

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

Advance DevOps Practical Examination

Name: Nidhi Pednekar
Roll No: 47

Div: D15B
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.

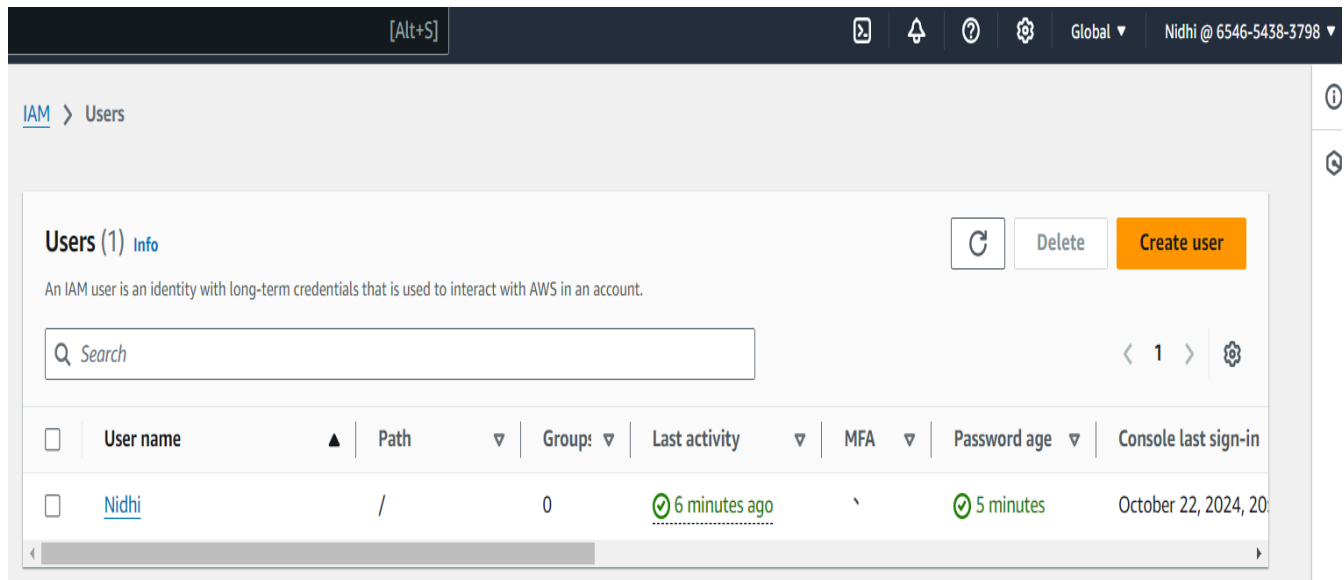
4. Attach Policies:

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.



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:

```
main.tf X
C: > Users > pedne > my-k8s-cluster > main.tf
1  # Configure AWS Provider
2  provider "aws" {
3    region = "us-east-1"
4    # Credentials will be configured via AWS CLI
5  }
6
7  # VPC Configuration
8  resource "aws_vpc" "eks_vpc" {
9    cidr_block      = "10.0.0.0/16"
10   enable_dns_hostnames = true
11   enable_dns_support  = true
12
13   tags = {
14     Name = "eks-vpc"
15   }
16 }
17
18 # Create 2 Public Subnets
19 resource "aws_subnet" "public_1" {
20   vpc_id            = aws_vpc.eks_vpc.id
21   cidr_block        = "10.0.1.0/24"
22   availability_zone = "us-east-1a"
23   map_public_ip_on_launch = true
24
25   tags = {
26     Name = "public-us-east-1a"
27     "kubernetes.io/cluster/eks" = "shared"
28   }
29 }
30
31 resource "aws_subnet" "public_2" {
32   vpc_id            = aws_vpc.eks_vpc.id
33   cidr_block        = "10.0.2.0/24"
34   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"

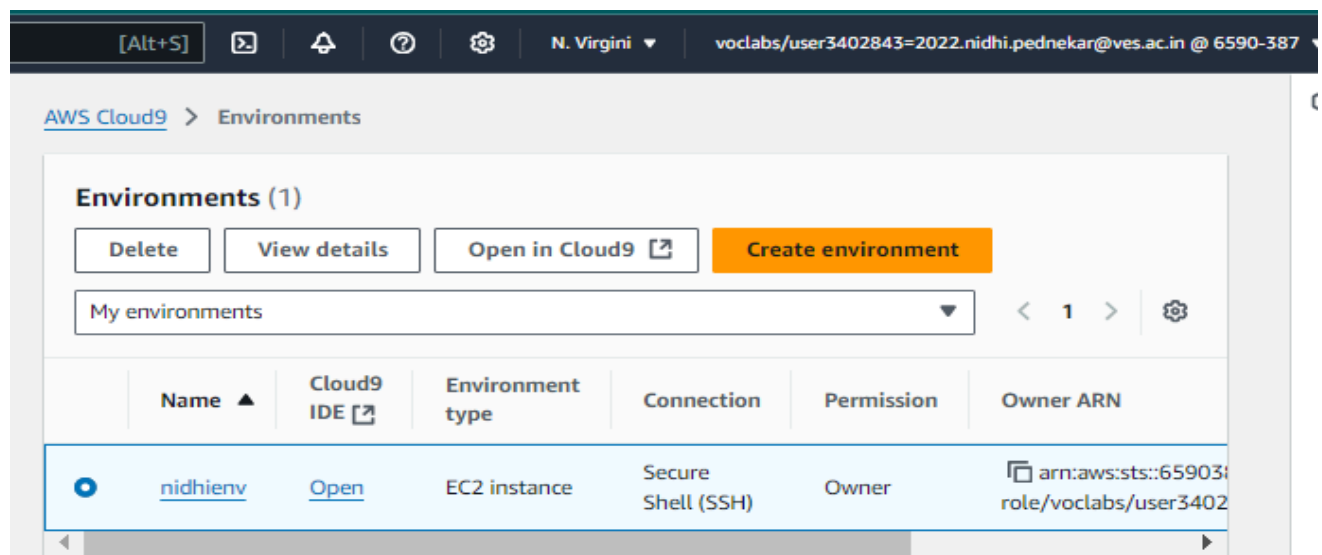
C:\Users\pedne\my-k8s-cluster>

```

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.



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
```

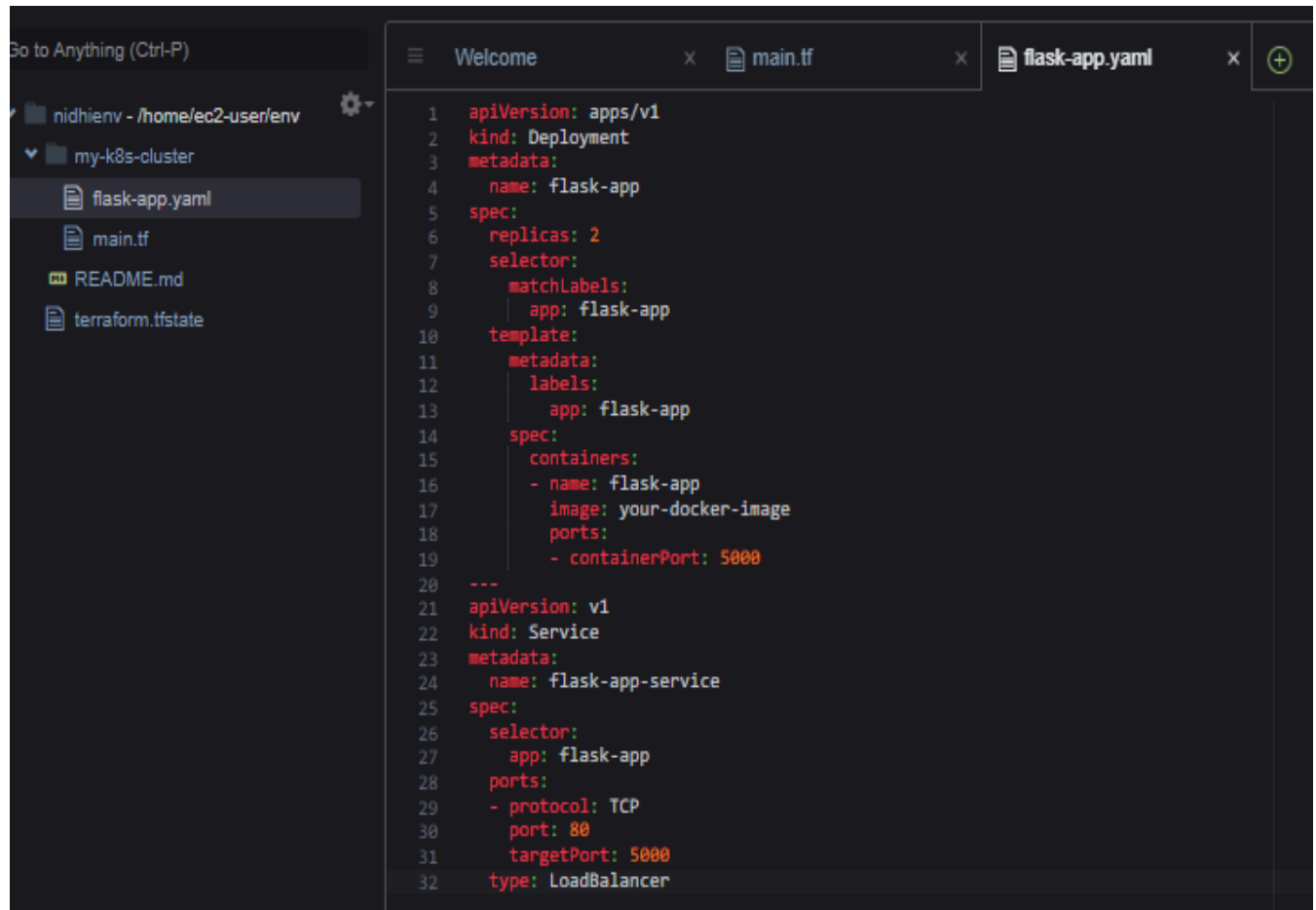
```
C:\Users\pedne\my-k8s-cluster>aws eks --region us-east-1 update-kubeconfig --name my-eks-cluster
Added new context arn:aws:eks:us-east-1:008971651210:cluster/my-eks-cluster to C:\Users\pedne\.kube\config
```

```
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:

A screenshot of a code editor interface. On the left, a file explorer shows a directory structure with files like 'flask-app.yaml', 'main.tf', 'README.md', and 'terraform.tfstate'. The main editor area displays the content of 'flask-app.yaml'. The file contains two Kubernetes manifests: a Deployment and a Service. The Deployment is named 'flask-app' and has 2 replicas. The Service is named 'flask-app-service' and is a LoadBalancer type. The code is as follows:

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: flask-app
5  spec:
6    replicas: 2
7    selector:
8      matchLabels:
9        app: flask-app
10   template:
11     metadata:
12       labels:
13         app: flask-app
14     spec:
15       containers:
16       - name: flask-app
17         image: your-docker-image
18         ports:
19         - containerPort: 5000
20   ---
21   apiVersion: v1
22   kind: Service
23   metadata:
24     name: flask-app-service
25   spec:
26     selector:
27       app: flask-app
28     ports:
29     - protocol: TCP
30       port: 80
31       targetPort: 5000
32   type: LoadBalancer
```

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.