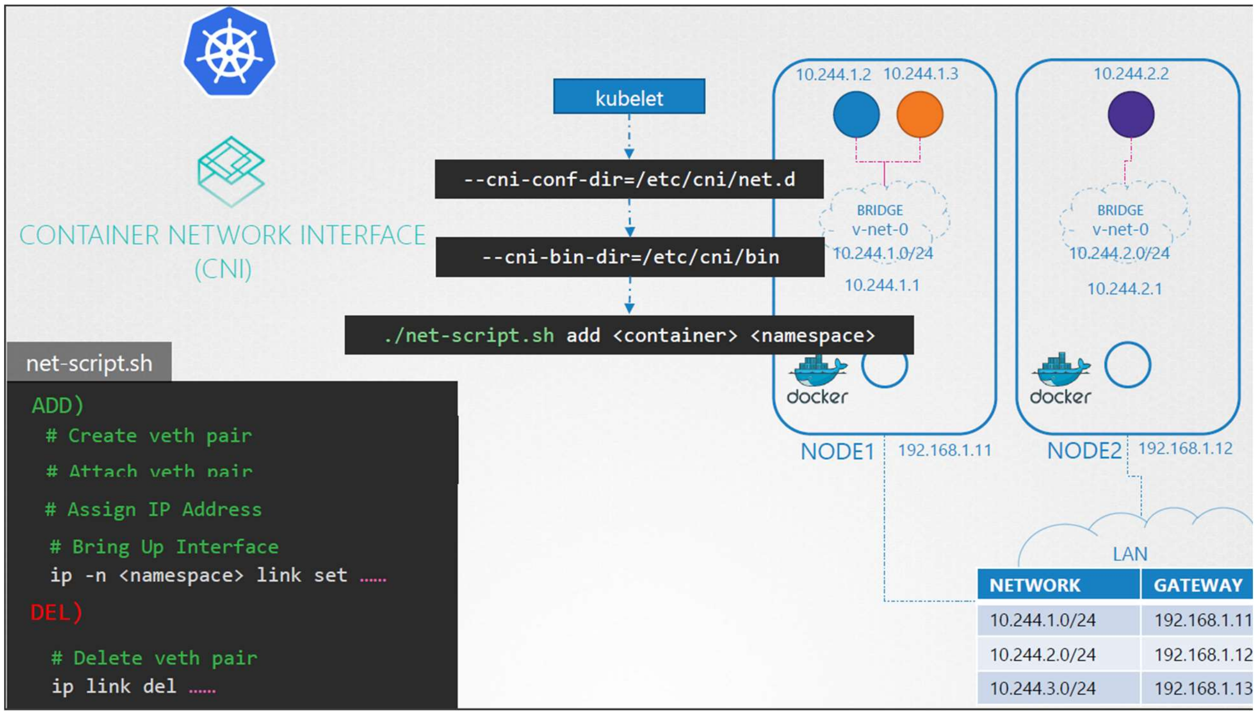
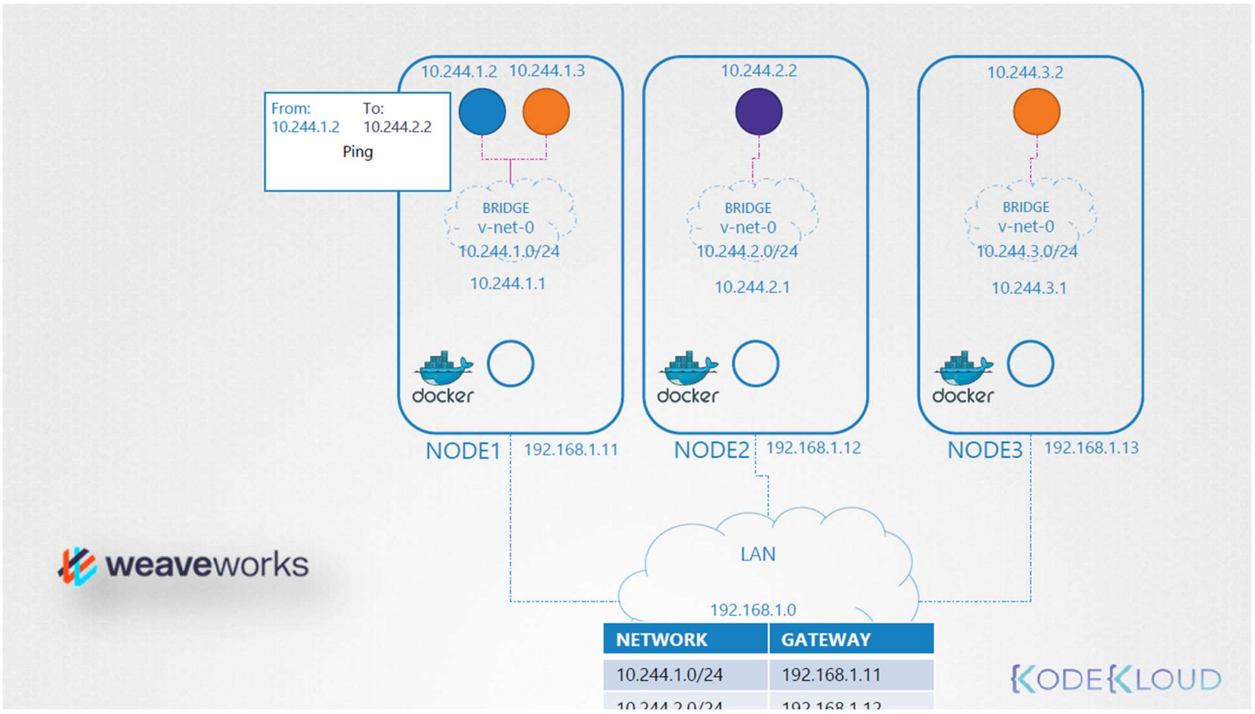
In this lecture we will discuss about one solution based on CNI, in particular Weaveworks. The Weaveworks with CNI plugin. In the previous practice test, we saw how it is configured. Now, we will see more details about how it works. We will start where we left off in the pod networking concept section. We had our own custom CNI script that we've built and integrated into kubelet through CNI.



```
ADD)
 # Create veth pair
 # Attach veth pair
 # Assign IP Address
 # Bring Up Interface
 ip -n <namespace> link set ……
DEL)
 # Delete veth pair
 ip link del ……
```

NETWORK	GATEWAY
10.244.1.0/24	192.168.1.11
10.244.2.0/24	192.168.1.12
10.244.3.0/24	192.168.1.13

In the previous lecture, we saw how instead of our own custom script, we integrated the Weave plugin.

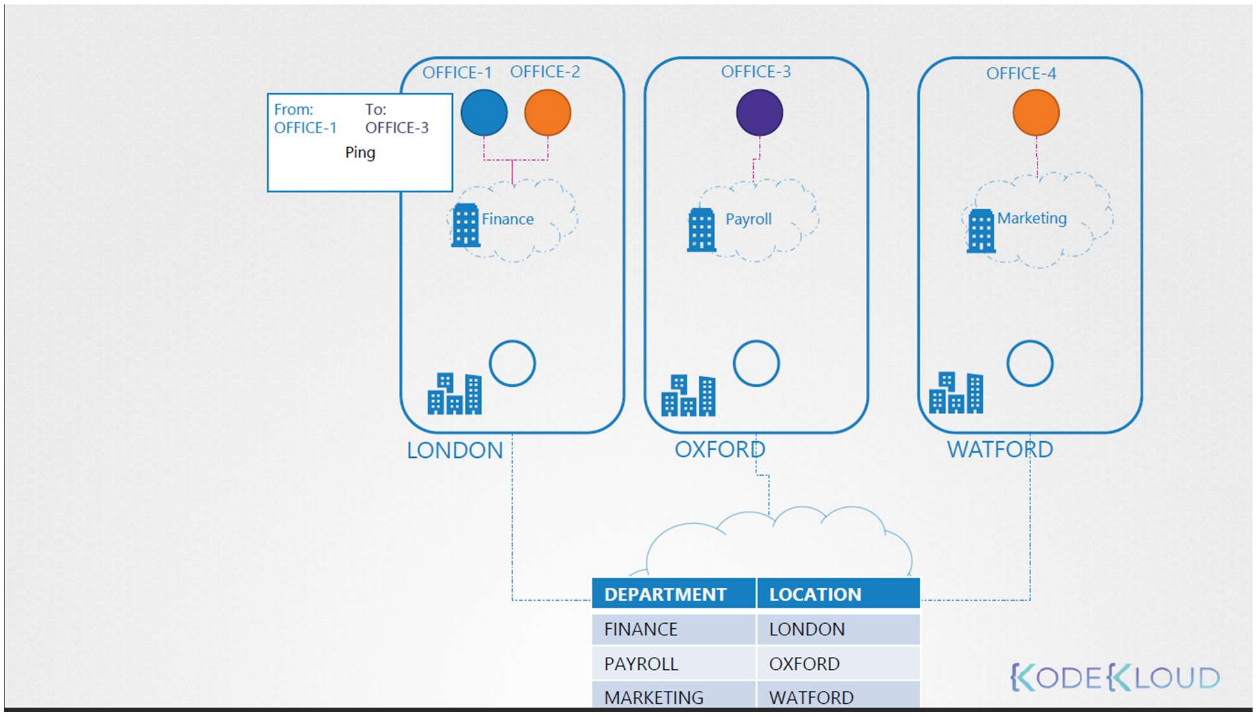


NETWORK	GATEWAY
10.244.1.0/24	192.168.1.11
10.244.2.0/24	192.168.1.12

Let us now see how the Weave solution works as it is important to understand at least one solution well. You should then be able to relate this to other solutions as well.

So the networking solution we set up manually had a routing table which mapped what networks are on what hosts. So when a packet is sent from one pod to the other, it goes out to the network, to the router and finds it's way to the node that hosts that pod. Now that works for a small environment and in a simple network, but in larger environments with hundreds of nodes in a cluster and hundreds of pods on each node, this is not practical. The routing table may not support so many entries and that is where you need to get creative and look for other solutions.

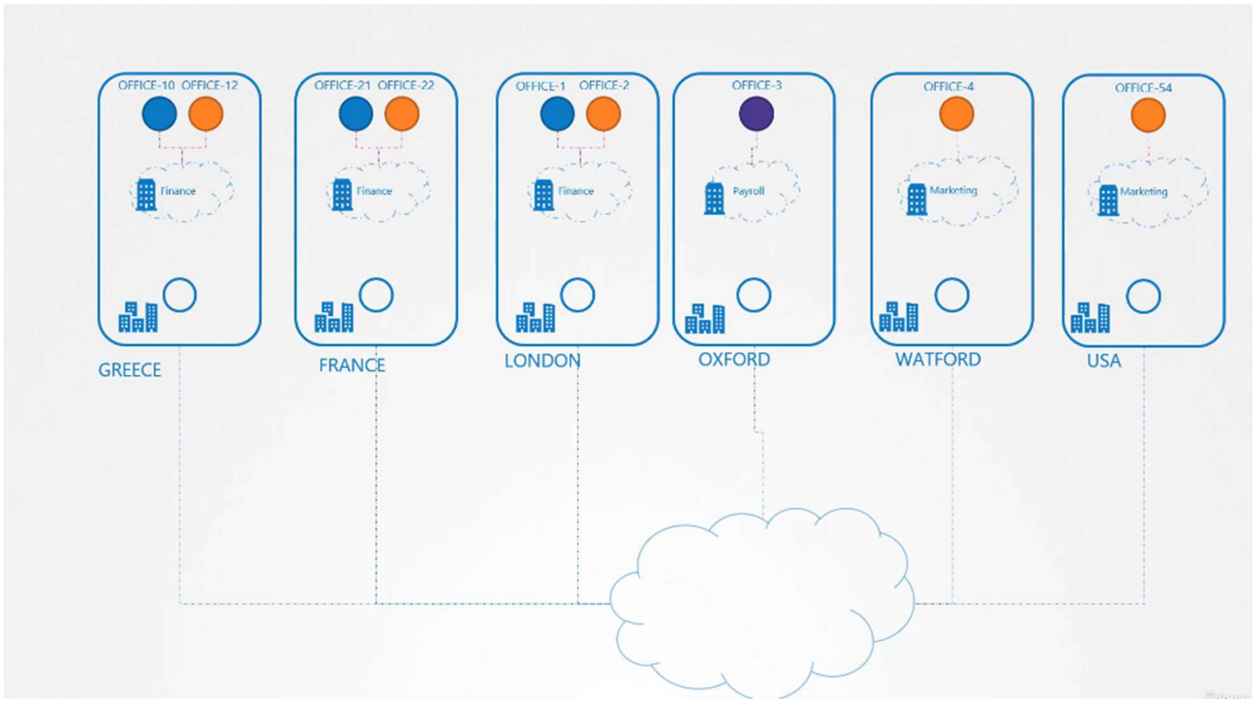
Think of the Kubernetes cluster as our company and the nodes as different office sites. With each site, we have different departments and within each department we have different offices.



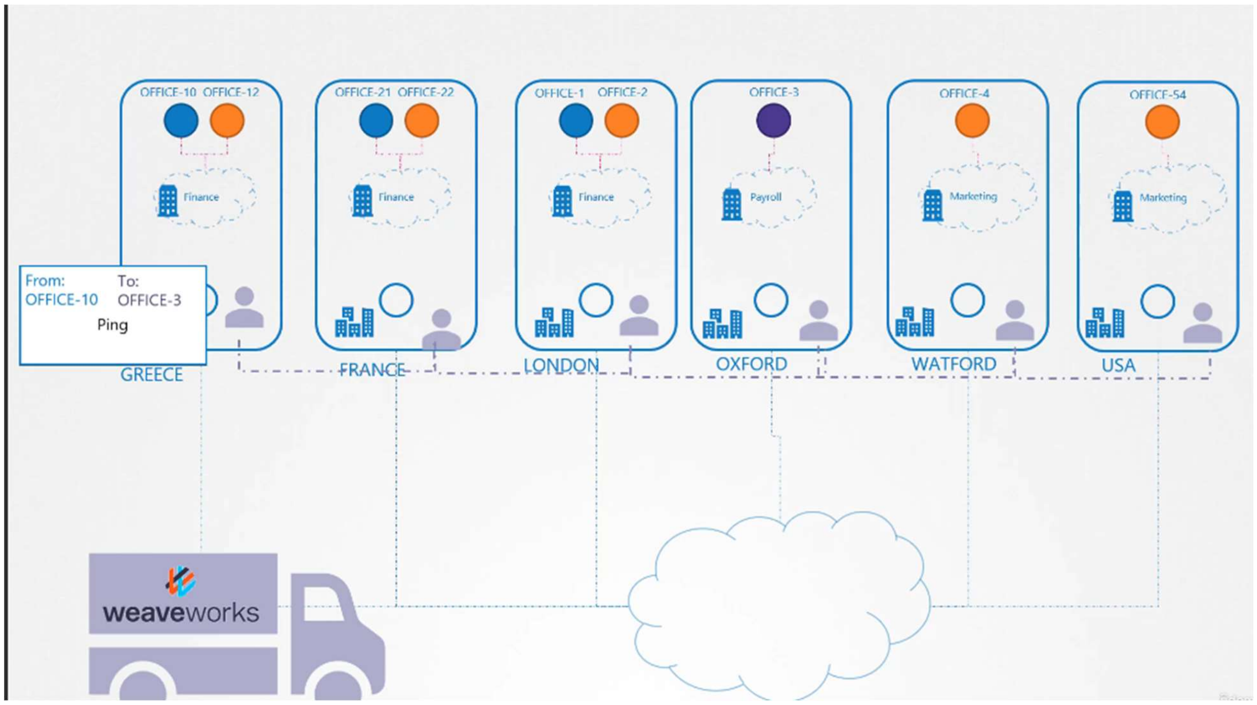
DEPARTMENT	LOCATION
FINANCE	LONDON
PAYROLL	OXFORD
MARKETING	WATFORD

Someone in office one, wants to send a packet to office three and hands it over to the office boy. All he knows is it needs to go to office three and he doesn't care who or how it is transported. The office boy takes the package, gets in his car, looks at the address for the target office in GPS, uses directions on the streets and finds his way to the destination site, delivers the package to the payroll department who in turn forwards the package to office three, this works just fine for now.

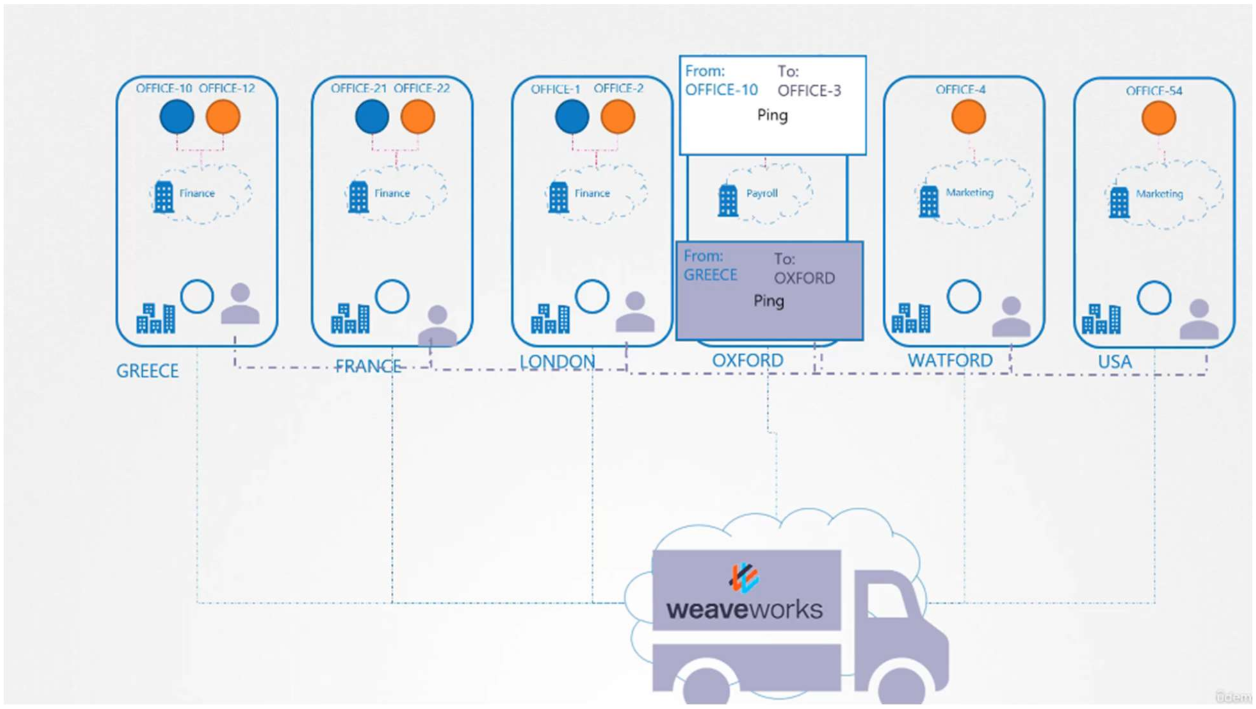
We soon expand to different regions and countries and this process no longer works. It's hard for the office boy to keep track of so many routes to these large number of offices across different countries and of course, he can't drive to these offices by himself.



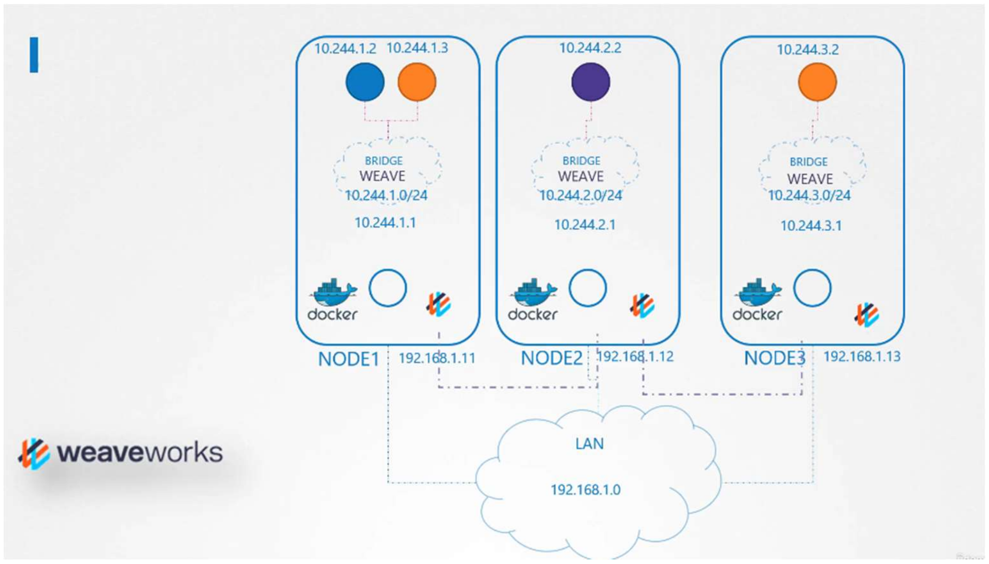
That's where we decide to outsource all mailing and shipping activities to a company who does it best. Once the shipping company is engaged, the first thing that they do is place their agents in each of our company's sites. These agents are responsible for managing all shipping activities between sites. They also keep talking to each other and are well connected. So they all know about each other's sites, the departments in them and the offices in them.



And so when a package is sent from, say, office 10 to office three, the shipping agent in that site intercepts the package and looks at the target office name. He knows exactly in which site and department that office is in, through his little internal network with his peers on the other sites, he then places this package into his own new package with the destination address set to the target sites' location and then sends the package through. Once the package arrives at the destination it is again intercepted by the agent on that site, he opens the packet, retrieves the original packet and delivers it to the right department.



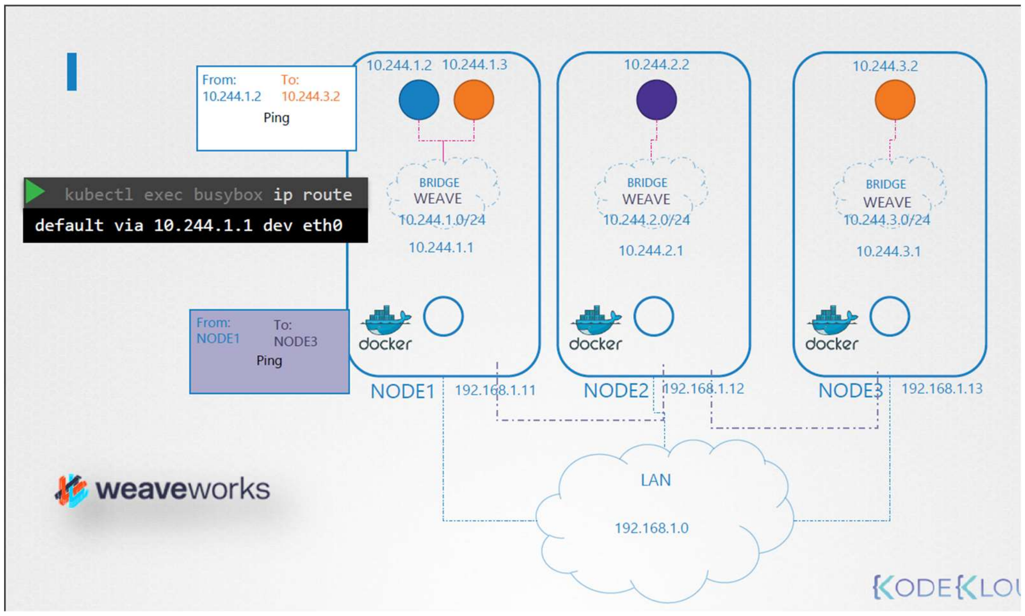
Back to our world where the Weave CNI plugin is deployed on a cluster, it deploys an agent or service on each node. They communicate with each other to exchange information regarding the nodes and networks and pods within them. Each agent or peer stores (indistinct) of the entire setup that way they know the pods and their IPs on the other nodes. Weave creates its own bridge on the nodes and names. It, Weave then assigns IP address to each network.



The IPs shown here are just examples. In the upcoming practice test, you will figure out the exact range of IP addresses, Weave assigns on each node. We will talk about IP address management and how IP addresses are handed out to pods and containers in the next lecture.

Remember that a single pod may be attached to multiple bridge networks. For example, you could have a pod attached to the Weave Bridge as well as the Docker bridge created by Docker. What path a packet takes to reach destination depends on the route configured on the container.

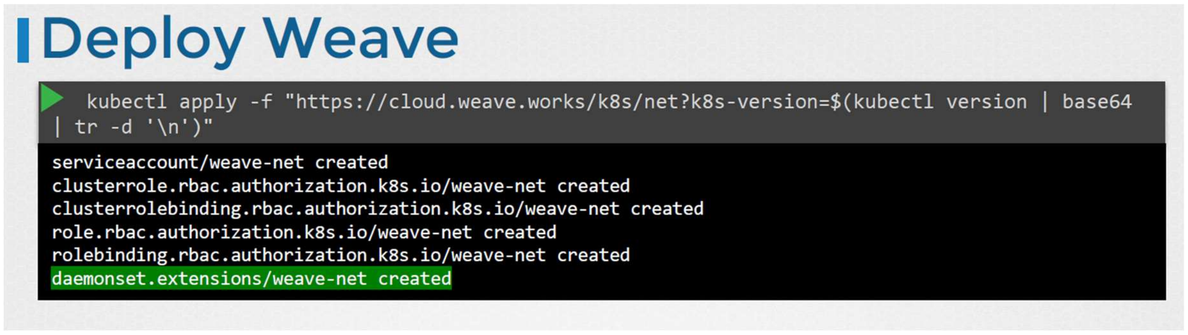
Weave make sure that pods gets the correct route configured to reach the agent and the agent then takes care of other parts. Now, when a packet is sent from one pod to another on another node, Weave intercepts the packet and identifies that it's on a separate network. It then encapsulates this packet into a new one with new source and destination and sends it across the network. Once on the other side the other Weave agent retrieves the packet, decapsulates it and routes the packet to the right pod.



```
kubectl exec busybox ip route
default via 10.244.1.1 dev eth0
```

So how do we deploy Weave on a Kubernetes cluster? Weave and weave peers can be deployed as services or Daemons on each node in the cluster manually or if Kubernetes is set up already then an easier way to do that is to deploy it as pods in the cluster. Once the base Kubernetes system is ready with nodes and networking configured correctly between the node and the basic control plane components are deployed, Weave can be deployed in the cluster with a single kube control apply command.

This deploys all the necessary components required for Weave in the cluster. Most importantly, the Weave peers are deployed as a DaemonSet. A DaemonSet ensures that one part of the given kind is deployed on all nodes in the cluster. This works perfectly for the Weave peers.

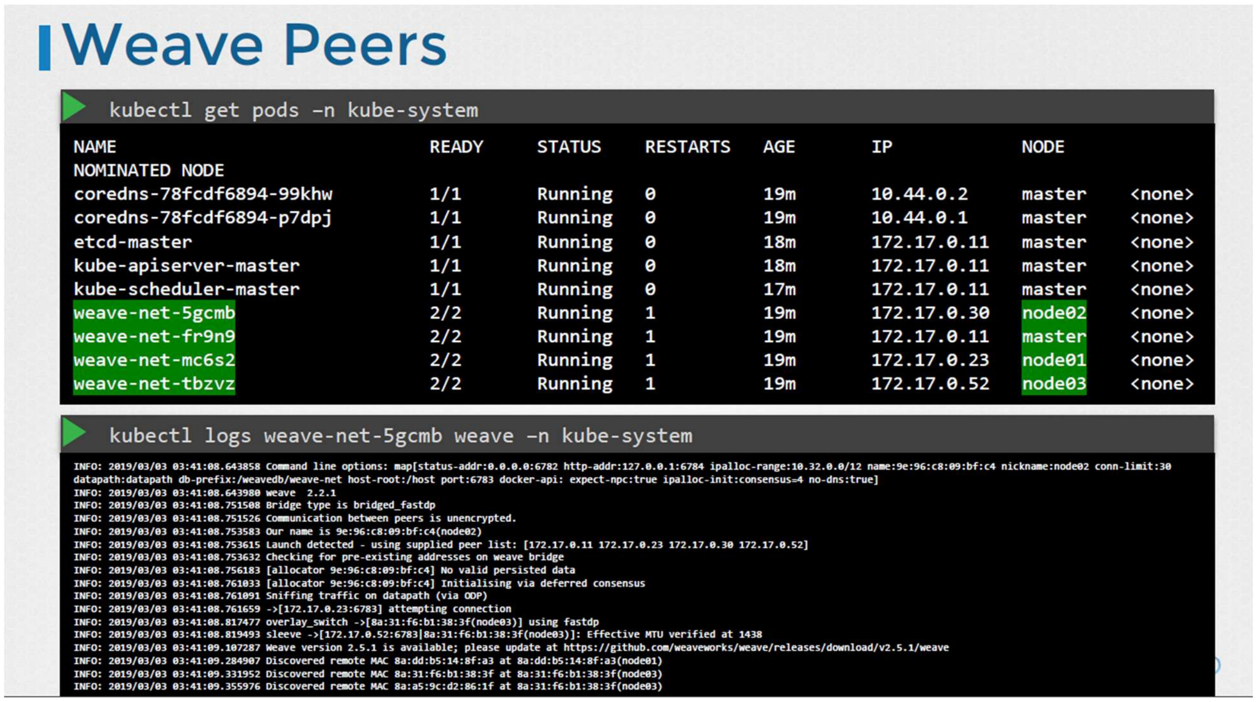


Deploy Weave

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

```
serviceaccount/weave-net created
clusterrole.rbac.authorization.k8s.io/weave-net created
clusterrolebinding.rbac.authorization.k8s.io/weave-net created
role.rbac.authorization.k8s.io/weave-net created
rolebinding.rbac.authorization.k8s.io/weave-net created
daemonset.extensions/weave-net created
```

If you deployed your cluster with a kubeadm tool and Weave plugin, you can see the Weave peers as parts deployed on each node. For troubleshooting purpose, view the logs using the kube control logs command.



Weave Peers

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

```
NAME                              READY     STATUS     RESTARTS    AGE      IP             NODE
NOMINATED NODE
coredns-78fcdf6894-99khw          1/1       Running    0           19m      10.44.0.2      master    <none>
coredns-78fcdf6894-p7dpj          1/1       Running    0           19m      10.44.0.1      master    <none>
etcd-master                       1/1       Running    0           18m      172.17.0.11    master    <none>
kube-apiserver-master             1/1       Running    0           18m      172.17.0.11    master    <none>
kube-scheduler-master             1/1       Running    0           17m      172.17.0.11    master    <none>
weave-net-5gcmb                   2/2       Running    1           19m      172.17.0.30    node02    <none>
weave-net-fr9n9                   2/2       Running    1           19m      172.17.0.11    master    <none>
weave-net-mc6s2                   2/2       Running    1           19m      172.17.0.23    node01    <none>
weave-net-tbzvz                   2/2       Running    1           19m      172.17.0.52    node03    <none>
```

```
kubectl logs weave-net-5gcmb weave –n kube-system
```

```
INFO: 2019/03/03 03:41:08.643858 Command line options: map[status-addr:0.0.0.0:6782 http-addr:127.0.0.1:6784 ipalloc-range:10.32.0.0/12 name:9e:96:c8:09:bf:c4 nickname:node02 conn-limit:30
datapath:datapath db-prefix:/weavedb/weave-net host-root:/host port:6783 docker-api: expect-npc:true ipalloc-init:consensus=4 no-dns:true]
INFO: 2019/03/03 03:41:08.643980 weave  2.2.1
INFO: 2019/03/03 03:41:08.751508 Bridge type is bridged_fastdp
INFO: 2019/03/03 03:41:08.751526 Communication between peers is unencrypted.
INFO: 2019/03/03 03:41:08.753583 Our name is 9e:96:c8:09:bf:c4(node02)
INFO: 2019/03/03 03:41:08.753615 Launch detected - using supplied peer list: [172.17.0.11 172.17.0.23 172.17.0.30 172.17.0.52]
INFO: 2019/03/03 03:41:08.753632 Checking for pre-existing addresses on weave bridge
INFO: 2019/03/03 03:41:08.756183 [allocator 9e:96:c8:09:bf:c4] No valid persisted data
INFO: 2019/03/03 03:41:08.761033 [allocator 9e:96:c8:09:bf:c4] Initialising via deferred consensus
INFO: 2019/03/03 03:41:08.761091 Sniffing traffic on datapath (via ODP)
INFO: 2019/03/03 03:41:08.761659 ->[172.17.0.23:6783] attempting connection
INFO: 2019/03/03 03:41:08.817477 overlay_switch ->[8a:31:f6:b1:38:3f(node03)] using fastdp
INFO: 2019/03/03 03:41:08.819493 sleeve ->[172.17.0.52:6783|8a:31:f6:b1:38:3f(node03)]: Effective MTU verified at 1438
INFO: 2019/03/03 03:41:09.107287 Weave version 2.5.1 is available; please update at https://github.com/weaveworks/weave/releases/download/v2.5.1/weave
INFO: 2019/03/03 03:41:09.284907 Discovered remote MAC 8a:dd:b5:14:8f:a3 at 8a:dd:b5:14:8f:a3(node01)
INFO: 2019/03/03 03:41:09.331952 Discovered remote MAC 8a:31:f6:b1:38:3f at 8a:31:f6:b1:38:3f(node03)
INFO: 2019/03/03 03:41:09.355976 Discovered remote MAC 8a:a5:9c:d2:86:1f at 8a:31:f6:b1:38:3f(node03)
```