Kubernetes Assignment

Task 1: Create a Kubernetes Cluster Using Minikube (On Azure VM)

Objective

Simulate a local development environment by running Minikube (a single-node Kubernetes cluster) inside an Azure VM.

Tools

- Azure VM (Ubuntu 22.04)
- Docker (Container Runtime)
- Minikube (Single-node Kubernetes)
- kubectl (Kubernetes CLI)

Steps

Step 1: Create Ubuntu VM on Azure

- Go to Azure Portal → Virtual Machines → +Create
- Choose Ubuntu 22.04 LTS
- Size: Standard B2s
- Allow SSH (port 22)

Step 2: Connect to the VM via SSH

```
ssh azureuser@<your-public-ip>
```

Step 3: Install Dependencies

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y curl docker.io conntrack
sudo usermod -aG docker $USER && newgrp docker
```

Step 4: Install Minikube and kubectl

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/
```

```
stable.txt)/bin/linux/amd64/kubectl"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Step 5: Start Minikube Cluster

```
minikube start --driver=docker
```

Step 6: Verify and Access Dashboard

```
kubectl get nodes
minikube dashboard
```

Output

- Single-node Kubernetes cluster using Docker inside Azure VM
- Dashboard for visual management

Task 2: Create Kubernetes Cluster Using kubeadm (On Azure VMs)

Objective

Deploy a production-like Kubernetes cluster manually using kubeadm for better understanding of control plane and node setup.

Tools

- 2 Azure Ubuntu VMs (Master & Worker)
- kubeadm, kubelet, kubectl, Docker

Steps

Step 1: Create Two VMs in Same VNet

- VM1: `k8s-master`
- VM2: `k8s-worker`
- Open ports: 22 (SSH), 6443 (API), 10250, 10255

Step 2: Install Dependencies on Both VMs

```
sudo apt update && sudo apt install -y apt-transport-https curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key
add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/
```

```
apt/sources.list.d/kubernetes.list
sudo apt update
sudo apt install -y kubelet kubeadm kubectl docker.io
sudo systemctl enable docker && sudo systemctl start docker
sudo swapoff -a
```

Step 3: Initialize Master Node (on k8s-master)

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/16
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

Step 4: Install Calico CNI

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

Step 5: Join Worker Node (on k8s-worker)

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

Output

- Multi-node Kubernetes cluster deployed manually
- Clear understanding of control plane, networking, and kubeadm bootstrapping

---

Task 3: Deploy AKS Cluster + Access Dashboard + RBAC

Objective

Deploy a fully managed Kubernetes cluster (AKS), set up GUI dashboard, and implement Role-Based Access Control (RBAC).

Steps

Step 1: Create AKS Cluster (Azure Portal)

- Go to Azure Portal → Kubernetes Services → +Create
- Set Region, Node count: 2, Node size: Standard_DS2_v2
- Enable RBAC, System Assigned Managed Identity

Step 2: Connect via Azure CLI

```
az login
az aks get-credentials --resource-group <rg> --name <aks>
kubectl get nodes
```

Step 3: Install Kubernetes Dashboard

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.7.0/
aio/deploy/recommended.yaml
kubectl proxy
```

Access via:

```
http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/
https:kubernetes-dashboard:/proxy/
```

Step 4: Assign RBAC Role (Cloud Shell or Local CLI)

Create a role (dev-role.yaml):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: dev-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

Create a binding (dev-binding.yaml):

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: dev-binding
  namespace: default
subjects:
- kind: User
  name: user@example.com
  apiGroup: rbac.authorization.k8s.io
```

```
roleRef:
  kind: Role
  name: dev-role
  apiGroup: rbac.authorization.k8s.io
```

Apply:

```
kubectl apply -f dev-role.yaml
kubectl apply -f dev-binding.yaml
```

Output

- AKS cluster running
- Dashboard installed
- User-specific access controlled by RBAC

Task 4: Deploy Microservice on AKS and Access Publicly

Objective

Deploy a simple microservice to AKS and expose it over the internet.

Steps

Step 1: Create Deployment YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
      - name: hello
        image: nginxdemos/hello
```

```
        ports:
        - containerPort: 80
```

Step 2: Expose with LoadBalancer

```
apiVersion: v1
kind: Service
metadata:
  name: hello-service
spec:
  type: LoadBalancer
  selector:
    app: hello
  ports:
  - port: 80
    targetPort: 80
```

Step 3: Apply and Verify

```
kubectl apply -f hello-deployment.yaml
kubectl apply -f hello-service.yaml
kubectl get svc hello-service
```

Output

- Microservice deployed
- Public IP provided by Azure Load Balancer

Task 5: Expose Services: NodePort, ClusterIP, LoadBalancer

Objective

Compare different Kubernetes service types and their accessibility

Steps

ClusterIP (internal only)

```
kubectl expose deployment hello-deployment --type=ClusterIP --port=80 --
name=svc-clusterip
```

NodePort (external via VM IP)

```
kubectl expose deployment hello-deployment --type=NodePort --port=80 --name=svc-
nodeport
kubectl get svc svc-nodeport
```

Access at: `<node-ip>:<node-port>`

LoadBalancer (external IP via Azure)

```
kubectl expose deployment hello-deployment --type=LoadBalancer --port=80 --
name=svc-lb
```

Output

- Internal, semi-public, and public service exposure verified