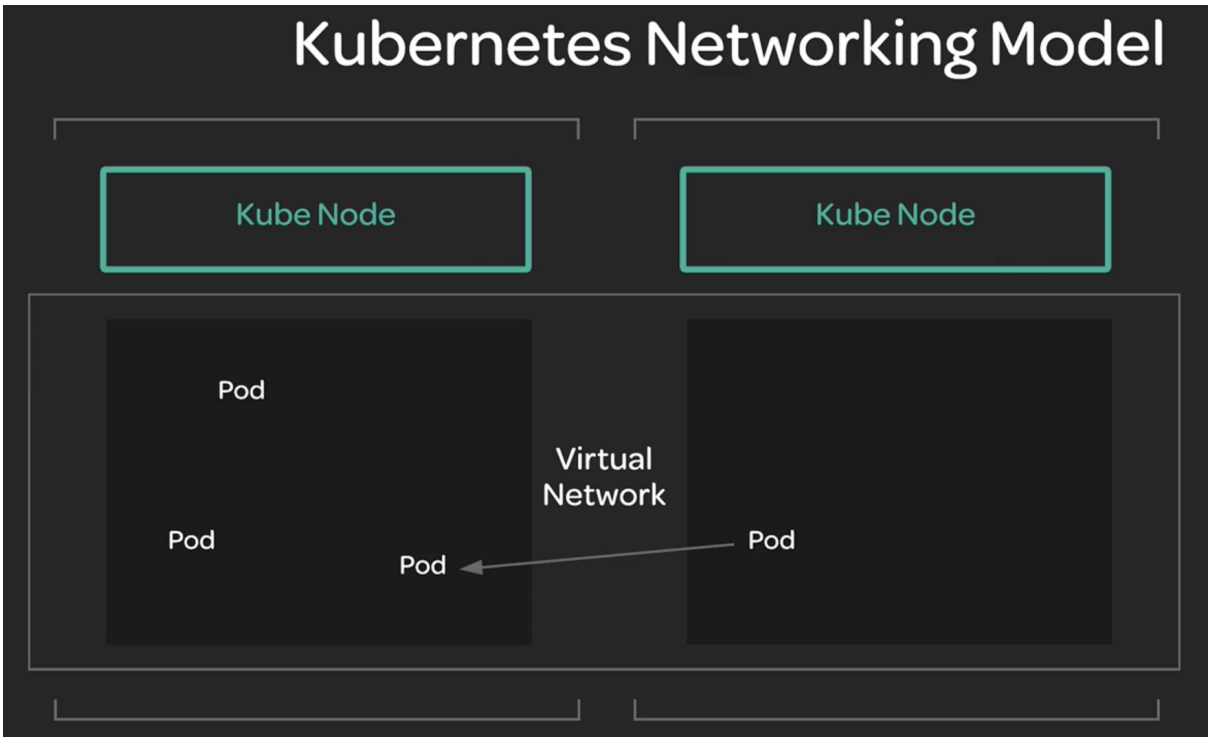
Cloud Provider:AWS
Server:Ubuntu 18.04 x6
Server List: 2 Controller(Master), 2(Worker), 1 Nginx(Reverse Proxy), 1 Ubuntu 18.04(local)

The Kubernetes Networking Model

- One virtual network for the whole cluster.
- Each pod has a unique IP within the cluster.
- Each service has a unique IP that is in a different range than pod IPs.



Cluster Network Architecture

Some Important CIDR ranges:

- Cluster CIDR: IP range used to assign IPs to pods in the cluster. For this lab, we'll be using a cluster CIDR of 10.200.0.0/16
- Service Cluster IP Range: IP range for services in the cluster. This should not overlap with the cluster CIDR range! For this lab, we'll be using 10.32.0.0/24
- Pod CIDR: IP range for pods on a specific worker node. This range should fall within the cluster CIDR but not overlap with the pod CIDR of any other worker node. For this lab, our networking plugin will automatically handle IP allocation to nodes, so we do not need to manually set a pod CIDR.

Installing Weave Net

// We are now ready to set up networking in our Kubernetes cluster. This lesson guides you through the process of installing Weave Net in the cluster. It also shows you how to test your cluster network to make sure that everything is working as

expected so far. After completing this lesson, you should have a functioning cluster network within your Kubernetes cluster.

// You can configure Weave Net like this:

// First, log in to both worker nodes and enable IP forwarding:

$ sudo sysctl net.ipv4.conf.all.forwarding=1

$ echo "net.ipv4.conf.all.forwarding=1" | sudo tee -a /etc/sysctl.conf

// The remaining commands can be done using kubectl. To connect with kubectl, you can either log in to one of the control nodes and run kubectl there or open an SSH tunnel for port 6443 to the load balancer server and use kubectl locally.

// You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>

// Install Weave Net like this:

$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')&env.IPALLOC_RANGE=10.200.0.0/16"

// Now Weave Net is installed, but we need to test our network to make sure everything is working.

// First, make sure the Weave Net pods are up and running:

$ kubectl get pods -n kube-system

// This should return two Weave Net pods, and look something like this:

```
ubuntu@ip-10-100-0-97:~$ kubectl get pods -n kube-system
NAME              READY     STATUS    RESTARTS   AGE
weave-net-7dzh8   2/2       Running   1          25m
weave-net-plh25   2/2       Running   1          25m
```

// Next, we want to test that pods can connect to each other and that they can connect to services. We will set up two Nginx pods and a service for those two pods. Then, we will create a busybox pod and use it to test connectivity to both Nginx pods and the service.

// First, create an Nginx deployment with 2 replicas:

```
$ cat << EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      run: nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
EOF
```

// Next, create a service for that deployment so that we can test connectivity to services as well:

```
$ kubectl expose deployment/nginx
```

// Now let's start up another pod. We will use this pod to test our networking. We will test whether we can connect to the other pods and services from this pod.

```
$ kubectl run busybox --image=radial/busyboxplus:curl --command -- sleep 3600

$ POD_NAME=$(kubectl get pods -l run=busybox -o jsonpath="{.items[0].metadata.name}")
```

// Now let's get the IP addresses of our two Nginx pods:

```
$ kubectl get ep nginx
```

// There should be two IP addresses listed under ENDPOINTS, for example:

```
ubuntu@ip-10-100-0-97:~$ kubectl get ep nginx
NAME      ENDPOINTS                         AGE
nginx     10.200.0.2:80,10.200.192.1:80     25m
```

// Now let's make sure the busybox pod can connect to the Nginx pods on both of those IP addresses.

$ kubectl exec $POD_NAME -- curl <first nginx pod IP address>
$ kubectl exec $POD_NAME -- curl <second nginx pod IP address>

// Both commands should return some HTML with the title "Welcome to Nginx!" This means that we can successfully connect to other pods.
// Now let's verify that we can connect to services.

$ kubectl get svc

// This should display the IP address for our Nginx service. For example, in this case, the IP is 10.32.0.54:

```
ubuntu@ip-10-100-0-97:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.32.0.1      <none>        443/TCP   17h
nginx        ClusterIP   10.32.0.183    <none>        80/TCP    29m
```

// Let's see if we can access the service from the busybox pod!

$ kubectl exec $POD_NAME -- curl <nginx service IP address>

// This should also return HTML with the title "Welcome to Nginx!"
// This means that we have successfully reached the Nginx service from inside a pod and that our networking configuration is working!

Clean Up

// Now that we have networking set up in the cluster, we need to clean up the objects that were created in order to test the networking. These objects could get in the way or become confusing in later lessons, so it is a good idea to remove them from the cluster before proceeding. After completing this lesson, your networking should still be in place, but the pods and services that were used to test it will be cleaned up.

// To do this, you will need to use kubectl. To connect with kubectl, you can either log in to one of the control nodes and run kubectl there or open an SSH tunnel for port 6443 to the load balancer server and use kubectl locally.

// You can open the SSH tunnel by running this in a separate terminal. Leave the session open while you are working to keep the tunnel active:

$ ssh -L 6443:localhost:6443 cloud_user@<your Load balancer cloud server public IP>

// You can clean up the testing objects from the previous lesson like so:

```
$ kubectl delete deployment busybox
$ kubectl delete deployment nginx
$ kubectl delete svc nginx
```

DNS In a Kubernetes Pod Network

What does DNS do inside a pod network?

- Provides a DNS service to be used by pods within the network.
- Configures containers to use the DNS service to perform DNS lookups.
    - For example:
        - You can access services using DNS names assigned to them.
        - You can access other pods using DNS names.