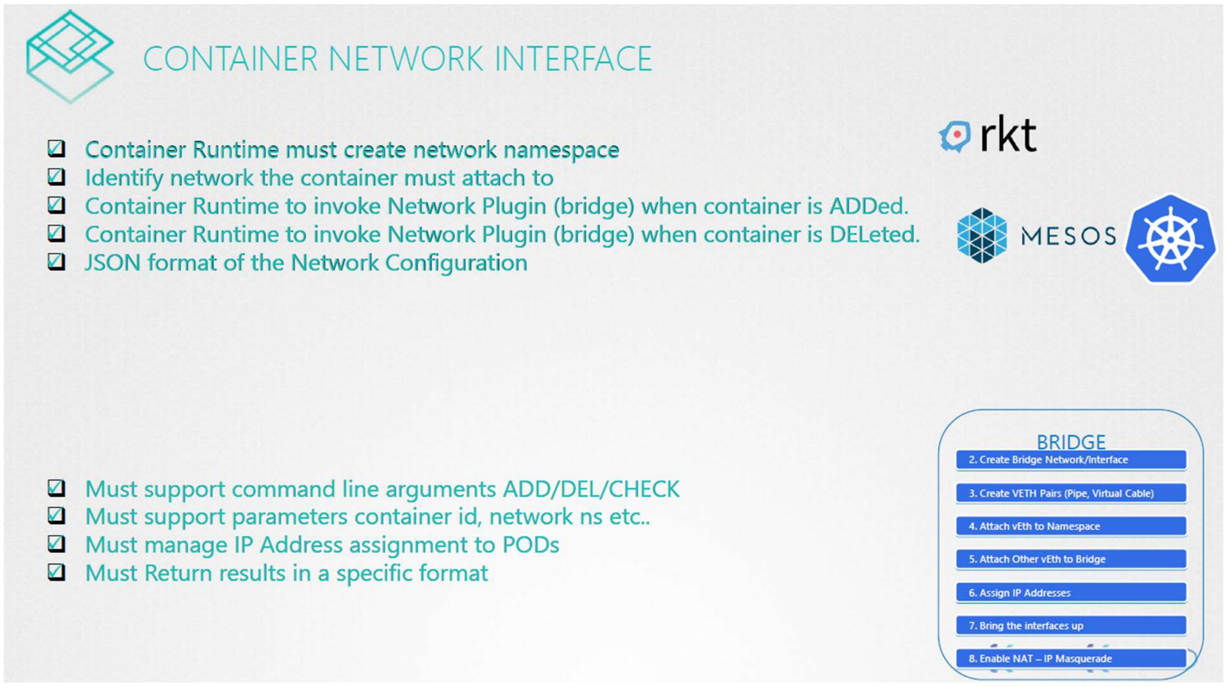
In this lecture we'll discuss about CNI in Kubernetes. In the prerequisite lectures, we started all the way from the absolute basics of network name spaces. Then we saw how it is done in Docker. We then discussed why you need standards for networking containers and how the container network interface came to be. And then we saw a list of supported plugins available with CNI.

In this lecture, we will see how Kubernetes is configured to use these network plugins. As we discussed in the prerequisite lecture CNI defines the responsibilities of container run time as per CNI container run times. In our case, Kubernetes is responsible for creating container network name spaces; identifying and attaching those name spaces to the right network by calling the right network plugin.



## CONTAINER NETWORK INTERFACE

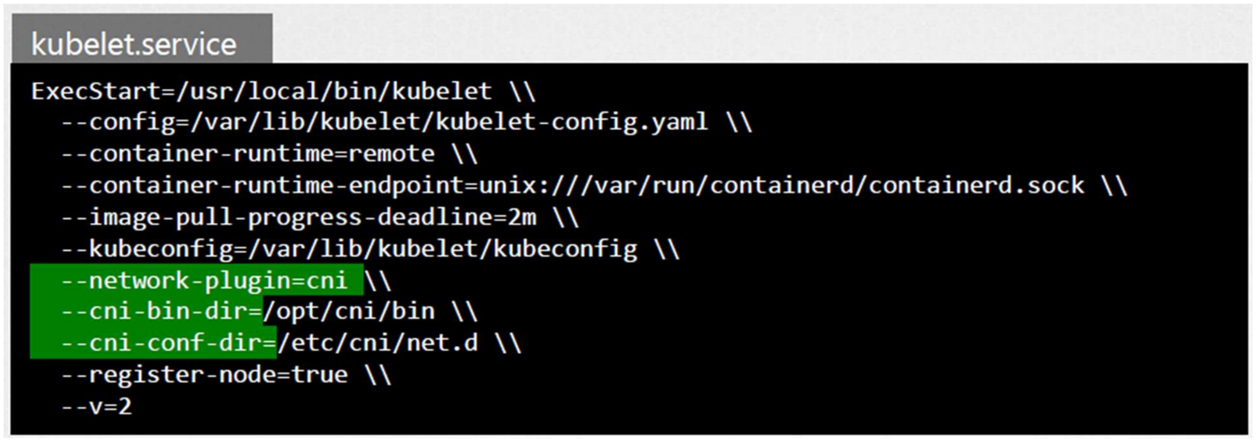
- ☑ Container Runtime must create network namespace
- ☑ Identify network the container must attach to
- ☑ Container Runtime to invoke Network Plugin (bridge) when container is ADDed.
- ☑ Container Runtime to invoke Network Plugin (bridge) when container is DELeted.
- ☑ JSON format of the Network Configuration

- ☑ Must support command line arguments ADD/DEL/CHECK
- ☑ Must support parameters container id, network ns etc..
- ☑ Must manage IP Address assignment to PODs
- ☑ Must Return results in a specific format

rkt
MESOS
BRIDGE
2. Create Bridge Network/Interface
3. Create VETH Pairs (Pipe, Virtual Cable)
4. Attach vEth to Namespace
5. Attach Other vEth to Bridge
6. Assign IP Addresses
7. Bring the interfaces up
8. Enable NAT – IP Masquerade

So where do we specify the CNI plugins for Kubernetes to use? The CNI plugin must be invoked by the component within Kubernetes that is responsible for creating containers because that component must then invoke the appropriate network plugin after the container is created.

The CNI plugin is configured in the kubelet Service on each node in the cluster. If you look at the kubelet Service file you will see an option called network plugin set to CNI.

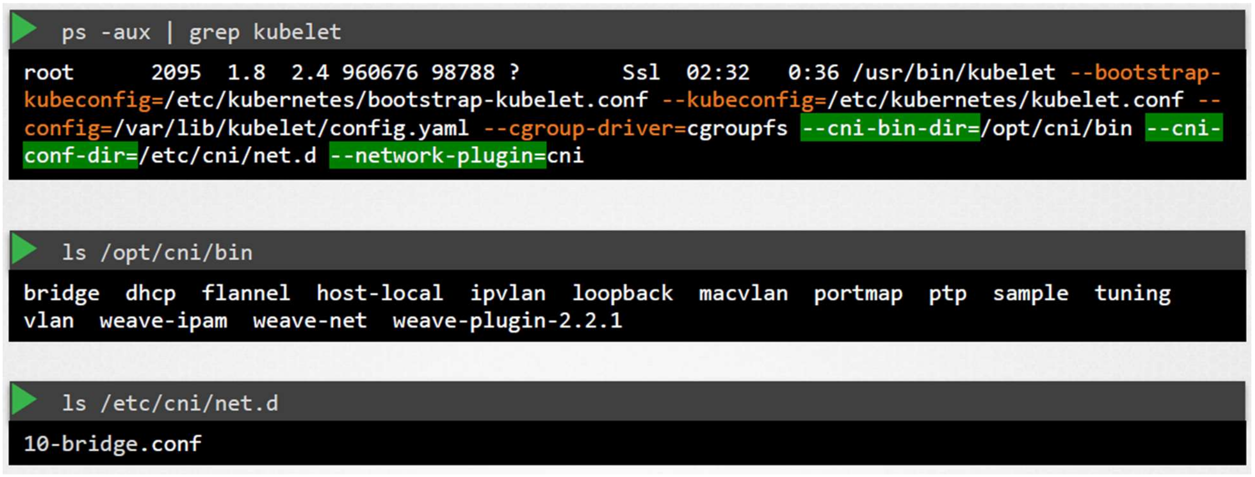


kubelet.service

```
ExecStart=/usr/local/bin/kubelet \\
  --config=/var/lib/kubelet/kubelet-config.yaml \\
  --container-runtime=remote \\
  --container-runtime-endpoint=unix:///var/run/containerd/containerd.sock \\
  --image-pull-progress-deadline=2m \\
  --kubeconfig=/var/lib/kubelet/kubeconfig \\
  --network-plugin=cni \\
  --cni-bin-dir=/opt/cni/bin \\
  --cni-conf-dir=/etc/cni/net.d \\
  --register-node=true \\
  --v=2
```

You can see the same information on viewing the Running kubelet service. You can see the network plugin set to CNI and a few other options related to CNI, such as the CNI bin directory and the CNI config directory.

The CNI bin directory has all the supported CNA plugins as executables, such as the bridge, dscp, flannel, et cetera. The CNI config directory has a set of configuration files. This is where kubelet looks to find out which plugin needs to be used. In this case, it finds the bridge configuration file. If there are multiple files here it will choose the one in alphabetical order.

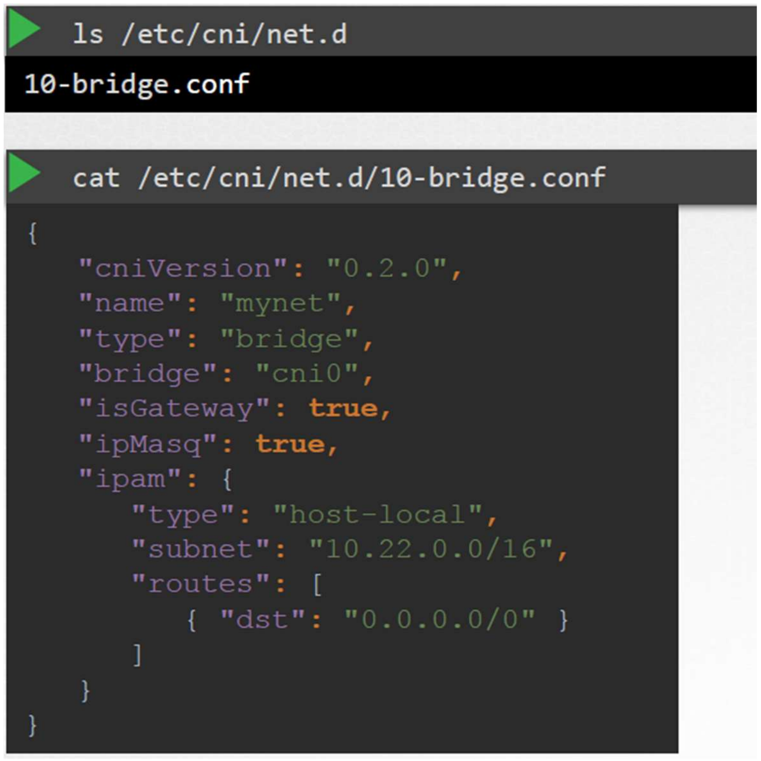


```
ps -aux | grep kubelet
root      2095  1.8  2.4 960676 98788 ?       Ssl  02:32   0:36 /usr/bin/kubelet --bootstrap-
kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf --
config=/var/lib/kubelet/config.yaml --cgroup-driver=cgroupfs --cni-bin-dir=/opt/cni/bin --cni-
conf-dir=/etc/cni/net.d --network-plugin=cni
```

```
ls /opt/cni/bin
bridge  dhcp  flannel  host-local  ipvlan  loopback  macvlan  portmap  ptp  sample  tuning
vlan  weave-ipam  weave-net  weave-plugin-2.2.1
```

```
ls /etc/cni/net.d
10-bridge.conf
```

If you look at the bridge conf file, it looks like this. This is a format defined by the CNI standard for a plug-in configuration file.



```
ls /etc/cni/net.d
10-bridge.conf
```

```
cat /etc/cni/net.d/10-bridge.conf
{
   "cniVersion": "0.2.0",
   "name": "mynet",
   "type": "bridge",
   "bridge": "cni0",
   "isGateway": true,
   "ipMasq": true,
   "ipam": {
      "type": "host-local",
      "subnet": "10.22.0.0/16",
      "routes": [
         { "dst": "0.0.0.0/0" }
      ]
   }
}
```

It's name is MyNet. Type is bridge. It also has a set of other configurations which can be related to the concepts we discussed in the prerequisite lectures on bridging routing, and masquerading. In that the IS gateway defines whether the bridge network should get an IP address assigned to it so that it can act as a gateway.

The IP masquerade defines if a NAT rule should be added for IP masquerading. The IPAM section defines IPAM configuration. This is where you specify the subnet or the range of IP addresses that will be assigned to pods and any necessary routs.

The type host local indicates that the IP addresses are managed locally on this host unlike DHCP server, maintaining it remotely. The type can also be set to DHCP to configure an external DHCP server.