Name: Harsh Dalvi Subject: Advance DevOps , Sem: SEM V
Roll No: 13 Class / Batch: TE-IT / Batch B

Experiment No. 4

Aim: To install Kubectl and execute Kubectl commands to manage the Kubernetes cluster and deploy your First Kubernetes Application. **(LO1, LO2)**

Theory:

- **What is Kubernetes?**
  Kubernetes is open-source software that allows you to deploy and manage containerized applications at scale. Kubernetes manages clusters of Amazon Elastic Compute Cloud (EC2) compute instances and runs containers on those instances with processes for deployment, maintenance, and scaling. Using Kubernetes, you can run any type of containerized applications using the same toolset on-premises and in the cloud. AWS makes it easy to run Kubernetes in the cloud with scalable and highly available virtual machine infrastructure, community-backed service integrations, and Amazon Elastic Kubernetes Service (EKS), a certified conformant, managed Kubernetes service.

- **What is Kubernetes Cluster?**
  A Kubernetes cluster is a logical grouping of EC2 compute instances that run your containers. A cluster consists of the control plane (the instances that control how, when, and where your containers run), and the data plane (the instances where your containers run). You must define a cluster before you can run containers or services with Kubernetes.

- **How Kubernetes works?**
  Kubernetes works by managing a cluster of compute instances and scheduling containers to run on the cluster based on the available compute resources and the resource requirements of each container. Containers are run in logical groupings called pods and you can run and scale one or many containers together as a pod. Kubernetes control plane software decides when and where to run your pods, manages traffic routing, and scales your pods based on utilization or other metrics that you define. Kubernetes automatically starts pods on your cluster based on their resource requirements and automatically restarts pods if they or the instances they are running on fail. Each pod is given an IP address and a single DNS name, which Kubernetes uses to connect your services with each other and external traffic.

- **Kubectl:**
  The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs. Kubectl is installable on a variety of Linux platforms, macOS and Windows.

- **Need of Kubernetes:**
  Containers are a good way to bundle and run your applications. In a production environment, you need to manage the containers that run the applications and ensure that there is no downtime. For example, if a container goes down, another container needs to start. Wouldn't it be easier if this behaviour was handled by a system? That's how Kubernetes comes to the rescue! Kubernetes provides you with a framework to run distributed systems resiliently. It takes care of scaling and failover for your application, provides deployment patterns, and more. For example, Kubernetes can easily manage a canary deployment for your system.

Kubernetes provides you with:

1. **Service discovery and load balancing:** Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.

2. **Storage orchestration:** Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.

3. **Automated rollouts and rollbacks:** You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.

4. **Automatic bin packing:** You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.

5. **Self-healing:** Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

6. **Secret and configuration management:** Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration.

Steps to perform the Experiment:

Step 1: Install Kubernetes tools Commands:

Both **Master** and **Slave:-**

1. **sudo apt-get install kubeadm kubelet kubectl –y**
2. **sudo apt-mark hold kubeadm kubelet kubectl**

```
root@ip-172-31-33-239:/home/ubuntu# sudo apt-get install kubeadm kubelet kubectl -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools ebtables kubernetes-cni socat
The following NEW packages will be installed:
  conntrack cri-tools ebtables kubeadm kubectl kubelet kubernetes-cni socat
```

```
root@ip-172-31-35-118:/home/ubuntu# sudo apt-get install kubeadm kubelet kubectl -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  conntrack cri-tools ebtables kubernetes-cni socat
The following NEW packages will be installed:
  conntrack cri-tools ebtables kubeadm kubectl kubelet kubernetes-cni socat
0 upgraded, 8 newly installed, 0 to remove and 87 not upgraded.
```

```
root@ip-172-31-33-239:/home/ubuntu# sudo apt-mark hold kubeadm kubelet kubectl
kubeadm set on hold.
kubelet set on hold.
kubectl set on hold.
root@ip-172-31-33-239:/home/ubuntu# ^C
```

```
root@ip-172-31-35-118:/home/ubuntu# sudo apt-mark hold kubeadm kubelet kubectl
kubeadm set on hold.
kubelet set on hold.
kubectl set on hold.
```

Step 2: Verify the installation Both **Master** and **Slave: -**

Commands: kubeadm version

```
root@ip-172-31-33-239:/home/ubuntu# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.1", GitCommit:"e4d4e1ab7cf1bf15273ef9730
3551b279f0920a9", GitTreeState:"clean", BuildDate:"2022-09-14T19:47:53Z", GoVersion:"go1.19.1", Compiler:"gc", P
latform:"linux/amd64"}
root@ip-172-31-33-239:/home/ubuntu#
```

```
root@ip-172-31-35-118:/home/ubuntu# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.1", GitCommit:"e4d4e1ab7cf1bf15273ef973
3551b279f0920a9", GitTreeState:"clean", BuildDate:"2022-09-14T19:47:53Z", GoVersion:"go1.19.1", Compiler:"gc",
latform:"linux/amd64"}
root@ip-172-31-35-118:/home/ubuntu#
```

Step 3: Begin Kubernetes deployment

Both **Master** and **Slave: -**

Commands: swapoff –a

```
root@ip-172-31-33-239:/home/ubuntu# swapoff --a
```

```
root@ip-172-31-35-118:/home/ubuntu# swapoff --a
```

Step 4: Assign Unique Hostname for Each Server Node Decide which server to set as the master node. Next, set a worker node hostname.

On Master: -

Command: sudo hostnamectl set-hostname master-node

On Worker: -

Command: sudo hostnamectl set-hostname worker01

```
root@ip-172-31-33-239:/home/ubuntu# sudo hostnamectl set-hostname master-node
root@ip-172-31-33-239:/home/ubuntu# exit
exit
```

```
root@ip-172-31-35-118:/home/ubuntu# sudo hostnamectl set-hostname worker01
root@ip-172-31-35-118:/home/ubuntu# exit
exit
```

Step 5: Initialize Kubernetes on Master Node.

On Master: -

Command:

1. **sudo su**
2. **sudo kubeadm init --pod-network-cidr=10.244.0.0/16**

```
ubuntu@ip-172-31-35-22:~$ sudo su
root@master-node:/home/ubuntu# sudo kubeadm init --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.25.2
[preflight] Running pre-flight checks
        [WARNING SystemVerification]: missing optional cgroups: blkio
```

If you are trying to run this on EC2 you'll get an error message saying less cpu and memory to override the error run the above command with --ignore-preflight-errors=all

On Master: -

Command: sudo kubeadm init--pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all

```
root@master-node:/home/ubuntu# sudo kubeadm init --pod-network-cidr=10.244.0.0/16 --ignore-preflight-errors=all
[init] Using Kubernetes version: v1.25.1
```

Once this command finishes, it will display a kubeadm join message at the end. Make a note of the whole entry. This will be used to join the worker nodes to the cluster. Next, create a directory for the cluster.

On Master: -
Command:

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

```
root@master-node:/home/ubuntu# mkdir -p $HOME/.kube
root@master-node:/home/ubuntu# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root@master-node:/home/ubuntu# sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Step 6: Deploy Pod Network to Cluster.

On Master: -
Command: sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

```
root@master-node:/home/ubuntu# sudo kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
```

Allow the process to complete. Verify that everything is running and communicating.
On Master: -
Command: kubectl get pods --all-namespaces

```
root@master-node:/home/ubuntu# kubectl get pods --all-namespaces
NAMESPACE      NAME                                 READY   STATUS              RESTARTS         AGE
kube-flannel   kube-flannel-ds-6pjhz                0/1     PodInitializing     0                32s
kube-system    coredns-565d847f94-dkckf             0/1     Pending             0                2m59s
kube-system    coredns-565d847f94-197mn             0/1     Pending             0                2m59s
kube-system    etcd-master-node                     1/1     Running             1 (4m6s ago)     4m9s
kube-system    kube-apiserver-master-node           1/1     Running             1 (3m36s ago)    3m31s
kube-system    kube-controller-manager-master-node  1/1     Running             1 (4m6s ago)     2m20s
kube-system    kube-proxy-bnmg6                     0/1     CrashLoopBackOff    2 (7s ago)       2m59s
kube-system    kube-scheduler-master-node           0/1     CrashLoopBackOff    4 (4s ago)       4m9s
```

Step 7: Join Worker Node to Cluster. you can enter the kubeadm join command on each worker node to connect it to the cluster. Switch to the worker01 system.

On Worker: -
Command:

1. **sudo su**
2. **kubeadm join 172.31.44.201:6443 –token iv4qnb.7aczfwllmfs4o7sc --discovery-token-ca-cert-hash sha256:33faf5ca90d079f20f6d1d48ca1bd225bc858462dbe0881418e7eeae4c4db075**

```
ubuntu@ip-172-31-45-44:~$ sudo su
root@worker01:/home/ubuntu# kubeadm join 172.31.35.22:6443 --token gufzye.7s8b8n08y8drrit0 \
        --discovery-token-cakubeadm join 172.31.35.22:6443 --token gufzye.7s8b8n08y8drrit0 \fa8b398bdd17072b8f
        --discovery-token-ca-cert-hash sha256:fc8aebfc006538ff988cc8a58a3a8a0bd27357a2f2e2b7fa8b398bdd17072b8f
[preflight] Running pre-flight checks
        [WARNING SystemVerification]: missing optional cgroups: blkio
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

Switch to Master server.

The system should display the worker nodes that you joined to the cluster.

On Master: -

Command: kubectl get nodes

```
root@master-node:/home/ubuntu# kubectl get nodes
NAME          STATUS     ROLES           AGE     VERSION
master-node   Ready      control-plane   6m58s   v1.25.2
worker01      NotReady   <none>          32s     v1.25.2
```

Step 8: Running An Application on the Cluster. Deploy any containerized application to your cluster.

On Master: -

Command: kubectl create deployment nginx –image=nginx

```
root@master-node:/home/ubuntu# kubectl get deployment deployment.apps/nginx
error: there is no need to specify a resource type as a separate argument when passing arguments in resource
'kubectl get resource resource/<resource_name>'
```

Step 9: Create a service named nginx that will expose the app publicly.

On Master: -

Command: kubectl expose deploy nginx --port 80 --target-port 80 --type NodePort

```
root@master-node:/home/ubuntu# kubectl expose deploy ngnix --port 80 --target-port 80 --type NodePort
service/ngnix exposed
```

Step 10: Services are another type of Kubernetes object that expose cluster internal services to clients, both internal and external.

On Master: -

Command: kubectl get services

```
root@master-node:/home/ubuntu# kubectl get services
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        13m
ngnix        NodePort    10.107.233.116   <none>        80:30769/TCP   18s
```

From the output you can retrieve the port that Nginx is running on. To test that everything is working, visit http://worker_1_ip:nginx_port

Step 11: Check the deployed container on the worker node.

On Worker: -

Command: docker ps

Step 12: To scale up the replicas for a deployment (nginx in our case)

On Master: -

Command:

1. **kubectl scale --current-replicas=1 --replicas=2**
2. **kubectl get pods**
3. **kubectl describe deployment/nginx**

Step 13: Remove the Nginx application. So first delete the nginx service from the master node. Then check that the service has been deleted.

On Master: -

Command:

1. **kubectl delete service nginx**
2. **kubectl get services**

Step 14: Delete the deployment. Then check if it has been deleted.

On Master: -

Command:

1. **kubectl delete deployment nginx**
2. **kubectl get deployments**

Step 15: Delete the node by Finding it, Draining it and then finally deleting it.

On Master: -

Command:

1. **kubectl get nodes**
2. **kubectl drain nodetoberemoved**
3. **kubectl delete node nodetoberemoved**

Step 16: Remove join/init setting from Worker node

On Worker: -

Command:

1. **kubeadm reset**
2. **docker ps On Master.**

Command: kubectl get nodes

Conclusion: From this experiment it is concluded that, we have successfully installed Kubectl and executed Kubectl commands to manage the kubernetes cluster and deploy our First Kubernetes Application. Hence, we have successfully achieved the Lab Outcome One and Lab Outcome Two (LO1 and LO2). Also, we have achieved PO1, PO2, PO3, PO4, PO5, PO9, PO10 and PO12 from this experiment.