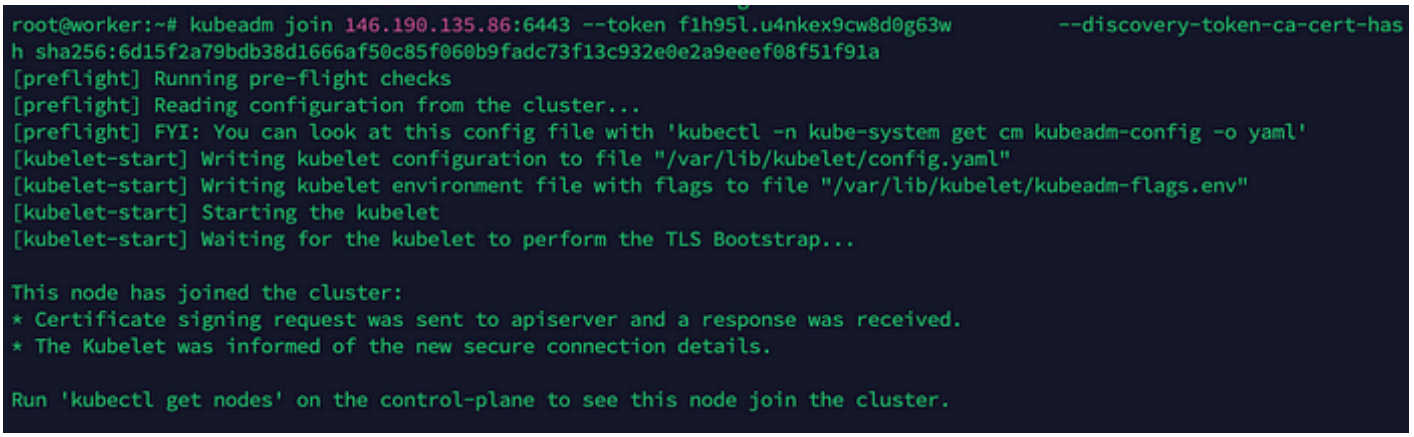

How to Install Kubernetes Cluster on Ubuntu 22.04 (Step-by-Step Guide)

Introduction

Hakan Bayraktar

Follow

~6 min read · November 2, 2023 (Updated: November 3, 2023) · Free: Yes

Kubernetes is a powerful container orchestration platform used for automating the deployment, scaling, and management of containerized applications. In this guide, we will walk you through the step-by-step process of installing Kubernetes on Ubuntu 22.04. This cluster configuration includes a master node and worker nodes, allowing you to harness the full power of Kubernetes.

Kubernetes Nodes

In a Kubernetes cluster, you will encounter two distinct categories of nodes:

Master Nodes: These nodes play a crucial role in managing the control API calls for various components within the Kubernetes cluster. This includes overseeing pods, replication controllers, services, nodes, and more.

Worker Nodes: Worker nodes are responsible for providing runtime environments for containers. It's worth noting that a group of container pods can extend across multiple worker nodes, ensuring optimal resource allocation and management.

Prerequisites

Before diving into the installation, ensure that your environment meets the following prerequisites:

- An Ubuntu 22.04 system.
- Privileged access to the system (root or sudo user).
- Active internet connection.
- Minimum 2GB RAM or more.
- Minimum 2 CPU cores (or 2 vCPUs).
- 20 GB of free disk space on /var (or more).

Step 1: Update and Upgrade Ubuntu (all nodes)

Begin by ensuring that your system is up to date. Open a terminal and execute the following commands:

Copy

```
sudo apt update
sudo apt upgrade
```

Step 2: Disable Swap (all nodes)

To enhance Kubernetes performance, disable swap and set essential kernel parameters. Run the following commands on all nodes to disable all swaps:

Copy

```
```

```
sudo swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Step 3: Add Kernel Parameters (all nodes)

Load the required kernel modules on all nodes:

Copy

```
sudo tee /etc/modules-load.d/containerd.conf <<EOF
overlay
br_netfilter
EOF
sudo modprobe overlay
sudo modprobe br_netfilter
```

Configure the critical kernel parameters for Kubernetes using the following:

Copy

```
sudo 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
```

Then, reload the changes:

Copy

```
sudo sysctl --system
```

Step 4: Install Containerd Runtime (all nodes)

We are using the containerd runtime. Install containerd and its dependencies with the following commands:

Copy

```
sudo apt install -y curl gnupg2 software-properties-common apt-transport-http
```

Enable the Docker repository:

Copy

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --d
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux
```

Update the package list and install containerd:

Copy

```
sudo apt update
sudo apt install -y containerd.io
```

Configure containerd to start using systemd as cgroup:

Copy

```
containerd config default | sudo tee /etc/containerd/config.toml >/dev/null
sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/contain
```

Restart and enable the containerd service:

Copy

```
sudo systemctl restart containerd
sudo systemctl enable containerd
```

Step 5: Add Apt Repository for Kubernetes (all nodes)

Kubernetes packages are not available in the default Ubuntu 22.04 repositories. Add the Kubernetes repositories with the following commands:

Copy

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo gpg --
sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial ma
```

Step 6: Install Kubectl, Kubeadm, and Kubelet (all nodes)

After adding the repositories, install essential Kubernetes components, including kubectl, kubelet, and kubeadm, on all nodes with the following commands:

Copy

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

Step 7: Initialize Kubernetes Cluster with Kubeadm (master node)

With all the prerequisites in place, initialize the Kubernetes cluster on the master node using the following Kubeadm command:

Copy

```
sudo kubeadm init
```

Copy

```
root@master:~# sudo kubeadm init
[init] Using Kubernetes version: v1.28.3
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your i
[preflight] You can also perform this action in beforehand using 'kubeadm con
W1102 19:06:53.288119   10840 checks.go:835] detected that the sandbox image
[certs] Using certificateDir folder "/etc/kubernetes/pki"
[certs] Generating "ca" certificate and key
[certs] Generating "apiserver" certificate and key
[certs] apiserver serving cert is signed for DNS names [kubernetes kubernetes
```

```
[certs] Generating "apiserver-kubelet-client" certificate and key
[certs] Generating "front-proxy-ca" certificate and key
[certs] Generating "front-proxy-client" certificate and key
[certs] Generating "etcd/ca" certificate and key
[certs] Generating "etcd/server" certificate and key
[certs] etcd/server serving cert is signed for DNS names [localhost master] a
[certs] Generating "etcd/peer" certificate and key
[certs] etcd/peer serving cert is signed for DNS names [localhost master] and
[certs] Generating "etcd/healthcheck-client" certificate and key
[certs] Generating "apiserver-etcd-client" certificate and key
[certs] Generating "sa" key and public key
[kubeconfig] Using kubeconfig folder "/etc/kubernetes"
[kubeconfig] Writing "admin.conf" kubeconfig file
[kubeconfig] Writing "kubelet.conf" kubeconfig file
[kubeconfig] Writing "controller-manager.conf" kubeconfig file
[kubeconfig] Writing "scheduler.conf" kubeconfig file
[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manife
[control-plane] Using manifest folder "/etc/kubernetes/manifests"
[control-plane] Creating static Pod manifest for "kube-apiserver"
[control-plane] Creating static Pod manifest for "kube-controller-manager"
[control-plane] Creating static Pod manifest for "kube-scheduler"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/confi
[kubelet-start] Starting the kubelet
[wait-control-plane] Waiting for the kubelet to boot up the control plane as
[apiclient] All control plane components are healthy after 8.002720 seconds
[upload-config] Storing the configuration used in ConfigMap "kubeadm-config"
[kubelet] Creating a ConfigMap "kubelet-config" in namespace kube-system with
[upload-certs] Skipping phase. Please see --upload-certs
[mark-control-plane] Marking the node master as control-plane by adding the l
[mark-control-plane] Marking the node master as control-plane by adding the t
[bootstrap-token] Using token: f1h95l.u4nkex9cw8d0g63w
[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to get
[bootstrap-token] Configured RBAC rules to allow Node Bootstrap tokens to pos
[bootstrap-token] Configured RBAC rules to allow the csrapprover controller a
[bootstrap-token] Configured RBAC rules to allow certificate rotation for all
[bootstrap-token] Creating the "cluster-info" ConfigMap in the "kube-public"
[kubelet-finalize] Updating "/etc/kubernetes/kubelet.conf" to point to a rota
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:
```

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

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each

kubeadm join 146.190.135.86:6443 --token f1h95l.u4nkex9cw8d0g63w \
        --discovery-token-ca-cert-hash sha256:6d15f2a79bdb38d1666af50c85f060b
```

After the initialization is complete make a note of the `kubeadm join` command for future reference.

Run the following commands on the master node:

Copy

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

Next, use `kubectl` commands to check the cluster and node status:

Copy

```
kubectl get nodes
```

```
root@master:~# kubectl get nodes
NAME     STATUS     ROLES           AGE     VERSION
master   NotReady   control-plane   3m21s   v1.28.2
root@master:~#
```

Step 8: Add Worker Nodes to the Cluster (worker nodes)

On each worker node, use the `kubeadm join` command you noted down earlier:

Copy

```
kubeadm join 146.190.135.86:6443 --token f1h95l.u4nkex9cw8d0g63w        --
```

```
root@worker:~# kubeadm join 146.190.135.86:6443 --token f1h95l.u4nkex9cw8d0g63w        --discovery-token-ca-cert-has
h sha256:6d15f2a79bdb38d1666af50c85f060b9fadc73f13c932e0e2a9eeef08f51f91a
[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.
```

Step :9 Install Kubernetes Network Plugin (master node)

To enable communication between pods in the cluster, you need a network plugin. Install the Calico network plugin with the following command from the master node:

Copy

```
kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.
```

Step 10: Verify the cluster and test (master node)

Finally, we want to verify whether our cluster is successfully created.

Copy

```
kubectl get pods -n kube-system
kubectl get nodes
```

```
root@master:~# kubectl get nodes
NAME     STATUS   ROLES           AGE    VERSION
master   Ready    control-plane   2d6h   v1.28.2
worker   Ready    <none>          2d6h   v1.28.2
root@master:~# kubectl get po -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
calico-kube-controllers-658d97c59c-xqj9p   1/1     Running   0          2d6h
calico-node-kp5kh                      1/1     Running   0          2d6h
calico-node-t6csv                      1/1     Running   0          2d6h
coredns-5dd5756b68-klbdw               1/1     Running   0          2d6h
coredns-5dd5756b68-wxgx8               1/1     Running   0          2d6h
etcd-master                            1/1     Running   0          2d6h
kube-apiserver-master                  1/1     Running   0          2d6h
kube-controller-manager-master         1/1     Running   0          2d6h
kube-proxy-4gm7j                       1/1     Running   0          2d6h
kube-proxy-fwdnv                       1/1     Running   0          2d6h
kube-scheduler-master                  1/1     Running   0          2d6h
root@master:~#
```

Step 11: Deploy test application on cluster (master node)

Copy

```
kubectl run nginx --image=nginx
```

```
root@master:~# kubectl get po
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          46s
root@master:~#
```

#kubernetes-cluster #ubuntu #kubernetes-installation #kubeadm #containers