Procedure.

kubeadm is a tool that allows you to easily create Kubernetes clusters that adhere to best practices. It can also perform a variety of cluster lifecycle functions, such as upgrading and downgrading the version of Kubernetes on nodes in the cluster. You will use kubeadm to create a Kubernetes cluster from scratch in this Lab. Creating clusters with kubeadm is the recommended way for learning Kubernetes, creating small clusters, and as a piece of a more complex systems for more enterprise-ready clusters.

Installing kubeadm and Its Dependencies

MASTER NODE:

1. Enter the following command to update the system's apt package manager index and update packages required to install Docker:

```
# Update the package index
sudo apt-get update
# Update packages required for HTTPS package repository access
sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common
```

2. Install Docker community edition using Ubuntu's apt package manager and the official Docker repository:

```
# Add Docker's GPG key

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Configure the stable Docker release repository

sudo add-apt-repository \

 "deb [arch=amd64] https://download.docker.com/linux/ubuntu \

 $(lsb_release -cs) \

 stable"

# Update the package index to include the stable Docker repository

sudo apt-get update

# Install Docker

sudo apt-get install -y docker-ce=17.03.2~ce-0~ubuntu-xenial
```

The installation instructions are based on Docker's official instructions. Although it is possible to install Ubuntu's docker.io package, to control the version and use a version that is officially supported by Kubernetes, Docker community edition 17.03 is installed directly from Docker's package repository.

3. Confirm Docker 17.03 is installed:

```
docker -version
```

4. Install kubeadm, kubectl, and kubelet from the official Kubernetes package repository:

```
# Add the Google Cloud packages GPG key
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

```
# Add the Kubernetes release repository
sudo add-apt-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"
# Update the package index to include the Kubernetes repository
sudo apt-get update
# Install the packages
sudo apt-get install -y kubeadm=1.13.4-00 kubelet=1.13.4-00 kubectl=1.13.4-00
kubernetes-cni=0.6.0-00
# Prevent automatic updates to the installed packages
sudo apt-mark hold kubelet kubeadm kubectl
```

5. Display the help page for kubeadm:

```
kubeadm
```

Initializing the Kubernetes Master Node

You will use kubeadm to initialize the master node in this Lab Step. The initialization process will create a certificate authority for secure cluster communication and authentication, and start all the node components (kubelet), master components (API server, controller manager, scheduler, etcd), and common add-ons (kube-proxy, DNS). You will see how easy the initialization process is with kubeadm.
The initialization uses sensible default values that adhere to best practices. However, many command options are available to configure the process, including if you want to provide your own certificate authority or if you want to use an external etcd key-value store. One option that you will use is required by the pod network plugin that you will install after the master is initialized. kubeadm does not install a default network plugin for you. You will use Weave as the pod network plugin. Weave supports Kubernetes network policies. For network policies to function properly, you must use the --pod-network-cidr option to specify a range of IP addresses for the pod network when initializing the master node with kubeadm.

1. Initialize the master node using the init command:

```
 sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --kubernetes-version=stable-
1.13
```

The pod network CIDR block (192.168.0.0/16) is the default used by Weave.

2*. **Copy the kubeadm join command at the end of the output*** and store it somewhere you can access later.

Example:

```
kubeadm join 10.0.0.77:6443 --token 0e72cy.zhvb54ol7cggl6od --discovery-token-ca-
cert-hash
sha256:3a79dd47e12755f207dff1c4a36b406fd138642ee3a04c3f7d5e6c1ab33c8ad4
```

3. Initialize your user's default kubectl configuration using the admin kubeconfig file generated by kubeadm:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

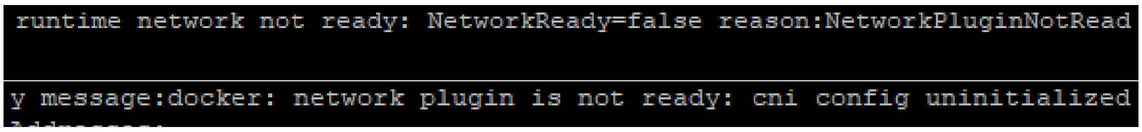
4. Confirm you can use kubectl to get the cluster component statuses:

kubectl get componentstatuses

5. Get the nodes in the cluster:
kubectl get nodes
The **master** node is reporting a **STATUS** of **NotReady**. Notice kubeadm gives the node a **NAME** based on its IP address. The --node-name option can be used to override the default behavior.

6. Describe the node to probe deeper into its NotReady status:
kubectl describe nodes

In the **Conditions** section of the output, observe the **Ready** condition is False, and read the **Message**:



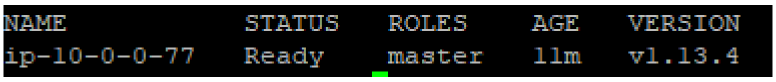
```
runtime network not ready: NetworkReady=false reason:NetworkPluginNotRead

y message:docker: network plugin is not ready: cni config uninitialized
```

The kubelet is not ready because the network plugin is not ready. The **cni config uninitialized** refers to the container network interface (CNI) and is a related problem. Network plugins implement the CNI interface. You will resolve the issue by initializing the Weave network plugin.

7. Enter the following commands to install the Weave pod network plugin:
kubectl apply -f https://docs.projectcalico.org/v3.1/getting-started/kubernetes/installation/hosted/rbac-kdd.yaml
kubectl apply -f https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')&env.IPALLOC_RANGE=192.168.0.0/16

The commands first install the cluster roles and bindings that are used by Weave (rbac-kdd.yaml). Then a variety of resources are created to support pod networking. A **daemonset** is used to run a **Weave-node** pod on each node in the cluster. The resources include several custom resources (**customresourcedefinition**) that extend the Kubernetes API, for example, to support network policies (**networkpolicies.crd.projectWeave.org**). Many network plugins have a similar installation procedure.

8. Check the status of the nodes in the cluster:
kubectl get nodes



```
NAME            STATUS    ROLES     AGE    VERSION
ip-10-0-0-77    Ready     master    11m    v1.13.4
```

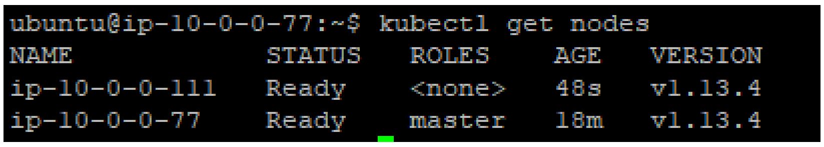
Joining a Worker Node to the Kubernetes Cluster

The process of adding a worker node with kubeadm is even simpler than initializing a master node. You will join a worker node to the cluster using the command that kubeadm init provided in this Lab Step.

WORKER NODE:

2. Enter sudo followed by the kubeadm join command that you stored from the output of kubeadm init. It resembles:
Example:
sudo kubeadm join 10.0.0.77:6443 --token 0e72cy.zhvb54ol7cggl6od --discovery-token-ca-cert-hash sha256:3a79dd47e12755f207dff1c4a36b406fd138642ee3a04c3f7d5e6c1ab33c8ad4

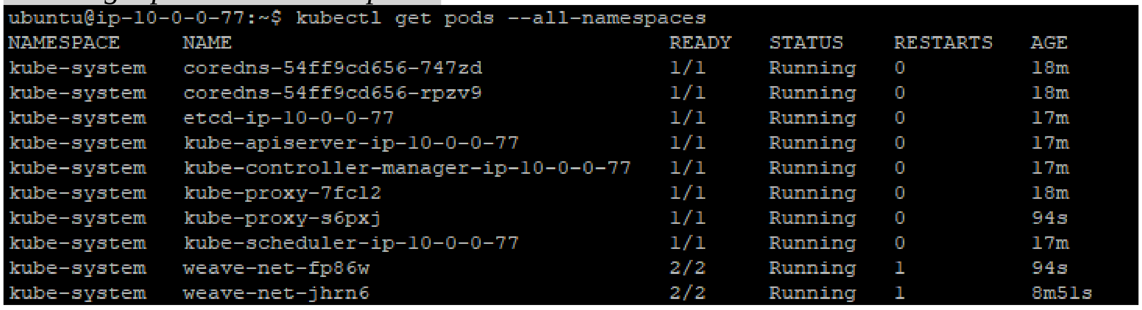
MASTER NODE:
kubectl get nodes



```
ubuntu@ip-10-0-0-77:~$ kubectl get nodes
NAME             STATUS   ROLES    AGE   VERSION
ip-10-0-0-111    Ready    <none>   48s   v1.13.4
ip-10-0-0-77     Ready    master   18m   v1.13.4
```

4. Confirm that all the pods in the cluster are running:

kubectl get pods --all-namespaces



```
ubuntu@ip-10-0-0-77:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                   READY   STATUS    RESTARTS   AGE
kube-system   coredns-54ff9cd656-747zd               1/1     Running   0          18m
kube-system   coredns-54ff9cd656-rpzv9               1/1     Running   0          18m
kube-system   etcd-ip-10-0-0-77                      1/1     Running   0          17m
kube-system   kube-apiserver-ip-10-0-0-77            1/1     Running   0          17m
kube-system   kube-controller-manager-ip-10-0-0-77   1/1     Running   0          17m
kube-system   kube-proxy-7fcl2                       1/1     Running   0          18m
kube-system   kube-proxy-s6pxj                       1/1     Running   0          94s
kube-system   kube-scheduler-ip-10-0-0-77            1/1     Running   0          17m
kube-system   weave-net-fp86w                        2/2     Running   1          94s
kube-system   weave-net-jhrn6                        2/2     Running   1          8m51s
```

Backing Up and Restoring Kubernetes Clusters

The state information of a Kubernetes cluster is stored in etcd. You can back up a Kubernetes cluster and restore it to an earlier state by using the snapshot and restore functionality of etcd. You will explore this functionality in this Lab Step. You will use a command-line client named etcdctl to interact with the Kubernetes ectd key-value store. Rather than install etcdctl on the host machine, which is a viable option, you will use pods and containers to perform the backup and restore operations.

1. In the master node SSH shell, create a deployment of the Nginx application with two replicas:

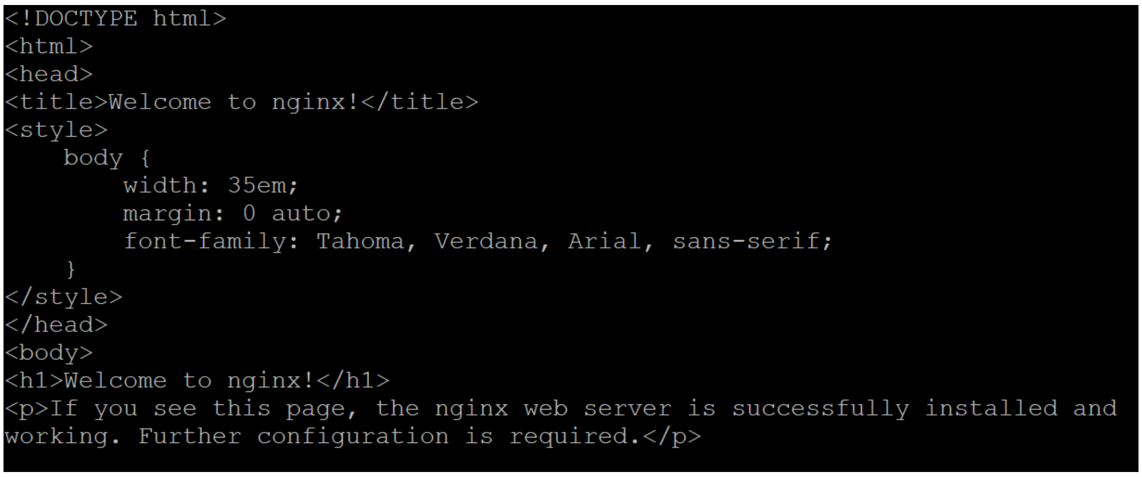
```
kubectl create deployment nginx --image=nginx
kubectl scale deployment nginx --replicas=2
```

2. Expose the deployment using a ClusterIP service:

```
kubectl expose deployment nginx --type=ClusterIP --port=80 --target-port=80 --
name=web
```

3. Send a HTTP request to the web service:

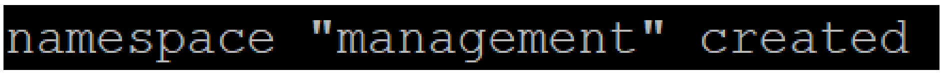
```
# Get the Cluster IP of the service
service_ip=$(kubectl get service web -o jsonpath='{.spec.clusterIP}')
# Use curl to send an HTTP request to the service
curl $service_ip
```



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>
```

4. 4. Create a management namespace:

```
kubectl create namespace management
```



```
namespace "management" created
```

5. Create a job that creates a pod, and issues the etcdctl snapshot save command to back up the cluster:

```
cat <<EOF | kubectl create -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: backup
  namespace: management
spec:
  template:
    spec:
      containers:
      # Use etcdctl snapshot save to create a snapshot in the /snapshot directory
      - command:
        - /bin/sh
        args:
        - -ec
        - etcdctl --cacert=/etc/kubernetes/pki/etcd/ca.crt --
cert=/etc/kubernetes/pki/etcd/peer.crt --key=/etc/kubernetes/pki/etcd/peer.key snapshot
save /snapshots/backup.db
        # The same image used by the etcd pod
        image: k8s.gcr.io/etcd-amd64:3.1.12
        name: etcdctl
        env:
        # Set the etcdctl API version to 3 (to match the version of etcd installed by
kubeadm)
        - name: ETCDCTL_API
          value: '3'
        volumeMounts:
        - mountPath: /etc/kubernetes/pki/etcd
          name: etcd-certs
          readOnly: true
        - mountPath: /snapshots
          name: snapshots
      # Use the host network where the etcd port is accessible (etcd pod uses host
network)
      # This allows the etcdctl to connect to etcd that is listening on the host network
      hostNetwork: true
      affinity:
        # Use node affinity to schedule the pod on the master (where the etcd pod is)
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: node-role.kubernetes.io/master
                operator: Exists
      restartPolicy: OnFailure
      tolerations:
      # tolerate the master's NoSchedule taint to allow scheduling on the master
```

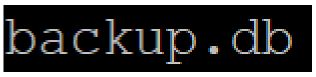
```
   - effect: NoSchedule
     operator: Exists
   volumes:
   # Volume storing the etcd PKI keys and certificates
   - hostPath:
       path: /etc/kubernetes/pki/etcd
       type: DirectoryOrCreate
     name: etcd-certs
   # A volume to store the backup snapshot
   - hostPath:
       path: /snapshots
       type: DirectoryOrCreate
     name: snapshots
EOF
```

You could also create a CronJob Kubernetes resource instead of a one-off Job to periodically perform the backup operation. A Job is sufficient for this Lab. Read through the Job manifest, and use the comments to help understand what it does.

The etcdctl command (see spec.template.spec.containers.args) requires the certificate authority certificate, a client key, and a client certificate to encrypt the etcd traffic. kubeadm configures etcd to listen to HTTPS only as a security best practice. The snapshot save command creates a snapshot of the entire key-value store at the given location (/snapshots/backup.db).

6. List the contents of the /snapshots directory:

```
ls /snapshots
```



```
backup.db
```

The etcd snapshot saved by the pod is present. You will now cause the master to fail and remove the data files of the etcd key-value store to simulate a substantial cluster failure.

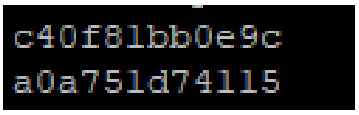
7. Stop the master's kubelet:

```
sudo systemctl stop kubelet.service
```

The kubelet will automatically try to restart the etcd pod if it detects that it has been deleted. You need to stop the kubelet to prevent this.

8. Delete the etcd containers in Docker that are created by the Kubernetes etcd pod:

```
sudo docker ps | grep etcd | cut -d' ' -f1 | xargs sudo docker rm -f
```



```
c40f81bb0e9c
a0a751d74115
```

The container IDs of the deleted etcd containers are displayed.

9. Delete the etcd data files persisted to disk:

```
sudo rm -rf /var/lib/etcd/*
```

The etcd pod mounts /var/lib/etcd to persist its data to disk.

10. Use a Docker container to restore the /var/lib/etcd data from the backup snapshot:

```
sudo docker run --rm \
   -v '/snapshots:/snapshots' \
   -v '/var/lib/etcd:/var/lib/etcd' \
   -e ETCDCTL_API=3 \
   'k8s.gcr.io/etcd-amd64:3.1.12' \
   /bin/sh -c "etcdctl snapshot restore '/snapshots/backup.db' && mv /default.etcd/member
/var/lib/etcd"
```

You need to directly use Docker instead of creating a pod in Kubernetes because Kubernetes will not function with the kubelet and etcd offline. The kubelet will recreate the etcd pod from the static pod manifest in /etc/kubernetes/manifests/etcd.yaml. The etcdctl snapshot restore command performs the restore operation.

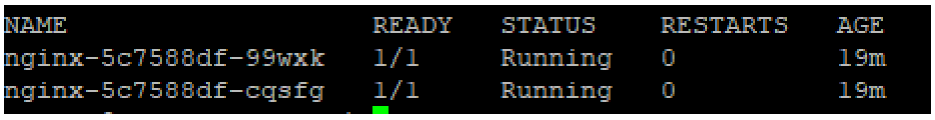
11. Start the kubelet:

```
sudo systemctl start kubelet
```

The kubelet automatically recreates the missing etcd pod containers. The pod will use the restored data files created from the backup, and Kubernetes will have a restored view of the cluster.

12. Confirm the Nginx pods are running:

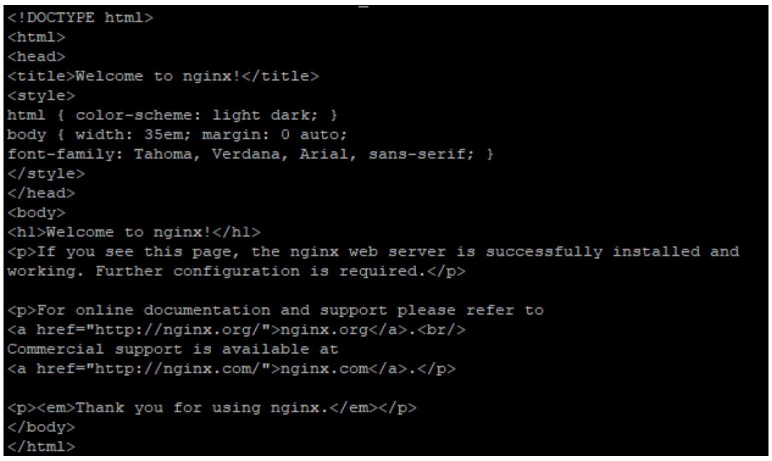
```
kubectl get pods
```



```
NAME                    READY   STATUS    RESTARTS   AGE
nginx-5c7588df-99wxk    1/1     Running   0          19m
nginx-5c7588df-cqsfg    1/1     Running   0          19m
```

13. Confirm the web service works:

```
service_ip=$(kubectl get service web -o jsonpath='{.spec.clusterIP}')
curl $service_ip
```



```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Upgrading Kubernetes Clusters with kubeadm

kubeadm supports upgrading Kubernetes clusters. In this Lab Step, you will be upgrading Kubernetes from version 1.13.4 to version 1.14.1. Although upgrading is supported, you should always take care to understand any changes between releases by reading the release notes and how they could impact your workloads. You should always backup important data before upgrading, and test upgrades before deploying them to production.

The upgrade process follows the general procedure of:

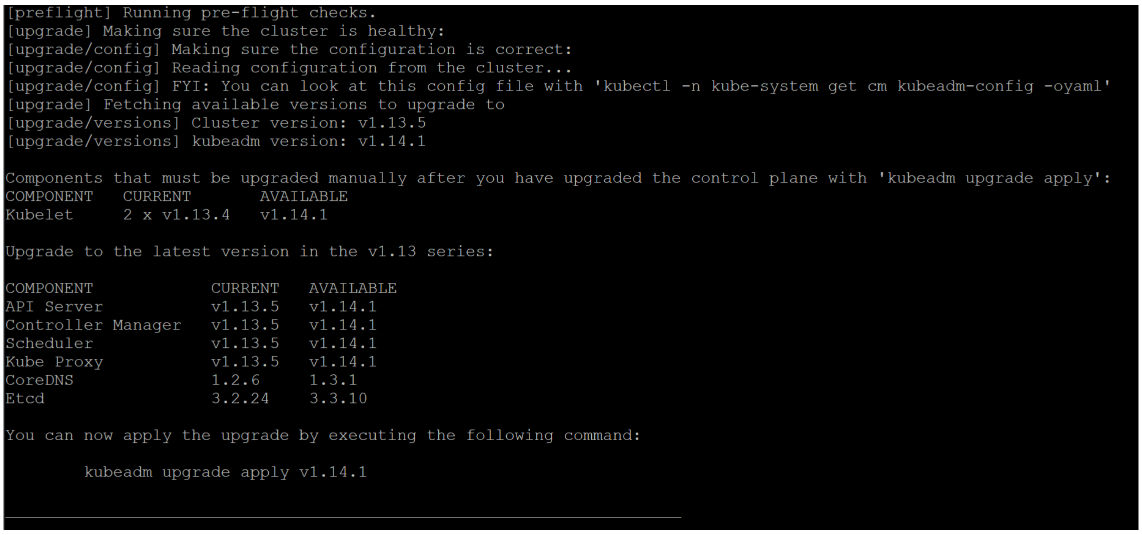
1. Upgrading the Kubernetes control plane with kubeadm (Kubernetes components and add-ons excluding the CNI)
2. Manually upgrading the CNI network plugin, if applicable (For this Lab, the installed version 3.1 of Calico is already the appropriate one for Kubernetes version 1.11.1)
3. Upgrading the Kubernetes packages (kubelet, kubeadm, kubectl) on the master and worker nodes
4. Upgrading the kubelet config on worker nodes with kubeadm

1. Download version 1.14.1 of kubeadm:

# Update the kubeadm binary with version 1.14.1
sudo curl -sSL https://dl.k8s.io/release/v1.14.1/bin/linux/amd64/kubeadm -o /usr/bin/kubeadm

2. Generate an upgrade plan for upgrading Kubernetes to version 1.14.1:

sudo kubeadm upgrade plan v1.14.1



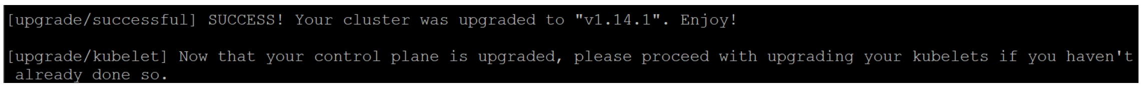
```
[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[upgrade] Fetching available versions to upgrade to
[upgrade/versions] Cluster version: v1.13.5
[upgrade/versions] kubeadm version: v1.14.1

Components that must be upgraded manually after you have upgraded the control plane with 'kubeadm upgrade apply':
COMPONENT   CURRENT       AVAILABLE
Kubelet     2 x v1.13.4   v1.14.1

Upgrade to the latest version in the v1.13 series:

COMPONENT            CURRENT   AVAILABLE
API Server           v1.13.5   v1.14.1
Controller Manager   v1.13.5   v1.14.1
Scheduler            v1.13.5   v1.14.1
Kube Proxy           v1.13.5   v1.14.1
CoreDNS              1.2.6     1.3.1
Etcd                 3.2.24    3.3.10

You can now apply the upgrade by executing the following command:

        kubeadm upgrade apply v1.14.1

_____________________________________________________________________
```

As the output explains, several checks are performed, and the requirements for upgrading the cluster are first verified. A reminder that you need to manually upgrade the kubelet on each node in the cluster is then displayed. Future versions may remove this manual step. Finally, a summary of the planned version changes for all the cluster components (**COMPONENT**) is presented.

3. Apply the upgrade plan by issuing the following command and entering y when prompted:

sudo kubeadm upgrade apply v1.14.1

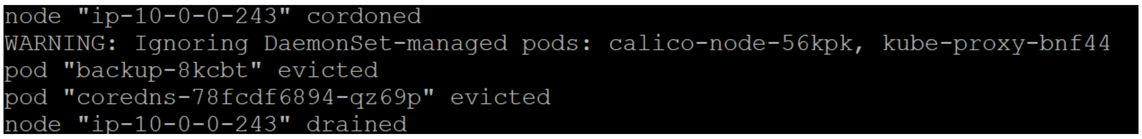
kubeadm begins upgrading the cluster components on the master node. Read through the output to understand what steps are being performed. It takes approximately four minutes to complete. You will see the following success message to know everything went as expected:



```
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.14.1". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't
 already done so.
```

4. Prepare to upgrade the master node's kubelet by draining the node:

kubectl drain $HOSTNAME --ignore-daemonsets



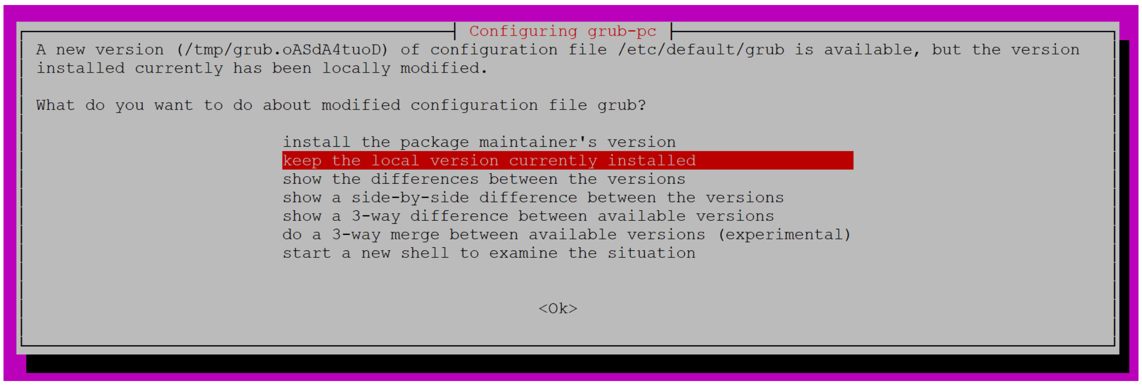
```
node "ip-10-0-0-243" cordoned
WARNING: Ignoring DaemonSet-managed pods: calico-node-56kpk, kube-proxy-bnf44
pod "backup-8kcbt" evicted
pod "coredns-78fcdf6894-qz69p" evicted
node "ip-10-0-0-243" drained
```

5. Upgrade the kubelet, kubeadm, and kubectl apt packages:

sudo apt-get update
*sudo apt-get upgrade -y --allow-change-held-packages *
kubelet=1.14.1-00 kubeadm=1.14.1-00 kubectl=1.14.1-00 kubernetes-cni=0.7.5-00

The upgrade may take a few minutes to complete.

Note: If you see a Configuring grub-pc menu, select Keep the local version currently installed:



```
                          Configuring grub-pc
A new version (/tmp/grub.oASdA4tuoD) of configuration file /etc/default/grub is available, but the version
installed currently has been locally modified.

What do you want to do about modified configuration file grub?

                         install the package maintainer's version
                         keep the local version currently installed
                         show the differences between the versions
                         show a side-by-side difference between the versions
                         show a 3-way difference between available versions
                         do a 3-way merge between available versions (experimental)
                         start a new shell to examine the situation


                                            <Ok>
```

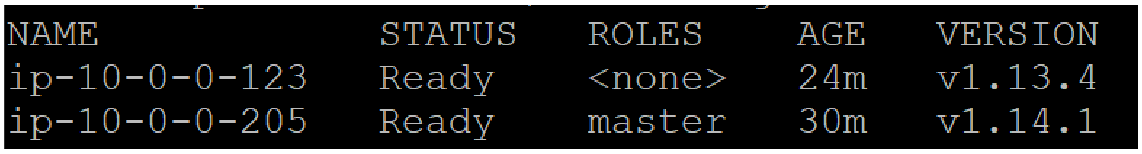
Press *space* to select the first device, followed by *enter* to press the <**Ok**> button:

6. Uncordon the master to allow pods to be scheduled on it now that is has been upgraded:

```
kubectl uncordon $HOSTNAME
```

7. Get the node information to confirm that the version of the master is 1.14.1:

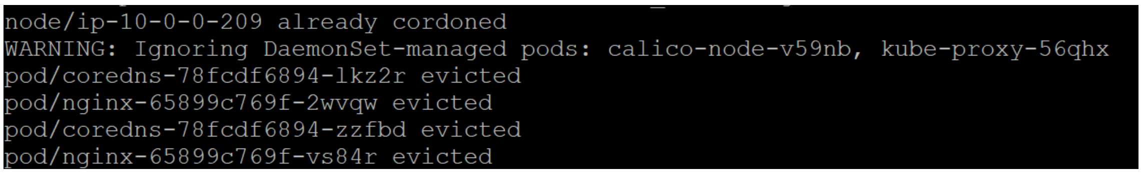
```
kubectl get nodes
```



```
NAME              STATUS    ROLES     AGE    VERSION
ip-10-0-0-123     Ready     <none>    24m    v1.13.4
ip-10-0-0-205     Ready     master    30m    v1.14.1
```

8. Drain the worker node to prepare it for upgrading:

```
# Get the worker's name
worker_name=$(kubectl get nodes | grep \<none\> | cut -d' ' -f1)
# Drain the worker node
kubectl drain $worker_name --ignore-daemonsets
```



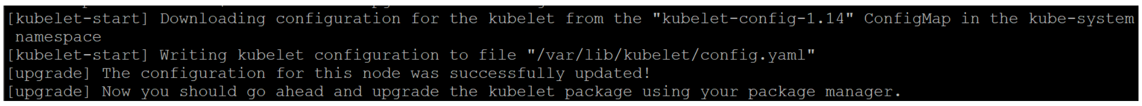
```
node/ip-10-0-0-209 already cordoned
WARNING: Ignoring DaemonSet-managed pods: calico-node-v59nb, kube-proxy-56qhx
pod/coredns-78fcdf6894-lkz2r evicted
pod/nginx-65899c769f-2wvqw evicted
pod/coredns-78fcdf6894-zzfbd evicted
pod/nginx-65899c769f-vs84r evicted
```

9. In the SSH shell connected to the worker node, drain the node and upgrade the Kubernetes packages:

```
sudo apt-get update
sudo apt-get upgrade -y --allow-change-held-packages \
    kubelet=1.14.1-00 kubeadm=1.14.1-00 kubectl=1.14.1-00 kubernetes-cni=0.7.5-00
```

10. Upgrade the worker node's kubelet config using kubeadm:

```
sudo kubeadm upgrade node config --kubelet-version v1.14.1
```



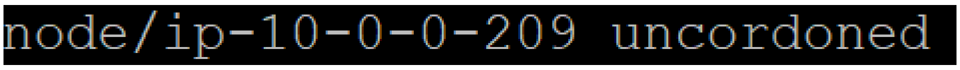
```
[kubelet-start] Downloading configuration for the kubelet from the "kubelet-config-1.14" ConfigMap in the kube-system
 namespace
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[upgrade] The configuration for this node was successfully updated!
[upgrade] Now you should go ahead and upgrade the kubelet package using your package manager.
```

11. Restart the worker node's kubelet:

sudo systemctl restart kubelet

12. Return to the master's SSH shell and uncordon the worker node:

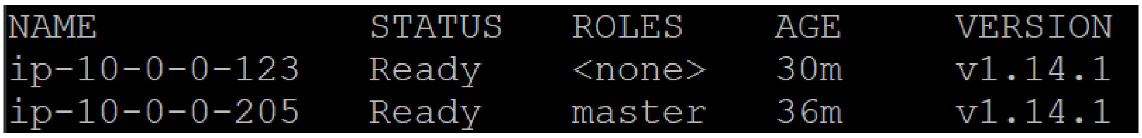
kubectl uncordon $worker_name



```
node/ip-10-0-0-209 uncordoned
```

13. Confirm the worker node is ready and running version 1.14.1:

kubectl get nodes



```
NAME             STATUS   ROLES    AGE    VERSION
ip-10-0-0-123    Ready    <none>   30m    v1.14.1
ip-10-0-0-205    Ready    master   36m    v1.14.1
```