# How to Install a Kubernetes Cluster on CentOS 7 via Kubeadm Tool

Donated by **Google** to the Opensource community, **Kubernetes** has now become the container management tool of choice.

A typical **Kubernetes** cluster would generally have a master node and several worker-nodes or Minions. The worker-nodes are then managed from the master node, thus ensuring that the cluster is managed from a central point.

It’s also important to mention that you can also deploy a single-node Kubernetes cluster which is generally recommended for very light, non-production workloads. For this, you can use **Minikube**, which is a tool that runs a single-node Kubernetes cluster in a virtual machine on your node.

For this tutorial, we will walk-through a multi-node **Kubernetes** cluster installation on **CentOS 7** Linux With Kubeadm Tool.

By Using kubeadm Tool we can automate the installation Process.

# Prerequisites

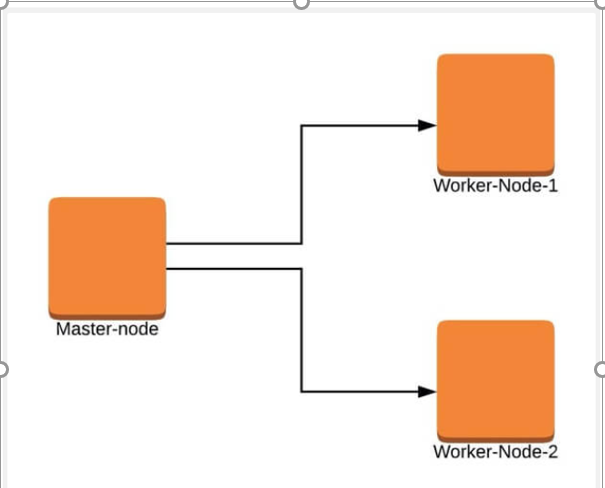
1. Multiple servers running **Centos 7** (**1 Master Node**, **2 Worker Nodes**). It is recommended that your **Master Node** have at least 2 CPUs swap should be off parmanentaly , though this is not a strict requirement.
2. Internet connectivity on all your nodes. We will be fetching **Kubernetes** and **docker** packages from the repository. Equally, you will need to make sure that the yum Package Manager By default is installed by default and can fetch packages remotely.
3. While creating Virtual machines create below networks and Attach two NIC Cards to VM’s in vmware workstation or oracle virtual box

a.Host only Network (it acts as internal Network to your Kubernetes VMs)

b.NAT or Bridged Network(it help to get internet access to download required packages for installation)

# Our Desighn

Please find the below design that we are going to implement



# Installation of Kubernetes Cluster on Master-Node

For **Kubernetes** to work, you will need a containerization engine. For this installation, we will use **docker** as it is the most popular.

The following steps will run on the **Master-Node**.

## 4.1 : Prepare Hostname, Firewall and SELinux

On your master node, set the hostname and if you don’t have a DNS server, then also update your **/etc/hosts** file.

# hostnamectl set-hostname master-node

# cat <<EOF>> /etc/hosts

10.0.0.4 master-node

10.0.0.5 worker-node-1

10.0.0.6 worker-node-2

EOF

You can ping **worker-node-1** and **worker-node-2** to test if your updated hostfile is fine using [ping command](https://www.tecmint.com/linux-ping-command-examples/).

# ping 10.128.0.29

# ping 10.128.0.30

Next, disable **SElinux** and update your firewall rules.

# setenforce 0

# sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux

# reboot

Set the following firewall rules on ports. Make sure that each [firewall-cmd command](https://www.tecmint.com/install-configure-firewalld-in-centos-ubuntu/), returns a success.

firewall-cmd --permanent --add-port=6443/tcp

firewall-cmd --permanent --add-port=2379-2380/tcp

firewall-cmd --permanent --add-port=10250/tcp

firewall-cmd --permanent --add-port=10251/tcp

firewall-cmd --permanent --add-port=10252/tcp

firewall-cmd --permanent --add-port=10255/tcp

firewall-cmd –reload

modprobe br\_netfilter

echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables

## 4.2 Setup the Kubernetes Repo

You will need to add **Kubernetes** repositories manually as they do not come installed by default on **CentOS 7**.

Note: you can search for latest Repos always.

cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

## **4.3 Install Kubeadm and Docker**

With the package repo now ready, you can go ahead and install **kubeadm** and **docker** packages.2

# yum install kubeadm docker -y

When the installation completes successfully, enable and start both services.

# systemctl enable kubelet

# systemctl start kubelet

# systemctl enable docker

# systemctl start docker

## 4.4 Initialize Kubernetes Master and Setup Default User

Now we are ready to initialize kubernetes master, but before that you need to disable swap in order to run “**kubeadm init**“ command.

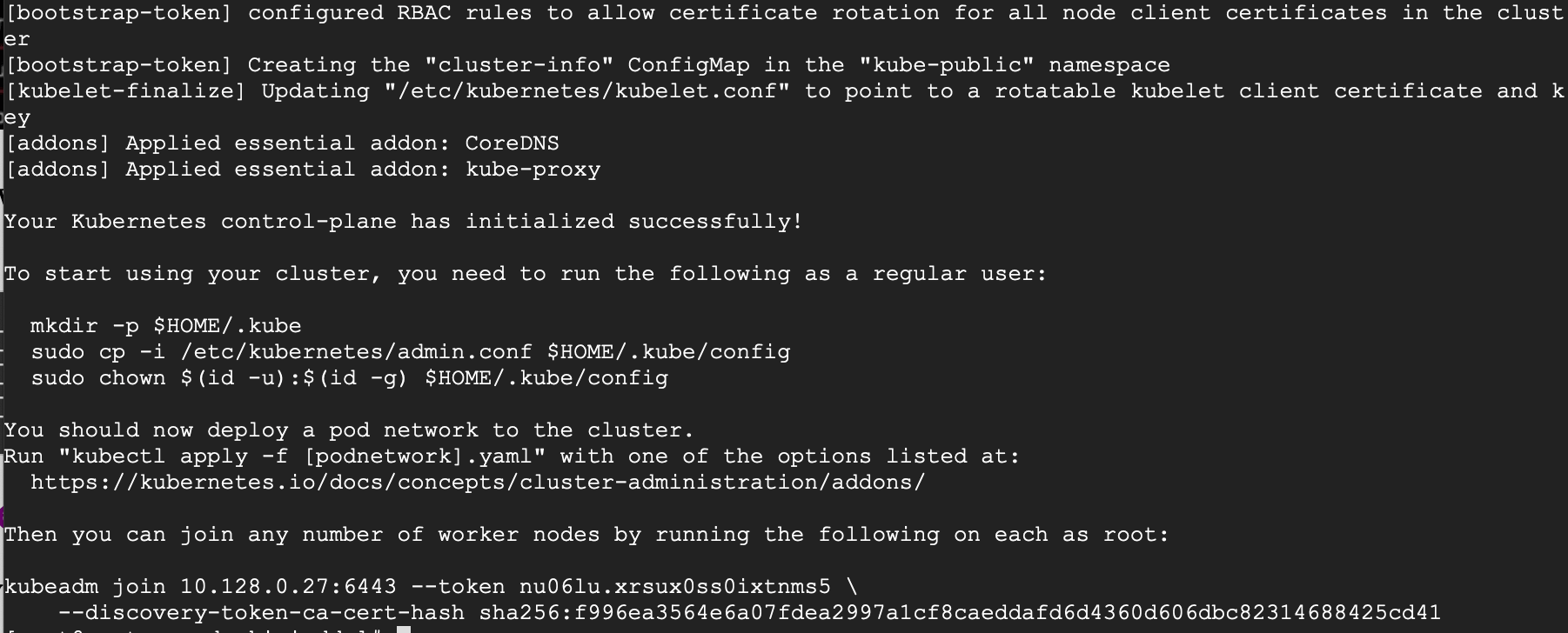
# swapoff -a

Initializing Kubernetes master is a fully automated process that is managed by the Kubeadm Tool with “**kubeadm init**“ command which you will run.

# kubeadm init



### **4.4.1 Example O/P from Kubeadm init:**



Note:All ways use Hostonly Ips for no Conflict Issues

You may want to copy the last line and save it somewhere because you will need to run it on the **worker nodes**.

kubeadm join 10.128.0.27:6443 --token nu06lu.xrsux0ss0ixtnms5 \ --discovery-tok

Having initialized **Kubernetes** successfully, you will need to allow your user to start using the cluster. In our case, we want to run this installation as **root** user, therefore we will go ahead and run these commands as root. You can change to a Sudo Enabled User you prefer and run the below using sudo.

To use **root**, run:

mkdir -p $HOME/.kube

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

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

To use a **sudo enabled user**, run:

mkdir -p $HOME/.kube

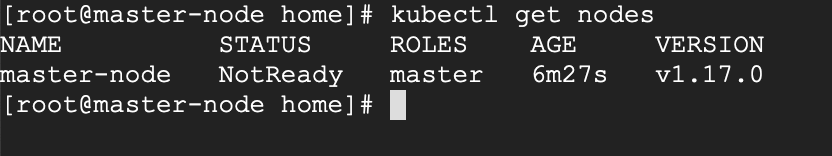
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

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

Now check to see if the **kubectl** command is activated.

# kubectl get nodes

Example O/P:



At this point, you will also notice that the status of the **master-node** is ‘**NotReady**’. This is because we are yet to deploy the **pod network** to the cluster.

The **pod Network** is the overlay network for the cluster, that is deployed on top of the present node network. It is designed to allow connectivity across the pod.

# Setup Your Pod Network ( Network for your Containers)

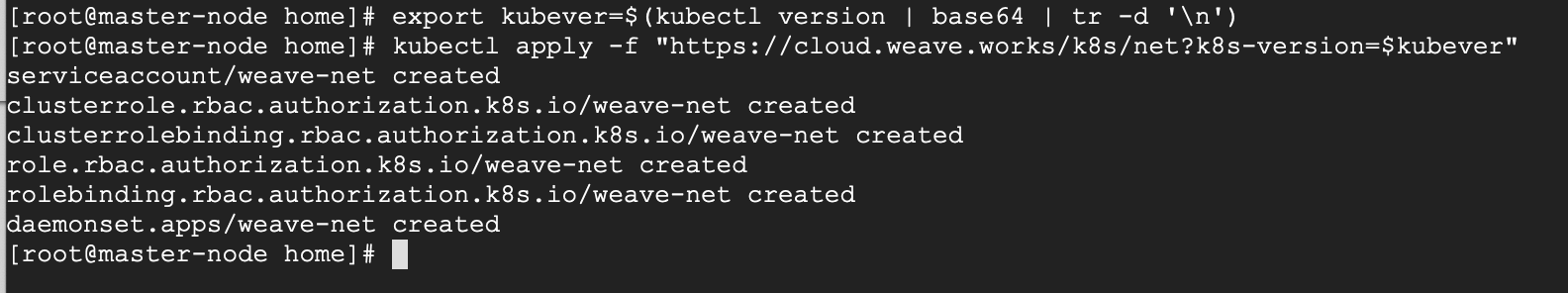
Deploying the network cluster is a highly flexible process depending on your needs and there are many options available. Since we want to keep our installation as simple as possible, we will use **Weavenet** plugin which does not require any configuration or extra code and it provides one IP address per pod which is great for us. If you want to see more options, please [check here](https://kubernetes.io/docs/concepts/cluster-administration/networking/).

These commands will be important to get the pod network setup.

# export kubever=$(kubectl version | base64 | tr -d '\n')

# kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$kubever"

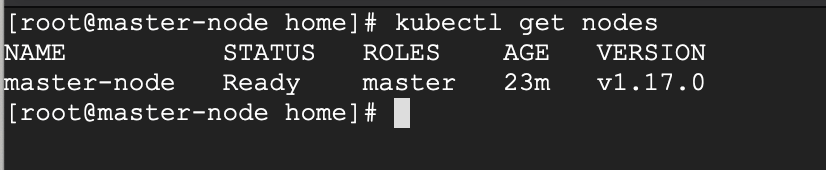
Eample O/P:



Now if you check the status of your **master-node**, it should be ‘**Ready**’.

# kubectl get nodes

**Eample O/P:**



Next, we add the **worker nodes** to the cluster.

# 6 Setting Up Worker Nodes to Join Kubernetes Cluster

The following steps will run on the **worker nodes**. These steps should be run on every **worker node** when joining the **Kubernetes** cluster.

## **6.1 Prepare Hostname, Firewall and SELinux**

On your **worker-node-1** and **worker-node-2**, set the hostname and in case you don’t have a DNS server, then also update your master and worker nodes on **/etc/hosts** file.

# hostnamectl set-hostname 'worker-node-1'

# cat <<EOF>> /etc/hosts

10.0.0.4 master-node

10.0.0.5 worker-node-1

10.0.0.6 worker-node-2

EOF

You can ping **master-node** to test if your updated hostfile is fine.

Next, disable **SElinux** and update your firewall rules.

# setenforce 0

# sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux

Set the following firewall rules on ports. Make sure that all firewall-cmd commands, return success.

firewall-cmd --permanent --add-port=6783/tcp

firewall-cmd --permanent --add-port=10250/tcp

firewall-cmd --permanent --add-port=10255/tcp

firewall-cmd --permanent --add-port=30000-32767/tcp

firewall-cmd  --reload

# echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables

## **6.2 Setup the Kubernetes Repo**

You will need to add **Kubernetes** repositories manually as they do not come pre-installed on **CentOS 7**.

cat <<EOF > /etc/yum.repos.d/kubernetes.repo

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

EOF

## **6.3 Install Kubeadm and Docker**

With the package repo now ready, you can go ahead and install **kubeadm** and **docker** packages.

# yum install kubeadm docker -y

Start and enable both the services.

# systemctl enable docker

# systemctl start docker

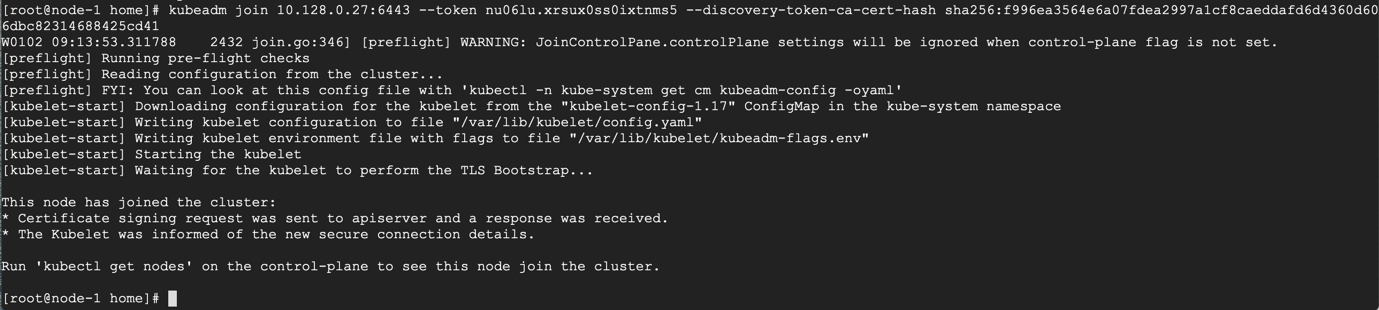
# systemctl enable kubelet

# systemctl start kubelet

## **6.4 Join the Worker Node to the Kubernetes Cluster**

We now require the token that **kubeadm** init generated, to join the cluster. You can copy and paste it to your **node-1** and **node-2** if you had copied it somewhere.

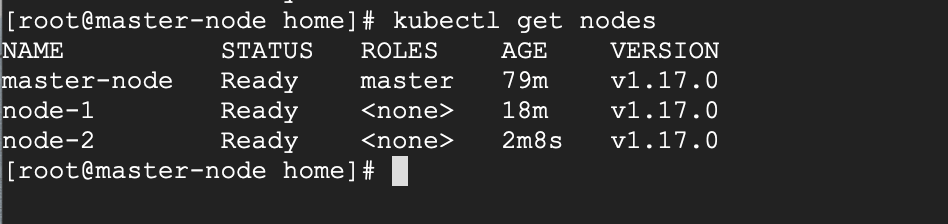
# kubeadm join 10.128.0.27:6443 --token nu06lu.xrsux0ss0ixtnms5 --discovery-token-ca-cert-hash sha256:f996ea3564e6a07fdea2997a1cf8caeddafd6d4360d606dbc82314688425cd41

[](https://www.tecmint.com/wp-content/uploads/2020/01/Join-Nodes-to-Kubernets-Cluster.png)

Join Nodes to Kubernets Cluster

As suggested on the last line, go back to your **master-node** and check if **worker node-1** and **worker node-2** have joined the cluster using the following command.

# kubectl get nodes

[](https://www.tecmint.com/wp-content/uploads/2020/01/Check-All-Nodes-Status-in-Kubernetes-Cluster.png)

Check All Nodes Status in Kubernetes Cluster

If all the steps run successfully, then, you should see **node-1** and **node-2** in ready status on the **master-node**.

# 7 How to Deploy Nginx on a Kubernetes Cluster

I will run this deployment on a Virtual Machine Hosted by a public cloud provider. As it is with many public cloud services, many generally maintain a public and private IP scheme for their Virtual Machines.

Testing Environment

Master Node - Public IP: 104.197.170.99 and Private IP: 10.128.15.195

Worker Node 1 - Public IP: 34.67.149.37 and Private IP: 10.128.15.196

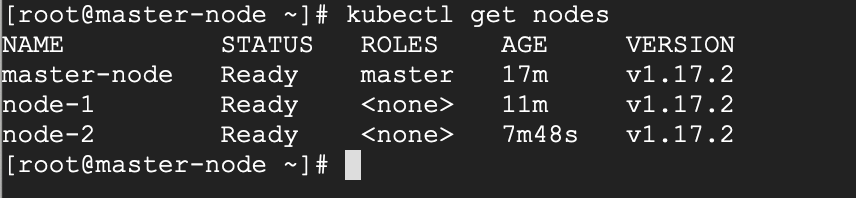
Worker Node 2 - Public IP: 35.232.161.178 and Private IP: 10.128.15.197

## 7.1 Deploying NGINX on a Kubernetes Cluster

We will run this deployment from the master-node.

Let’s begin by checking the status of the cluster. All your nodes should be in a **READY** state.

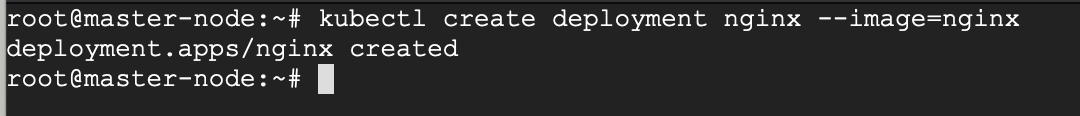
# kubectl get nodes

[](https://www.tecmint.com/wp-content/uploads/2020/02/Check-Kubernetes-Cluster-Status.png)

Check Kubernetes Cluster Status

We create a deployment of **NGINX** using the **NGINX** image.

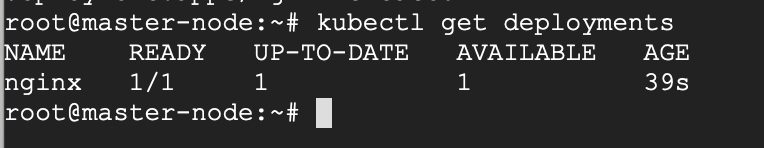
# kubectl create deployment nginx --image=nginx

[](https://www.tecmint.com/wp-content/uploads/2020/02/Deploying-Nginx-on-Kubernetes.png)

Deploying Nginx on Kubernetes

You can now see the state of your deployment.

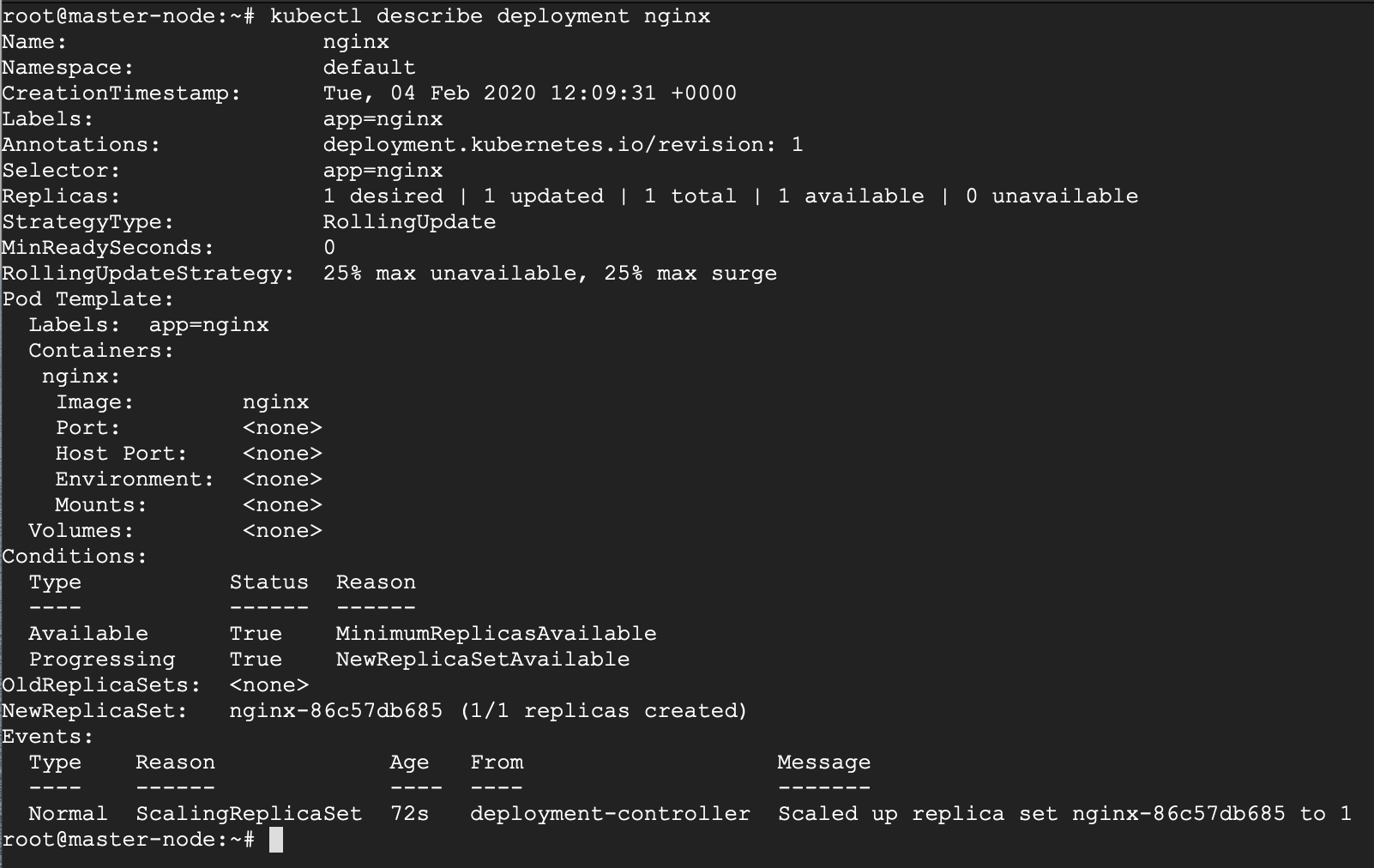
# kubectl get deployments

[](https://www.tecmint.com/wp-content/uploads/2020/02/Check-Deployment-of-Nginx-on-Kubernetes.png)

Check Deployment of Nginx on Kubernetes

If you’d like to see more detail about your deployment, you can run the **describe** command. For example, it is possible to determine how many replicas of the deployment are running. In our case, we expect to see a replica of 1 running (i.e **1/1** replicas).

# kubectl describe deployment nginx

[](https://www.tecmint.com/wp-content/uploads/2020/02/Check-Nginx-Deployment-Details.png)

Check Nginx Deployment Details

Now your Nginx deployment is active, you may want to expose the **NGINX** service to a public IP reachable on the internet.

## **7.2 Exposing Your Nginx Service to Public Network**

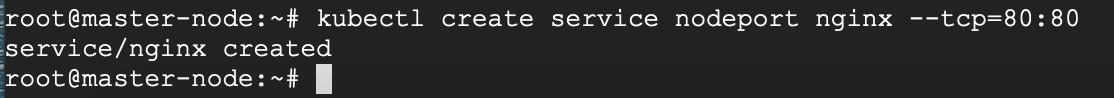
Kubernetes offers several options when exposing your service based on a feature called Kubernetes Service-types and they are:

1. **ClusterIP** – This Service-type generally exposes the service on an internal IP, reachable only within the cluster, and possibly only within the cluster-nodes.
2. **NodePort** – This is the most basic option of exposing your service to be accessible outside of your cluster, on a specific port (called the **NodePort**) on every node in the cluster. We will illustrate this option shortly.
3. **LoadBalancer** – This option leverages on external Load-Balancing services offered by various providers to allow access to your service. This is a more reliable option when thinking about high availability for your service, and has more feature beyond default access.
4. **ExternalName** – This service does traffic redirect to services outside of the cluster. As such the service is thus mapped to a DNS name that could be hosted out of your cluster. It is important to note that this does not use proxying.

The default Service-type is **ClusterIP**.

In our scenario, we want to use the **NodePort Service-type** because we have both a public and private IP address and we do not need an external load balancer for now. With this service-type, Kubernetes will assign this service on ports on the **30000+** range.

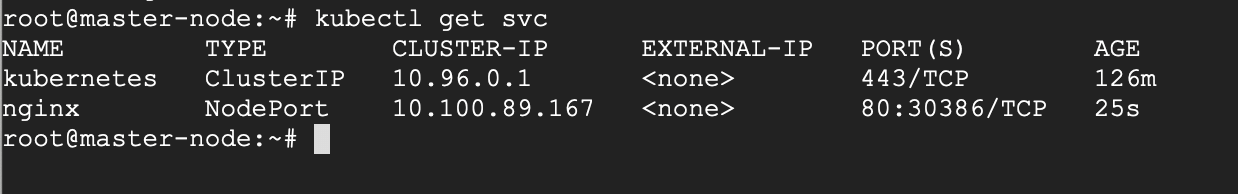
# kubectl create service nodeport nginx --tcp=80:80

[](https://www.tecmint.com/wp-content/uploads/2020/02/Create-NodePort-Service-to-Expose-nginx.png)

Create NodePort Service to Expose Nginx

Run the **get svc** command to see a summary of the service and the ports exposed.

# kubectl get svc

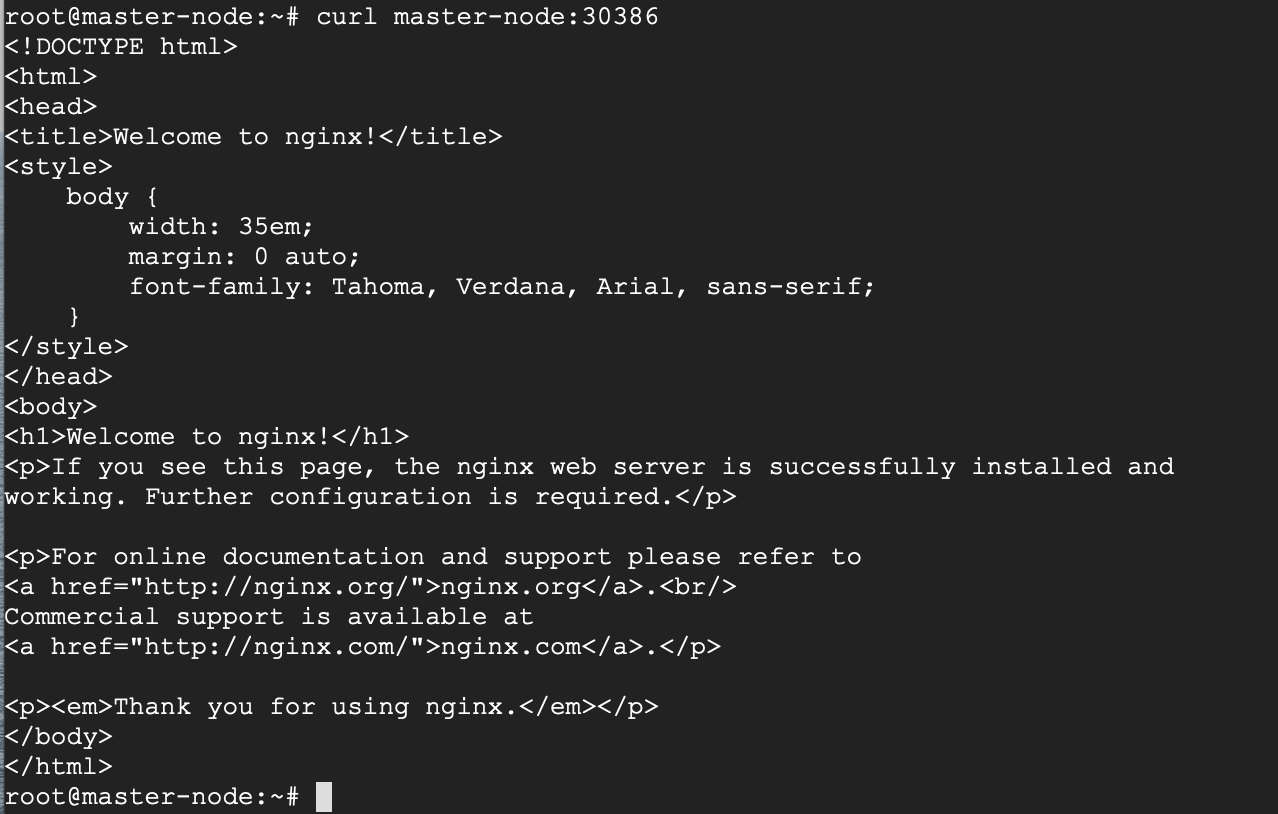
[](https://www.tecmint.com/wp-content/uploads/2020/02/Check-Nginx-Service-and-Port.png)Check Nginx Service and Port

Now you can verify that the **Nginx** page is reachable on all nodes using curl

# curl master-node:30386

# curl node-1:30386

# curl node-2:30386

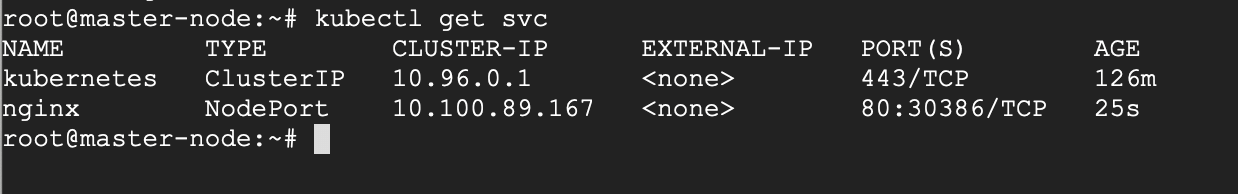


As you can see, the “**WELCOME TO NGINX!**” page can be reached.

Reaching Ephemeral PUBLIC IP Addresses

As you may have noticed, **Kubernetes** reports that I have no active Public IP registered, or rather no EXTERNAL-IP registered.

# kubectl get svc

[](https://www.tecmint.com/wp-content/uploads/2020/02/Check-Nginx-Service-and-Port.png)

Check Nginx Service and Port

Let’s verify if it is indeed true, that I have no EXTERNAL IP attached to my interfaces using ip cmd

# ip a

Check IP Addresses

No public IP as you can see.

As mentioned earlier, I am currently running this deployment on a **Virtual Machine** offered by a public cloud provider. So, while there’s no particular interface assigned a public IP, the VM provider has issued an Ephemeral external IP address.

An ephemeral external IP address is a temporary IP address that remains attached to the VM until the virtual instance is stopped. When the virtual instance is restarted, a new external IP is assigned. Basically put, it’s a simple way for service providers to leverage on idle public IPs.

The challenge here, other than the fact that your public IP is not static, is that the **Ephemeral Public IP** is simply an extension (or proxy) of the Private IP, and for that reason, the service will only be accessed on port 30386. That means that the service will be accessed on the URL <PublicIP:InternalPort>, that is **104.197.170.99:30386**, which if you check your browser, you should be able to see the welcome page.

Check Nginx Page Deployed on Kubernetes

With that, we have successfully deployed **NGINX** on our 3-node Kubernetes cluster.

