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

Advance DevOps Practical Examination

Name: Nidhi Pednekar **Div:** D15B
Roll No: 47 **Date of exam:** 24/10/2024

Case Study 7:

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

Theory:
This case study explores the process of creating and managing a Kubernetes cluster on AWS using Terraform. Kubernetes is an open-source platform designed to automate deploying, scaling, and operating application containers. Terraform, a popular infrastructure as code tool, is used to set up the underlying infrastructure, while AWS Cloud9 IDE is utilized for developing and managing the application. The study demonstrates how this integration enables infrastructure scalability and container orchestration, showcasing modern DevOps practices.

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**.

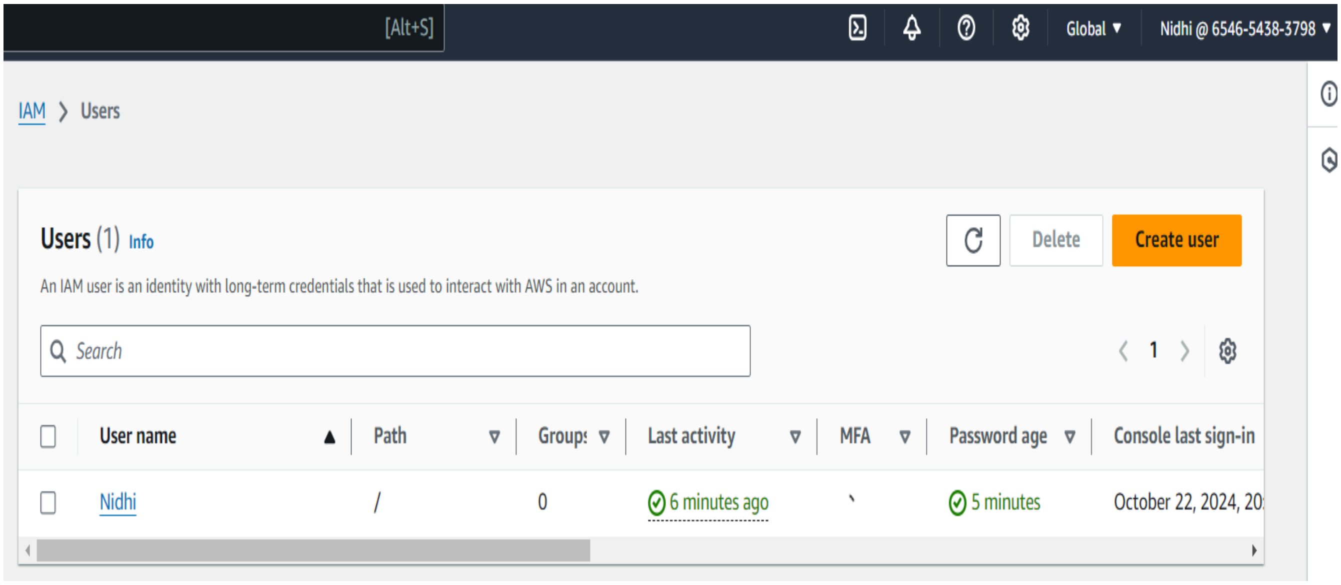
3. **Create a New IAM User:**
   - Name the user terraform-user.
   - Enable **Programmatic Access** to generate access keys.
4. **Attach Policies:**

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.



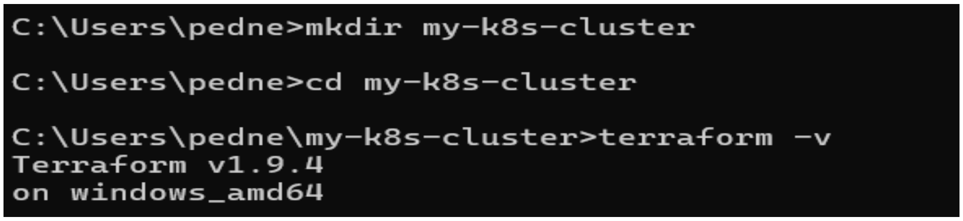
Users (1) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

	User name ▲	Path	Group:	Last activity	MFA	Password age	Console last sign-in
□	Nidhi	/	0	✓ 6 minutes ago	-	✓ 5 minutes	October 22, 2024, 20

2. Creating the Terraform Script

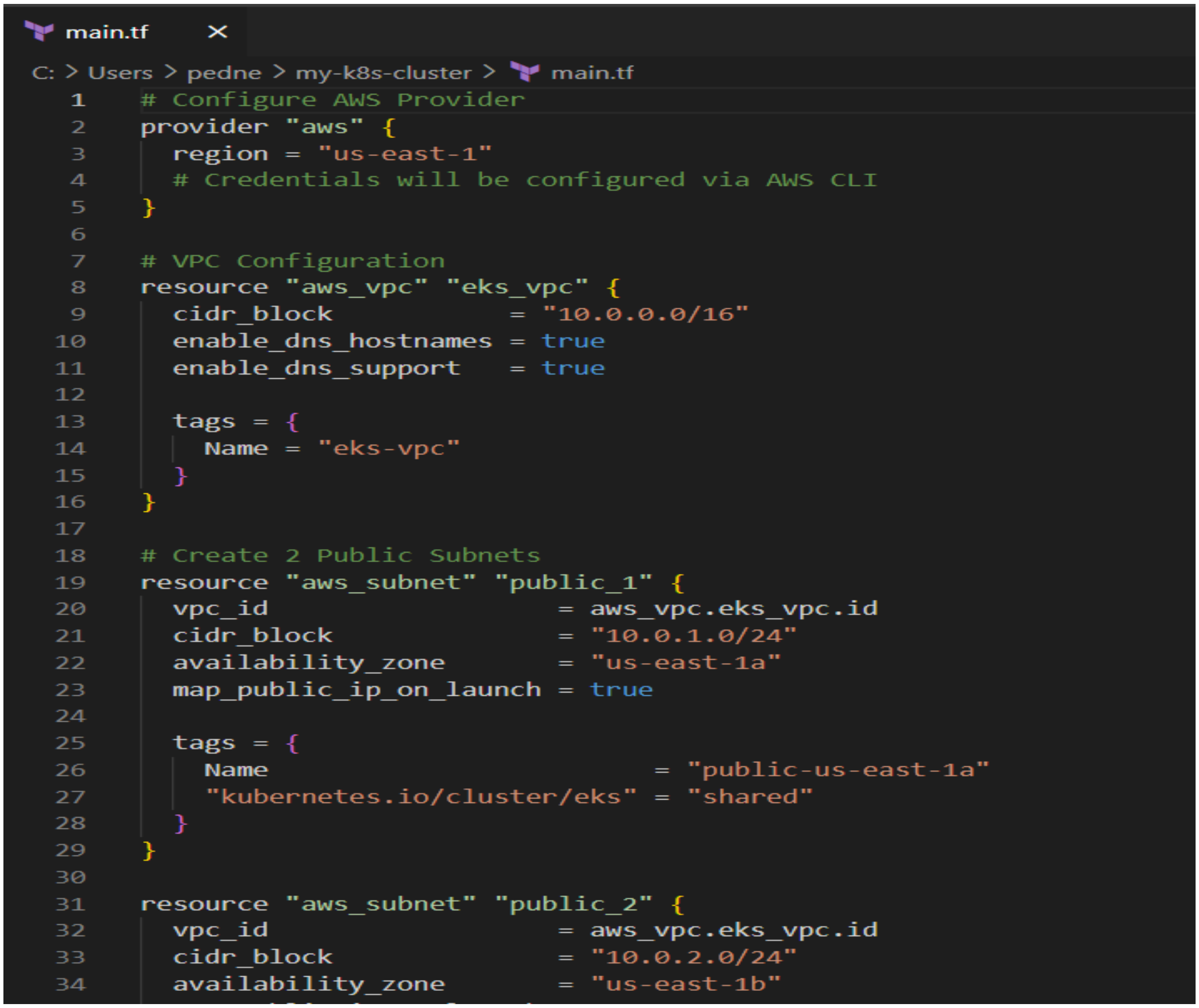
1. **Create a Directory:**
   - Open a terminal and create a new directory for your Terraform project: mkdir my-k8s-cluster
   - cd my-k8s-cluster



```
C:\Users\pedne>mkdir my-k8s-cluster

C:\Users\pedne>cd my-k8s-cluster

C:\Users\pedne\my-k8s-cluster>terraform -v
Terraform v1.9.4
on windows_amd64
```

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

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



```
# Configure AWS Provider
provider "aws" {
  region = "us-east-1"
  # Credentials will be configured via AWS CLI
}

# 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" {
  vpc_id                  = aws_vpc.eks_vpc.id
  cidr_block              = "10.0.2.0/24"
  availability_zone       = "us-east-1b"
```

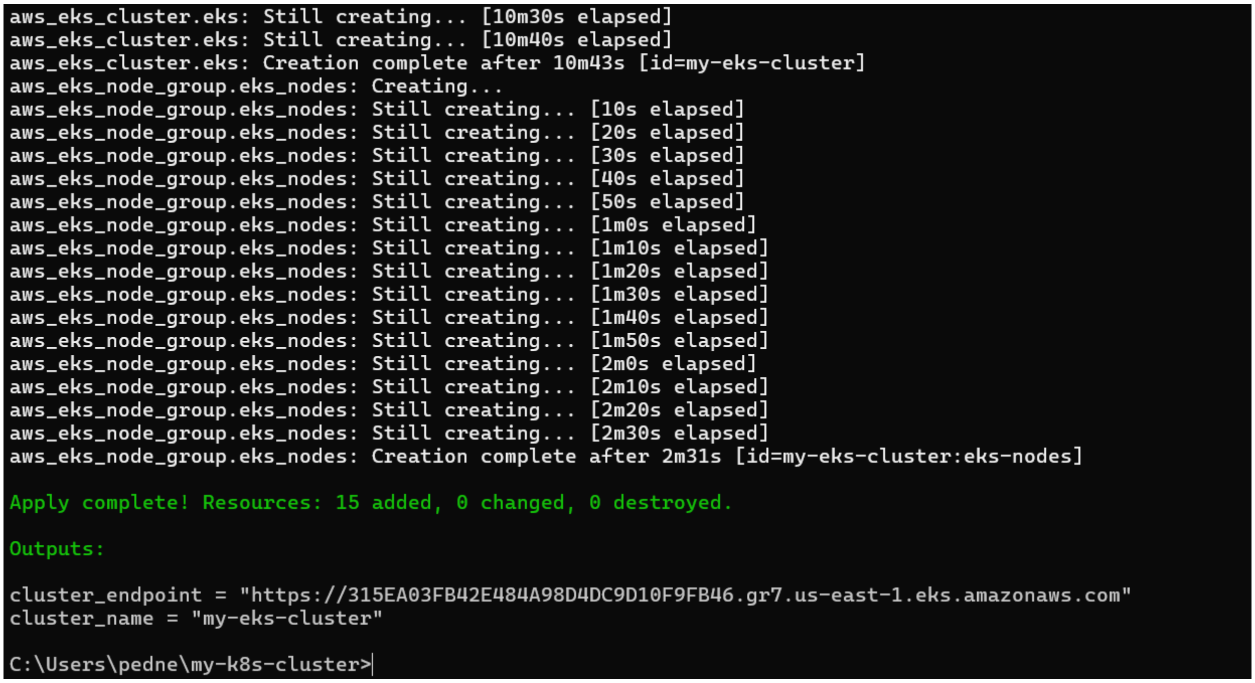
3. Initializing and Applying Terraform

Initialize Terraform:

```
terraform init
```

Apply the Terraform Configuration:

```
terraform apply
```

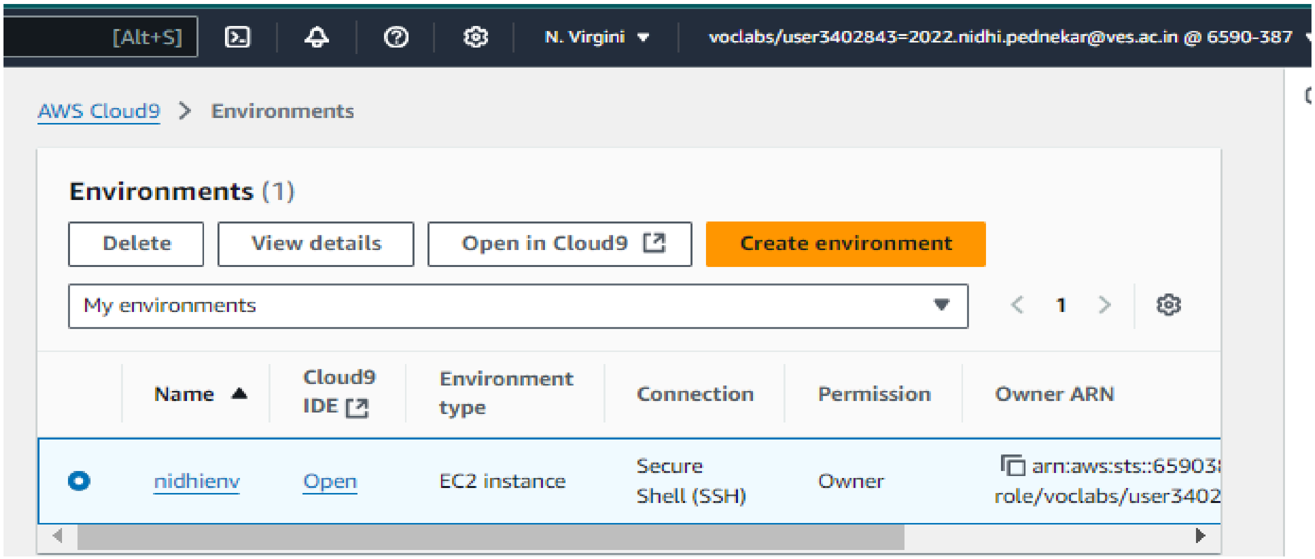


```
aws_eks_cluster.eks: Still creating... [10m30s elapsed]
aws_eks_cluster.eks: Still creating... [10m40s elapsed]
aws_eks_cluster.eks: Creation complete after 10m43s [id=my-eks-cluster]
aws_eks_node_group.eks_nodes: Creating...
aws_eks_node_group.eks_nodes: Still creating... [10s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [20s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [30s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [40s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [50s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [1m0s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [1m10s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [1m20s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [1m30s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [1m40s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [1m50s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [2m0s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [2m10s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [2m20s elapsed]
aws_eks_node_group.eks_nodes: Still creating... [2m30s elapsed]
aws_eks_node_group.eks_nodes: Creation complete after 2m31s [id=my-eks-cluster:eks-nodes]

Apply complete! Resources: 15 added, 0 changed, 0 destroyed.

Outputs:

cluster_endpoint = "https://315EA03FB42E484A98D4DC9D10F9FB46.gr7.us-east-1.eks.amazonaws.com"
cluster_name = "my-eks-cluster"

C:\Users\pedne\my-k8s-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.



[Alt+S]
N. Virgini ▼
voclabs/user3402843=2022.nidhi.pednekar@ves.ac.in @ 6590-387
AWS Cloud9 > Environments
Delete
View details
Open in Cloud9
Create environment
My environments
1

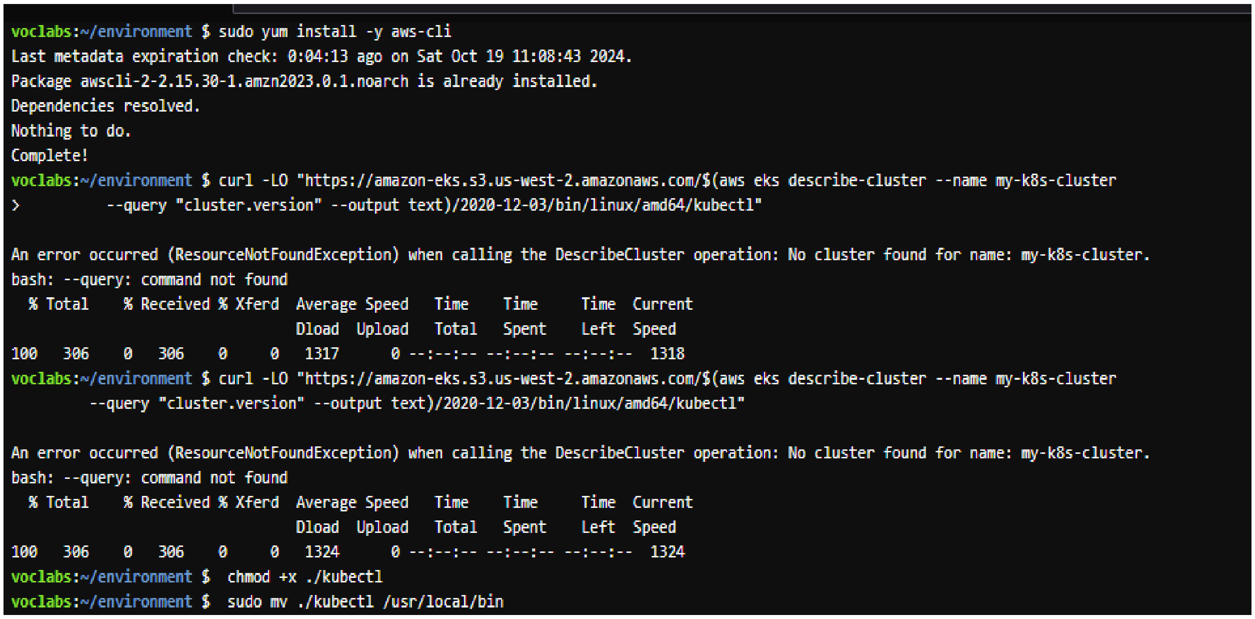
Environments (1)

	Name ▲	Cloud9 IDE	Environment type	Connection	Permission	Owner ARN
●	nidhienv	Open	EC2 instance	Secure Shell (SSH)	Owner	arn:aws:sts::65903 role/voclabs/user3402

5. Configuring kubectl

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

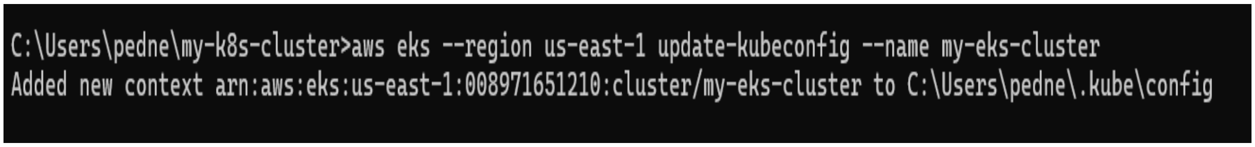
```
sudo yum install -y aws-cli

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
```



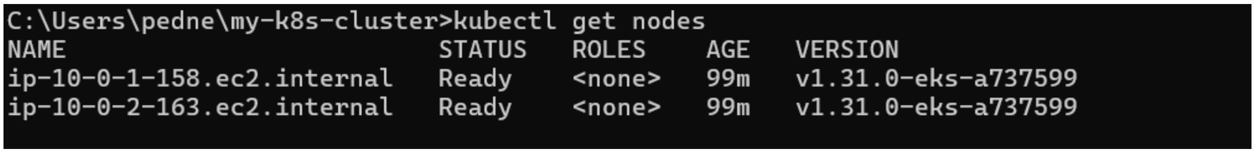
```
voclabs:~/environment $ sudo yum install -y aws-cli
Last metadata expiration check: 0:04:13 ago on Sat Oct 19 11:08:43 2024.
Package awscli-2-2.15.30-1.amzn2023.0.1.noarch is already installed.
Dependencies resolved.
Nothing to do.
Complete!
voclabs:~/environment $ curl -LO "https://amazon-eks.s3.us-west-2.amazonaws.com/$(aws eks describe-cluster --name my-k8s-cluster
>          --query "cluster.version" --output text)/2020-12-03/bin/linux/amd64/kubectl"

An error occurred (ResourceNotFoundException) when calling the DescribeCluster operation: No cluster found for name: my-k8s-cluster.
bash: --query: command not found
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   306    0   306    0     0   1317      0 --:--:-- --:--:-- --:--:--  1318
voclabs:~/environment $ 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"

An error occurred (ResourceNotFoundException) when calling the DescribeCluster operation: No cluster found for name: my-k8s-cluster.
bash: --query: command not found
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   306    0   306    0     0   1324      0 --:--:-- --:--:-- --:--:--  1324
voclabs:~/environment $  chmod +x ./kubectl
voclabs:~/environment $  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
```



```
C:\Users\pedne\my-k8s-cluster>aws eks --region us-east-1 update-kubeconfig --name my-eks-cluster
Added new context arn:aws:eks:us-east-1:008971651210:cluster/my-eks-cluster to C:\Users\pedne\.kube\config
```

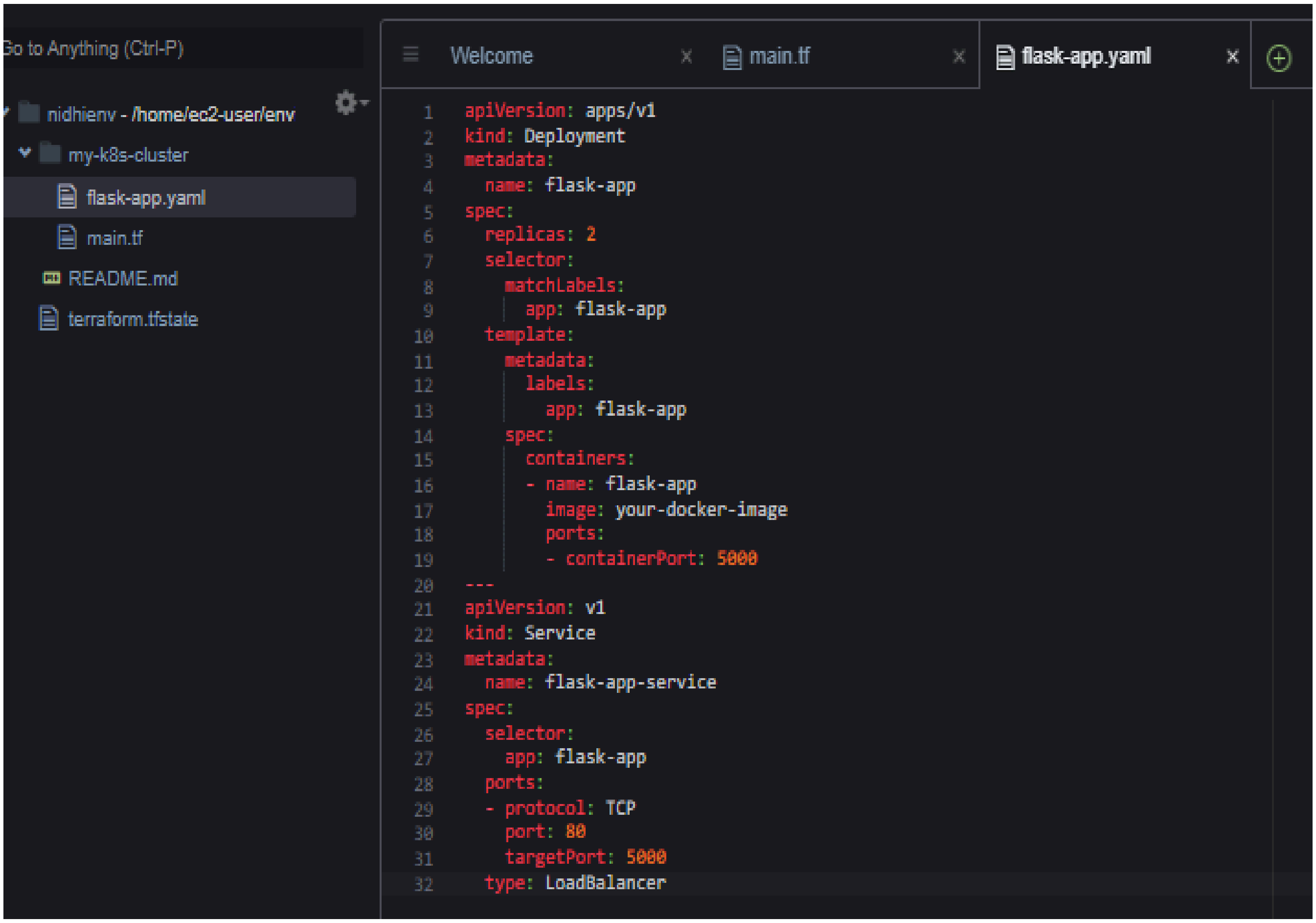


```
C:\Users\pedne\my-k8s-cluster>kubectl get nodes
NAME                         STATUS   ROLES    AGE   VERSION
ip-10-0-1-158.ec2.internal   Ready    <none>   99m   v1.31.0-eks-a737599
ip-10-0-2-163.ec2.internal   Ready    <none>   99m   v1.31.0-eks-a737599
```

6. Creating the Flask Application

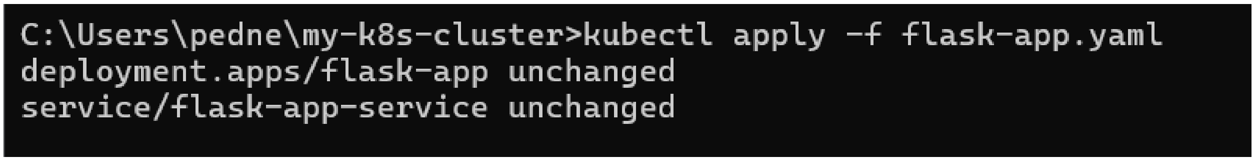
1. **Create a Deployment YAML file:**

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



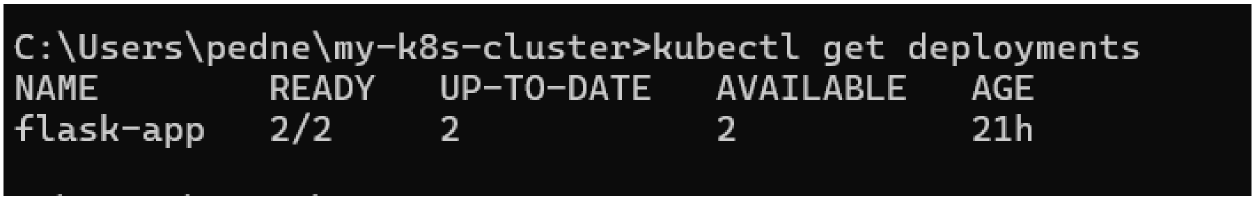
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: your-docker-image
        ports:
        - containerPort: 5000
---
apiVersion: v1
kind: Service
metadata:
  name: flask-app-service
spec:
  selector:
    app: flask-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 5000
  type: LoadBalancer
```

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



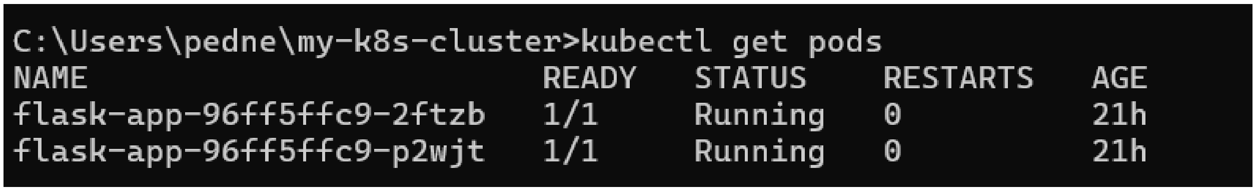
```
C:\Users\pedne\my-k8s-cluster>kubectl apply -f flask-app.yaml
deployment.apps/flask-app unchanged
service/flask-app-service unchanged
```

# Check the deployment status
kubectl get deployments



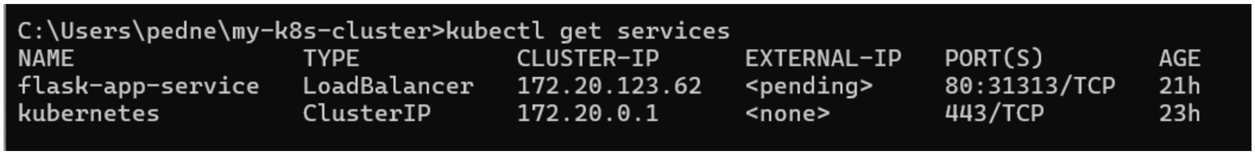
```
C:\Users\pedne\my-k8s-cluster>kubectl get deployments
NAME        READY   UP-TO-DATE   AVAILABLE   AGE
flask-app   2/2     2            2           21h
```

kubectl get pods



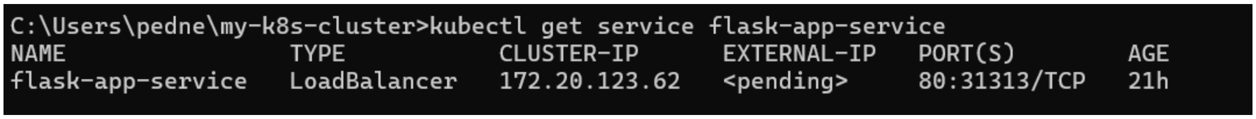
```
C:\Users\pedne\my-k8s-cluster>kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
flask-app-96ff5ffc9-2ftzb   1/1     Running   0          21h
flask-app-96ff5ffc9-p2wjt   1/1     Running   0          21h
```

kubectl get services



```
C:\Users\pedne\my-k8s-cluster>kubectl get services
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
flask-app-service   LoadBalancer   172.20.123.62   <pending>     80:31313/TCP   21h
kubernetes          ClusterIP      172.20.0.1      <none>        443/TCP        23h
```

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



```
C:\Users\pedne\my-k8s-cluster>kubectl get service flask-app-service
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
flask-app-service   LoadBalancer   172.20.123.62   <pending>     80:31313/TCP   21h
```

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.