A1 TRAINING INSTITUTE
(Only Coding)

# Kubernetes Cluster Setup Guide

# Scenario:

You have been hired by a company to set up a Kubernetes cluster to deploy a web application that consists of a frontend service, a backend service, and a database. The application should be scalable, highly available, and resilient to failures.

A1 TRAINING INSTITUTE
(Only Coding)

# Tasks:

Cluster Setup: Set up a Kubernetes cluster using a managed Kubernetes service using kubeadm. Provide details of the cluster setup, including the number of nodes and their specifications.

A1 TRAINING INSTITUTE
(Only Coding)

## Namespace Creation:

Create a separate namespace for the project to isolate the resources.

## Deploying the Application:

Deploy a simple frontend application (e.g., an Nginx web server) in the frontend namespace.

Deploy a backend application (e.g., a Node.js or Java application) in the backend namespace.
Deploy a database (e.g., MySQL or PostgreSQL) in the database namespace.

## Service Exposure:

Expose the frontend service using a Kubernetes Service of type LoadBalancer or NodePort.

Ensure that the backend service is accessible only within the cluster. Configure the database service to be accessible by the backend service.

## Scaling and High Availability:

Implement horizontal scaling for the frontend and backend services using Deployments and set up appropriate ReplicaSets.

Ensure that the database is set up with high availability (e.g., using StatefulSets, if applicable).

## ConfigMaps and Secrets:

Use ConfigMaps to manage configuration data for the backend service. Use Secrets to securely store and manage sensitive information such as database credentials.

A1 TRAINING INSTITUTE
(Only Coding)

# Rolling Updates and Rollbacks:

Demonstrate how to perform a rolling update for the backend service without downtime. Show how to roll back to a previous version in case of issues.

# Cluster Setup Documentation

## 1. Cluster Specification:

- Control Plane Nodes: 3
  - CPU: 4 vCPUs
  - Memory: 8 GB RAM
  - Disk: 50 GB SSD

# Cluster Setup Documentation

- Worker Nodes: 3
  - CPU: 4 vCPUs
  - Memory: 16 GB RAM
  - Disk: 100 GB SSD

# Cluster Setup Documentation

2. Setting up the Control Plane Nodes:
   - Initialize the Kubernetes control plane on the first master node:

```bash
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

# Cluster Setup Documentation

- Configure Kubect1 on the control plane node:

```bash
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

# Cluster Setup Documentation

- Install a pod network add-on (e.g., Flannel)

bash

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

# Cluster Setup Documentation

- Join the other control plane nodes using the token and hash from the kubeadm init output:

```bash
sudo kubeadm join <control-plane-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash> --control-plane
```

# Cluster Setup Documentation

## 3. Setting up the Worker Nodes:

- Join the worker nodes to the cluster using the token and hash from the kubeadm init output:

```bash
sudo kubeadm join <control-plane-ip>:6443 --token <token> --discovery-token-ca-cert-hash sha256:<hash>
```

# Namespace Creation:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: frontend
---
apiVersion: v1
kind: Namespace
metadata:
  name: backend
---
apiVersion: v1
kind: Namespace
metadata:
  name: database
```

# Deploying the Application:

## Frontend (Nginx)

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  namespace: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

# Deploying the Application:

## Frontend (Nginx)

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
  namespace: frontend
spec:
  type: LoadBalancer
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

A1 TRAINING INSTITUTE
Only Coding

# Backend (Node.js):

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  namespace: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: node:14
        ports:
```

# Backend (Node.js):

```
          - containerPort: 3000
          envFrom:
          - configMapRef:
              name: backend-config
          - secretRef:
              name: db-credentials
---
apiVersion: v1
kind: Service
metadata:
  name: backend-service
  namespace: backend
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  clusterIP: None
```

# Database (PostgreSQL):

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  namespace: database
spec:
  serviceName: postgres
  replicas: 3
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:13
        ports:
```

# Database (PostgreSQL):

```
        - containerPort: 5432
        envFrom:
        - secretRef:
            name: db-credentials
---
apiVersion: v1
kind: Service
metadata:
  name: postgres
  namespace: database
spec:
  ports:
    - port: 5432
  clusterIP: None
  selector:
    app: postgres
```

# ConfigMaps and Secrets

## ConfigMap for Backend:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-config
  namespace: backend
data:
  DATABASE_HOST: postgres.database.svc.cluster.local
  DATABASE_PORT: "5432"
```

A1 TRAINING INSTITUTE (Only Coding)

# Secret for Database Credentials:

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
  namespace: backend
type: Opaque
data:
  POSTGRES_USER: <base64-encoded-username>
  POSTGRES_PASSWORD: <base64-encoded-password>
---
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
  namespace: database
type: Opaque
data:
  POSTGRES_USER: <base64-encoded-username>
  POSTGRES_PASSWORD: <base64-encoded-password>
```

# Rolling Updates and Rollbacks

## Rolling Update for Backend:

1. Update the backend deployment with a new image version:

A1 TRAINING INSTITUTE
Only Coding

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  namespace: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: node:14.17.0
        ports:
        - containerPort: 3000
        envFrom:
        - configMapRef:
            name: backend-config
        - secretRef:
            name: db-credentials
```

# 2. Apply the Update:

```bash
kubectl apply -f backend-deployment.yaml
```

# Rollback Procedure:

1. List the rollout history:

bash Copy code

```
kubectl rollout history deployment/backend-deployment -n backend
```

2. Rollback to a previous revision:

bash Copy code

```
kubectl rollout undo deployment/backend-deployment -n backend --to-revision=<revision>
```

A1 TRAINING INSTITUTE
(Only Coding)

# THANKS FOR WATCHING