https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

Task:1

Install the `kubeadm` package on the `controlplane` and `node01`.

0.5)

controlplane $ **cat /etc/os-release**

NAME="Ubuntu"

VERSION="18.04.5 LTS (Bionic Beaver)"

ID=ubuntu

ID_LIKE=debian

PRETTY_NAME=**"Ubuntu 18.04.5 LTS"**

VERSION_ID="18.04"

HOME_URL="https://www.ubuntu.com/"

SUPPORT_URL="https://help.ubuntu.com/"

BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"

PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"

VERSION_CODENAME=bionic

UBUNTU_CODENAME=bionic

controlplane $

1)

Follow the steps in the official documentation.

These steps have to be performed on both nodes.

1.`set net.bridge.bridge-nf-call-iptables to 1`:

```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF

cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sudo sysctl --system
```

2.

The `docker` runtime has already been installed on both nodes, so you may skip this step.

root@controlplane:~# docker --version

Docker version 19.03.0, build aeac949

root@controlplane:~#

Installing runtime

To run containers in Pods, Kubernetes uses a container runtime.

Linux nodes | other operating systems

By default, Kubernetes uses the Container Runtime Interface (CRI) to interface with your chosen container runtime.

If you don't specify a runtime, kubeadm automatically tries to detect an installed container runtime by scanning through a list of well known Unix domain sockets. The following table lists container runtimes and their associated socket paths:

Runtime	Path to Unix domain socket
Docker	/var/run/dockershim.sock
containerd	/run/containerd/containerd.sock
CRI-O	/var/run/crio/crio.sock

If both Docker and containerd are detected, Docker takes precedence. This is needed because Docker 18.09 ships with containerd and both are detectable even if you only installed Docker. If any other two or more runtimes are detected, kubeadm exits with an error.

The kubelet integrates with Docker through the built-in `dockershim` CRI implementation.

See container runtimes for more information.

3. Install `kubeadm, kubectl and kubelet` on all nodes:

```
3.1
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
```

3.2

```
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

3.3

```
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

3.4

```
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

4. Set the correct cgroup driver: [new setting in k8s v1.22]

https://kubernetes.io/docs/setup/production-environment/container-runtimes/

Before Update:

root@controlplane:~#
root@controlplane:~# cat /etc/docker/daemon.json | grep -i driver
  "native.cgroupdriver=**cgroupfs**"
root@controlplane:~#
root@controlplane:~#

```
root@controlplane:~# vi /etc/docker/daemon.json
```

After Update:
root@controlplane:~# vi /etc/docker/daemon.json

Save & Exit.

root@controlplane:~# cat /etc/docker/daemon.json | grep -i driver
  "native.cgroupdriver=**systemd**"
root@controlplane:~#

Restart Docker :

```
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

Repeat above steps on, all the nodes of the cluster :

controlplane# **ssh node01**
node01#
.
.
.
.
4. Set the correct cgroup driver: [new setting in k8s v1.22]

https://kubernetes.io/docs/setup/production-environment/container-runtimes/

Before Update:
root@node01:~#
root@node01:~# cat /etc/docker/daemon.json | grep -i driver
"native.cgroupdriver=**cgroupfs**"
root@node01:~#
root@node01:~#

After Update:
root@**node01**:~# vi /etc/docker/daemon.json

Save & Exit.

root@**node01**:~# cat /etc/docker/daemon.json | grep -i driver
"native.cgroupdriver=**systemd**"
root@**node01**:~#
.
.
.
.
.
root@node01# exit

controlplane#

5)
Lets now bootstrap a `kubernetes` cluster using `kubeadm`.

The latest version of Kubernetes will be installed.

Initialize `Control Plane Node (Master Node)`. Use the following options:

`apiserver-advertise-address` - Use the IP address allocated to eth0 on the controlplane node

`apiserver-cert-extra-sans` - Set it to `controlplane`

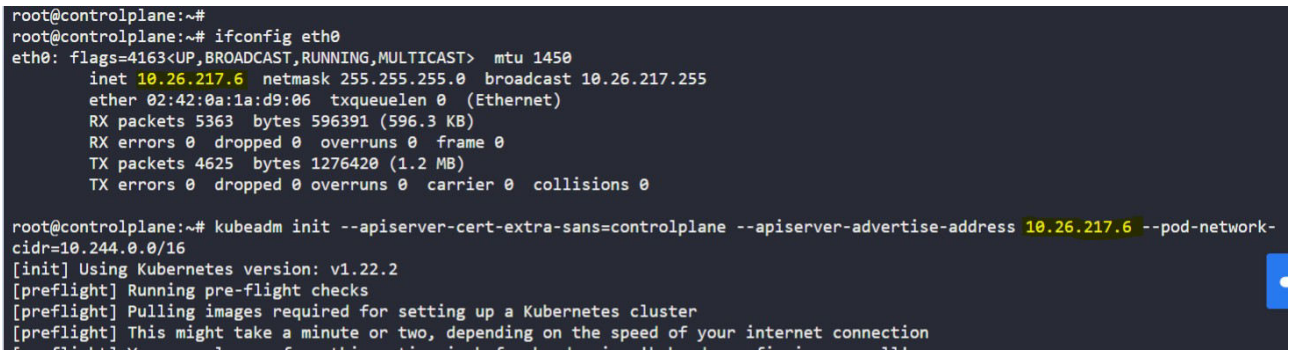
`pod-network-cidr` - Set to `10.244.0.0/16`

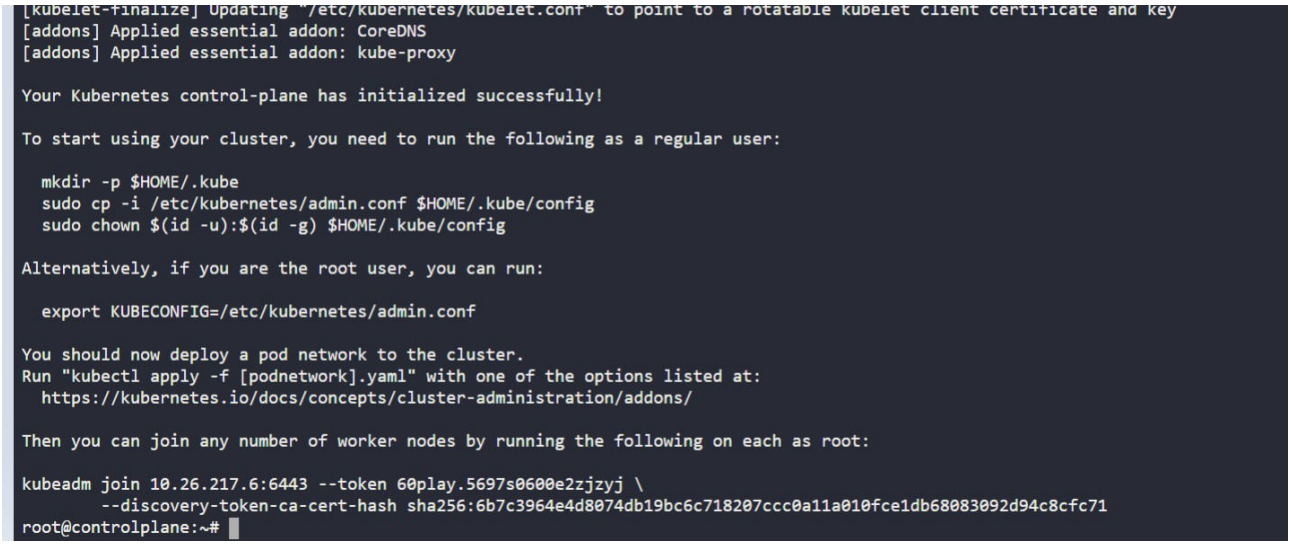
Once done, set up the `default kubeconfig` file and wait for node to be part of the cluster.

Hint:
run `kubeadm init --apiserver-cert-extra-sans=controlplane --apiserver-advertise-address 10.26.217.6 --pod-network-cidr=10.244.0.0/16`
The IP address used here is just an example. It will change for your lab session. Make sure to check the IP address allocated to eth0 by running:

```
root@controlplane:~# ifconfig eth0
```

In this example, the IP address is `10.26.217.6`





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

kubeadm join 10.26.217.6:6443 --token 60play.5697s0600e2zjzyj \
        --discovery-token-ca-cert-hash sha256:6b7c3964e4d8074db19bc6c718207ccc0a11a010fce1db68083092d94c8cfc71
root@controlplane:~#
root@controlplane:~#
root@controlplane:~#
root@controlplane:~#  mkdir -p $HOME/.kube
root@controlplane:~#   sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root@controlplane:~#   sudo chown $(id -u):$(id -g) $HOME/.kube/config
root@controlplane:~#
root@controlplane:~# kubectl get nodes
NAME           STATUS     ROLES                  AGE   VERSION
controlplane   NotReady   control-plane,master   76s   v1.22.2
root@controlplane:~#
root@controlplane:~# kubectl get po -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
coredns-78fcd69978-g99qh               0/1     Pending   0          75s
coredns-78fcd69978-pmncz               0/1     Pending   0          74s
etcd-controlplane                      1/1     Running   0          83s
kube-apiserver-controlplane            1/1     Running   0          89s
kube-controller-manager-controlplane   1/1     Running   0          83s
kube-proxy-hf55b                       1/1     Running   0          75s
kube-scheduler-controlplane            1/1     Running   0          83s
root@controlplane:~#
root@controlplane:~# ssh node01
Last login: Sat Oct 23 15:54:27 2021 from 10.26.217.7
root@node01:~#
root@node01:~# kubeadm join 10.26.217.6:6443 --token 60play.5697s0600e2zjzyj \
```

```
root@controlplane:~# ssh node01
Last login: Sat Oct 23 15:54:27 2021 from 10.26.217.7
root@node01:~#
root@node01:~# kubeadm join 10.26.217.6:6443 --token 60play.5697s0600e2zjzyj \
>         --discovery-token-ca-cert-hash sha256:6b7c3964e4d8074db19bc6c718207ccc0a11a010fce1db68083092d94c8cfc71
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.

root@node01:~#
```

6)

Install a Network Plugin. As a default, we will go with **`flannel`**

On the controlplane, run: `kubectl apply -f`

`https://raw.githubusercontent.com/coreos/flannel/master/Documentation/`

`kube-flannel.yml`

```
root@controlplane:~# kubectl  get po -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
coredns-78fcd69978-g99qh               1/1     Running   0          20m
coredns-78fcd69978-pmncz               1/1     Running   0          20m
etcd-controlplane                      1/1     Running   0          20m
kube-apiserver-controlplane            1/1     Running   0          20m
kube-controller-manager-controlplane   1/1     Running   0          20m
kube-proxy-hf55b                       1/1     Running   0          20m
kube-proxy-wd4r9                       1/1     Running   0          18m
kube-scheduler-controlplane            1/1     Running   0          20m
root@controlplane:~#
root@controlplane:~#
root@controlplane:~# kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25+
podsecuritypolicy.policy/psp.flannel.unprivileged created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
root@controlplane:~#
```

```
root@controlplane:~#
root@controlplane:~# kubectl  get po -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
coredns-78fcd69978-g99qh               1/1     Running   0          20m
coredns-78fcd69978-pmncz               1/1     Running   0          20m
etcd-controlplane                      1/1     Running   0          20m
kube-apiserver-controlplane            1/1     Running   0          21m
kube-controller-manager-controlplane   1/1     Running   0          20m
kube-flannel-ds-jgch8                  1/1     Running   0          26s
kube-flannel-ds-lqkmr                  1/1     Running   0          26s
kube-proxy-hf55b                       1/1     Running   0          20m
kube-proxy-wd4r9                       1/1     Running   0          19m
kube-scheduler-controlplane            1/1     Running   0          20m
root@controlplane:~#
root@controlplane:~#
root@controlplane:~# kubectl get nodes
NAME           STATUS   ROLES                  AGE   VERSION
controlplane   Ready    control-plane,master   21m   v1.22.2
node01         Ready    <none>                 19m   v1.22.2
root@controlplane:~#
root@controlplane:~#
```

```
root@controlplane:~# kubectl  get po -n kube-system -owide
NAME                                   READY   STATUS    RESTARTS   AGE     IP           NODE           NOMINATED NODE   READINESS
 GATES
coredns-78fcd69978-g99qh               1/1     Running   0          25m     10.32.0.2    controlplane   <none>           <none>
coredns-78fcd69978-pmncz               1/1     Running   0          25m     10.32.0.3    controlplane   <none>           <none>
etcd-controlplane                      1/1     Running   0          25m     10.26.217.6  controlplane   <none>           <none>
kube-apiserver-controlplane            1/1     Running   0          25m     10.26.217.6  controlplane   <none>           <none>
kube-controller-manager-controlplane   1/1     Running   0          25m     10.26.217.6  controlplane   <none>           <none>
kube-flannel-ds-jgch8                  1/1     Running   0          4m58s   10.26.217.9  node01         <none>           <none>
kube-flannel-ds-lqkmr                  1/1     Running   0          4m58s   10.26.217.6  controlplane   <none>           <none>
kube-proxy-hf55b                       1/1     Running   0          25m     10.26.217.6  controlplane   <none>           <none>
kube-proxy-wd4r9                       1/1     Running   0          23m     10.26.217.9  node01         <none>           <none>
kube-scheduler-controlplane            1/1     Running   0          25m     10.26.217.6  controlplane   <none>           <none>
root@controlplane:~#
```

Reference:

Configuring a cgroup driver

Both the container runtime and the kubelet have a property called "cgroup driver", which is important for the management of cgroups on Linux machines.

> **Warning:**
> Matching the container runtime and kubelet cgroup drivers is required or otherwise the kubelet process will fail.
>
> See Configuring a cgroup driver for more details.

https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/configure-cgroup-driver/

Configuring the container runtime cgroup driver

The Container runtimes page explains that the `systemd` driver is recommended for kubeadm based setups instead of the `cgroupfs` driver, because kubeadm manages the kubelet as a systemd service.

The page also provides details on how to setup a number of different container runtimes with the `systemd` driver by default.

Cgroup drivers

Control groups are used to constrain resources that are allocated to processes.

When systemd is chosen as the init system for a Linux distribution, the init process generates and consumes a root control group (`cgroup`) and acts as a cgroup manager. Systemd has a tight integration with cgroups and allocates a cgroup per systemd unit. It's possible to configure your container runtime and the kubelet to use `cgroupfs` . Using `cgroupfs` alongside systemd means that there will be two different cgroup managers.

A single cgroup manager simplifies the view of what resources are being allocated and will by default have a more consistent view of the available and in-use resources. When there are two cgroup managers on a system, you end up with two views of those resources. In the field, people have reported cases where nodes that are configured to use `cgroupfs` for the kubelet and Docker, but `systemd` for the rest of the processes, become unstable under resource pressure.

Changing the settings such that your container runtime and kubelet use `systemd` as the cgroup driver stabilized the system. To configure this for Docker, set `native.cgroupdriver=systemd` .

> **Caution:**
> Changing the cgroup driver of a Node that has joined a cluster is a sensitive operation. If the kubelet has created Pods using the semantics of one cgroup driver, changing the container runtime to another cgroup driver can cause errors when trying to re-create the Pod sandbox for such existing Pods. Restarting the kubelet may not solve such errors.
>
> If you have automation that makes it feasible, replace the node with another using the updated configuration, or reinstall it using automation.

Docker

1. On each of your nodes, install the Docker for your Linux distribution as per Install Docker Engine. You can find the latest validated version of Docker in this dependencies file.

2. Configure the Docker daemon, in particular to use systemd for the management of the container's cgroups.

```
sudo mkdir /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
```

> **Note:** `overlay2` is the preferred storage driver for systems running Linux kernel version 4.0 or higher, or RHEL or CentOS using version 3.10.0-514 and above.

3. Restart Docker and enable on boot:

```
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```