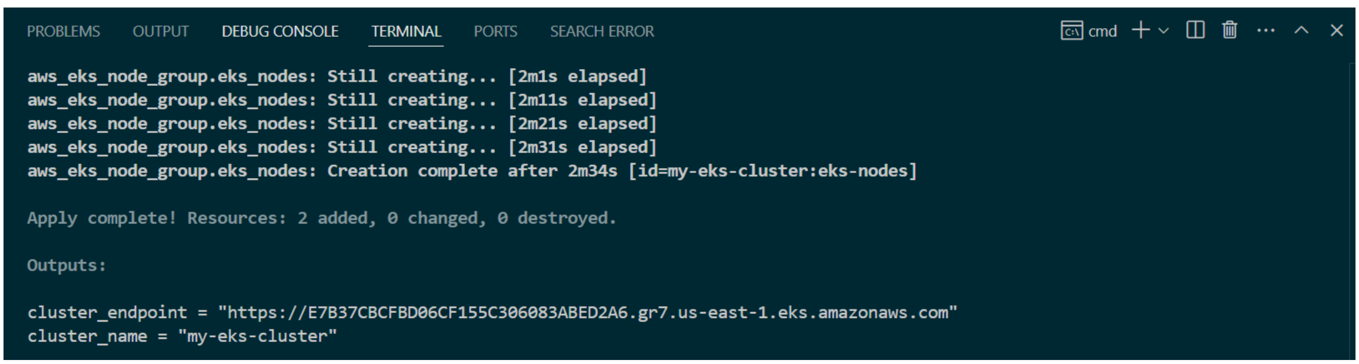
```
terraform init
```

Plan the Terraform changes:

```
Terraform plan
```

Apply the Terraform Configuration:

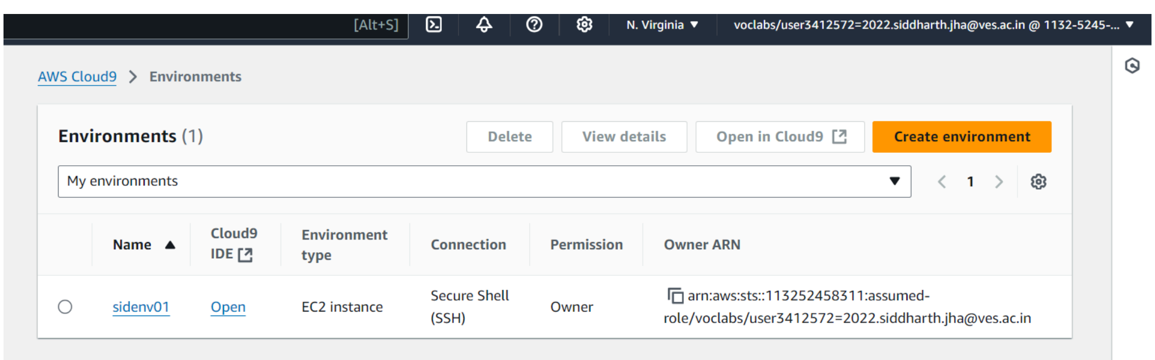
```
terraform apply
```



4. Setting Up AWS Cloud9

Create a Cloud9 Environment:

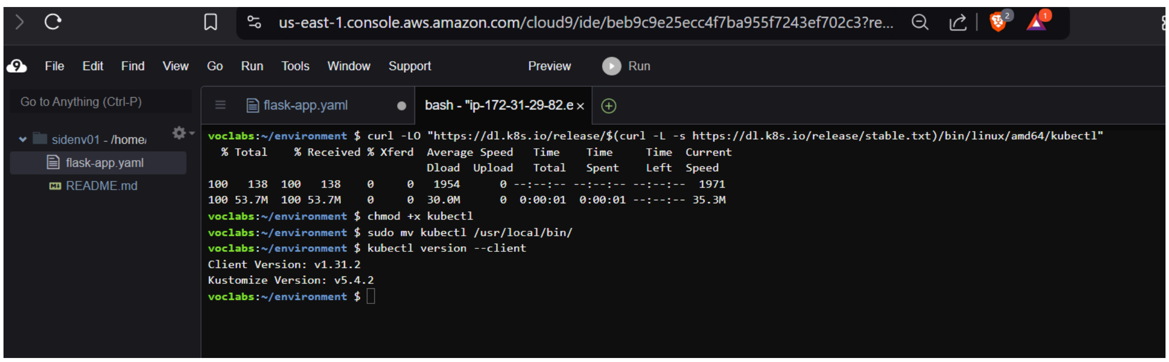
- Go to the **AWS Cloud9** console.
- Create a new Cloud9 environment, using instance type `t3.small` and default settings.



5. Configuring kubectl

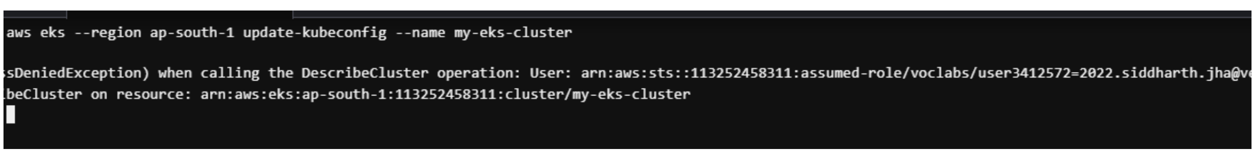
1 .**Install AWS CLI and kubectl** in the Cloud9 terminal:

```
sudo yum install -y aws-cli

curl -LO "https://amazon-eks.s3.us-west-2.amazonaws.com/$(aws eks
describe-cluster --name my-k8s-cluster --query "cluster.version"
--output text)/2020-12-03/bin/linux/amd64/kubectl"

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin
```



2. **Update kubeconfig** to interact with the Kubernetes cluster:

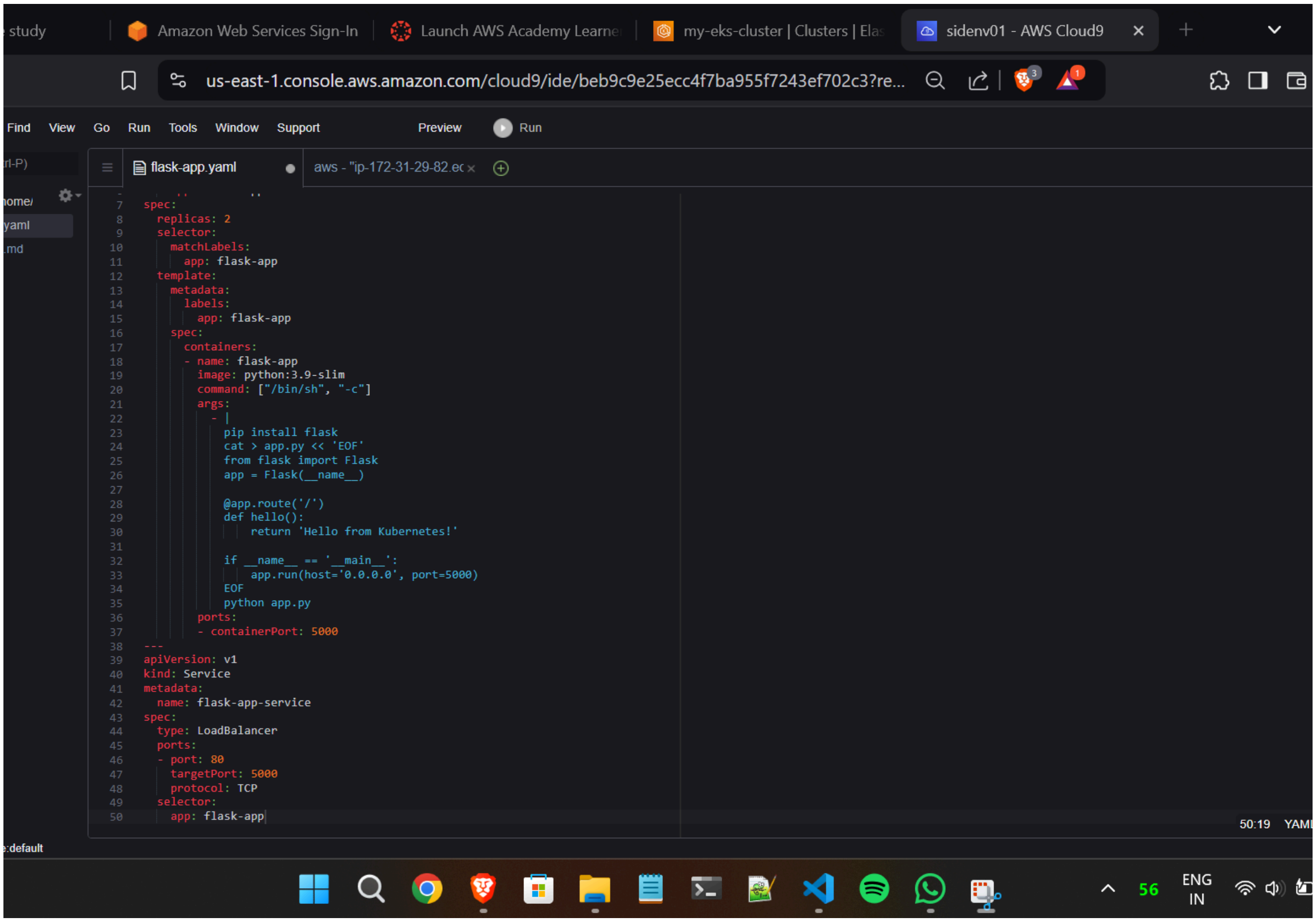
```
aws eks --region us-east-1 update-kubeconfig --name my-eks-cluster
```



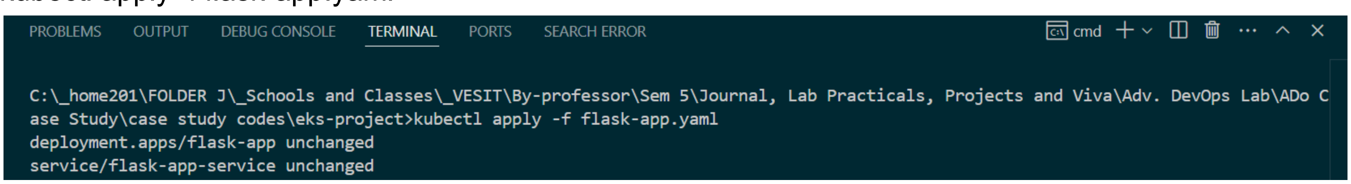
6. Creating the Flask Application

1. **Create a Deployment YAML file:**

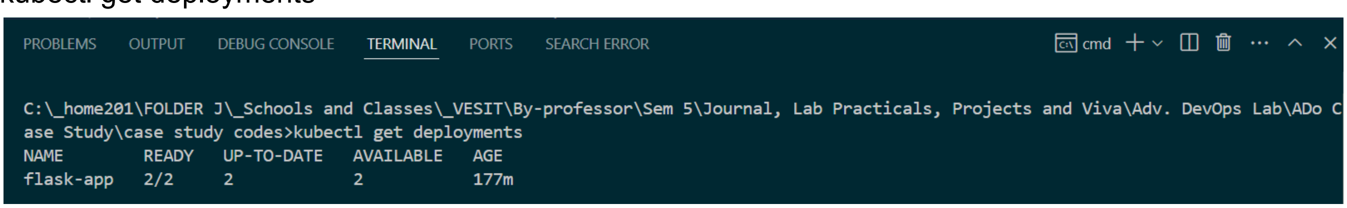
Create a file named `flask-app.yaml` for deploying the Flask application:



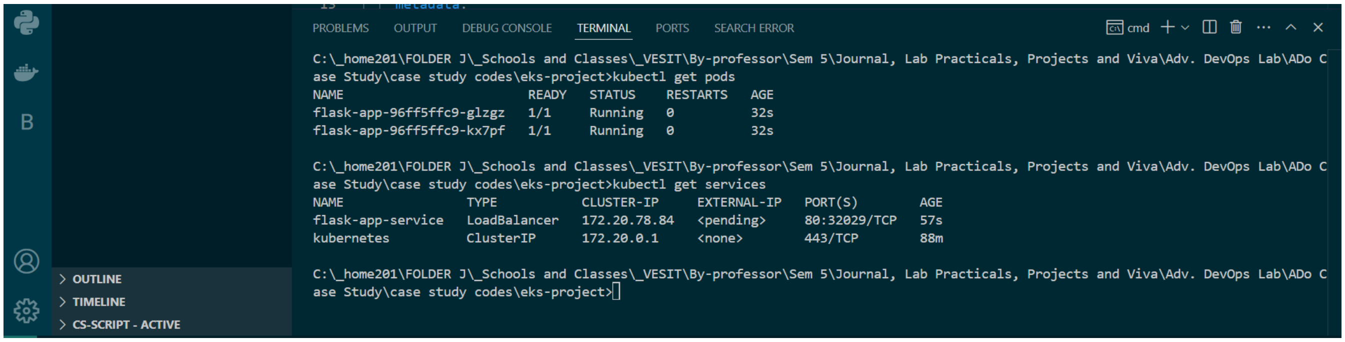
Save the manifest as flask-app.yaml
kubectl apply -f flask-app.yaml



Check the deployment status
kubectl get deployments



kubectl get pods
kubectl get services



Get the LoadBalancer URL (may take a few minutes to provision)
kubectl get service flask-app-service

4. Key Features and Applications

- **Infrastructure as Code:**
  - Terraform allows for version-controlled infrastructure definitions
  - Enables reproducible and consistent environment setup
- **Scalability:**
  - Kubernetes enables automatic scaling of applications based on demand
  - Load balancing through Kubernetes services
- **Flexibility:**
  - Deploy applications in a cloud-agnostic manner
  - Easy migration between different cloud providers

5. Conclusion

This case study illustrates the integration of Terraform, AWS, and Kubernetes to provision a scalable and manageable infrastructure for deploying applications. It emphasizes the importance of infrastructure as code in modern cloud environments. Through this implementation, we demonstrated the practical application of DevOps principles and tools in creating a production-ready container orchestration platform.