Hands-on lab designed to guide students through setting up a Kubernetes cluster with one master node and two worker nodes on Ubuntu virtual machines (VMs) in Azure, using kubeadm. The lab covers cluster installation, configuration, and additional operations like deploying an application, scaling, monitoring, and troubleshooting, aligning with the concepts. The lab is beginner-friendly but includes practical tasks to reinforce key Kubernetes concepts such as pods, services, deployments, and kubectl commands.

Kubernetes Lab: Setting Up a 1 Master + 2 Worker Nodes Cluster on Azure with Ubuntu

Lab Objective

This lab will guide students to:

- Provision three Ubuntu VMs in Azure (1 master node, 2 worker nodes).
- Install and configure a Kubernetes cluster using kubeadm.
- Deploy a sample application, expose it via a service, scale it, monitor the cluster, and troubleshoot issues.
- Clean up resources to avoid unnecessary costs.

Lab Prerequisites

- **Azure Account**: An active Azure subscription (free tier or paid). Refer to the Azure Free Trial Account for setup if needed.
- **Hardware Requirements** (per VM):
  - VM Size: Standard_B1ms (1 vCPU, 2GB RAM, for both worker nodes)
  - VM Size: Standard_B2ms (2 vCPUs, 8GB RAM, for Master node).
  - OS: Ubuntu 22.04 LTS.
- **Software**:
  - Azure CLI installed locally (az command-line tool).
  - SSH client (e.g., OpenSSH for Linux/macOS, PuTTY for Windows).
  - kubectl installed locally (optional for managing the cluster from your local machine).
- **Access**: SSH key pair generated (~/.ssh/id_rsa.pub and ~/.ssh/id_rsa).
- **Pre-reading**: Review the provided document sections on clusters, nodes, services, deployments, and kubectl commands.

Lab Setup

1. **Install Azure CLI**:
   - Install Azure CLI on your local machine: Azure CLI Installation Guide.
   - Verify: az --version.
2. **Generate SSH Key Pair** (if not already done):

```
ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa
```

3. **Log in to Azure**:

```
az login
```

Lab Tasks

The lab is divided into tasks to set up the cluster and perform Kubernetes operations. Each task includes commands, expected outcomes, and alignment with the provided document.

Task 1: Provision Azure VMs

Goal: Create three Ubuntu 22.04 VMs in Azure (1 master, 2 workers). **Concepts Covered**: Infrastructure setup for Kubernetes nodes (from document section on Nodes). **Steps**:

1. Create a resource group:

   ```
   az group create --name k8s-cluster --location eastus
   ```

   - Expected Output: Resource group k8s-cluster created.
2. Create the master node VM:

   ```
   az vm create \

     --resource-group k8s-cluster \

     --name k8s-master \

     --image Ubuntu2204 \

     --size Standard_B2ms \

     --admin-username azureuser \

     --generate-ssh-keys \

     --public-ip-sku Standard
   ```

   - Expected Output: VM k8s-master created with public IP.
3. Create two worker node VMs:

   ```
   az vm create \

     --resource-group k8s-cluster \

     --name k8s-worker1 \

     --image Ubuntu2204 \

     --size Standard_B1ms \

     --admin-username azureuser \

     --ssh-key-values ~/.ssh/id_rsa.pub
   ```

```
az vm create \

  --resource-group k8s-cluster \

  --name k8s-worker2 \

  --image Ubuntu2204 \

  --size Standard_B1ms \

  --admin-username azureuser \

  --ssh-key-values ~/.ssh/id_rsa.pub
```

  - Expected Output: VMs k8s-worker1 and k8s-worker2 created.

4. Open necessary ports for Kubernetes (e.g., 6443 for API server, as per document's mention of networking):

```
az network nsg rule create \

  --resource-group k8s-cluster \

  --nsg-name k8s-masterNSG \

  --name kubernetes-api \

  --protocol tcp \

  --priority 1000 \

  --destination-port-range 6443 \

  --access Allow \

  --source-address-prefix '*'
```

  - Note: For testing, we open port 6443; in production, restrict source-address-prefix to specific IPs.
  - Expected Output: Security rule created.

Learning Outcome: Students learn to provision cloud infrastructure for a Kubernetes cluster.

Task 2: Prepare All Nodes

Goal: Install prerequisites (containerd, kubeadm, kubelet, kubectl) on all three VMs. **Concepts Covered**: Node components, container runtime (from document sections on Node Components and Container Runtime). **Steps** (Perform on all VMs: k8s-master, k8s-worker1, k8s-worker2):

1. SSH into each VM:

   ```
   ssh azureuser@<public-ip-of-vm>
   ```

   - Replace <public-ip-of-vm> with the public IP from az vm list-ip-addresses -g k8s-cluster.
2. Update packages and disable swap:

   ```
   sudo apt-get update && sudo apt-get upgrade -y

   sudo swapoff -a

   sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
   ```

   - Note: Kubernetes requires swap to be disabled (document section on Node Components).
3. Install containerd:

   ```
   sudo apt-get install -y containerd

   sudo mkdir -p /etc/containerd

   containerd config default | sudo tee /etc/containerd/config.toml

   sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g'
   /etc/containerd/config.toml

   sudo systemctl restart containerd

   sudo systemctl enable containerd
   ```

   - Note: Configures containerd with SystemdCgroup for Kubernetes compatibility.
4. Install kubeadm, kubelet, and kubectl:

   ```
   curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --
   dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

   echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
   https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee
   /etc/apt/sources.list.d/kubernetes.list

   sudo apt-get update

   sudo apt-get install -y kubeadm=1.28.1-1.1 kubelet=1.28.1-1.1 kubectl=1.28.1-1.1
   ```

```
sudo apt-mark hold kubeadm kubelet kubectl
```

    - Note: Locks versions to prevent unintended upgrades.

5. Enable required kernel modules:

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

overlay

br_netfilter

EOF

sudo modprobe overlay

sudo modprobe br_netfilter

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-iptables = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.ipv4.ip_forward = 1

EOF

sudo sysctl --system
```

Learning Outcome: Students prepare nodes with the necessary software, understanding the role of container runtime and kubelet.

Task 3: Initialize the Master Node

Goal: Set up the Kubernetes control plane on the master node. **Concepts Covered**: Control plane components (kube-apiserver, etcd, kube-scheduler, kube-controller-manager) (from document section on Control Plane Components). **Steps** (Perform on k8s-master):

1. Get the master node's private IP:

```
MASTER_PRIVATE_IP=$(ip -4 addr show eth0 | grep -oP '(?<=inet\s)\d+(\.\d+){3}')

echo "Master node private IP is: $MASTER_PRIVATE_IP"
```

2. Initialize the control plane:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --apiserver-advertise-
address=$MASTER_PRIVATE_IP
```

   - Note: Uses Calico-compatible pod network CIDR (document section on Kubernetes Networking Basics).
   - Expected Output: Initialization completes with a kubeadm join command (save this for Task 4).
3. Set up kubeconfig for the user:

```
mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

4. Install Calico network plugin:

```
kubectl apply -f https://docs.projectcalico.org/v3.24/manifests/calico.yaml
```

   - Note: Calico enables pod-to-pod communication (document section on Kubernetes Networking Basics).
5. Verify cluster status:

```
kubectl get nodes

kubectl get pods --all-namespaces
```

   - Expected Output: Master node in NotReady state (until workers join) and control plane pods (e.g., etcd, kube-apiserver) running.

Learning Outcome: Students initialize the control plane and understand its components' roles.

Task 4: Join Worker Nodes to the Cluster

Goal: Connect worker nodes to the master node. **Concepts Covered**: Cluster, node joining (from document sections on Cluster and Node). **Steps** (Perform on k8s-worker1 and k8s-worker2):

1. SSH into each worker node:

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

2. Run the kubeadm join command from Task 3 (example):

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

- Replace <MASTER_PRIVATE_IP>, <token>, and <hash> with values from kubeadm init output.
- If the token expires, generate a new one on the master:

```
sudo kubeadm token create --print-join-command
```

3. On the master node, verify all nodes:

```
kubectl get nodes
```

- Expected Output:

```
NAME         STATUS   ROLES           AGE   VERSION

k8s-master   Ready    control-plane   10m   v1.28.1

k8s-worker1  Ready    <none>          2m    v1.28.1

k8s-worker2  Ready    <none>          1m    v1.28.1
```

Learning Outcome: Students learn to join worker nodes, forming a multi-node cluster.

Task 5: Deploy a Sample Application

Goal: Deploy an Nginx application using a Deployment and expose it via a Service. **Concepts Covered**: Pods, deployments, services, pod-to-pod communication (from document sections on Deployments and ReplicaSets, Introduction to Services). **Steps** (Perform on k8s-master):

1. Create a file nginx-deployment.yaml:

```
apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: nginx-deployment
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: nginx
    template:
      metadata:
        labels:
          app: nginx
      spec:
        containers:
        - name: nginx
          image: nginx:latest
          ports:
            - containerPort: 80
```

2. Deploy the application:

   ```
   kubectl apply -f nginx-deployment.yaml
   ```

   - Expected Output: deployment.apps/nginx-deployment created.
3. Verify the deployment and pods:

   ```
   kubectl get deployments
   ```

   ```
   kubectl get pods
   ```

   - Expected Output: Shows nginx-deployment with 2/2 replicas and two pods running.
4. Expose the deployment as a NodePort service:

   ```
   kubectl expose deployment nginx-deployment --type=NodePort --port=80
   ```

5. Access the service:
   - Get the NodePort:

     ```
     kubectl get services
     ```

     - Expected Output: Shows nginx-deployment service with a port (e.g., 30000–32767).
   - Get a worker node’s public IP:

     ```
     az vm list-ip-addresses -g k8s-cluster --name k8s-worker1
     ```

   - Access Nginx in a browser or via curl:

     ```
     curl http://<worker1-public-ip>:<node-port>
     ```

     - Expected Output: Nginx welcome page.

Learning Outcome: Students deploy and expose an application, understanding deployments and services.

Task 6: Scale and Monitor the Application

Goal: Scale the Nginx deployment and monitor the cluster. **Concepts Covered**: Scaling, monitoring, logging (from document sections on Scaling Applications and Monitoring and Logging Basics). **Steps** (Perform on k8s-master):

1. Scale the deployment to 4 replicas:

   ```
   kubectl scale deployment nginx-deployment --replicas=4
   ```

   - Expected Output: deployment.apps/nginx-deployment scaled.

2. Verify scaling:

   ```
   kubectl get pods
   ```

   - Expected Output: Shows four Nginx pods running.
3. View logs of a pod:

   ```
   kubectl logs <nginx-pod-name>
   ```

   - Replace <nginx-pod-name> with a pod name from kubectl get pods.
   - Expected Output: Nginx access logs.
4. Enable Metrics Server (optional, for resource monitoring):

   ```
   kubectl apply -f https://github.com/kubernetes-sigs/metrics-
   server/releases/latest/download/components.yaml
   ```

   - Note: May require patching for Azure compatibility (e.g., add --kubelet-insecure-tls to Metrics Server args).
   - Verify resource usage:

     ```
     kubectl top pods
     ```

     - Expected Output: CPU and memory usage for pods.

Learning Outcome: Students learn to scale applications and monitor resource usage.

Task 7: Troubleshoot a Pod Failure

Goal: Simulate and resolve a pod failure. **Concepts Covered**: Troubleshooting, self-healing (from document section on Common Troubleshooting Tips). **Steps** (Perform on k8s-master):

1. Delete a pod to simulate failure:

   ```
   kubectl delete pod <nginx-pod-name>
   ```

   - Expected Output: pod "<nginx-pod-name>" deleted.
2. Verify self-healing:

   ```
   kubectl get pods
   ```

   - Expected Output: A new pod is created by the deployment to maintain 4 replicas.
3. Check events for troubleshooting:

   ```
   kubectl describe pod <new-nginx-pod-name>
   ```

   - Expected Output: Shows events like pod creation or errors.
4. If nodes are in NotReady state, check Calico pods:

```
kubectl get pods -n kube-system | grep calico
```

   - Troubleshoot: Ensure Calico pods are running; if not, check logs (kubectl logs <calico-pod-name> -n kube-system).

Learning Outcome: Students learn to troubleshoot and observe Kubernetes' self-healing.

Task 8: Clean Up Resources

Goal: Remove Kubernetes resources and Azure VMs to avoid costs. **Concepts Covered**: Resource management (from document section on Basic kubectl Commands). **Steps**:

1. Delete Kubernetes resources (on k8s-master):

   ```
   kubectl delete deployment nginx-deployment
   ```

   ```
   kubectl delete service nginx-deployment
   ```

   - Expected Output: Resources deleted.
2. Reset the cluster (on all nodes):

   ```
   sudo kubeadm reset -f
   ```

3. Delete Azure VMs and resource group:

   ```
   az group delete --name k8s-cluster --yes --no-wait
   ```

   - Expected Output: Resource group deletion initiated.

Learning Outcome: Students learn to clean up resources properly.