**LAB 1**

**Nombres y Apellidos:** Jhordan Manuel Escobar Soto

**Procedure:**

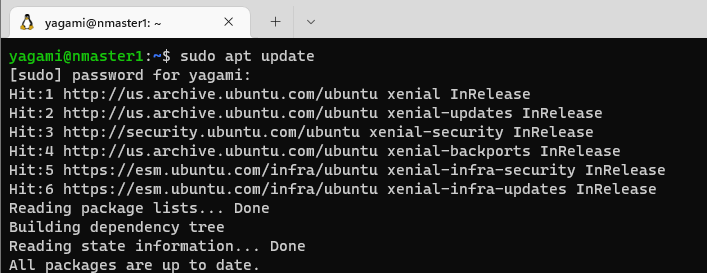
kubeadm is a tool that allows you to easily create Kubernetes clusters that adhere to best practices. It can also perform a variety of cluster lifecycle functions, such as upgrading and downgrading the version of Kubernetes on nodes in the cluster. You will use kubeadm to create a Kubernetes cluster 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**

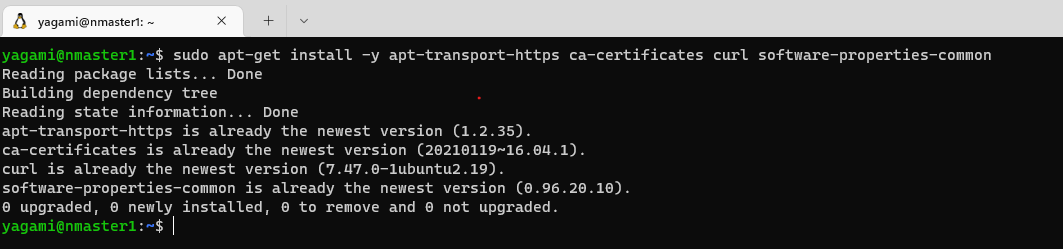
1. Enter the following command to update the system's apt package manager index and update packages required to install Docker:

# Update the package index

**sudo apt-get update**



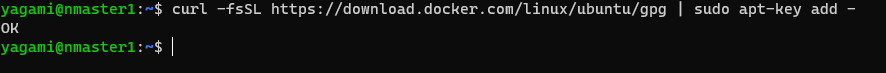
# Update packages required for HTTPS package repository access

**sudo apt-get install -y apt-transport-https ca-certificates curl software-properties-common**

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

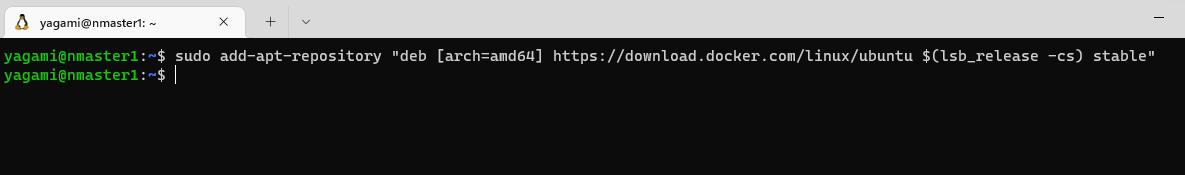
# Add Docker’s GPG key

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

**

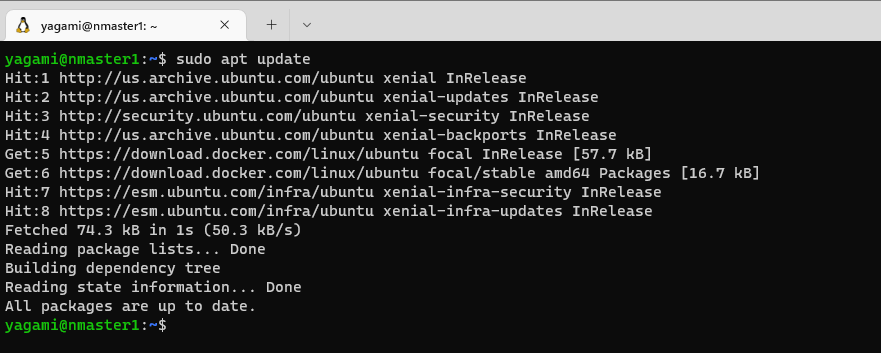
# Configure the stable Docker release repository

**sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"**



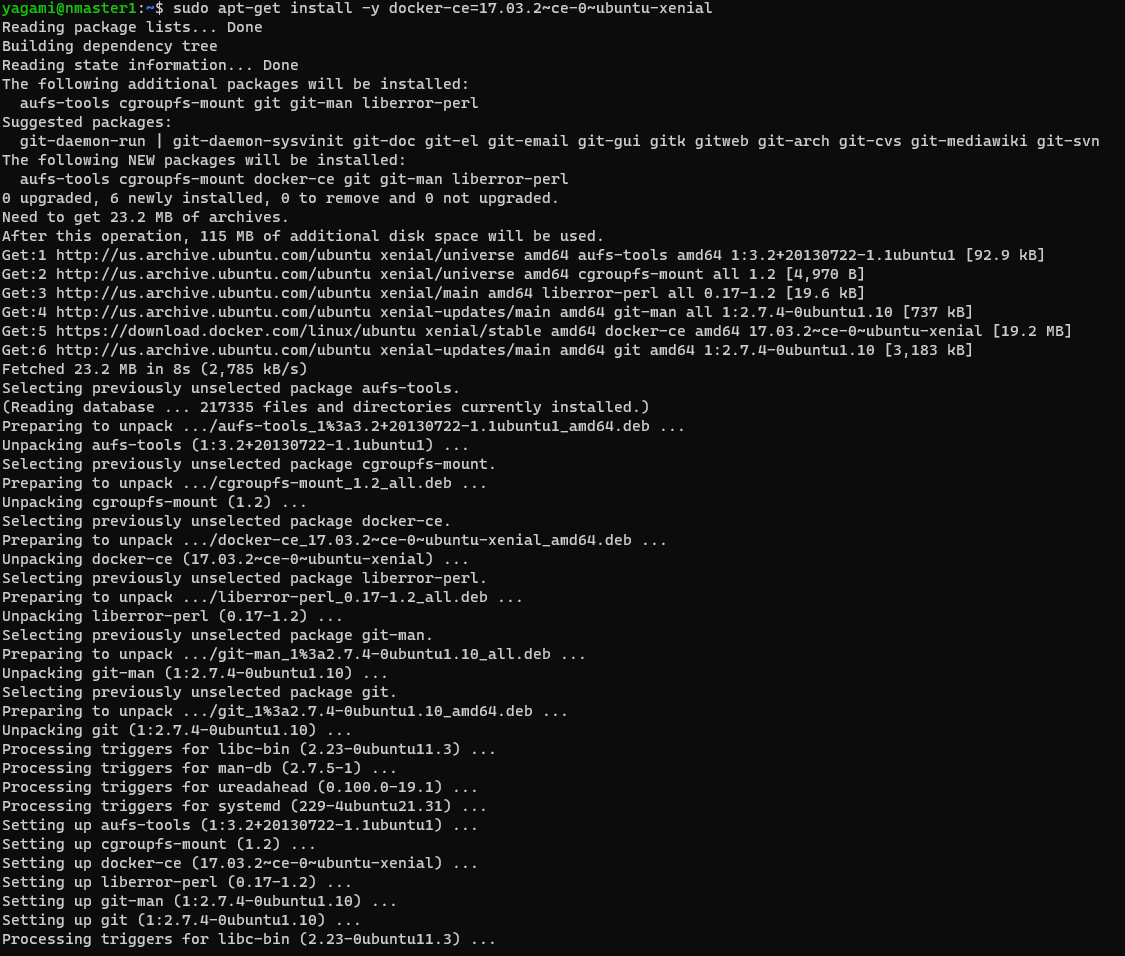
# Update the package index to include the stable Docker repository

**sudo apt update**

**

# Install Docker

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

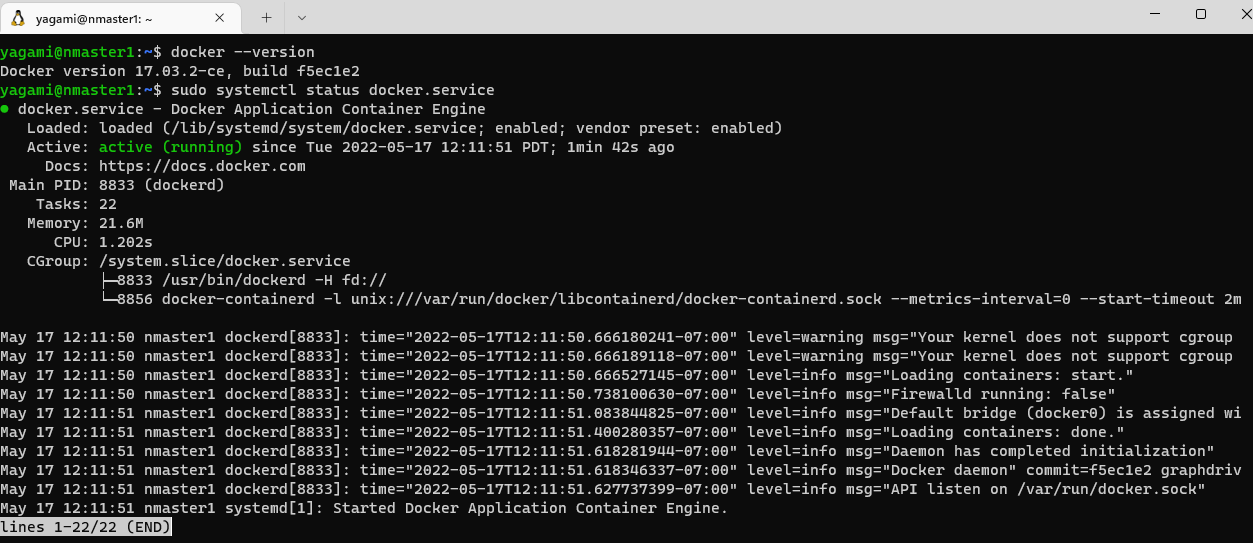


The installation instructions are based on Docker's official instructions. Although it is possible to install Ubuntu's docker.io package, to control the version and use a version that is officially supported by Kubernetes, Docker community edition 17.03 is installed directly from Docker's package repository.

1. Confirm Docker 17.03 is installed:

**docker -version**

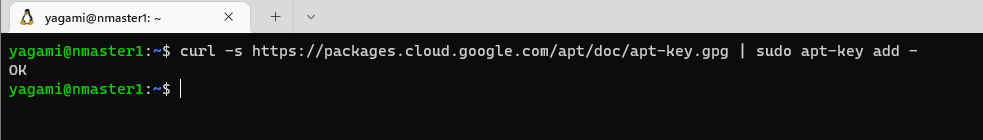
**systemctl status docker.service**

**

1. 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 –**



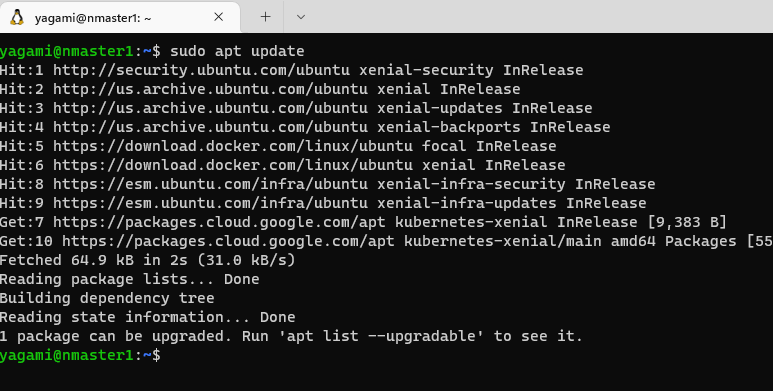
# Add the Kubernetes release repository

**sudo add-apt-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"**

****

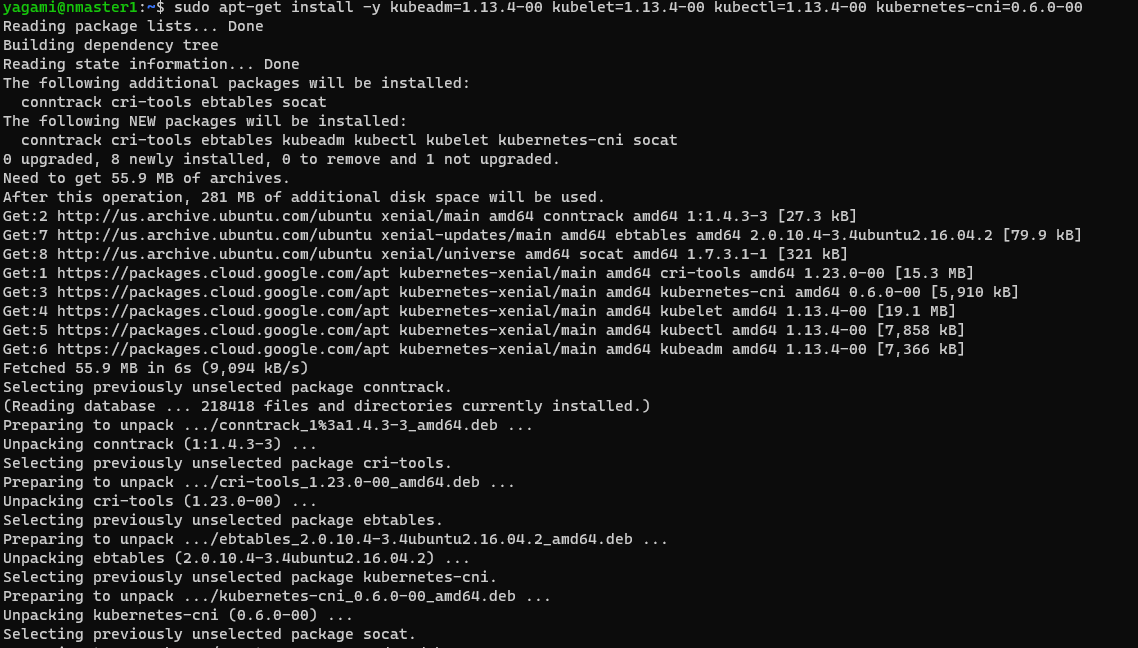
# Update the package index to include the Kubernetes repository

**sudo apt-get update**

**

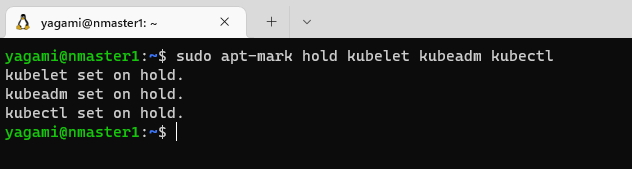
# Install the packages

**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**

**

1. Display the help page for kubeadm:

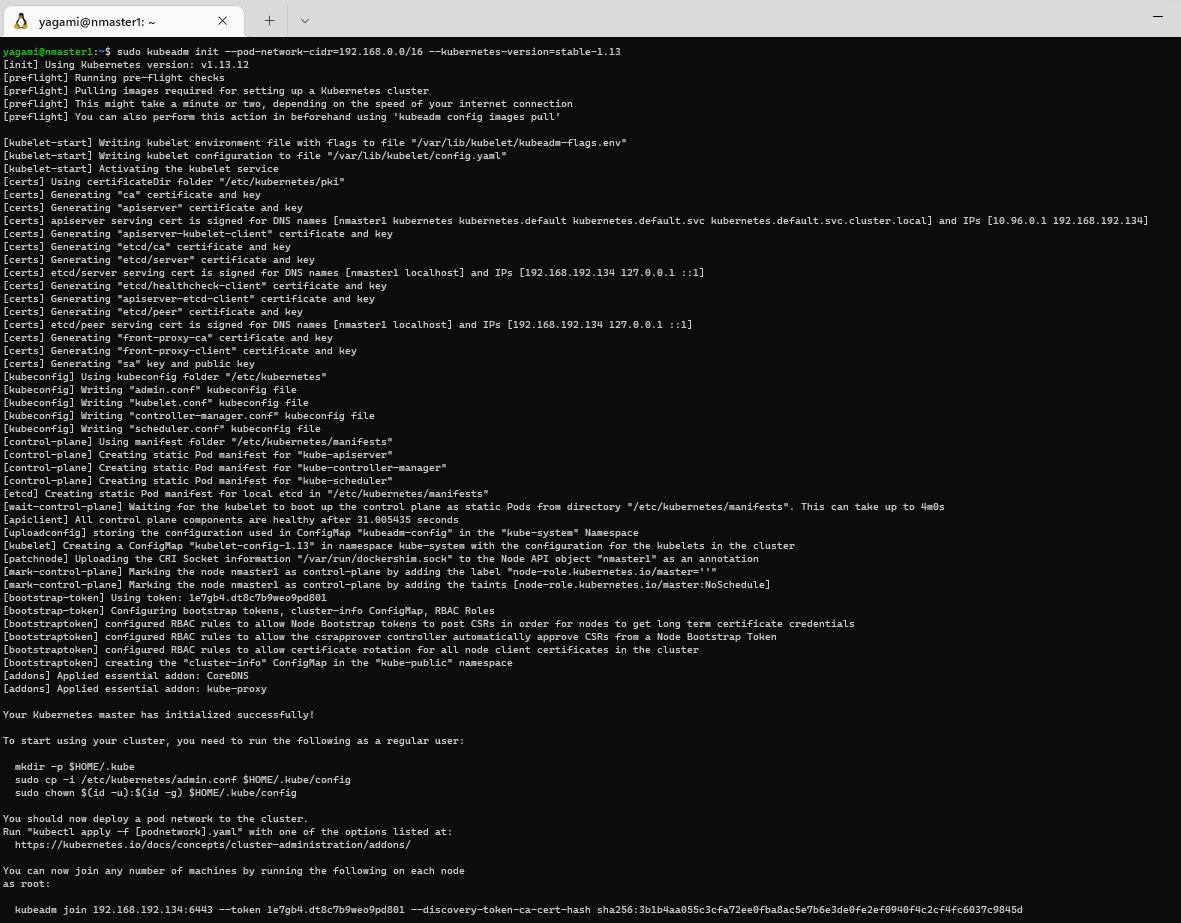
**kubeadm**

**Inicialización del nodo maestro de Kubernetes**

En este Paso del Laboratorio se utilizará kubeadm para inicializar el nodo maestro. El proceso de inicialización creará una autoridad de certificación para la comunicación segura del clúster y la autenticación, e iniciará todos los componentes del nodo (kubelet), los componentes maestros (servidor API, gestor de controladores, planificador, etc.) y los complementos comunes (kube-proxy, DNS). Verás lo fácil que es el proceso de inicialización con kubeadm.

La inicialización utiliza valores sensibles por defecto que se adhieren a las mejores prácticas. Sin embargo, hay muchas opciones de comandos disponibles para configurar el proceso, incluyendo si quieres proporcionar tu propia autoridad de certificados o si quieres usar un almacén de valores clave etcd externo. Una de las opciones que utilizará es la requerida por el plugin de red del pod que instalará después de que el maestro sea inicializado. kubeadm no instala un plugin de red por defecto para usted. Usted utilizará Weave como el plugin de red del pod. Weave soporta las políticas de red de Kubernetes. Para que las políticas de red funcionen correctamente, debe utilizar la opción --pod-network-cidr para especificar un rango de direcciones IP para la red del pod al inicializar el nodo maestro con kubeadm.

1. Inicialice el nodo maestro utilizando el comando init:

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

El bloque CIDR de la red del pod (192.168.0.0/16) es el que utiliza por defecto Weave.

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

Example:

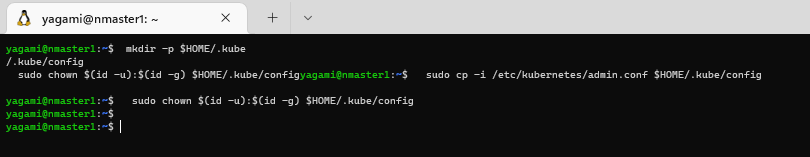
**kubeadm join 10.0.0.77:6443 --token 0e72cy.zhvb54ol7cggl6od --discovery-token-ca-cert-hash sha256:3a79dd47e12755f207dff1c4a36b406fd138642ee3a04c3f7d5e6c1ab33c8ad4**

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

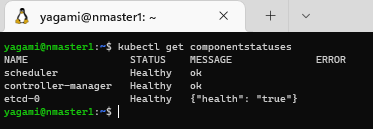
**mkdir -p $HOME/.kube**

**sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config**

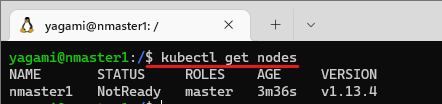
**sudo chown $(id -u):$(id -g) $HOME/.kube/config**

**

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

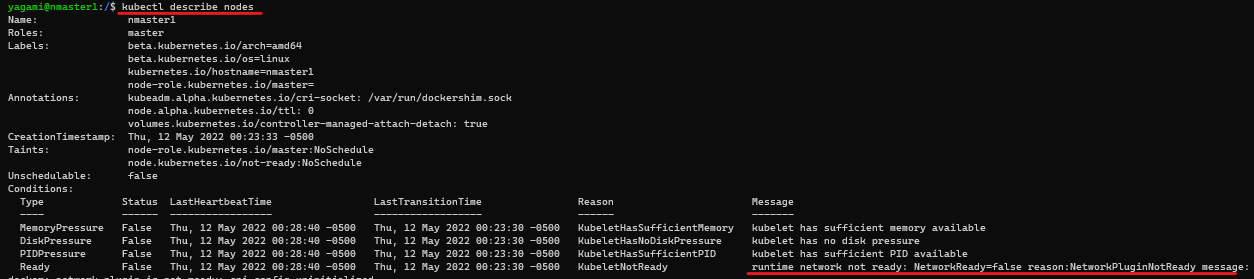
****kubectl get componentstatuses**

1. 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.

1. Describe the node to probe deeper into its NotReady status:

****kubectl describe nodes**

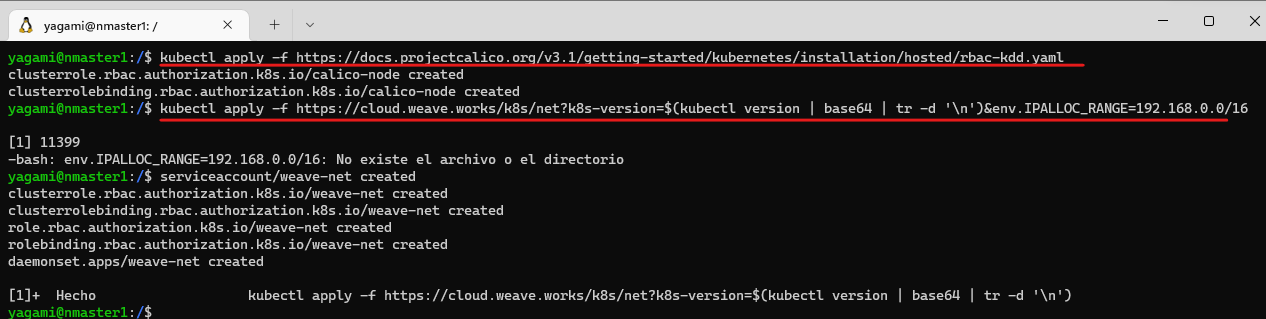
In the **Conditions** section of the output, observe the **Ready** condition is False, and read the **Message**:

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.

1. 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**

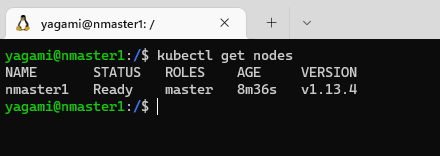
**kubectl apply -f https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')&env.IPALLOC\_RANGE=192.168.0.0/16**



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.

1. Check the status of the nodes in the cluster:

**kubectl get nodes**

**

**Joining a Worker Node to the Kubernetes Cluster**

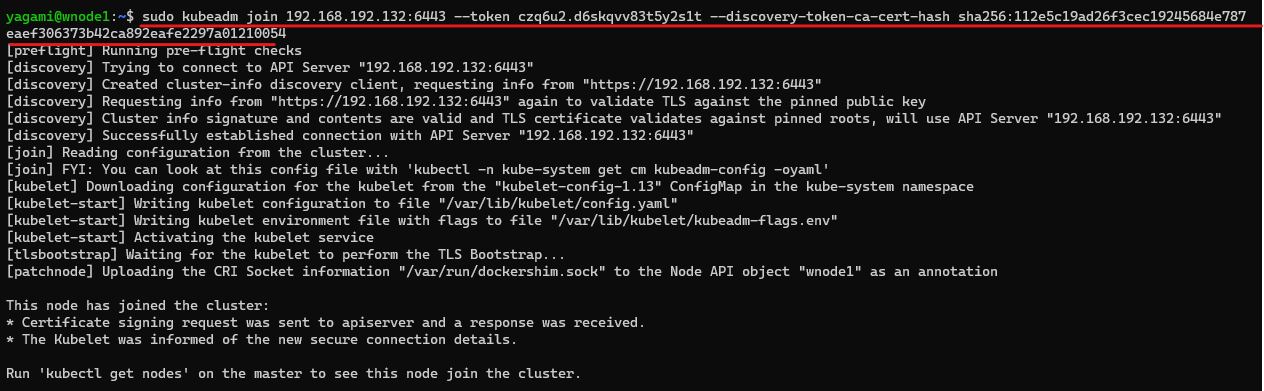
*The process of adding a worker node with kubeadm is even simpler than initializing a master node. You will join a worker node to the cluster using the command that kubeadm init provided in this Lab Step.*

**WORKER NODE:**

1. Enter sudo followed by the kubeadm join command that you stored from the output of kubeadm init. It resembles:

*Example:*

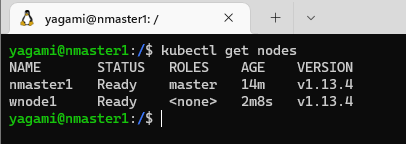
**kubeadm join 192.168.192.132:6443 --token czq6u2.d6skqvv83t5y2s1t --discovery-token-ca-cert-hash sha256:112e5c19ad26f3cec19245684e787eaef306373b42ca892eafe2297a01210054**

**

***MASTER NODE:***

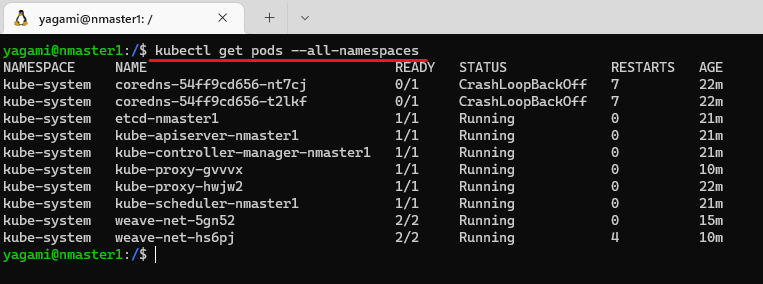
1. Confirm that all the pods in the cluster are running:

**kubectl get nodes**

**

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

**kubectl get pods --all-namespaces**

**

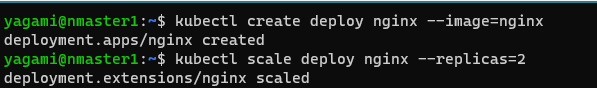
**Backing Up and Restoring Kubernetes Clusters**

*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

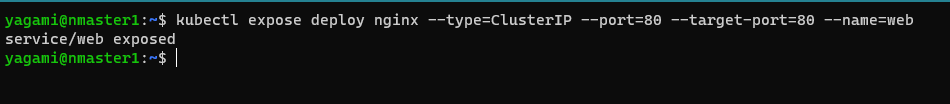
**kubectl create deployment nginx --image=nginx**

**kubectl scale deployment nginx --replicas=2**

**

**

1. Expose the deployment using a ClusterIP service:

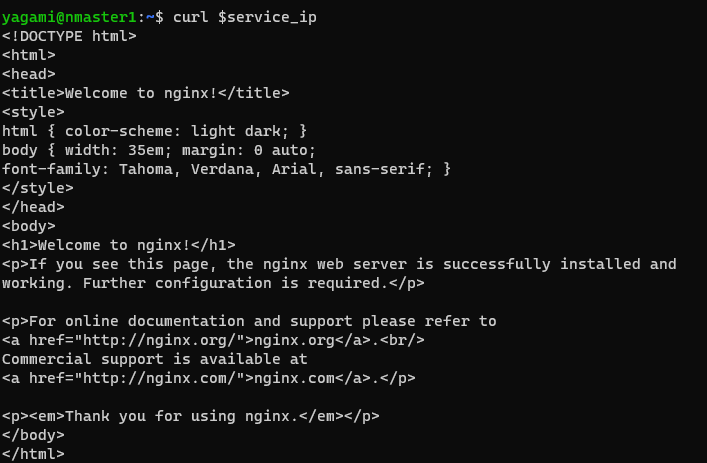
****kubectl expose deployment nginx --type=ClusterIP --port=80 --target-port=80 --name=web**

1. Send a HTTP request to the web service:

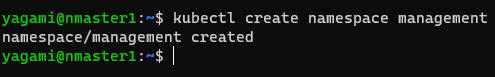
# Get the Cluster IP of the service

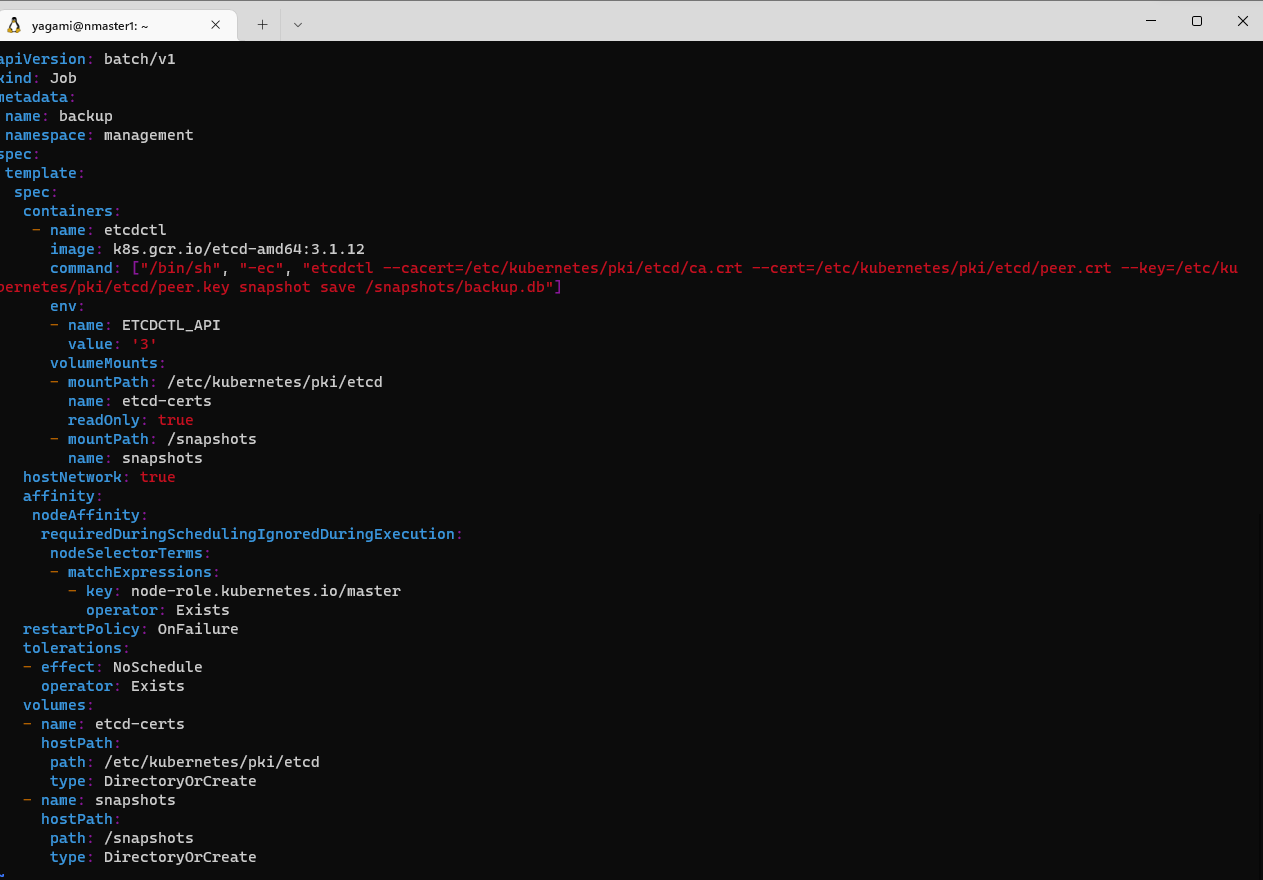
**service\_ip=$(kubectl get service web -o jsonpath='{.spec.clusterIP}')**

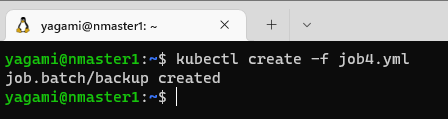
**# Use curl to send an HTTP request to the service curl $service\_ip

**

1. Create a management namespace:

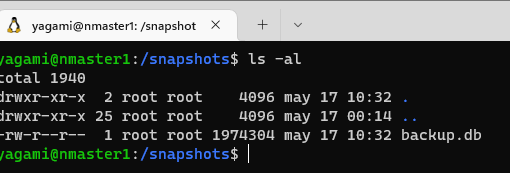
**kubectl create namespace management**

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

**

1. List the contents of the /snapshots directory:

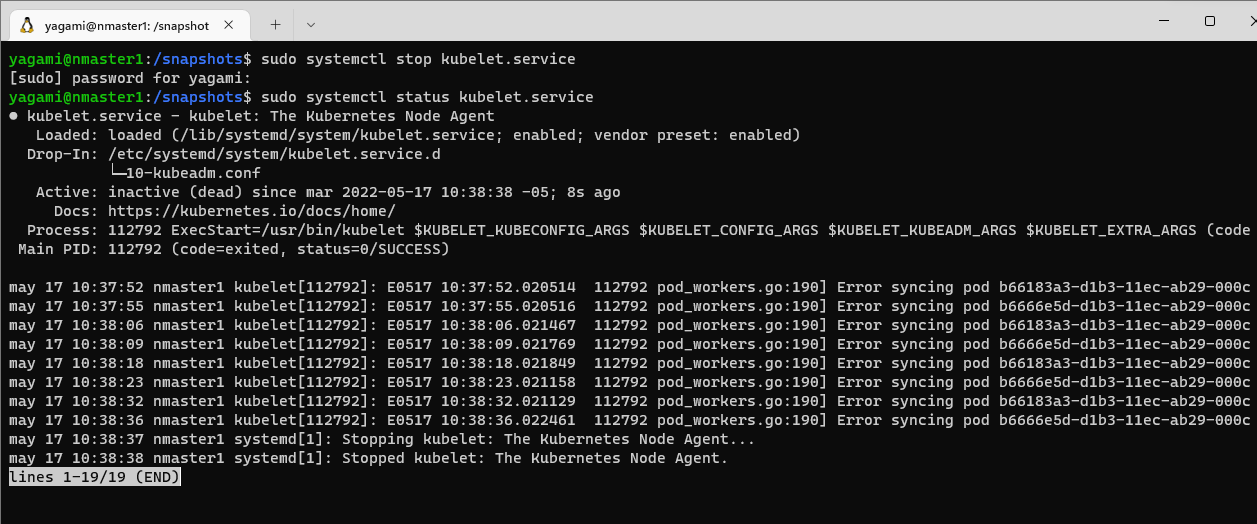
**ls -al /snapshots**



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.

1. Stop the master's kubelet:

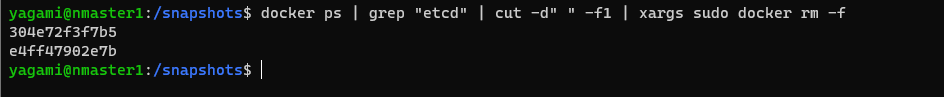
**sudo systemctl stop kubelet.service**



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.

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

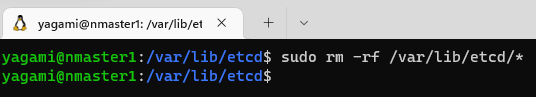
**sudo docker ps | grep etcd | cut -d” ” -f1 | xargs sudo docker rm -f**

****

The container IDs of the deleted etcd containers are displayed.

1. 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.

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

**sudo docker run --rm \**

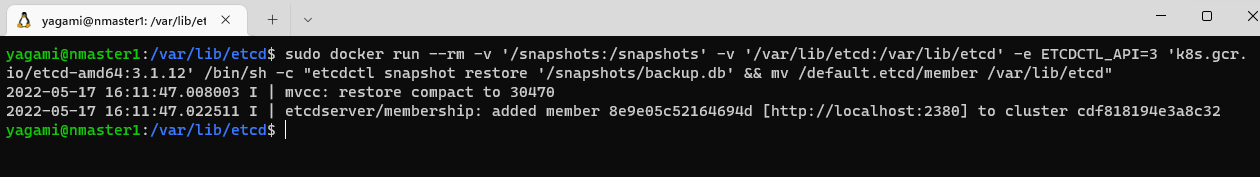
**-v '/snapshots:/snapshots' \**

**-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"**

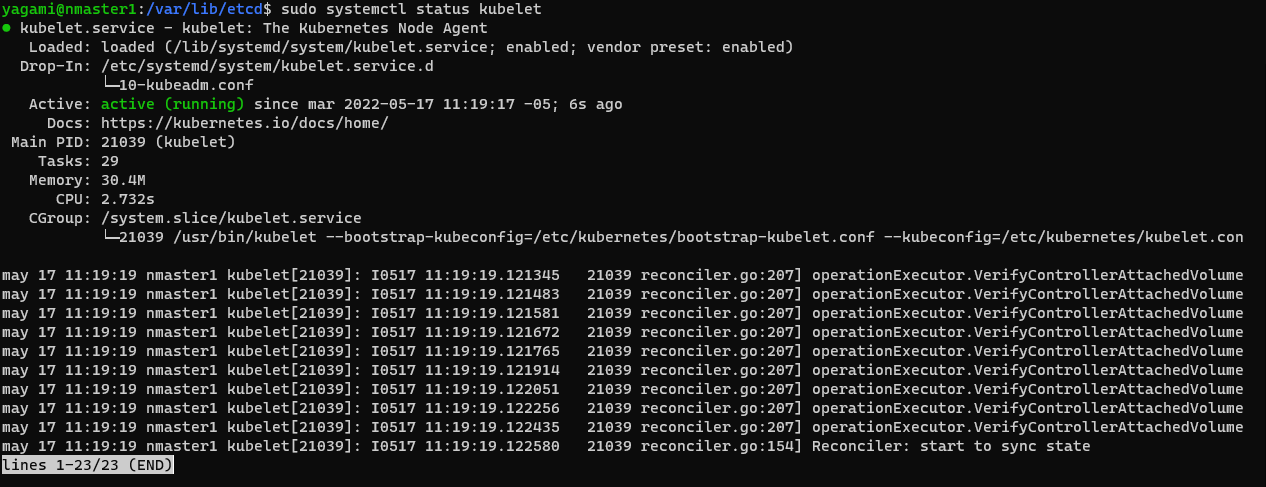
**

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.

1. Start the kubelet:

**sudo systemctl start kubelet**

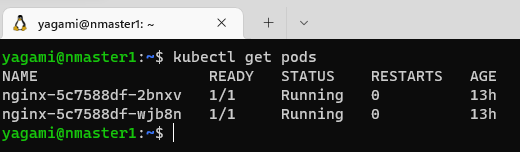
**sudo systemctl status kubelet**



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.

1. Confirm the Nginx pods are running:

**kubectl get pods**



1. Confirm the web service works:

**service\_ip=$(kubectl get service web -o jsonpath='{.spec.clusterIP}')**

**curl $service\_ip**



**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

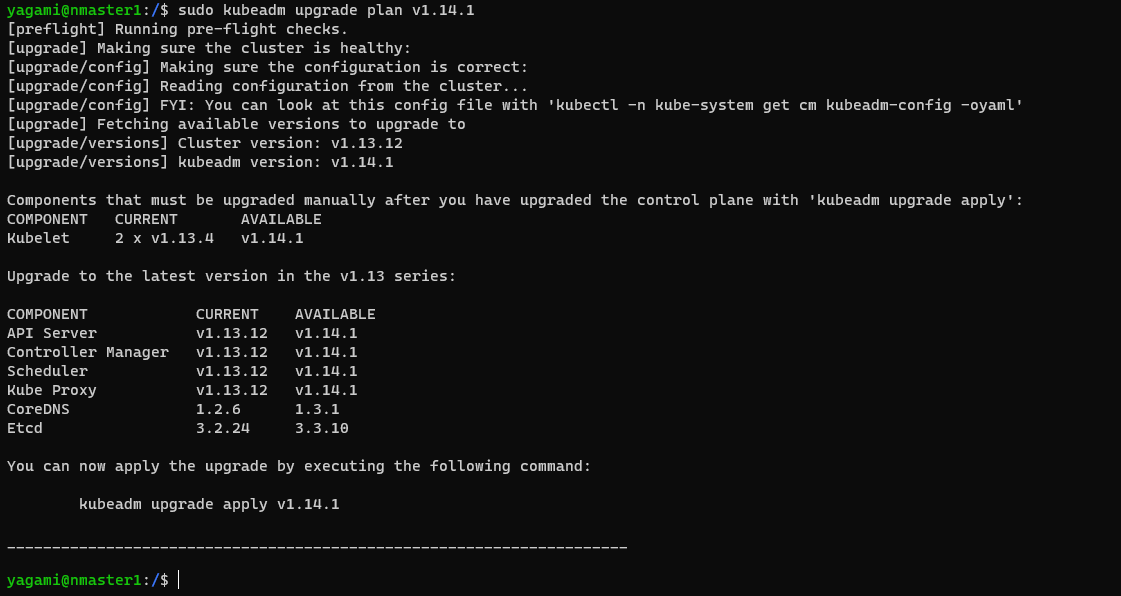
4. Upgrading the kubelet config on worker nodes with kubeadm

1. Download version 1.14.1 of kubeadm:

# Update the kubeadm binary with version 1.14.1

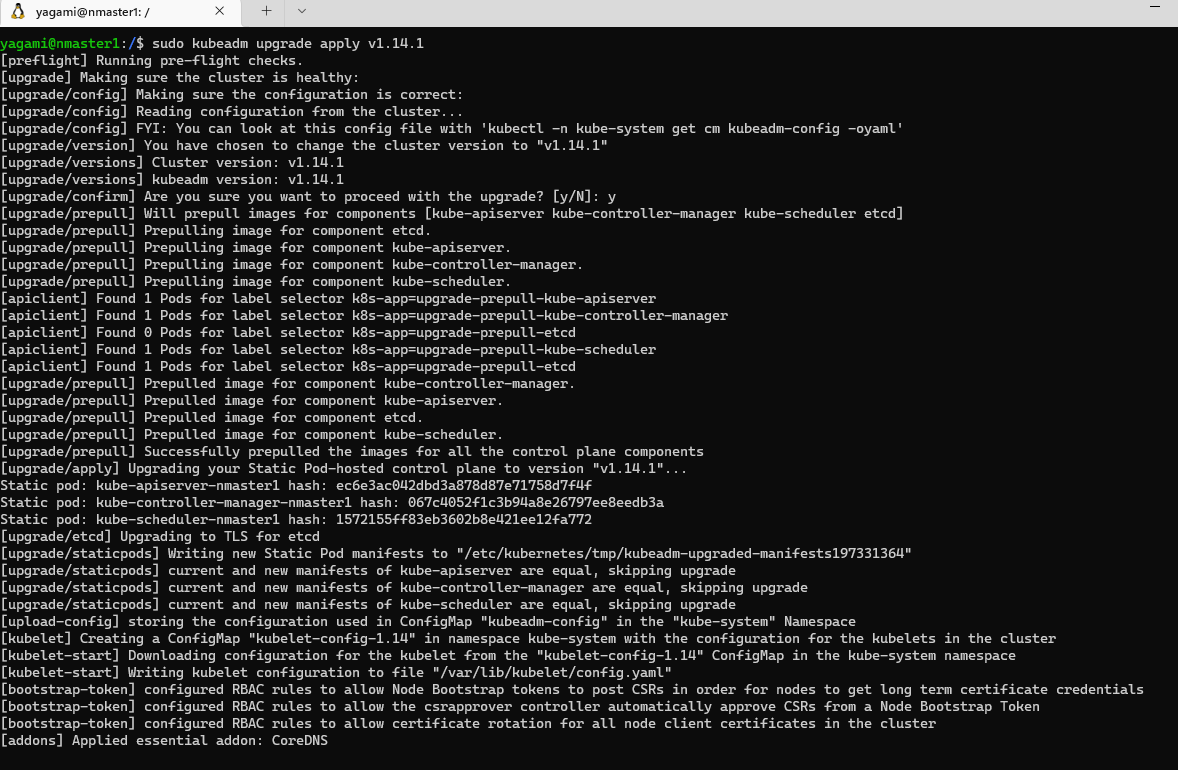
**sudo curl -sSL https://dl.k8s.io/release/v1.14.1/bin/linux/amd64/kubeadm -o /usr/bin/kubeadm**

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

**sudo kubeadm upgrade plan 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.

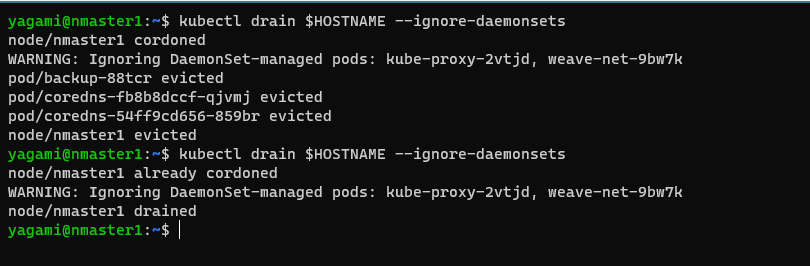
1. Apply the upgrade plan by issuing the following command and entering y when prompted:

**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:

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

**kubectl drain $HOSTNAME --ignore-daemonsets**

****

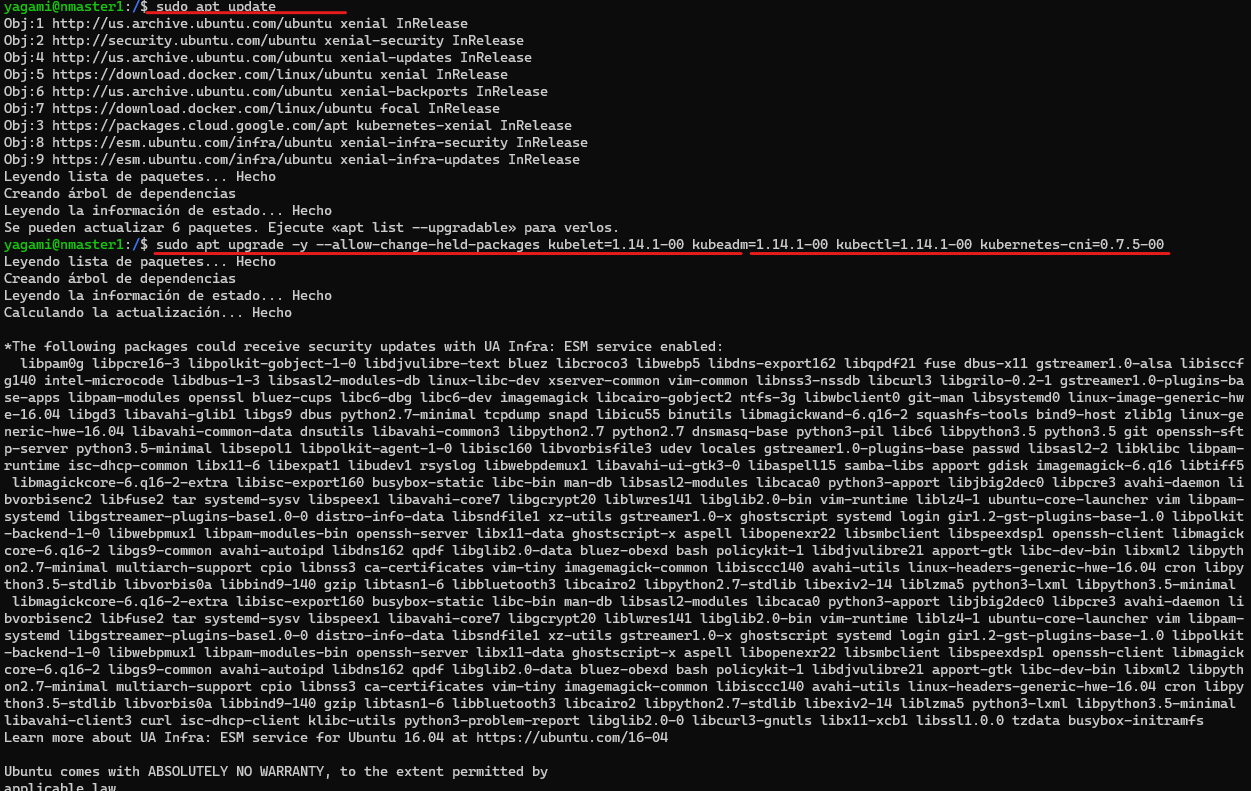
1. Upgrade the kubelet, kubeadm, and kubectl apt packages:

**sudo apt-get update**

**sudo apt-get upgrade -y --allow-change-held-packages \**

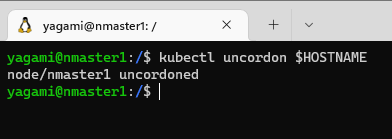
**kubelet=1.14.1-00 kubeadm=1.14.1-00 kubectl=1.14.1-00 kubernetes-cni=0.7.5-00**

The upgrade may take a few minutes to complete.

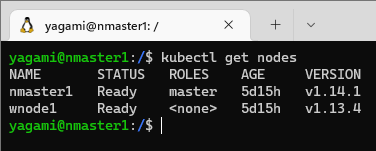


1. Uncordon the master to allow pods to be scheduled on it now that is has been upgraded:

**kubectl uncordon $HOSTNAME**



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

**kubectl get nodes**

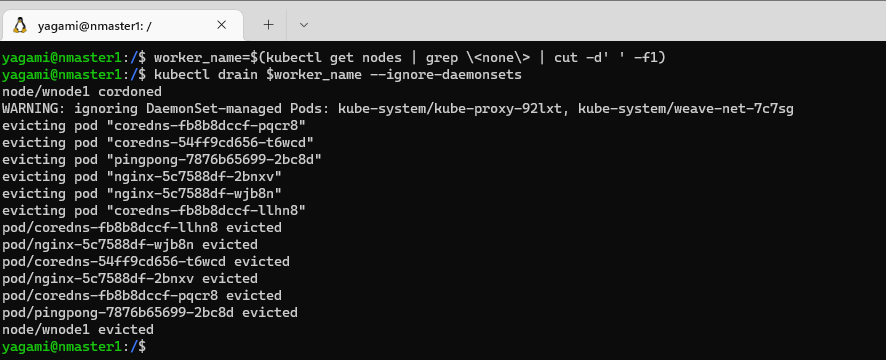
1. Drain the worker node to prepare it for upgrading:

# 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**

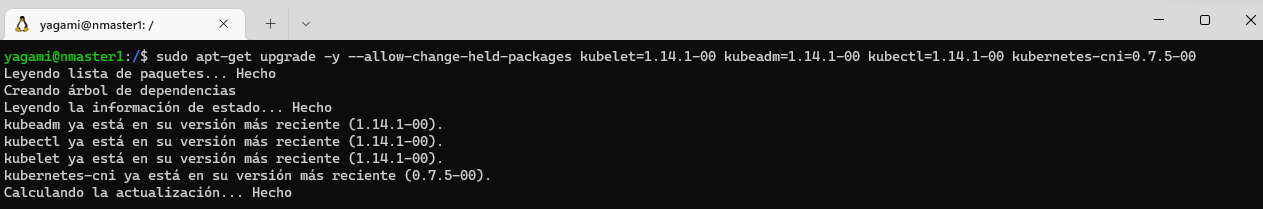


1. In the SSH shell connected to the worker node, drain the node and upgrade the Kubernetes packages:

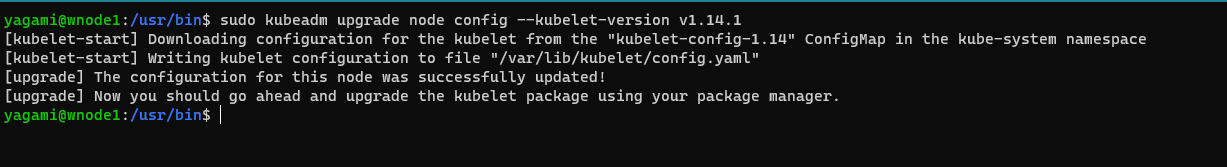
**sudo apt-get update**

**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**

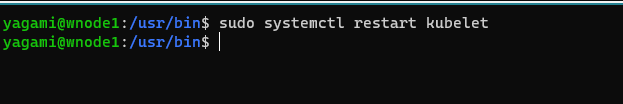


1. Upgrade the worker node's kubelet config using kubeadm:

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

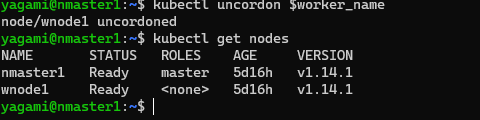
1. Restart the worker node's kubelet:

**sudo systemctl restart kubelet**

****

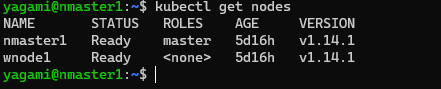
1. Return to the master's SSH shell and uncordon the worker node:

**kubectl uncordon $worker\_name**

****

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

**kubectl get nodes**

****