**How To Setup A Three Node Kubernetes Cluster For CKA: Step By Step**

https://secure.gravatar.com/avatar/972bd62722802ed51a78325d8539de00?s=60&d=mm&r=g

May 10, 2022 by [Nijaguna Darshan](https://k21academy.com/author/nijaguna-darshan/) [12 Comments](https://k21academy.com/docker-kubernetes/three-node-kubernetes-cluster/#comments)

 19943 views

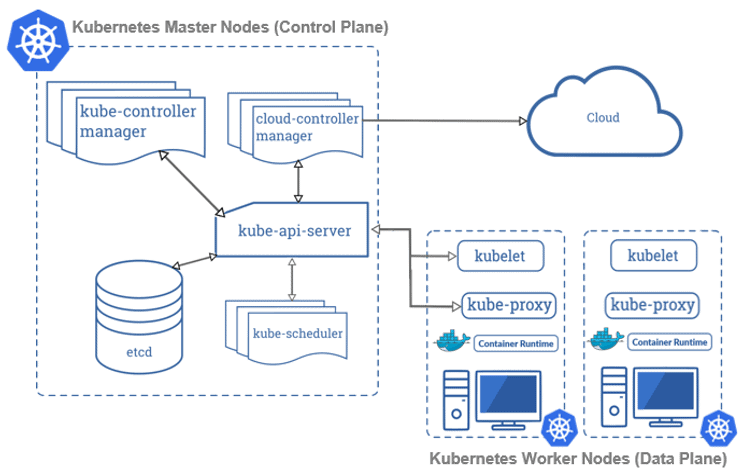
A **Kubernetes Cluster**is a group of node or machines running together. At the highest level of Kubernetes, there exist two kinds of servers, a **Master** and a **Worker node**. These servers can be Virtual Machine(VM) or physical servers(Bare metal). Together, these servers form a Kubernetes cluster and are controlled by the services that make up the Control Plane.

If you are new to Docker & Kubernetes world, then check out our blog on [**Kubernetes**](https://k21academy.com/docker-kubernetes/kubernetes-for-beginners/)for Beginners to get an idea about the components and concepts of Kubernetes.

In this blog, we will cover **How to install and configure a three-node cluster in Kubernetes**which is the first topic in Kubernetes. We have a set of [**Hands-on Labs**](https://k21academy.com/docker-kubernetes/certified-kubernetes-administrator-cka-certification-training-step-by-step-activity-guides-hands-on-lab-exercise/) that you must perform in order to learn **Docker & Kubernetes**and clear the [**CKA certification**](https://k21academy.com/docker-kubernetes/certified-kubernetes-administrator-cka-exam/) exam. **Cluster Architecture, Installation & Configuration** have a total weightage of **25%** in the Exam.



There are **3 ways** to deploy a Kubernetes cluster:  
1. By deploying all the components separately.  
2. Using Kubeadm.  
3. Using Managed Kubernetes Services



In this blog, we will be covering the following topics:

* [Prerequisites](https://k21academy.com/docker-kubernetes/three-node-kubernetes-cluster/#5)
* [Installing Containerd, Kubectl and Kubeadm Packages](https://k21academy.com/docker-kubernetes/three-node-kubernetes-cluster/#1)
* [Create a Kubernetes Cluster](https://k21academy.com/docker-kubernetes/three-node-kubernetes-cluster/#2)
* [Join Worker Nodes to the Kubernetes Cluster](https://k21academy.com/docker-kubernetes/three-node-kubernetes-cluster/#3)
* [Testing the Cluster](https://k21academy.com/docker-kubernetes/three-node-kubernetes-cluster/#4)

**Prerequisites For Cluster Setup**

Deploying three nodes on-premises can be hard and painful, so an alternate way of doing this can be using a Cloud Platform for deploying them. You can use any Cloud Platform, here we are using **Azure Cloud**. Before getting on with creating a cluster make sure you have the following setup ready:

**I)** Create an**Azure Free Account**, as we will use Azure Cloud for setting up a Kubernetes Cluster.

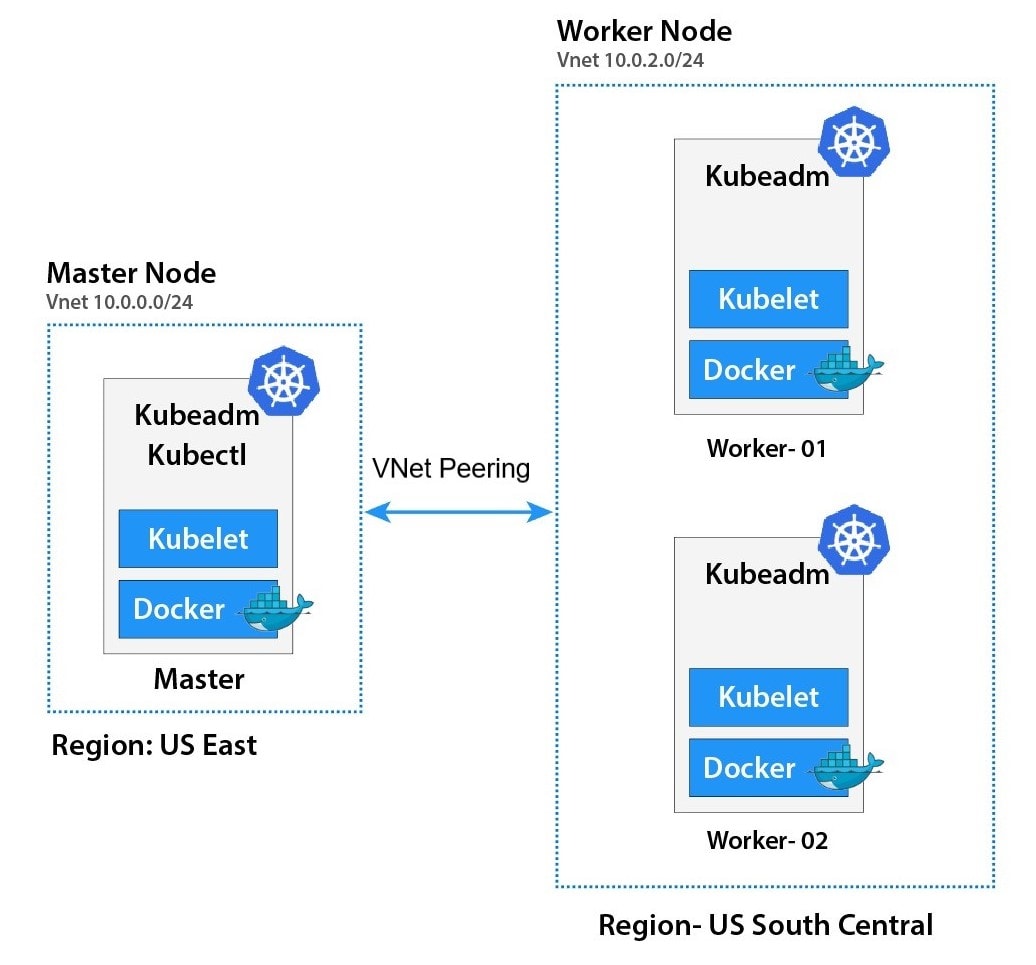
To create an Azure Free Account, check our blog on [**Azure Free Trial Account**](https://k21academy.com/microsoft-azure/create-free-microsoft-azure-trial-account/)**.**

**II)** Launch **3 Virtual Machines** – one Master Node and 2 Worker Nodes. We are launching these VMs in different regions because in the Azure Free tier account we can’t create 3 virtual machines in a single region due to the service limit. So we are creating One Master node in US East Region and Two Worker node (worker-1, worker-2 in US Southcentral Region)

To create an Ubuntu Virtual Machine, check our blog on [**Create An Ubuntu VM In Azure.**](https://k21academy.com/microsoft-azure/az-104/az-104-create-and-connect-an-ubuntu-virtual-machine-in-azure/)

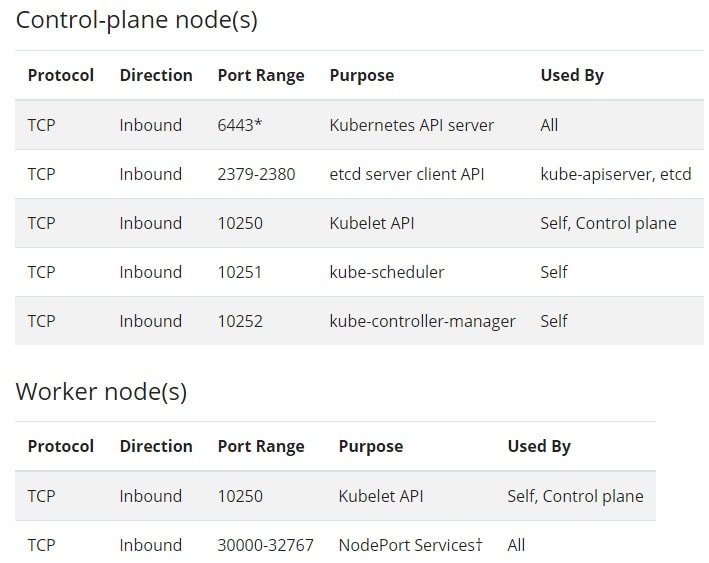
**III)** For connecting the worker node with the master node as they are in different regions and in different Vnet, we have to do **VNet Peering.**

To know more about Virtual Networks, refer to our blog on [**azure vnet peering**](https://k21academy.com/microsoft-azure/az-104/virtual-network-vnet-peering-in-azure/)

****

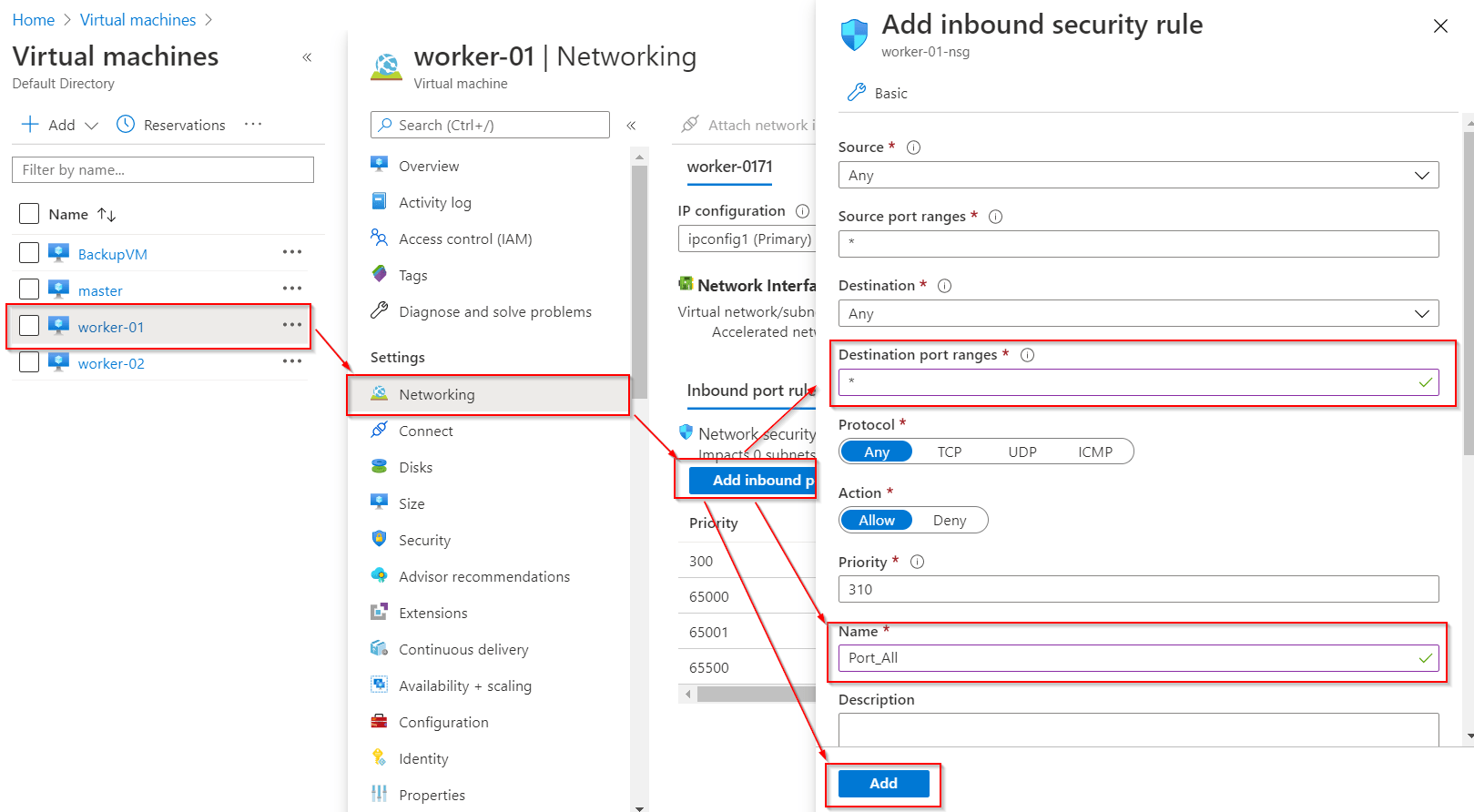
**Also Check:**Our [Kubernetes training](https://k21academy.com/kubernetes/) & understand Kubernetes basics in a better way.

**IV)** The Ports specified below are the default Port range for NodePort Services for Master and Worker Nodes.



Port numbers marked with **\*** are overridable, so we have to make sure that any custom ports we provide are open.

**Note**: As we are creating a cluster only for testing purpose, so we can open all the Ports rather than opening only specified Port.

The **specifications** required for a Node:

* One or more machines running a deb/rpm-compatible Linux OS; for example Ubuntu or CentOS.  
  (**Note**: We are going to use Ubuntu in this setup.)
* 8 GiB or more of RAM per machine.
* At least 4 CPUs on the machine that you use as a control-plane node.

**Also Read:**[**Kubernetes vs docker**](https://k21academy.com/docker-kubernetes/kubernetes-vs-docker/)**,**to know the major difference between them.

**Installing Containerd, Kubectl, And Kubeadm Packages**

After doing the above-mentioned process, we have to install some packages on our machines. These packages are:

* **kubeadm** – a CLI tool that will install and configure the various components of a cluster in a standard way.
* **kubelet** – a system service/program that runs on all nodes and handles node-level operations.
* **kubectl** – a CLI tool used for issuing commands to the cluster through its API Server.

In order to install these packages, follow the steps mentioned below on **Master as well as Worker nodes**:

**Step 1)** We have to do SSH to our virtual machines with the username and password. If you are a Linux or Mac user then use ssh command and if you are a Windows user then you can use Putty.

$ sudo -i

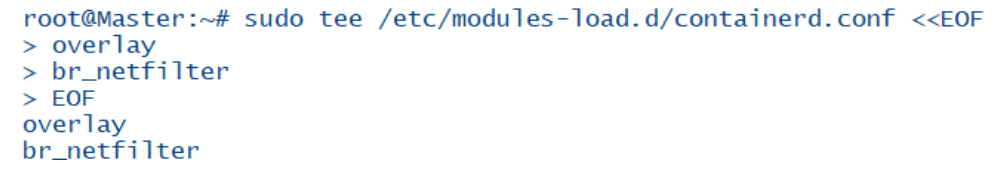
**Step 2)** Configure persistent loading of modules:

$ tee /etc/modules-load.d/containerd.conf <<EOF

overlay

br\_netfilter

EOF



To Install Docker on the local system, you can check out the following blog [**Install Docker**](https://k21academy.com/docker-kubernetes/docker-installation-overview/)

**Step 3)** Load at runtime:

$ modprobe overlay

$ modprobe br\_netfilter

**Step 4)**Update Iptables Settings:

**Note:** To ensure packets are properly processed by IP tables during filtering and port forwarding. Set the net.bridge.bridge-nf-call-iptables to ‘1’ in your sysctl config file.

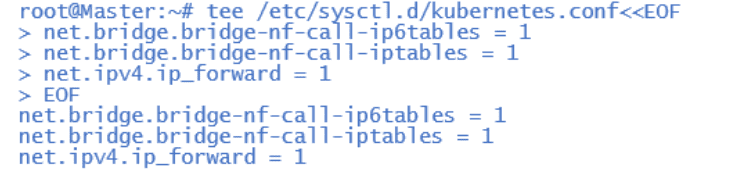
$ tee /etc/sysctl.d/kubernetes.conf<<EOF

net.bridge.bridge-nf-call-ip6tables = 1

net.bridge.bridge-nf-call-iptables = 1

net.ipv4.ip\_forward = 1

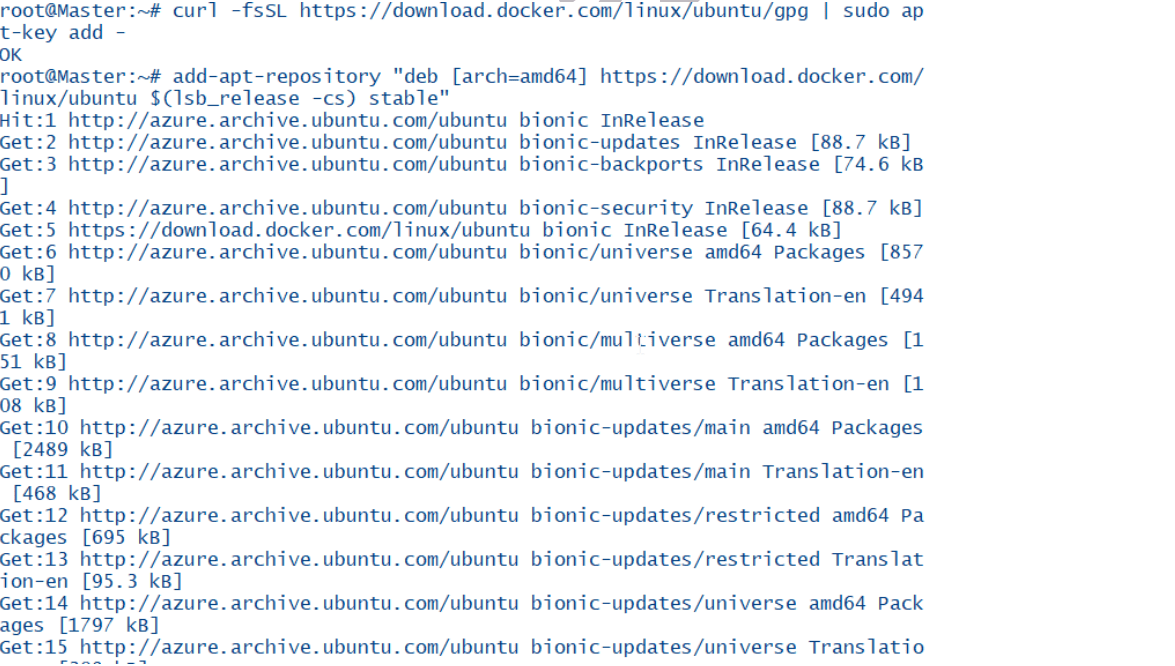
EOF

  
**Step 5) Reload configs:**

$ sysctl –system  
  
**Step 6)**Add Docker repo:

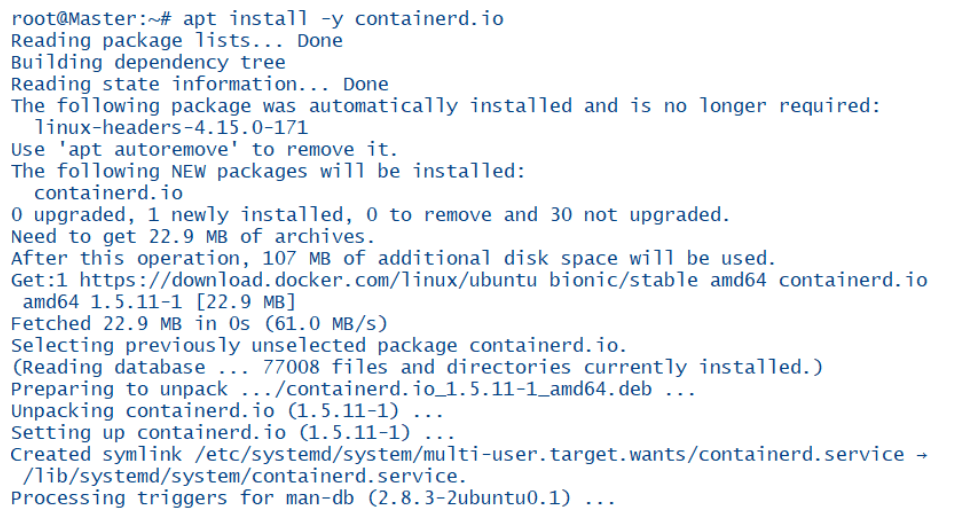
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

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

  
**Step 7)** Install **containerd:**

$ apt update

$ apt install -y containerd.io



**Step 8)**

Configure containerd and start service:

$ mkdir -p /etc/containerd

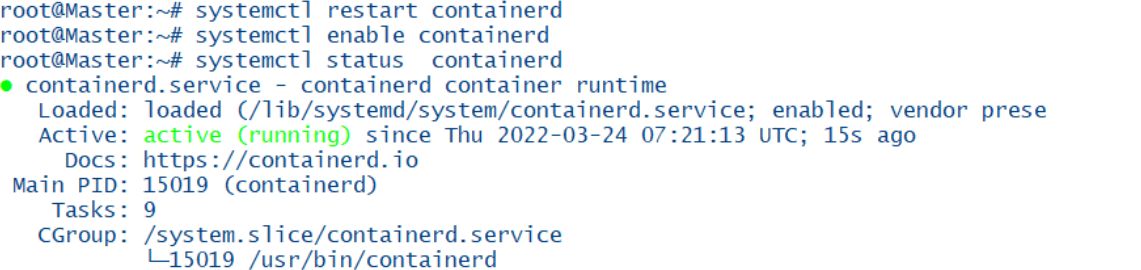
$ containerd config default>/etc/containerd/config.toml

**Step 9)** Restart containerd:

$ systemctl restart containerd

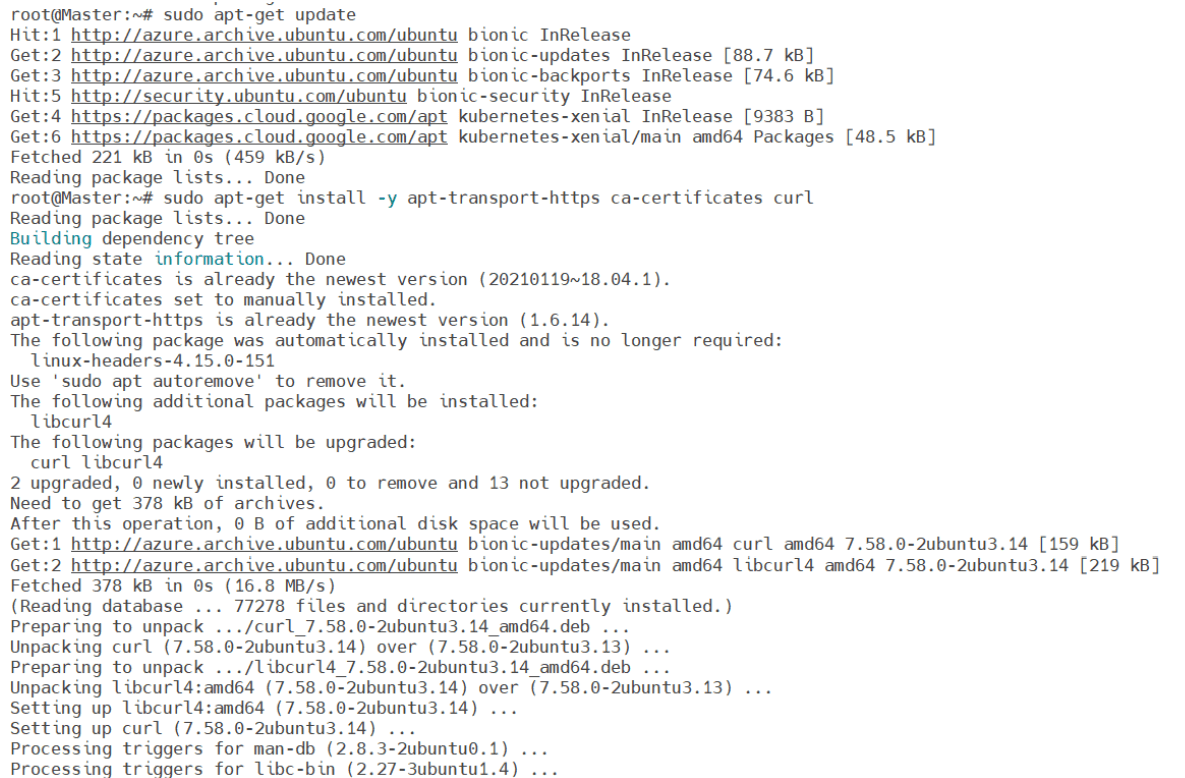
$ systemctl enable containerd

$ systemctl status containerd



**Step 10)** Update the apt package index and install packages needed to use the Kubernetes apt  
repository:

$ apt-get update && apt-get install -y apt-transport-https ca-certificates curl

****

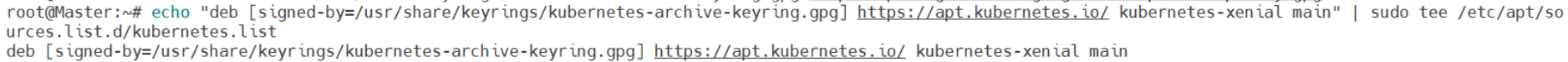
**Also Check:**Our blog post on [**Kubernetes Health Check**](https://k21academy.com/docker-kubernetes/kubernetes-readiness-and-livenessprobe/). Click here

**Step 11)** Download the Google Cloud public signing key:

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

**Step 12)**Add the Kubernetes apt repository:

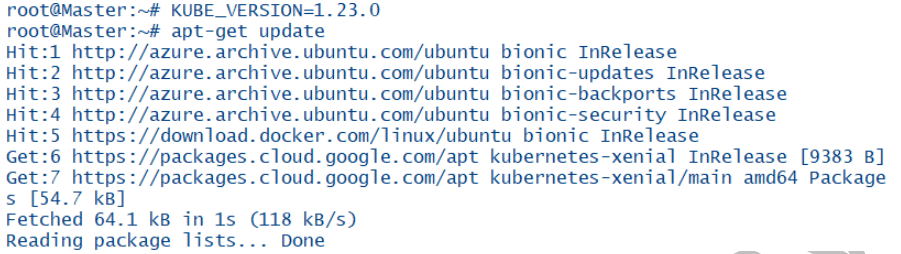
$ echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list



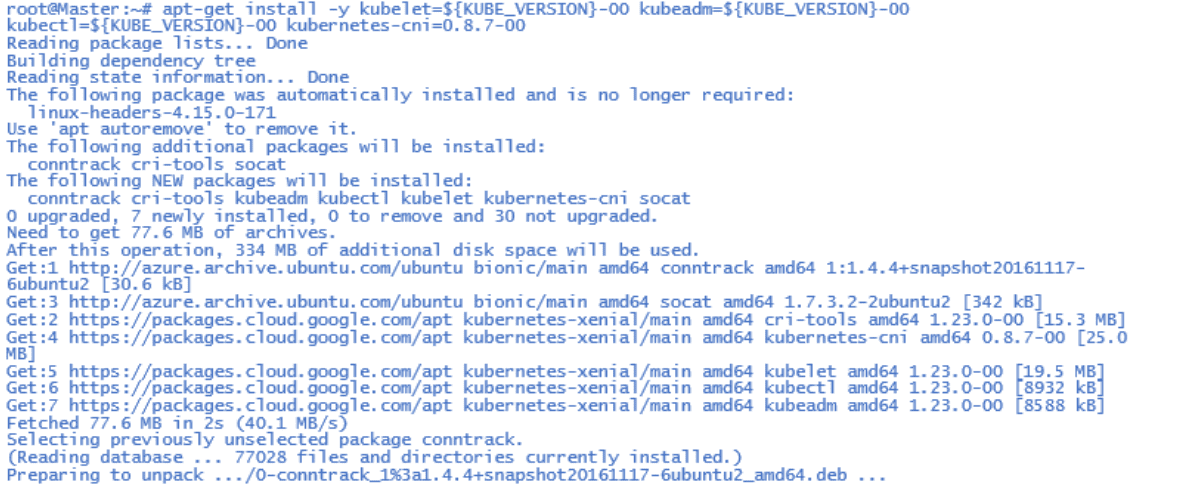
**Step 13)** Update apt package index, install kubelet, kubeadm and kubectl, and pin their version:

$ KUBE\_VERSION=1.23.0

$ apt-get update

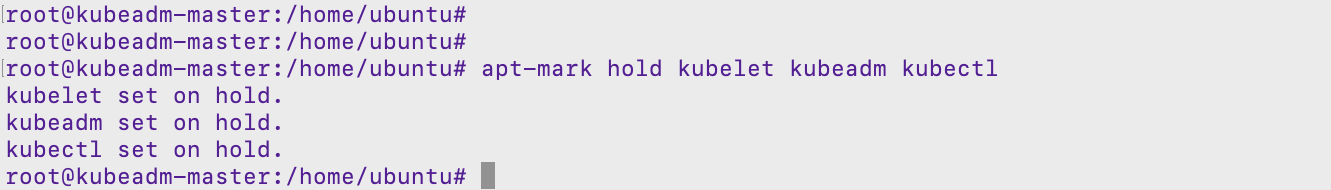


$ apt-get install -y kubelet=${KUBE\_VERSION}-00 kubeadm=${KUBE\_VERSION}-00 kubectl=${KUBE\_VERSION}-00 kubernetes-cni=0.8.7-00



To hold the installed packages at their installed versions, use the following command:

$ apt-mark hold kubelet kubeadm kubectl



Start the kubelet service is required on all the nodes:

$ systemctl enable kubelet && systemctl start kubelet

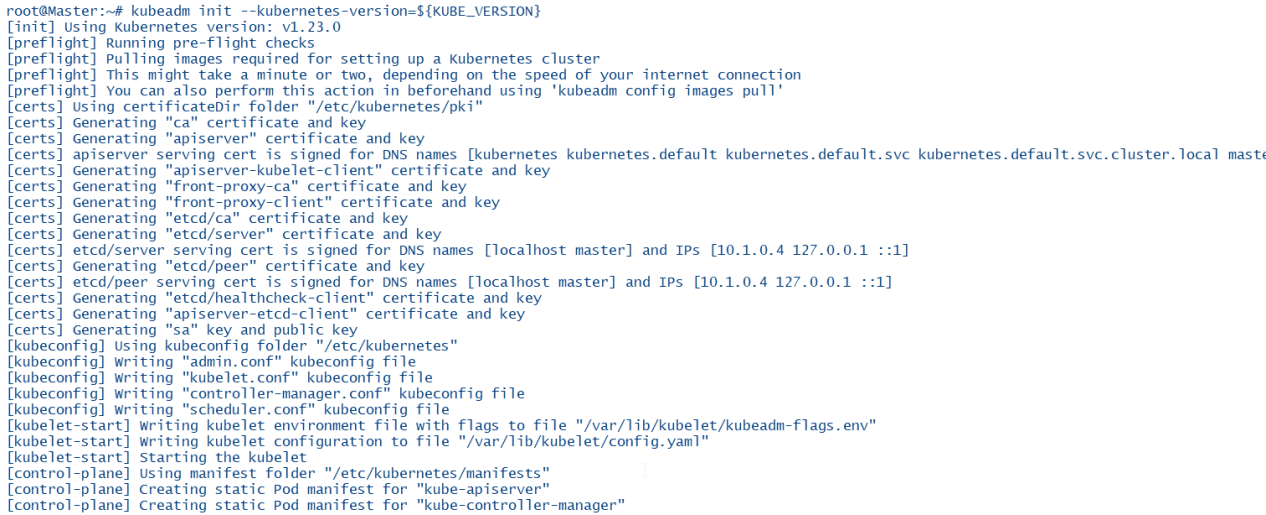
**Check Out:**Our blog post on Ingress Controller, to choose the [best ingress controller for Kubernetes](https://k21academy.com/docker-kubernetes/kubernetes-ingress-controllers/).

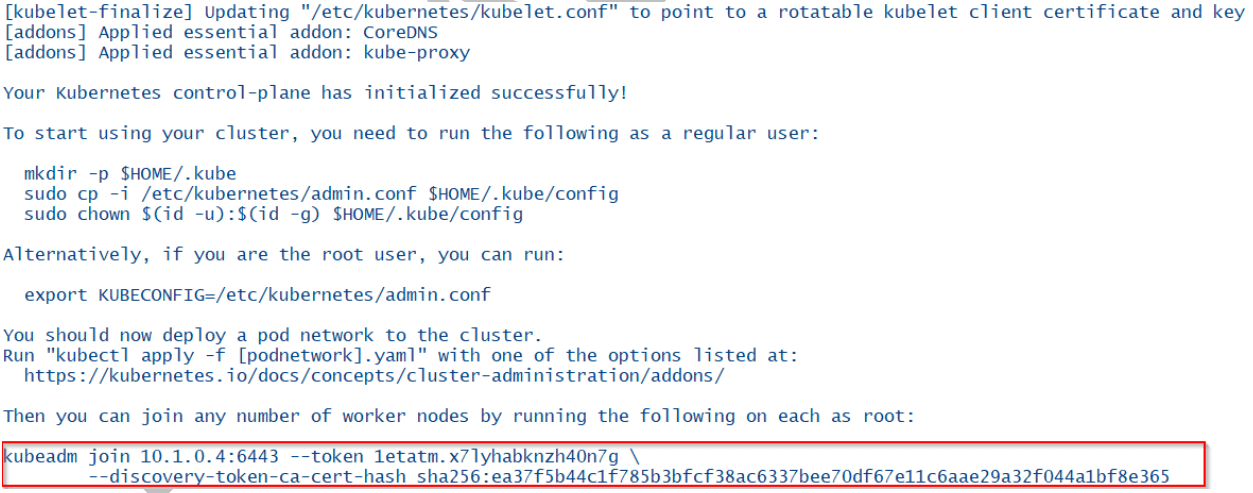
**Create A Kubernetes Cluster**

As we have successfully installed Kubeadm, next we will create a Kubernetes cluster using the following mentioned steps:

**Step 1)** We have to **initialize kubeadm** on the master node. This command will check against the node that we have all the required dependencies. If it is passed, then it will install control plane components.  
(**Note**: Run this command in **Master Node** only.)

$ kubeadm init --kubernetes-version=${KUBE\_VERSION}

  
You will see a similar output:



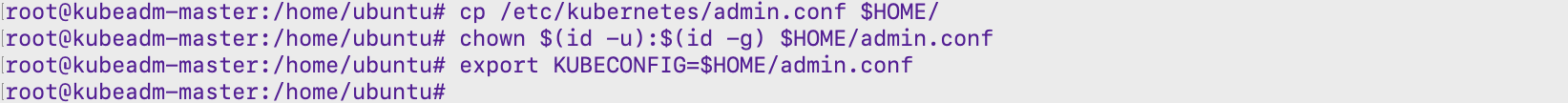
If cluster initialization has succeeded, then we will see a cluster join command. This command will be used by the worker nodes to join the Kubernetes cluster, so copy this command and save it for the future use.

**Step 2)** To start using the cluster, we have to set the environment variable on the master node.  
To temporarily set the environment variables on the master node, run the following commands:  
(**Note**: Every time you are starting the Master, you have to set these Environment Variables.)

$ cp /etc/kubernetes/admin.conf $HOME/

$ chown  $(id -u) $HOME/admin.conf

$ export KUBECONFIG=$HOME/admin.conf



**Also Check:**Our previous blog post on [**Kubernetes deployment**](https://k21academy.com/docker-kubernetes/kubernetes-deployment/). Click here

**Join Worker Nodes to the Kubernetes Cluster**

Now our Kubernetes master node is set up, we should join Worker nodes to our cluster. Perform the following same steps on all of the **worker nodes:**

**Step 1)** SSH into the Worker node with the username and password.

$ ssh <external ip of worker node>

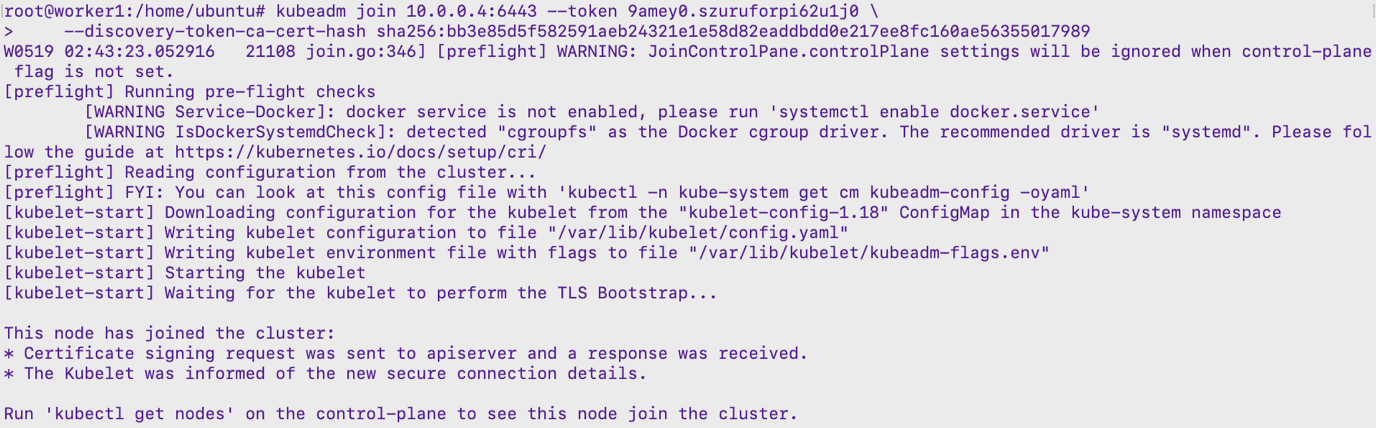
**Step 2)** Run the**kubeadm join** command that we have received and saved.

Note: This is above cluster command, you will get your command in your cluster so use that command not this command

$ kubeadm join 10.0.0.4:6443 --token 9amey0.szuruforpi62u1j0 \

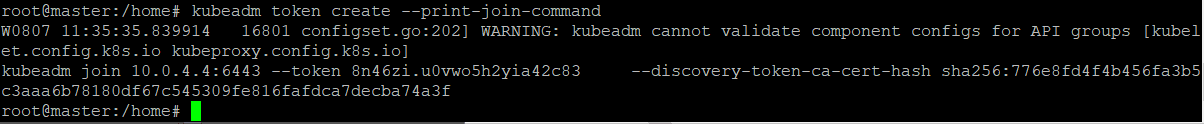
> --discovery-token-ca-cert-hash sha256:bb3e85d5f582591aeb24321e1e58d82eaddbdd0e217ee8fc160ae56355017989

(**Note:** Don’t use this same command, use the command that you have received and saved while doing **kubeadm init** command.)



If you have forgotten to save the above received kubeadm join command, then you can create a new token and use it for joining worker nodes to the cluster.

$ kubeadm token create --print-join-command



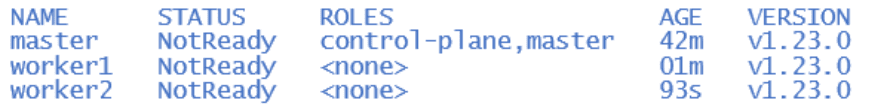
**Check Out:**[What is Kubernetes and Docker](https://k21academy.com/docker-kubernetes/docker-and-kubernetes/). Click here

**Testing the Kubernetes Cluster**

After creating the cluster and joining worker nodes, we have to make sure that everything is working properly. To see and verify the cluster status, we can use **kubectl** command on the master node:

Using **Kubectl get nodes**command, we can see the status of our Nodes (master and worker) whether they are ready or not.

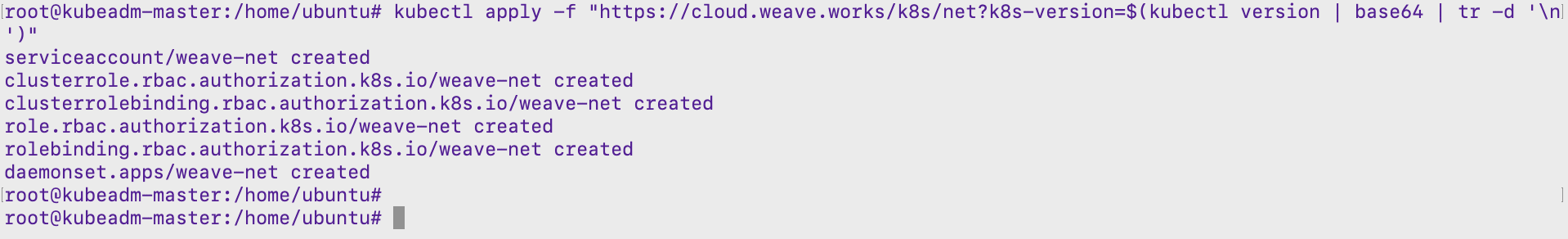
$ kubectl get nodes



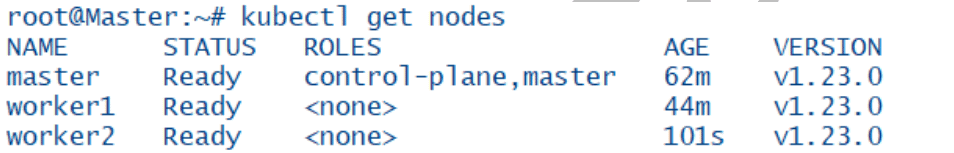
We have to install CNI so that pods can communicate across nodes and also **Cluster DNS** to start functioning. Apply **Weave CNI** (Container Network Interface) on the master node.

**Note:** If you want to know more about Network Policy, then check our blog on [**Kubernetes Network Policy**](https://k21academy.com/docker-kubernetes/network-policies-in-kubernetes/)

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

Wait for a few minutes and verify the cluster status by executing kubectl command on the master node and see that nodes come to the ready state.

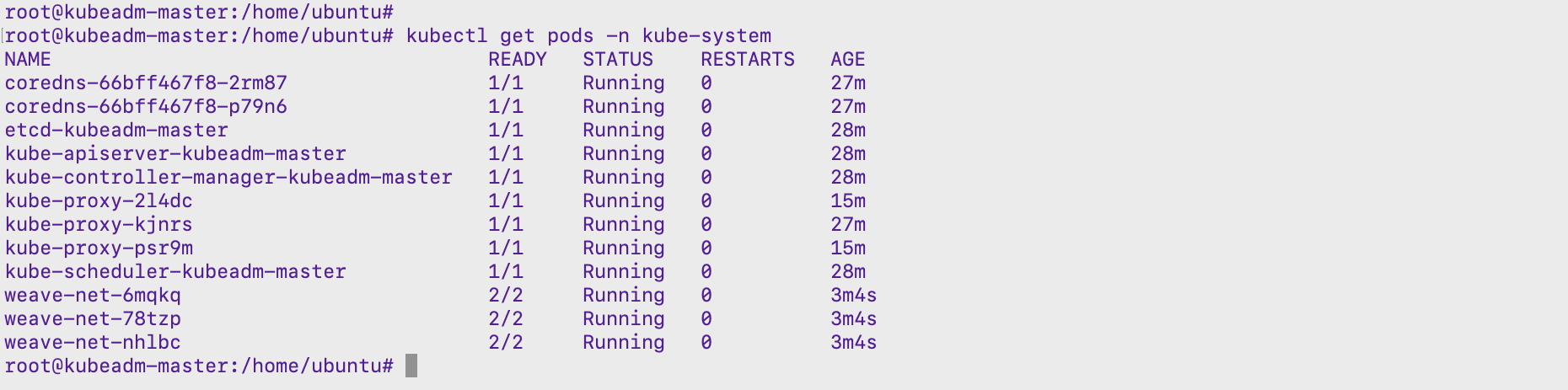
$ kubectl get nodes



To verify the status of the system pods like coreDNS, weave-net, Kube-proxy, and all other master node system processes, use the following command:

To know more about Pods in Kubernetes, check our blog on [**Kubernetes Pods**](https://k21academy.com/docker-kubernetes/kubernetes-pods-for-beginners/)for Beginners**.**

$ kubectl get pods -n kube-system

Your output should match with the shown output above. If not, you will have to check whether you have performed all the steps correctly and on the **mentioned node only**.

**Source:** https://k21academy.com/docker-kubernetes/three-node-kubernetes-cluster/