Kubernetes

Lab 1 Installing Kubernetes

Pre-requisites

- Create 3 VMs in the same Azure Region
- Size: DC1s_v3
- Location: UK South (or any)
- Ubuntu 22.04 LTS
- Username/Password
- VM names: master, node1, node2
- VNET: Accept the default 10.0.0.0/16

Install Kubernetes on master, node and node2

Execute these commands on all 3 vm's

```
# Update the VMs
sudo apt update
sudo apt-upgrade -y
sudo apt-get remove needrestart -y

# Add entries in hosts file
sudo sed -i '2i10.0.0.4 master' /etc/hosts
sudo sed -i '3i10.0.0.5 node1' /etc/hosts
sudo sed -i '4i10.0.0.6 node2' /etc/hosts

# Turn off swap (Most cloud VMs don't have swap turned on by default, but still)
sudo swapoff -a

# Adjust kernel settings
sudo tee /etc/modules-load.d/containerd.conf <<EOF
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

sudo tee /etc/sysctl.d/kubernetes.conf <<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF

# Reload system
sudo sysctl --system

# Install a few utils
sudo apt install -y curl gnupg2 software-properties-common apt-transport-https ca-certificates

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
| sudo gpg --dearmour -o /etc/apt/trusted.gpg.d/docker.gpg

```

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu  $(lsb_release -cs) stable" -y

# Update the system
sudo apt update

# Install container runtime
sudo apt install -y containerd.io

containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1

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

# Install kubernetes
sudo apt-get update

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/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:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Now create the cluster

Perform only on master node

```
# Configure Kubernetes on Master Node
# PERFORM THIS ON CONTROL-PLANE OR MASTER NODES ONLY
sudo kubeadm init --control-plane-endpoint=master

# If you have only 1 CPU for master node, use this command
sudo kubeadm init --control-plane-endpoint=master --ignore-preflight-errors=NumCPU

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

# Install Calico Pod Network Add-on
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.25.0/manifests/calico.yaml
```

Print the join command on the master node

Perform only on master node

```
# List existing tokens (optional)
kubeadm token list

# To print a join command for a new worker node use
kubeadm token create --print-join-command
```

Join nodes to cluster

Perform on both node and node2

Paste the output of earlier step and it looks like this.

```
# Join worker node to kubernetes cluster
# On the master node copy these lines and paste here. Your master node may generate a different output.
sudo kubeadm join master:6443 --token uf71xc.3g7yt5vl9zwdqc1a \
        --discovery-token-ca-cert-hash sha256:9aaba87d3dd19be12234eda79075a76bb6ebb6cdab1e0442c9bba2b33b63fbd7


```

Verify cluster

```
kubectl get nodes
kubectl get pods -n kube-system
```

Lab 2 Kubernetes Cluster Details

Get cluster details

```
kubectl cluster-info
```

List nodes

```
kubectl get nodes
```

List namespaces in the cluster

```
kubectl get namespaces
```

Get current deployments in the cluster

```
kubectl get deployments --all-namespaces
```

Get services in the cluster

```
kubectl get services --all-namespaces
```

Lab 3 Namespaces

List exisitng namespaces

```
kubectl get namespaces
```

Create a New Namespace

Find the API version to use for namespace

```
kubectl api-resources | grep namespaces
```

Create a YAML file called as dev.yaml

```
nano dev.yaml
```

```
apiVersion: v1
kind: Namespace
metadata:
 name: dev
```

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

Verify creation

```
kubectl get namespaces
```

Create namespace using create command

```
kubectl create namespace prod
```

Verify creation

```
kubectl get namespaces
```

Using a Github Repo link

```
kubectl apply -f \
https://raw.githubusercontent.com/reposforlabs/kubernetes/main/namespace.yaml
```

Verify creation

```
kubectl get ns
```

List pods in all namespaces

```
kubectl get pods --all-namespaces
```

Lab 4 Create PODs using RUN command

Using RUN command

In default namespace

```
kubectl run pod1 --image=nginx --port=80 --labels=owner=raghavendra
```

List

```
kubectl get pods
```

In dev namespace

```
kubectl run pod1 --image=nginx --port=80 --labels=type=dev -n dev
```

List

```
kubectl get pods -n dev
```

Find the nodes on which the pods are created

```
kubectl get pods
kubectl get pods -o wide
```

Viewing POD Labels

```
kubectl get pods --show-labels
kubectl get pods --show-labels -n default
kubectl get pods --show-labels -n dev
```

Describing a POD

```
kubectl describe pod pod1 -n default
```

Finding POD Logs

```
kubectl logs pod1 -n default
```

Delete a pod

```
kubectl delete pod pod1 -n default
kubectl delete pod pod1 -n dev
```

Lab 5 Creating PODs using POD Object

Create a YAML file and deploy

```
nano pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  labels:
    app: myapp
spec:
  containers:
  - name: mycontainer
    image: nginx:latest
    ports:
    - containerPort: 80
```

```
kubectl apply -f mypod.yaml
```

- `apiVersion` specifies the version of the Kubernetes API being used.
- `kind` specifies the type of Kubernetes resource, which in this case is a Pod.
- `metadata` contains information about the Pod, including its name and labels.
- `spec` specifies the desired state of the Pod, including the containers running in the Pod.
- `containers` contains a list of containers running in the Pod. Each container has a `name` and an `image` (the Docker image to use for the container).
- `ports` specifies the ports to expose from the container. In this case, port 80 is exposed.

How to know which API version to use

```
kubectl api-resources
```

List the pod status

```
kubectl get pods
```

Delete the pod

```
kubectl delete pod mypod -n default
```

Lab 6 Create PODs using ReplicationSet

Create a YAML and deploy

```
nano replicaset.yaml
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: mycontainer
        image: nginx:latest
        ports:
        - containerPort: 80
```

```
kubectl apply -f replicaset.yaml
```

```
kubectl get pods
```

```
kubectl get rs
```

Delete ReplicaSet and PODs

```
kubectl delete rs my-replicaset
```

Lab 7 Create PODs using DeploymentSet

Create a YAML file and deploy

```
nano deployment.yaml
```

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

- `apiVersion` specifies the version of the Kubernetes API being used.
- `kind` specifies the type of Kubernetes resource, which in this case is a Deployment.
- `metadata` contains information about the Deployment, including its name.
- `spec` specifies the desired state of the Deployment, including the number of replicas to maintain.
- `replicas` specifies the number of desired replicas, which in this case is 3.
- `selector` specifies the labels used to select pods managed by the Deployment.
- `template` contains the pod template used to create new pods.
- `metadata` specifies the labels to apply to pods created by the Deployment.
- `spec` specifies the pod specification, including the container definition.
- `containers` contains a list of containers running in the pod. Each container has a name and an image.

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

```
kubectl get deployment
kubectl get pods
kubectl get pods -o wide
kubectl get rs
```

Increase replicas to 4

Edit the deployment.yaml file and make replicas to 4.

Reapply

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

```
kubectl get rs
```

Delete Deployment

```
kubectl delete deployment my-deployment
```

Lab 8 Managing PODs

Deploy a pod

```
kubectl run pod1 --image=nginx --port=80 --labels=type=dev
```

Getting details about a particular POD

```
kubectl describe pod pod1
```

How to get the containers running on all PODs

```
kubectl get pods -o=custom-
columns=PodName:.metadata.name,Containers:.spec.containers[*].name,Image:.spec.containers[*].image
```

Attaching to a POD

```
kubectl get pods
kubectl exec --stdin --tty <POD Name> -- /bin/bash
```

Attaching to a container in a POD

```
kubectl exec -i -t <pod name> --container <container name> -- /bin/bash
```

Create a pod on a particular node

Create a file called as deploy-on-a-node.yml and paste the definition

```
nano deploy-on-a-node.yml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
spec:
  containers:
  - name: container1
    image: nginx
  nodeName: node2
```

The **nodeName field** instructs the **Scheduler** to **assign the POD** to node2 computer.

Deploy the POD:

```
kubectl apply -f deploy-on-a-node.yml
```

Verify that the POD is created on the selected node:

```
kubectl get pods
kubectl get pods -o wide
```

Add a label to a node

```
kubectl get nodes
kubectl get nodes -o wide
kubectl get nodes --show-labels
```

Choose one of your nodes, and add a label to it:

```
kubectl label nodes node2 disktype=ssd
```

Verify the label:

```
kubectl get nodes --show-labels
```

Filter all nodes that match your label:

```
kubectl get nodes --show-labels | grep disktype=ssd
```

Deploy a POD based on a label

We just added a label to node2: `disktype=ssd`

We want to deploy a POD to this node not by nodename but by the label.

Create a deployment file as shown below using nano:

```
nano deploy-with-label.yml
```

Paste the contents below:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    env: dev
spec:
  containers:
  - name: container1
    image: nginx
  nodeSelector:
    disktype: ssd
```

Perform the POD deployment:

```
kubectl apply -f deploy-with-label.yml
```

Verify the POD creation on node2, because it matches with the **NodeSelector** field `disktype` -

```
kubectl get pods -o wide
```

Remove a Label from a node

First find the node labels:

```
kubectl get nodes --show-labels
```

I want to delete the Label: `disktype`

```
kubectl label node <nodename> <labelname>-
```

```
kubectl label node workernode2 disktype-
node/workernode2 unlabeled
```

Verify:

```
kubectl get nodes --show-labels
```

Lab 9 Exposing PODs to Internet

Method-1 Using Load Balancer Service

Create a Deployment kind using this YAML

```
nano deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: website-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: website-container
        image: tanvisinghny/ssl-website
        ports:
        - containerPort: 80
        - containerPort: 443
```

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

Verify if the Pods are created on both nodes

```
kubectl get pods -o wide
NAME                                 READY   STATUS    RESTARTS   AGE   IP               NODE    NOMINATED
NODE   READINESS GATES
website-deployment-5fc6c46f9b-jw2xm   1/1     Running   0          40s   192.168.166.136   node1   <none>
<none>
website-deployment-5fc6c46f9b-s8s9c   1/1     Running   0          40s   192.168.104.21    node2   <none>
<none>

```

Create a Service object that exposes the deployment

```
kubectl expose deployment website-deployment --type=LoadBalancer --name=service-website-deployment
```

View the service

```
kubectl get services
NAME                        TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)                      AGE
kubernetes                  ClusterIP      10.96.0.1    <none>        443/TCP                      3d
```

```
4 service-website-deployment   LoadBalancer   10.99.3.87   <pending>     80:30923/TCP,443:30274/TCP   20s
```

Note that External IP is pending

Patch the service with node1 and node2 host IPs

```
1 kubectl patch svc service-website-deployment -n default -p '{"spec": {"type": "LoadBalancer", "externalIPs":
  ["10.0.0.5", "10.0.0.6"]}}'
```

View the service again

```
1 kubectl get services
2 NAME                       TYPE           CLUSTER-IP   EXTERNAL-IP         PORT(S)                      AGE
3 kubernetes                 ClusterIP      10.96.0.1    <none>              443/TCP                      3d
4 service-website-deployment LoadBalancer   10.99.3.87   10.0.0.5,10.0.0.6   80:30923/TCP,443:30274/TCP   3m21s
```

On the VMs open HTTP, HTTPS Ports and access the website using the node1 and node2 IP addresses

Method-2 Using NodePort Service

Delete the Service you created above

```
1 kubectl delete service service-website-deployment
```

View if the service is deleted

```
1 kubectl get services
2 NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
3 kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   3d
```

Create a Deployment kind using this YAML

```
nano deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: website-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: website-container
        image: tanvisinghny/ssl-website
        ports:
        - containerPort: 80
        - containerPort: 443
---
apiVersion: v1
kind: Service
metadata:
  name: website-service
spec:
  type: NodePort
  selector:
    app: webapp
  ports:
   - port: 80
     targetPort: 80
     nodePort: 30080
     protocol: TCP
     name: http
   - port: 443
     targetPort: 443
     nodePort: 30443
     protocol: TCP
     name: https
  externalIPs:
    - 10.0.0.5
    - 10.0.0.6
```

```
kubecl apply -f deployment.yaml
```

View the created objects

```
kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
website-deployment   2/2     2            2           14s

kubectl get service
```

```
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP          PORT(S)                      AGE
kubernetes        ClusterIP   10.96.0.1       <none>               443/TCP                      3d1h
website-service   NodePort    10.97.136.212   10.0.0.5,10.0.0.6    80:30080/TCP,443:30443/TCP   21s

kubectl get pods -o wide
NAME                                  READY   STATUS    RESTARTS   AGE   IP               NODE    NOMINATED
NODE   READINESS GATES
website-deployment-5fc6c46f9b-hfgzp   1/1     Running   0          81s   192.168.104.24   node2   <none>
<none>
website-deployment-5fc6c46f9b-lb59l   1/1     Running   0          81s   192.168.166.139  node1   <none>
<none>
```

Access the website in the browser

Cleanup

```
kubectl delete deployment website-deployment
kubectl delete service website-service
```

Method-3 Using ClusterIP

Create a Deployment kind using this YAML

```
nano deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: website-deployment
spec:
  replicas: 2
  selector:
```

```
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      containers:
      - name: website-container
        image: tanvisinghny/ssl-website
        ports:
        - containerPort: 80
        - containerPort: 443
---
apiVersion: v1
kind: Service
metadata:
  name: website-service
spec:
  type: ClusterIP
  selector:
    app: webapp
  ports:
   - port: 80
     protocol: TCP
     name: http
   - port: 443
     protocol: TCP
     name: https
  externalIPs:
    - 10.0.0.5
    - 10.0.0.6
```

```
kubecl apply -f deployment.yaml
```

View the created objects

```
kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
website-deployment   2/2     2            2           69s

kubectl get service
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP         PORT(S)          AGE
kubernetes        ClusterIP   10.96.0.1      <none>              443/TCP          3d2h
website-service   ClusterIP   10.99.86.166   10.0.0.5,10.0.0.6   80/TCP,443/TCP   16s
```

Access the website in the browser

Lab 10 Kubernetes Networking

Finding POD IP Range

```
kubectl get ipamblocks.crd.projectcalico.org
```

For example

```
kubectl get ipamblocks.crd.projectcalico.org
NAME                 AGE
192-168-104-0-26     82m
192-168-166-128-26   83m
192-168-219-64-26    101m
```

Default CIDR Block is /26 = 64 IPs

This allows to create 64 PODS only

Changing Block CIDR

Step-1 Install calicoctl as a Kubernetes pod

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

Verify

```
kubectl get pods -n kube-system
NAME                                       READY   STATUS    RESTARTS      AGE
calico-kube-controllers-658d97c59c-gzqz4   1/1     Running   0             111m
calico-node-9dnxq                          1/1     Running   0             90m
calico-node-jc25j                          1/1     Running   1 (96m ago)   111m
calico-node-k5l67                          1/1     Running   0             91m
calicoctl                                  1/1     Running   0             19s
coredns-76f75df574-44bz7                   1/1     Running   0             136m
coredns-76f75df574-44g76                   1/1     Running   0             136m
etcd-master                                1/1     Running   1 (96m ago)   136m
kube-apiserver-master                      1/1     Running   1 (96m ago)   136m
kube-controller-manager-master             1/1     Running   3 (96m ago)   136m
kube-proxy-4fwg6                           1/1     Running   0             90m
kube-proxy-64sfl                           1/1     Running   0             91m
kube-proxy-w68zd                           1/1     Running   1 (96m ago)   136m
kube-scheduler-master                      1/1     Running   3 (96m ago)   136m
```

Set an alias:

```
alias calicoctl="kubectl exec -i -n kube-system calicoctl -- /calicoctl "
```

Step-2 Add a new IP pool

```
calicoctl create -f -<<EOF
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
  name: new-pool
spec:
  cidr: 10.0.0.0/8
```

```
  ipipMode: Always
  natOutgoing: true
EOF
```

Lab 12 Taints and Tolerations

POD scheduling issues

- Pod scheduling is one of the most important aspects of Kubernetes administration.
- Effective scheduling can improve performance, reduce costs, and make clusters easier to manage.
- Taints and tolerations help prevent your pods from scheduling to undesirable nodes.
- When we create a pod, the scheduler in the control plane looks at the available nodes and verifies conditions before assigning a pod to the nodes.
- If there are no errors during the verification, the pod will be scheduled on the node.
- If the conditions of the verification aren't satisfied, then your pods will be put in a **Pending** state.
- You can use `kubectl describe pods <pod-Name>` and scroll down to Events to find the precise reasons for the pending state
- You can run `kubectl get events`.
- If it is a hardware resource issue, you may get some error about CPU or Memory or Storage.

Taint

Taints are a Kubernetes node property that enable nodes to repel certain pods.

Master Node Example:

By default except Kubernetes System PODs, User PODs are not created on Control-plane node.

This because the master node is has a Taint:

`node-role.kubernetes.io/control-plane:NoSchedule`

The NoSchedule Taint REPELS the Scheduler and does not allow POD creation.

Since Taints is a NODE Property, you can see a node taints using describe command:

```
kubectl describe node master
```

The taint effect defines how a tainted node reacts to a pod without appropriate toleration. It must be one of the following effects;

- **NoSchedule** — The pod will not get scheduled to the node without a matching toleration.
- **NoExecute** — This will immediately evict all the pods without the matching toleration from the node.
- **PerferNoSchedule** — This is a softer version of NoSchedule where the controller will not try to schedule a pod with the tainted node. However, it is not a strict requirement.

Toleration

- **Toleration** is applied to pods and allows the pods to schedule onto nodes with matching taints.

POD (Toleration) ===Scheduled===> NODE (Taint)

For the above control-plane example, to schedule PODs on control-plane node add a Toleration:

```
tolerations:
 - key: "node-role.kubernetes.io/control-plane"
   effect: "NoSchedule"

```

Actual script:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment1
spec:
  replicas: 20
  selector:
    matchLabels:
      app: app1
  template:
    metadata:
      labels:
        app: app1
    spec:
      containers:
      - name: app1-con
        image: nginx:latest
        ports:
        - containerPort: 80
      tolerations:
       - key: "node-role.kubernetes.io/control-plane"
         effect: "NoSchedule"
```

Taint a Node

```
kubectl taint nodes node1 gpu=true:NoSchedule
```

We want to ensure only Deployments with Tolerations matching `gpu=true:NoSchedule` should allow scheduler to deploy PODs on node1.

Try this script

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment1
spec:
  replicas: 20
  selector:
    matchLabels:
      app: app1
  template:
    metadata:
      labels:
        app: app1
    spec:
      containers:
      - name: app1-con
        image: nginx:latest
        ports:
        - containerPort: 80
      tolerations:
       - key: "gpu"
         operator: "Equal"
         value: "true"
         effect: "NoSchedule"
```

If that doesn't work, change Toleration as:

```
tolerations:
 - key: "gpu"
   effect: "NoSchedule"
```

Untaint a node

```
kubectl taint nodes node1 gpu=true:NoSchedule-
```

Lab 15 Kubernetes Backup

Download velero

```
curl -LO https://github.com/vmware-tanzu/velero/releases/download/v1.10.1/velero-v1.10.1-linux-amd64.tar.gz
```

Extract

```
tar zxvf velero-v1.10.1-linux-amd64.tar.gz
```

Move to bin path

```
sudo mv velero-v1.10.1-linux-amd64/velero /usr/local/bin/velero
```

Backup a namespace and it's objects

```
velero backup create <backup-name> --include-namespaces <namespace>
```

Backup all deployments in the cluster

```
velero backup create <backup-name> --include-resources deployments
```

Backup the deployments in a namespace

```
velero backup create <backup-name> --include-resources deployments --include-namespaces <namespace>
```

Backup entire cluster including cluster-scoped resources

```
velero backup create <backup-name>
```

Backup a namespace and include cluster-scoped resources

```
velero backup create <backup-name> --include-namespaces <namespace> --include-cluster-resources=true
```

Include resources matching the label selector

```
velero backup create <backup-name> --selector <key>=<value>
```

Include resources that are not matching the selector

```
velero backup create <backup-name> --selector "<key> notin (<value>)"
```

Exclude kube-system from the cluster backup

```
velero backup create <backup-name> --exclude-namespaces kube-system
```

Exclude secrets from the backup

```
velero backup create <backup-name> --exclude-resources secrets
```

Exclude secrets and rolebindings

```
velero backup create <backup-name> --exclude-resources secrets,rolebindings
```

Restore two namespaces and their objects

```
velero restore create <backup-name> --include-namespaces <namespace1>,<namespace2>
```

Restore all deployments and configmaps in the cluster

```
velero restore create <backup-name> --include-resources deployments,configmaps
```

Restore only namespaced resources in the cluster

```
velero restore create <backup-name> --include-cluster-resources=false
```