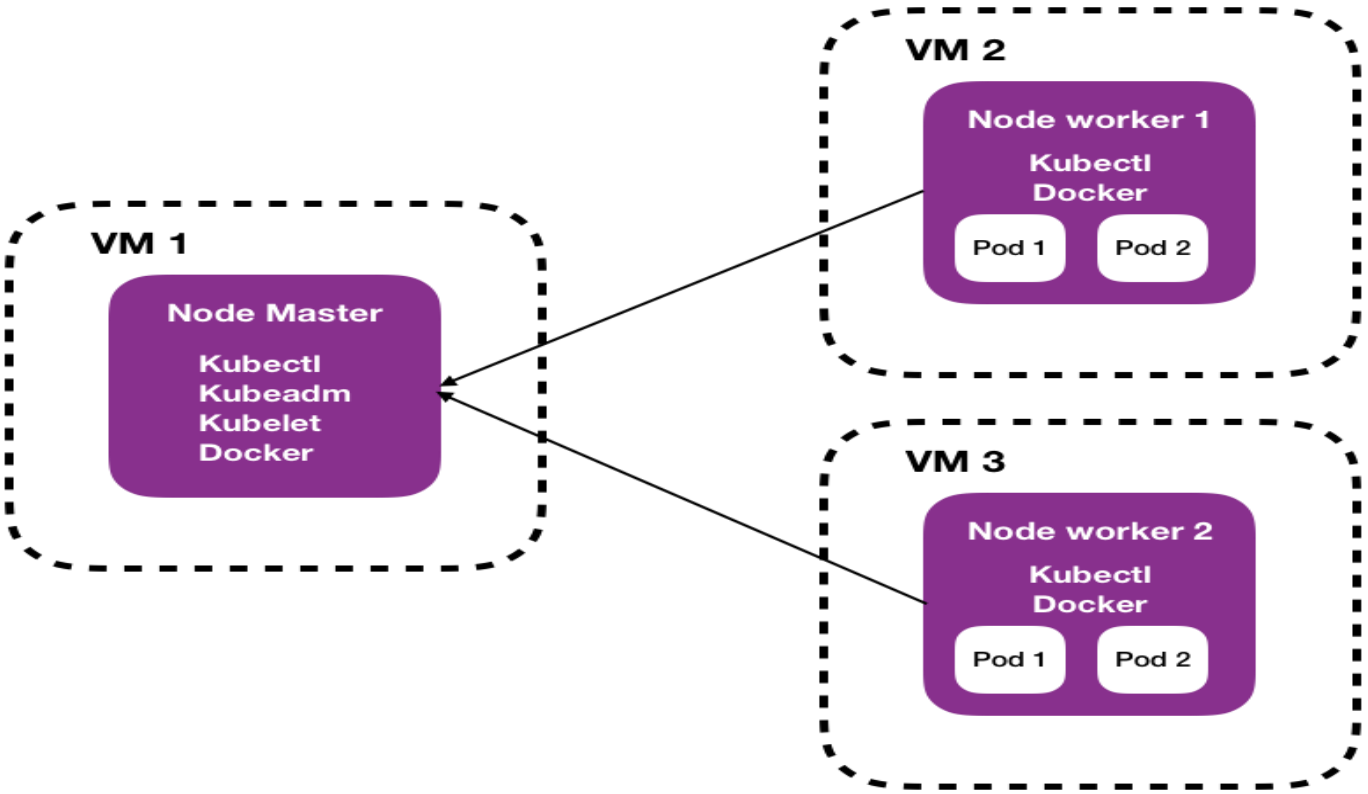
$ Create K8S Cluster using KubeAdm $

In this Project we will create K8S Cluster contain 3 servers (one as Master node “Control plane” & two as Worker nodes) using **KubeAdm**. Then we will deploy a Java App with it’s complete architecture services on this Cluster.



➢ Tools we need for this project.

1) **VMware Workstation :** (VMware Workstation is a Desktop Hypervisor products which let users run virtual machines, containers, and Kubernetes clusters. VMware Workstation is a virtual machine software that is used for x86 and x86-64 computers to run multiple operating systems over a single physical host computer. Each virtual machine can run a single instance of any operating system (Microsoft, Linux, etc.).

2) **Docker :** (Docker is an open source platform that enables developers to build, deploy, run, update and manage containers).

3) **KubeAdm :** (**Kubeadm** is a tool used to build Kubernetes (K8s) clusters. Kubeadm performs the actions necessary to get a minimum viable cluster up and running quickly. By design, it cares only about bootstrapping, not about provisioning machines (underlying worker and master nodes). Kubeadm is a really handy tool that can help you set up a Kubernetes cluster. Essentially, it automates the whole process by initializing the control plane and joining worker nodes to the cluster.).

- **Kubeadm's Features :-**

  I. Quick minimum viable cluster creation :> (Kubeadm is designed to have all the components you need in one place in one cluster regardless of where you are running them).

  II. Portable :> (Kubeadm can be used to set up a cluster anywhere whether it's your laptop, or public cloud infrastructure).

  III. Local development :> (As Kubeadm creates clusters with minimal dependencies and quickly, it's an ideal candidate for creating disposable clusters on local machines for development and testing needs).

  IV. Building block for other tools :> (Kubeadm is not just a K8s installer. It also serves as a building block for other tools like Kubespray).

➢ What is (MiniKube, Kind, MicroK8s, K3s).

1. MiniKube : (miniKube is the most widely used local Kubernetes installer. It offers an easy to use guide to installing and running single Kubernetes environments across multiple operating systems. It deploys Kubernetes as a container, VM or bare-metal and implements a Docker API endpoint that helps it push container images faster. It has advanced features like load balancing, filesystem mounts, and FeatureGates, making it a favorite for running Kubernetes locally).

2. Kind : (Primarily designed to test Kubernetes, Kind (Kubernetes in Docker) helps you run Kubernetes clusters locally and in CI pipelines using Docker containers as "nodes").

3. MicroK8s : (microK8S is a Kubernetes distribution designed to run fast, self-healing, and highly available Kubernetes clusters. It is optimized for quick and easy installation of single and multi-node clusters on multiple operating systems, including macOS, Linux, and Windows. It is ideal for running Kubernetes in the cloud, local development environments and Edge and IoT devices. It also works efficiently in standalone systems using ARM or Intel).

4. K3s : (K3s is a lightweight tool designed to run production-level Kubernetes workloads for low resourced and remotely located IoT and Edge devices.
   K3s helps you run a simple, secure and optimized Kubernetes environment on a local computer using a virtual machine such as VMWare or VirtualBox.).

➢ KubeAdm VS (MiniKube, Kind, MicroK8s, K3s).

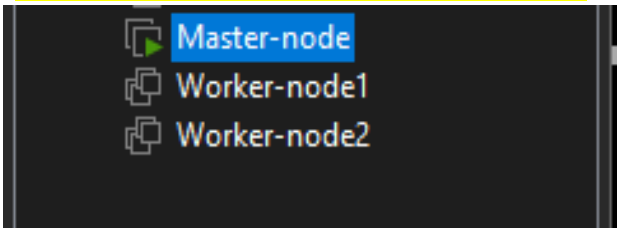
- **Kubeadm** allows us to customize and configure our clusters according to our needs and preferences. So it used widely in production environment.
- **Minikube** is a tool that allows us to run a single-node Kubernetes cluster on our local machine. It is mainly used for development and testing purposes, With Minikube, you can easily create a Kubernetes cluster, deploy applications, and test them before deploying them to a production environment.
- **Kind,** It is primarily optimized for CI pipelines. Due to starting a cluster in a docker container, it can start faster compared to the VM alternatives.
- **MicroK8s,** It is lightweight K8S, and It is suitable for Edge and IoT devices. It is mainly focused on simplicity and performance.
- **K3S,** It is lightweight and has a low memory footprint due to excluding some features from the Kubernetes binary. It provides a VM-based Kubernetes environment, So we should manually edit nodes and virtual machines for multiple K8S servers. It is suitable for testing the production environment locally, and generally used in performance-constraint environments like IoT devices.
-

❖Steps :

- Create 3 Virtual Machines using **VMware Workstation** (one as Master node and two as Worker nodes)**.**
- Then Set up the environment of Kubernetes cluster Using kubeAdm, One Node Act as control plane (master) and others join to the cluster as workers :-
  - On Master and Worker nodes :
    I. Turn off the swap & firewall.
    II. Update sysctl settings for Kubernetes networking.
    III. Set up and install Docker.
    IV. Install cri-dockerd.
    V. Installing kubeadm, kubelet and kubectl, install Kubernetes package repositories.
  - On Master Node :
    I. Initialize Kubernetes Cluster.
    II. Create a user for kubernetes administration and copy kube config file.
    III. Deploy Calico network as a kubeadmin user.
    IV. Cluster join command.
  - On Worker Nodes :
    I. Add worker nodes to cluster.
    II. Verifying the cluster To Get Nodes status.
- deploy a Java App with it's complete architecture services onto the Cluster:-
  - Create Yaml configuration files to create Deployment for each service, and to create service for each Deployment.
  - Apply this Yaml files to build our App architecture objects (deployments, replicasets, pods and services).

$~ Lab Setup ~$

➢ Create 3 Virtual Machines:



➢ Set up the environment of Kubernetes cluster Using kubeAdm.

- Perform all the commands as root user.

- On Master and Worker nodes :

I. Turn off the swap & firewall.

```
# Disable swap
-  sed -i '/swap/d' /etc/fstab
-   swapoff -a
# Disable SELinux
- setenforce 0
- sed -i --follow-symlinks 's/^SELINUX=enforcing/SELINUX=disabled/'
/etc/sysconfig/selinux
# Disable Firewall
-  systemctl disable firewalld
- systemctl stop firewalld
```

II. Update sysctl settings for Kubernetes networking.

```
# Forwarding IPv4 and letting iptables see bridged traffic
-  cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
-  modprobe overlay
-  modprobe br_netfilter
```

```
# sysctl params required by setup, params persist across reboots
-  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
# Apply sysctl params without reboot
-  sysctl –system
```

III. Set up and install Docker.

```
# Set up the repository.
- yum install -y yum-utils
- yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo
# Install Docker Engine, containerd, and Docker Compose.
-  yum install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
# Start Docker.
- systemctl enable docker
- systemctl start docker
```

IV. Install cri-dockerd.

```
# Install git and wget.
-  yum install -y git wget
# Download cri-dockerd source code.
-  git clone https://github.com/Mirantis/cri-dockerd.git
# Download and install GOlang.
-  wget https://go.dev/dl/go1.22.0.linux-amd64.tar.gz
-  tar -C /usr/local -xzf go1.22.0.linux-amd64.tar.gz
-  export PATH=$PATH:/usr/local/go/bin
-  go version
# Install cri-dockerd.
- cd cri-dockerd
- yum -y install make
- make cri-dockerd
- mkdir -p /usr/local/bin
- install -o root -g root -m 0755 cri-dockerd /usr/local/bin/cri-dockerd
- install packaging/systemd/* /etc/systemd/system
- sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,'
/etc/systemd/system/cri-docker.service
- systemctl daemon-reload
- systemctl enable --now cri-docker.socket
```

V. Installing kubeadm, kubelet and kubectl, install Kubernetes package repositories.

```
#  Add the Kubernetes yum repository.
-  cat  <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/
enabled=1
gpgcheck=1
gpgkey=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/repodata/repomd.xml.key
exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
EOF
# Install kubelet, kubeadm and kubectl.
-  yum install -y kubelet  kubeadm  kubectl  --disableexcludes=Kubernetes
-  systemctl enable --now kubelet
```

- On Master Node :

I. Initialize Kubernetes Cluster.

- kubeadm init --apiserver-advertise-address=<MasterServerIP> --pod-network-cidr=192.168.0.0/16 --cri-socket=unix:///var/run/cri-dockerd.sock

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.11.132:6443 --token ux9fzi.jfuwseklfyzzicqu \
        --discovery-token-ca-cert-hash sha256:620213e052c5086d16d24f4c6c0038d0f1cb1ea8b88a94ddf8ade9de710c3dcb
```

II. Create a user for kubernetes administration and copy kube config file.

- useradd kubeadmin
- mkdir -p /home/kubeadmin/.kube
- cp -i /etc/kubernetes/admin.conf /home/kubeadmin/.kube/config
- chown -R kubeadmin:kubeadmin /home/kubeadmin/.kube/config

III. Deploy Calico network as a kubeadmin user.

- sudo su – kubeadmin

Install the operator on your cluster.

- kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifests/tigera-operator.yaml

Download the custom resources necessary to configure Calico.

- curl https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifests/custom-resources.yaml -O

Create the manifest in order to install Calico.

- kubectl create -f custom-resources.yaml

Verify Calico installation in your cluster

- watch kubectl get pods -n calico-system

IV. Cluster join command.

Run the command that was output by kubeadm init .

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.11.132:6443 --token ux9fzi.jfuwseklfyzzicqu \
        --discovery-token-ca-cert-hash sha256:620213e052c5086d16d24f4c6c0038d0f1cb1ea8b88a94ddf8ade9de710c3dcb
```

Or you can create new token using this command.

- kubeadm token create --print-join-command

- **On Worker Nodes :**

I. Add worker nodes to cluster.

Use the output from kubeadm init or kubeadm token create command in previous step from the master server and run it in worker nodes.

II. Verifying the cluster To Get Nodes status.

- kubectl get nodes

```
[kubeadmin@master-node ~]$ kubectl get nodes
NAME           STATUS   ROLES           AGE     VERSION
master-node    Ready    control-plane   62m     v1.29.2
worker-node1   Ready    <none>          8m15s   v1.29.2
worker-node2   Ready    <none>          5m36s   v1.29.2
```

➢ Deploy a Java App with it's complete architecture services onto the Cluster.

- Our App services:
  - i. **K8S L.B** (**Load Balancer SVC**).
  - ii. **MariaDB** (**Database SVC**).
  - iii. **Memcached** (**DB Caching SVC**).
  - iv. **RabbitMQ** (**Broker/Queue SVC**).
  - v. **Tomcat** (**Application SVC**).

- Create Yaml configuration files to create Deployment for each service, and to create service for each Deployment:
  - i. App service Deployment Yaml file.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vproapp
  labels:
    app: vproapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vproapp
  template:
    metadata:
      labels:
        app: vproapp
    spec:
      containers:
      - name: vproapp
        image: alihassan895/app_tomcat
        ports:
          - containerPort: 8080
      initContainers:
      - name: init-mydb
        image: busybox
        command: ['sh', '-c', 'until nslookup vprodb.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.local; do
 echo waiting for mydb; sleep 2; done;']
      - name: init-memcache
        image: busybox
        command: ['sh', '-c', 'until nslookup vprocache01.$(cat /var/run/secrets/kubernetes.io/serviceaccount/namespace).svc.cluster.loca
l; do echo waiting for mydb; sleep 2; done;']
```

ii. RabbitMQ service Deployment Yaml file.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vpromq01
  labels:
    app: vpromq01
spec:
  selector:
    matchLabels:
      app: vpromq01
  replicas: 1
  template:
    metadata:
      labels:
        app: vpromq01
    spec:
      containers:
      - name: vpromq01
        image: rabbitmq
        ports:
          - containerPort: 15672
        env:
         - name: RABBITMQ_DEFAULT_PASS
           valueFrom:
             secretKeyRef:
               name: app-secret
               key: rmq-pass
         - name: RABBITMQ_DEFAULT_USER
           value: "guest"
```

iii. Memcached service Deployment Yaml file.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vpromc
  labels:
    app: vpromc
spec:
  selector:
    matchLabels:
      app: vpromc
  replicas: 1
  template:
    metadata:
      labels:
        app: vpromc
    spec:
      containers:
      - name: vpromc
        image: memcached
        ports:
          - containerPort: 11211
```

iv. MariaDB service Deployment Yaml file.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vprodb
  labels:
    app: vprodb
spec:
  selector:
    matchLabels:
      app: vprodb
  replicas: 1
  template:
    metadata:
      labels:
        app: vprodb
    spec:
      containers:
        - name: vprodb
          image: alihassan895/mariadb
          ports:
            - containerPort: 3306
          env:
            - name: MARIADB_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: app-secret
                  key: db-pass
      initContainers:
        - name: busybox
          image: busybox:latest
          args: ["rm", "-rf", "/var/lib/mysql/lost+found"]
```

v. K8S L.B service Yaml file.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: vproapp-service
spec:
  type: LoadBalancer
  selector:
    app: vproapp
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
```

vi. RabbitMQ Deployment service Yaml file.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: vpromq01
spec:
  type: ClusterIP
  selector:
    app: vpromq01
  ports:
  - port: 15672
    targetPort: 15672
    protocol: TCP
```

vii. Memcached Deployment service Yaml file.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: vprocache01
spec:
  type: ClusterIP
  selector:
    app: vpromc
  ports:
  - port: 11211
    targetPort: 11211
    protocol: TCP
```

viii. MariaDB Deployment service Yaml file.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: vprodb
spec:
  type: ClusterIP
  selector:
    app: vprodb
  ports:
  - port: 3306
    targetPort: 3306
    protocol: TCP
```

ix. App secret Yaml file.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
type: Opaque
data:
  db-pass: YWRtaW4xMjMK
  rmq-pass: dGVzdAo=
```

- Now we have our App Yaml files all ready.

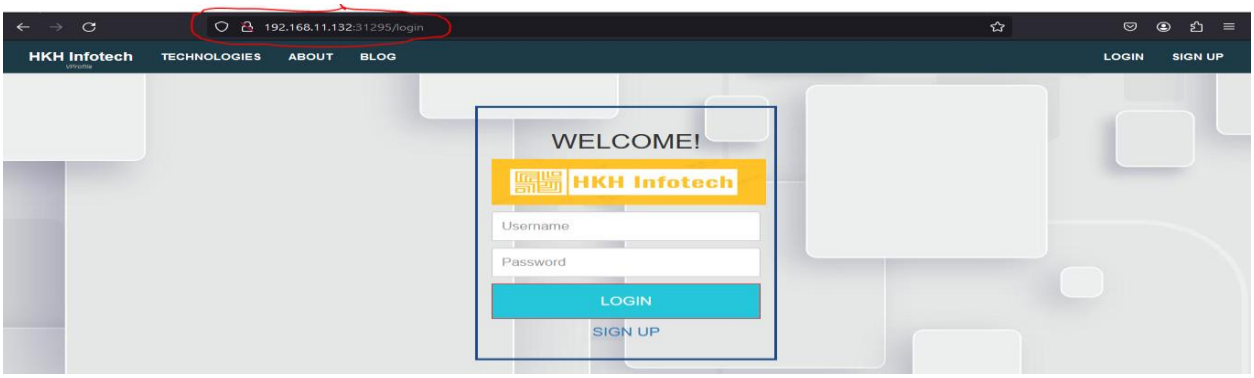
```
[kubeadmin@master-node kubernetes]$ ls
app-lb-service.yml  db-cip.yml          deployment-db.yml  deployment-rmq.yml  rmq-cip.yml
app-secret.yml      deployment-app.yml  deployment-mc.yml  mc-cip.yml
```

- To deploy our App onto the cluster we need to run this command: `kubectl apply -f .`

```
[kubeadmin@master-node kubernetes]$ kubectl apply -f .
service/vproapp-service created
secret/app-secret created
service/vprodb created
deployment.apps/vproapp created
deployment.apps/vprodb created
deployment.apps/vpromc created
deployment.apps/vpromq01 created
service/vprocache01 created
service/vpromq01 created
```

```
[kubeadmin@master-node kubernetes]$ kubectl get deployment
NAME       READY   UP-TO-DATE   AVAILABLE   AGE
vproapp    1/1     1            1           6m29s
vprodb     1/1     1            1           6m29s
vpromc     1/1     1            1           6m29s
vpromq01   1/1     1            1           6m29s
[kubeadmin@master-node kubernetes]$ kubectl get svc
NAME              TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes        ClusterIP      10.96.0.1        <none>        443/TCP        6d2h
vproapp-service   LoadBalancer   10.101.247.15    <pending>     80:31295/TCP   6m42s
vprocache01       ClusterIP      10.100.76.22     <none>        11211/TCP      6m41s
vprodb            ClusterIP      10.96.236.197    <none>        3306/TCP       6m42s
vpromq01          ClusterIP      10.96.173.131    <none>        15672/TCP      6m41s
[kubeadmin@master-node kubernetes]$ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
vproapp-576df59488-fs59b    1/1     Running   0          6m52s
vprodb-5b5d888ccb-xhdjx     1/1     Running   0          6m52s
vpromc-77b689879f-f27c5     1/1     Running   0          6m52s
vpromq01-65f95c5975-jmd49   1/1     Running   0          6m52s
```

- We have successfully deploy our App onto the cluster, and we can access it from any web browser using any node ip with our L.B Service port “31295”.



➢ Conclusion:

- In this project we have covered:
  1) What is (**KubeAdm**, **MiniKube**, **Kind**, **MicroK8s** and **K3s**).
  2) **KubeAdm** VS (**MiniKube**, **Kind**, **MicroK8s**, **K3s**).
  3) How to install and configure **KubeAdm**.
  4) How to create Kubernetes cluster Using **kubeAdm**.
  5) How to deploy a Java App with it’s complete architecture services onto the Cluster.