

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

Advance DevOps Practical Examination

Name: Siddharth Jha
Roll No: 27

Div: D15B
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.

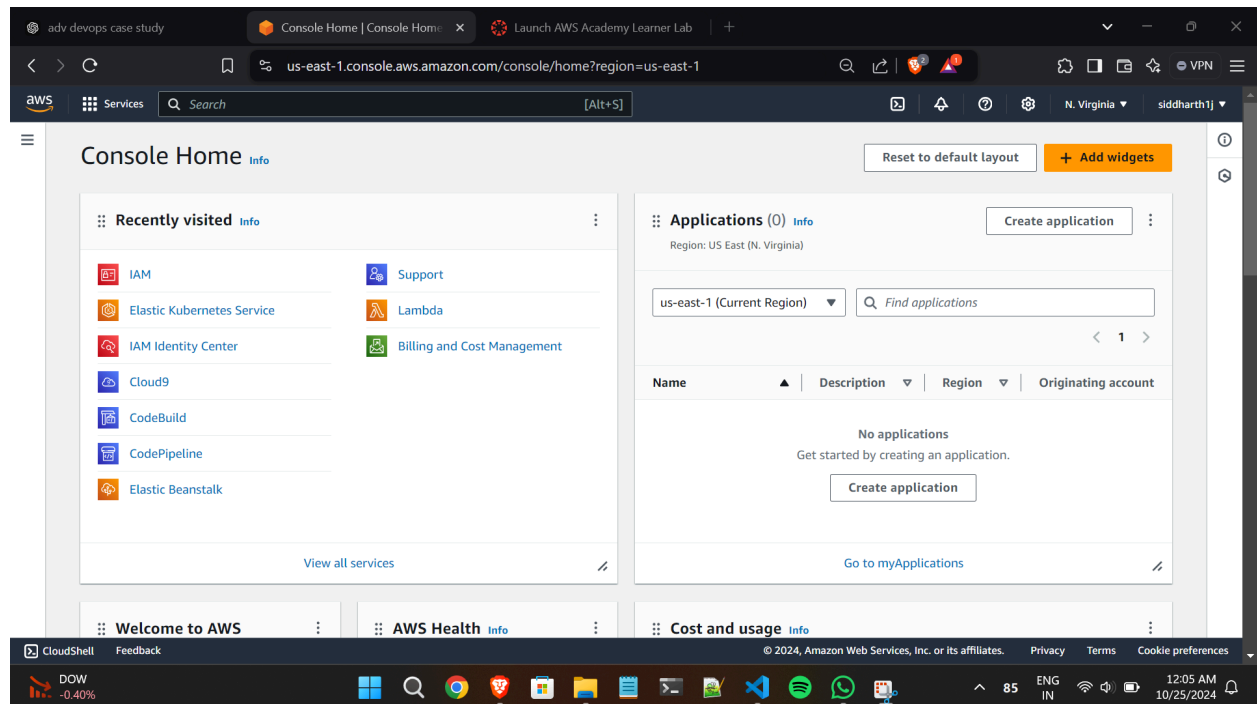
This study highlights how Terraform's integration with AWS and Kubernetes enables infrastructure scalability, container orchestration, and the implementation of modern DevOps practices such as continuous deployment, automation, and infrastructure management. By adopting such tools and practices, organizations 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**.

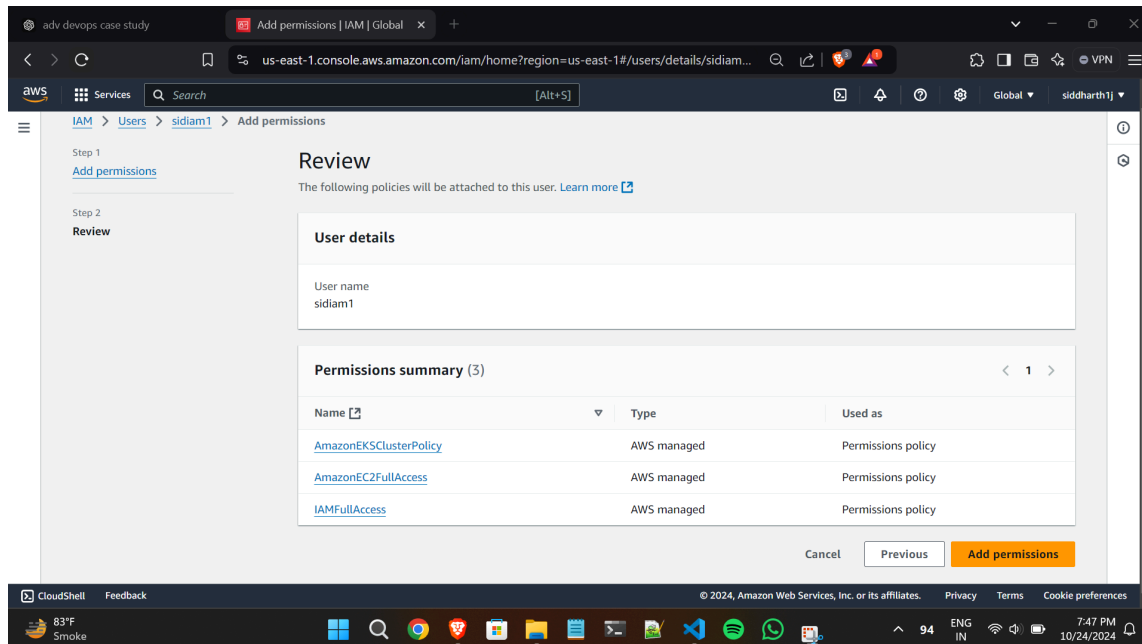
3. Create a New IAM User:

- Name the user terraform-user.
- Enable **Programmatic Access** to generate access keys.

4. Attach Policies:

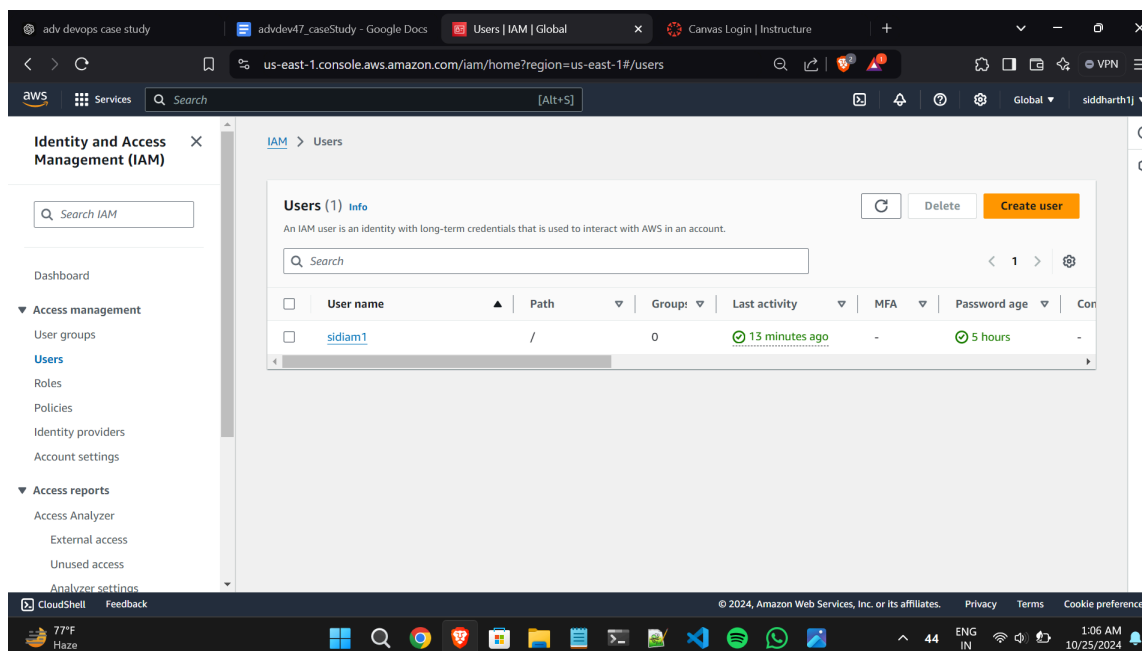
In the permissions section, select **Attach policies directly** and add the following policies:

- AmazonEKSClusterPolicy
- AmazonEKSServicePolicy
- AmazonEC2FullAccess



5. Download Credentials:

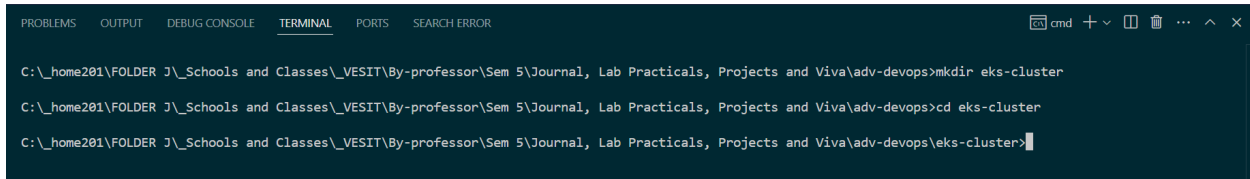
After creating the user, download the **Access Key ID** and **Secret Access Key**. This will be needed for configuring Terraform.



2. Creating the Terraform Script

1. Create a Directory:

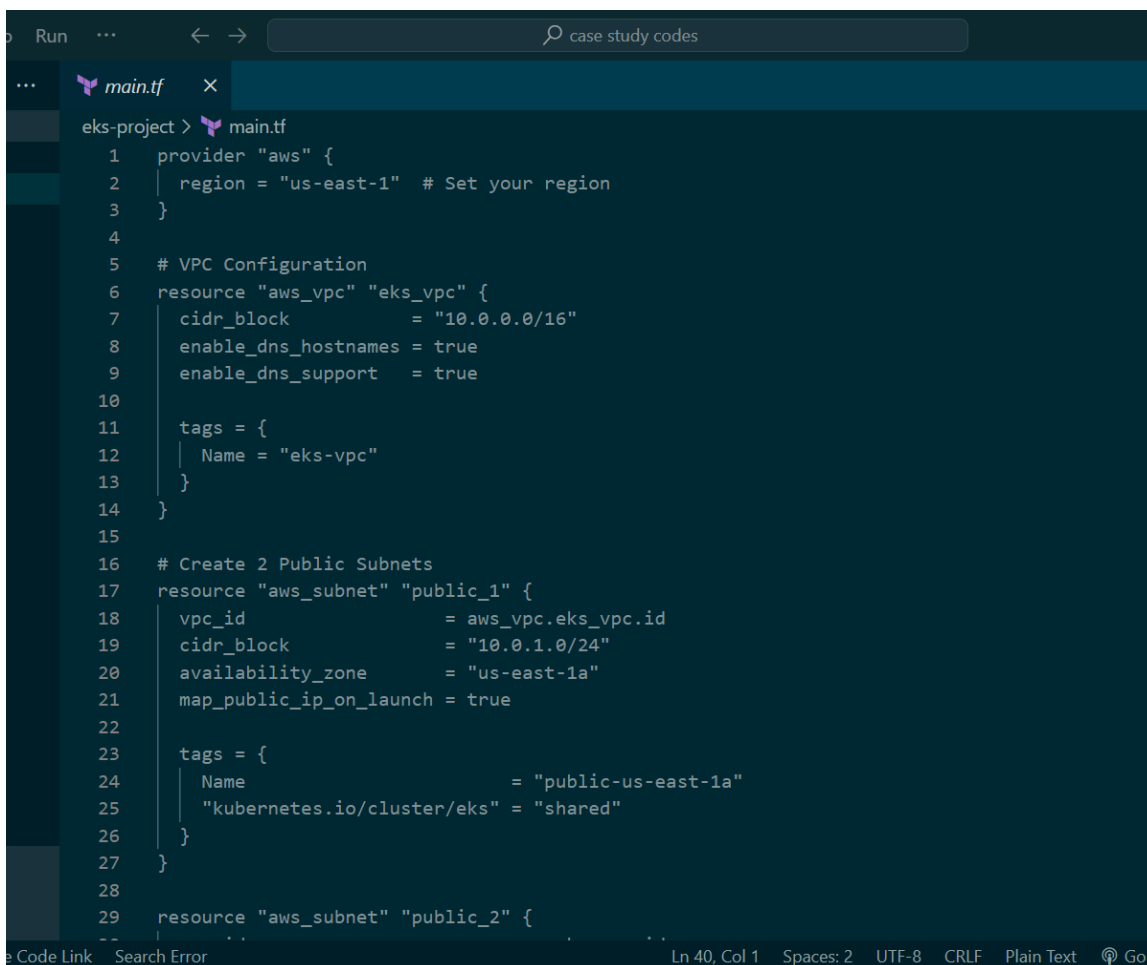
- Open a terminal and create a new directory for your Terraform project:
mkdir eks-cluster
- cd eks-cluster



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR
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:



```
Run ... < -> case study codes
... main.tf x
eks-project > main.tf
1 provider "aws" {
2   | region = "us-east-1" # Set your region
3 }
4
5 # VPC Configuration
6 resource "aws_vpc" "eks_vpc" {
7   | cidr_block           = "10.0.0.0/16"
8   | enable_dns_hostnames = true
9   | enable_dns_support   = true
10
11   | tags = {
12   |   | Name = "eks-vpc"
13   | }
14 }
15
16 # Create 2 Public Subnets
17 resource "aws_subnet" "public_1" {
18   | vpc_id             = aws_vpc.eks_vpc.id
19   | cidr_block          = "10.0.1.0/24"
20   | availability_zone   = "us-east-1a"
21   | map_public_ip_on_launch = true
22
23   | tags = {
24   |   | Name = "public-us-east-1a"
25   |   | "kubernetes.io/cluster/eks" = "shared"
26   | }
27 }
28
29 resource "aws_subnet" "public_2" {
```

3. Initializing and Applying Terraform

Initialize Terraform:

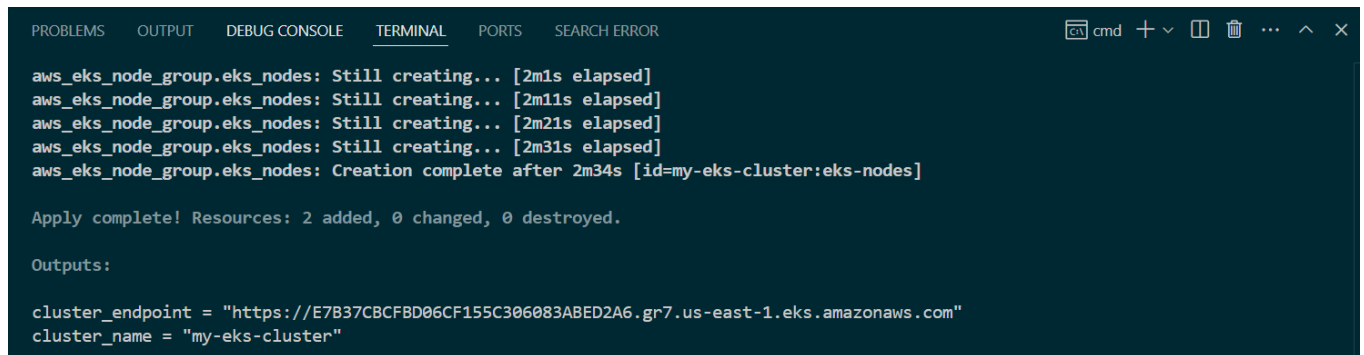
```
terraform init
```

Plan the Terraform changes:

```
Terraform plan
```

Apply the Terraform Configuration:

```
terraform apply
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SEARCH ERROR
aws_eks_node_group.eks_nodes: Still creating... [2m1s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [2m11s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [2m21s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [2m31s elapsed]
aws_eks_node_group.eks_nodes: Creation complete after 2m34s [id=my-eks-cluster:eks-nodes]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

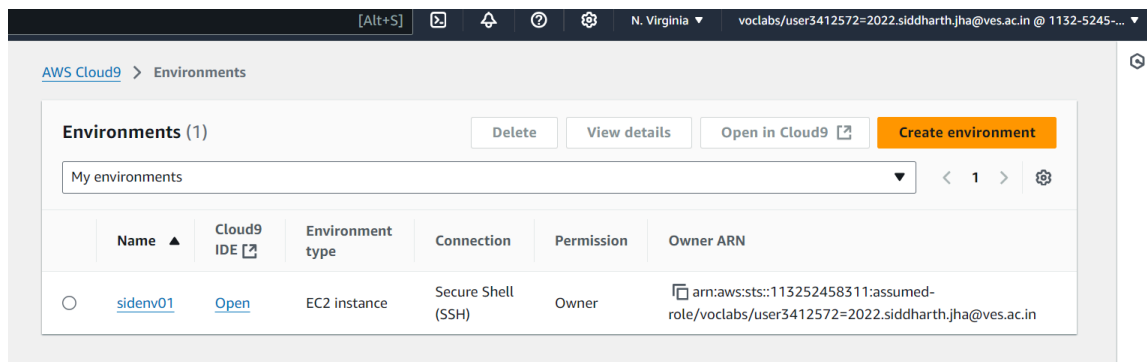
Outputs:

cluster_endpoint = "https://E7B37CBFCFBD06CF155C306083ABED2A6.gr7.us-east-1.eks.amazonaws.com"
cluster_name = "my-eks-cluster"
```

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

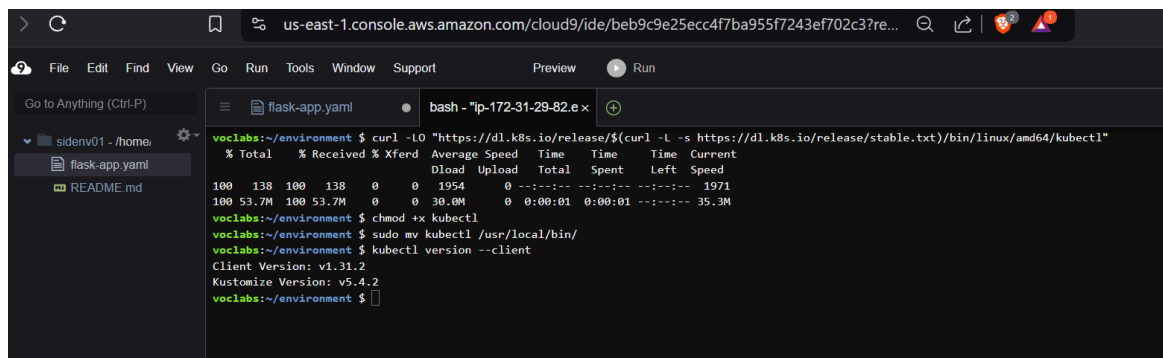
1 .Install AWS CLI and kubectl in the Cloud9 terminal:

```
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
```

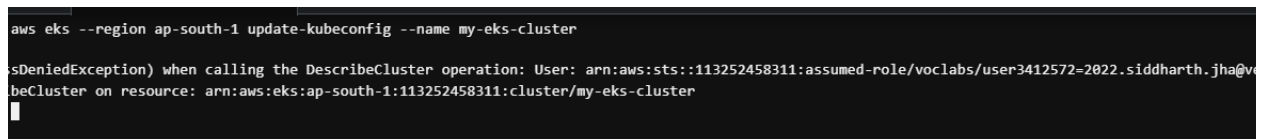
```
sudo mv ./kubectl /usr/local/bin
```



```
us-east-1.console.aws.amazon.com/cloud9/ide/beb9c9e25ecc4f7ba955f7243ef702c3?re...
File Edit Find View Go Run Tools Window Support Preview Run
Go to Anything (Ctrl-P)
sidenv01 - /home
flask-app.yaml
README.md
voclabs:~/environment $ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 138 100 138 0 0 1954 0 0:00:01 0:00:01 0:00:01 1971
100 53.7M 100 53.7M 0 0 30.0M 0 0:00:01 0:00:01 0:00:01 35.3M
voclabs:~/environment $ chmod +x kubectl
voclabs:~/environment $ sudo mv kubectl /usr/local/bin/
voclabs:~/environment $ kubectl version --client
Client Version: v1.31.2
Kustomize Version: v5.4.2
voclabs:~/environment $
```

2. Update kubeconfig to interact with the Kubernetes cluster:

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



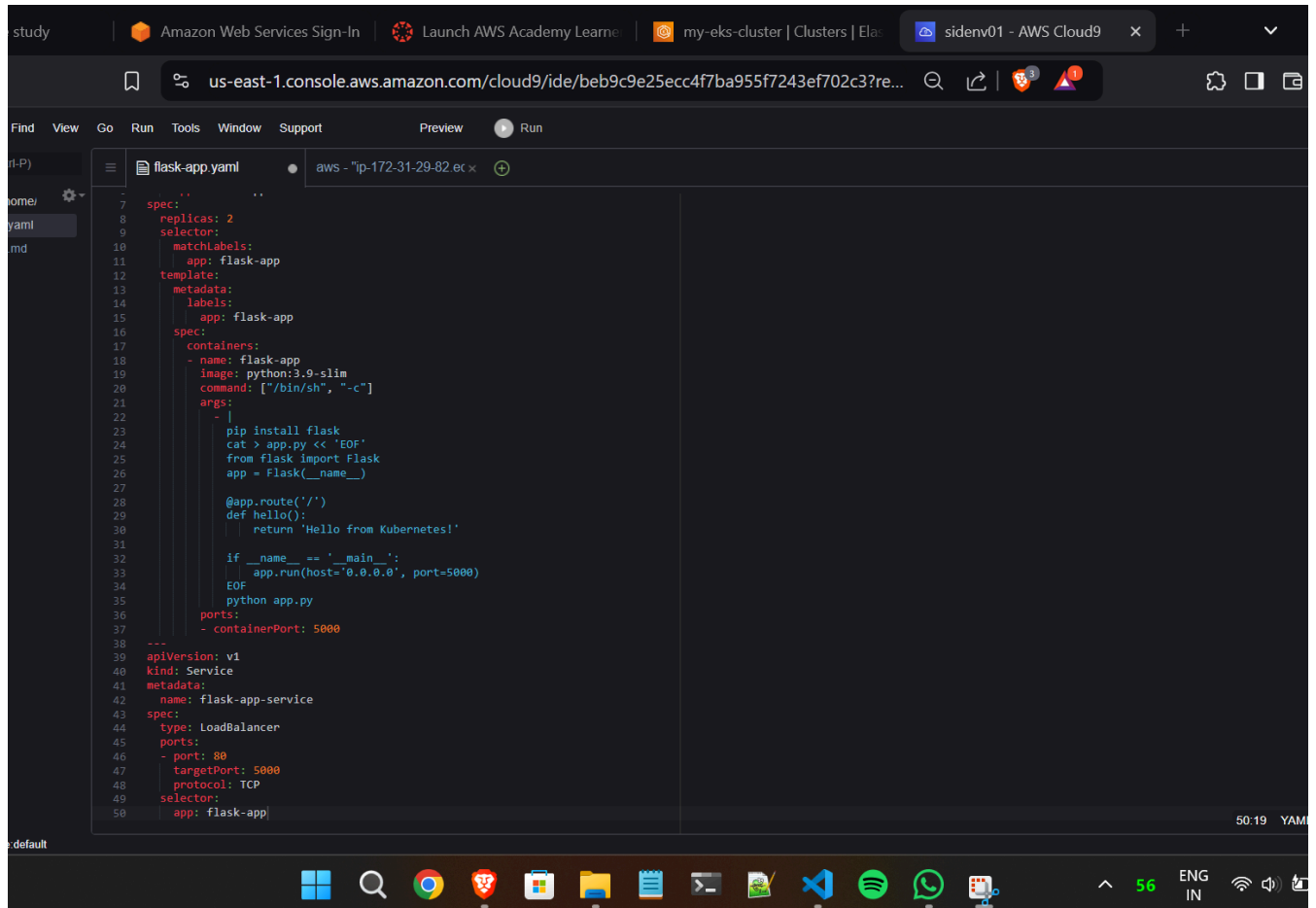
```
aws eks --region ap-south-1 update-kubeconfig --name my-eks-cluster

sDeniedException) when calling the DescribeCluster operation: User: arn:aws:sts::113252458311:assumed-role/voclabs/user3412572-2022.siddharth.jha@vo
beCluster on resource: arn:aws:eks:ap-south-1:113252458311:cluster/my-eks-cluster
```

6. Creating the Flask Application

1. Create a Deployment YAML file:

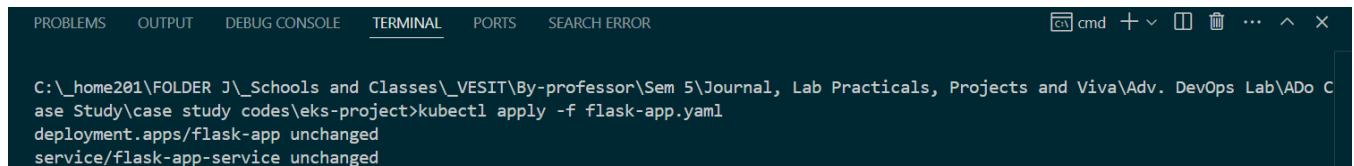
Create a file named `flask-app.yaml` for deploying the Flask application:



```
1 spec:
2   replicas: 2
3   selector:
4     matchLabels:
5       app: flask-app
6   template:
7     metadata:
8       labels:
9         app: flask-app
10    spec:
11      containers:
12        - name: flask-app
13          image: python:3.9-slim
14          command: ["/bin/sh", "-c"]
15          args:
16            - |
17              pip install flask
18              cat > app.py << 'EOF'
19              from flask import Flask
20              app = Flask(__name__)
21
22              @app.route('/')
23              def hello():
24                  return 'Hello from Kubernetes!'
25
26              if __name__ == '__main__':
27                  app.run(host='0.0.0.0', port=5000)
28              EOF
29              python app.py
30          ports:
31            - containerPort: 5000
32
33  ---
34  apiVersion: v1
35  kind: Service
36  metadata:
37    name: flask-app-service
38  spec:
39    type: LoadBalancer
40    ports:
41      - port: 80
42        targetPort: 5000
43        protocol: TCP
44    selector:
45      app: flask-app
```

Save the manifest as flask-app.yaml

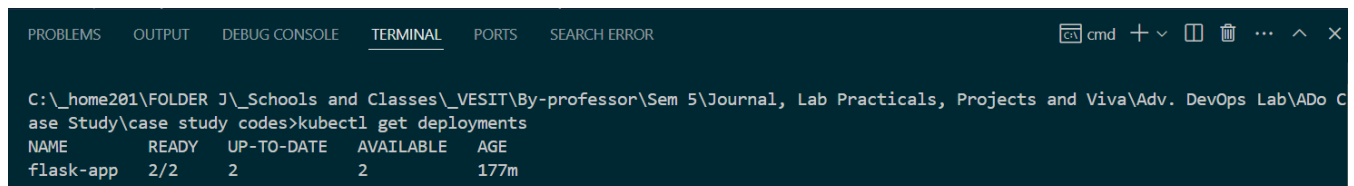
kubectl apply -f flask-app.yaml



```
C:\_home201\FOLDER J\Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\Adv. DevOps Lab\ADO C
ase Study\case study codes\eks-project>kubectl apply -f flask-app.yaml
deployment.apps/flask-app unchanged
service/flask-app-service unchanged
```

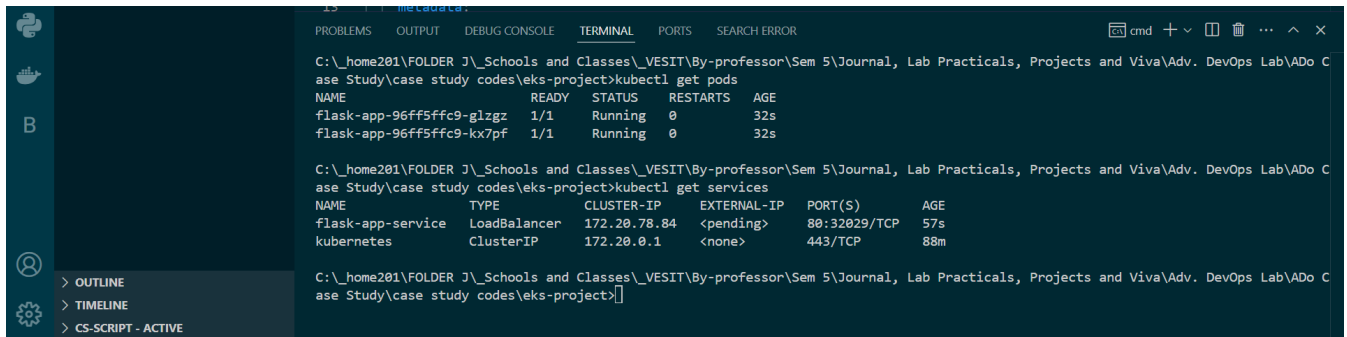
Check the deployment status

kubectl get deployments



```
C:\_home201\FOLDER J\Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\Adv. DevOps Lab\ADO C
ase Study\case study codes>kubectl get deployments
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
flask-app  2/2     2            2           177m
```

kubectl get pods
kubectl get services



```
C:\_home201\FOLDER J\_Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\Adv. DevOps Lab\ADO C
ase Study\case study codes\eks-project>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
flask-app-96ff5ffc9-glzgk           1/1     Running   0           32s
flask-app-96ff5ffc9-kx7pf           1/1     Running   0           32s

C:\_home201\FOLDER J\_Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\Adv. DevOps Lab\ADO C
ase Study\case study codes\eks-project>kubectl get services
NAME          TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
flask-app-service  LoadBalancer  172.20.78.84  <pending>      80:32029/TCP     57s
kubernetes      ClusterIP      172.20.0.1    <none>         443/TCP           88m

C:\_home201\FOLDER J\_Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\Adv. DevOps Lab\ADO C
ase Study\case study codes\eks-project>]
```

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



```
C:\_home201\FOLDER J\_Schools and Classes\_VESIT\By-professor\Sem 5\Journal, Lab Practicals, Projects and Viva\Adv. DevOps Lab\ADO C
ase Study\case study codes\eks-project>kubectl get service flask-app-service
NAME          TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
flask-app-service  LoadBalancer  172.20.78.84  <pending>      80:32029/TCP     152m
```

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.