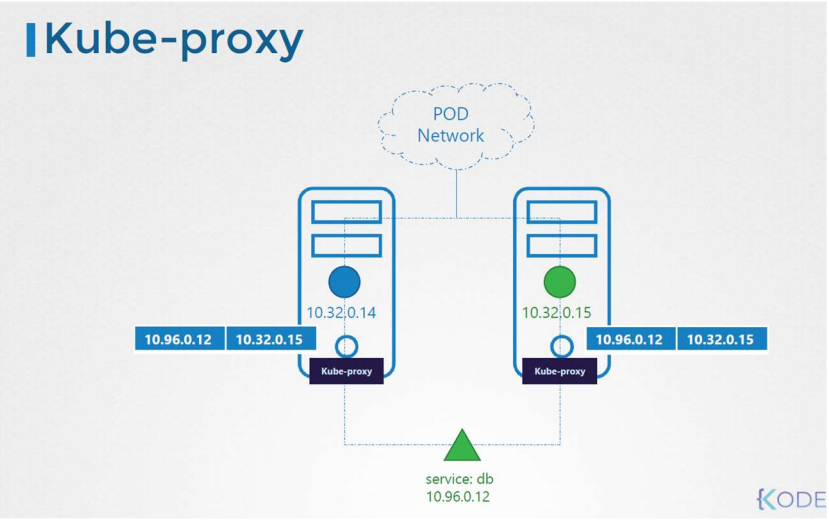
Kubeproxy

In this lecture, we will talk about kube-proxy. Within a Kubernetes cluster, every pod can reach every other pod. This is accomplished by deploying a pod networking solution to the cluster. A pod network is an internal virtual network that spans across all the nodes in the cluster to which all the pods connect to. Through this network, they're able to communicate with each other. There are many solutions available for deploying such a network.

In this case, I have a web application deployed on the first node and a database application deployed on the second. The web app can reach the database simply by using the IP of the pod, but there is no guarantee that the IP of the database pod will always remain the same. If you've gone through the lecture on services, as discussed in the beginner's course, you must know that a better way for the web application to access the database is using a service. So we create a service to expose the database application across the cluster. The web application can now access the database using the name of the service, DB. The service also gets an IP address assigned to it. Whenever a pod tries to reach the service using its IP or name, it forwards the traffic to the backend pod, in this case, the database.

But what is this service, and how does it get an IP? Does the service join the same pod network? The service cannot join the pod network because the service is not an actual thing. It is not a container like pods, so it doesn't have any interfaces or an actively listening process. It is a virtual component that only lives in the Kubernetes memory. But then, we also said that the service should be accessible across the cluster from any nodes. So how is that achieved? That's where kube-proxy comes in. Kube-proxy is a process that runs on each node in the Kubernetes cluster. Its job is to look for new services, and every time a new service is created, it creates the appropriate rules on each node to forward traffic to those services to the backend pods. One way it does this is using iptables rules. In this case, it creates an iptables rule on each node in the cluster to forward traffic heading to the IP of the service, which is 10.96.0.12, to the IP of the actual pod, which is 10.32.0.15. So that's how kube-proxy configures a service.



We will now see how to install kube-proxy. Download the kube-proxy binary from the Kubernetes release page, extract it, and run it as a service. The kubeadm tool deploys kube-proxy as pods on each node. In fact, it is deployed as a DaemonSet, so a single pod is always deployed on each node in the cluster.

```
wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-proxy
```

kube-proxy.service

```
ExecStart=/usr/local/bin/kube-proxy \\
  --config=/var/lib/kube-proxy/kube-proxy-config.yaml
Restart=on-failure
RestartSec=5
```

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

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	coredns-78fcdf6894-hwrq9	1/1	Running	0	16m
kube-system	coredns-78fcdf6894-rzhjr	1/1	Running	0	16m
kube-system	etcd-master	1/1	Running	0	15m
kube-system	kube-apiserver-master	1/1	Running	0	15m
kube-system	kube-controller-manager-master	1/1	Running	0	15m
kube-system	kube-proxy-lzt6f	1/1	Running	0	16m
kube-system	kube-proxy-zm5qd	1/1	Running	0	16m
kube-system	kube-scheduler-master	1/1	Running	0	15m
kube-system	weave-net-29z42	2/2	Running	1	16m
kube-system	weave-net-snmdl	2/2	Running	1	16m

```
kubectl get daemonset -n kube-system
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE SELECTOR	AGE
kube-proxy	2	2	2	2	2	beta.kubernetes.io/arch=amd64	1h