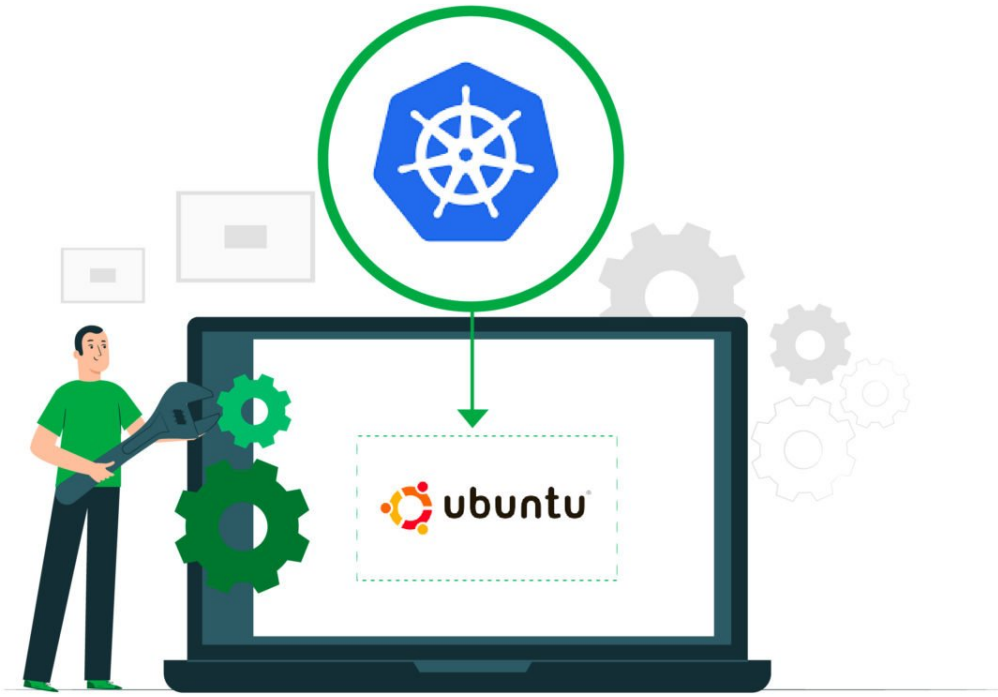


How to Install and Use Kubernetes on Ubuntu 20.04

Introduction

Kubernetes (https://kubernetes.io/) is an open-source tool that is crucial in container orchestration (https://docs.docker.com/get-started/orchestration/). Kubernetes works by orchestrating and managing clusters at scale across various cloud environments or even on-premise servers. A cluster is a set of hosts meant for running containerized applications and services. A cluster needs a minimum of two nodes to work – one `master node` and a `worker node`. Keeping scalability in mind, you have the option to expand the cluster with as many worker nodes as required.

A node in Kubernetes refers to a server. A master node is a server that manages the state of the cluster. Worker nodes are servers that run the workloads – these are typically containerized applications and services (https://www.cloudsigma.com/how-to-install-operate-docker-on-ubuntu-in-the-public-cloud/).

This **guide will walk you through the steps of installing and deploying a Kubernetes cluster consisting of two nodes on Ubuntu 20.04**. As mentioned, having two nodes is the most basic configuration when working with Kubernetes. You also have the option to add more worker nodes once you understand the fundamentals. We will further show you how to link the two servers to allow the master node to control the worker node.

To test our configuration, we will deploy a Docker (https://www.docker.com/) container running the Nginx webserver (https://www.cloudsigma.com/installing-nginx-on-ubuntu-18-04/) to the cluster. This is a typical real-life application of Kubernetes. You will learn more about some defining components of Kubernetes such as `kubectl` and `kubeadm` as we go along. It is also advisable to first get acquainted with our tutorial on getting to know Kubernetes tool-kit basics (https://www.cloudsigma.com/getting-to-know-kubernetes/) to familiarize yourself with the basics of the Kubernetes platform.

Now, let's start!

Prerequisites

You will need to provision two servers, running on `Ubuntu 20.04`. For best performance, the minimum system requirements for Kubernetes are 2GB of RAM and 2 CPUs. You may follow steps 1 to 4 of this step-by-step tutorial to help you set up your Ubuntu server (https://www.cloudsigma.com/how-to-set-up-your-ubuntu-18-04-server/) on CloudSigma. One server will be the master node, the other will be the worker node. We have aptly named our two servers as `kubernetes-master` and `kubernetes-worker`. This makes it easier to follow along with the tutorial. However, you are free to choose the `hostnames` you prefer.

- Ensure you add a user with `sudo` privileges on both nodes that we will use to run the commands as outlined in the tutorial above. Follow this tutorial on configuring the Linux sudoers file for instructions (https://www.cloudsigma.com/configuring-the-linux-sudoers-file/).
- Network connectivity – the servers in the cluster should be able to communicate. When you deploy your VMs from CloudSigma (https://zrh.cloudsigma.com/ui/4.0/login), they will be connected to the internet with a public IP by default. If you are working from a local network,

you may have to edit your /etc/hosts (https://linuxize.com/post/how-to-edit-your-hosts-file/) file in each server and link them appropriately.

- You will need to install and enable `Docker` on each of the nodes. Kubernetes relies on a container runtime to run containers in pods. While there are other container platforms to choose from, we will be using Docker in this tutorial. Docker will provide the runtime environment needed by Ubuntu. You may follow steps 1, 2, and 3 of our tutorial on installing and operating Docker (https://www.cloudsigma.com/how-to-install-operate-docker-on-ubuntu-in-the-public-cloud/).

Step 1: Install Kubernetes

In this step, we will be installing Kubernetes. Just like you did with Docker in the prerequisites, you must run the commands in both nodes to install Kubernetes. Use `ssh` to login into both nodes and proceed. You will start by installing the `apt-transport-https` package which enables working with `http` and `https` in Ubuntu's repositories. Also, install `curl` as it will be necessary for the next steps. Execute the following command:

```
1 sudo apt install apt-transport-https curl
```

Then, add the `Kubernetes signing key` to both nodes by executing the command:

```
1 curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add
```

Next, we add the `Kubernetes repository` as a package source on both nodes using the following command:

```
1 echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" >> ~/kubernetes.list
2 sudo mv ~/kubernetes.list /etc/apt/sources.list.d
```

After that, update the nodes:

```
1 sudo apt update
```

- **Install Kubernetes tools**

Once the update completes, we will install Kubernetes. This involves installing the various tools that make up Kubernetes: `kubeadm`, `kubelet`, `kubectl`, and `kubernetes-cni`. These tools are installed on both nodes. We define each tool below:

- **kubelet** – an agent that runs on each node and handles communication with the master node to initiate workloads in the container runtime. Enter the following command to install kubelet:


```
1 sudo apt install kubelet
```

- **kubeadm** – part of the Kubernetes project and helps initialize a Kubernetes cluster. Enter the following command to install the kubeadm:

```
sudo apt install kubeadm
```

- **kubectl** – the Kubernetes command-line tool that allows you to run commands inside the Kubernetes clusters. Execute the following command to install kubectl:

```
sudo apt install kubectl
```

- **kubernetes-cni** – enables networking within the containers ensuring containers can communicate and exchange data. Execute the following command to install:

```
sudo apt-get install -y kubernetes-cni
```

Optionally, you can install all four in a single command:

```
sudo apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```

Step 2: Disabling Swap Memory

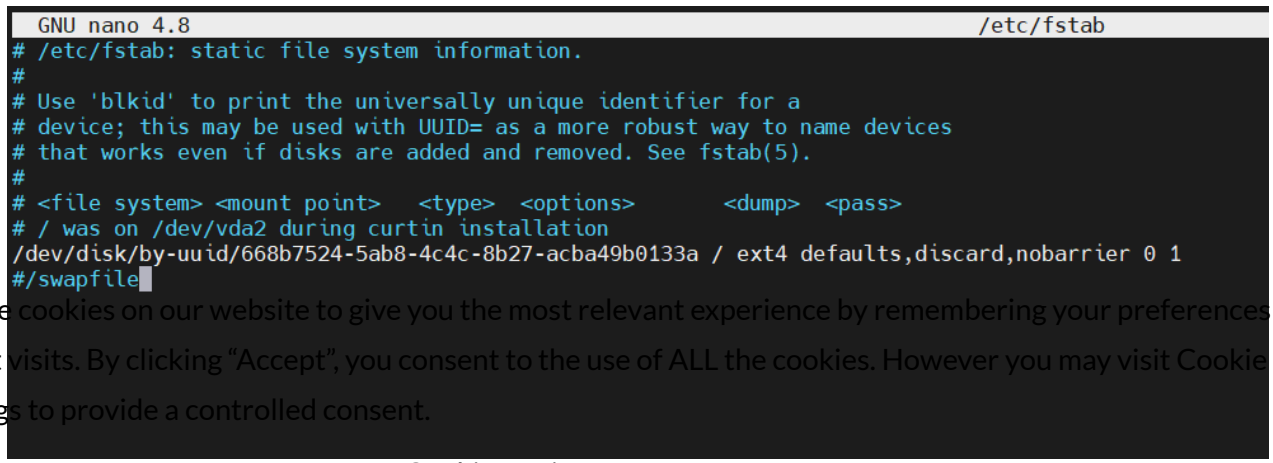
Kubernetes fails to function in a system that is using `swap memory`. Hence, it must be disabled in the master node and all worker nodes. Execute the following command to disable swap memory:

```
sudo swapoff -a
```

This command disables swap memory until the system is rebooted. We have to ensure that it remains off even after reboots. This has to be done on the master and all worker nodes. We can do this by editing the `fstab file` and commenting out the `/swapfile` line with a `#`. Open the file with the nano text editor by entering the following command:

```
sudo nano /etc/fstab
```

Inside the file, comment out the `swapfile` line as shown in the screenshot below:



If you do not see the swapfile line, just ignore it. Save and close the file when you are done editing. Follow the same process for both nodes. Now, swap memory settings will remain off, even after your server reboots.

Step 3: Setting Unique Hostnames

Your nodes must have unique hostnames for easier identification. If you are deploying a cluster with many nodes, you can set it to identify names for your worker nodes such as node-1, node-2, etc. As we had mentioned earlier, we have named our nodes as `kubernetes-master` and `kubernetes-worker`. We have set them at the time of creating the server. However, you can adjust or set yours if you had not already done so from the command line. To adjust the hostname on the master node, run the following command:

```
1 sudo hostnamectl set-hostname kubernetes-master
```

On the worker node, run the following command:

```
1 sudo hostnamectl set-hostname kubernetes-worker
```

You may close the current terminal session and `ssh` back into the server to see the changes.

Step 4: Letting Iptables See Bridged Traffic

For the master and worker nodes to correctly see bridged traffic, you should ensure `net.bridge.bridge-nf-call-iptables` is set to 1 in your config. First, ensure the `br_netfilter` module is loaded. You can confirm this by issuing the command:

```
1 lsmod | grep br_netfilter
```

Optionally, you can explicitly load it with the command:

```
1 sudo modprobe br_netfilter
```

Now, you can run this command to set the value to 1:

```
1 sudo sysctl net.bridge.bridge-nf-call-iptables=1
```

Step 5: Changing Docker Cgroup Driver

By default, Docker installs with `"cgroupfs"` as the cgroup driver. Kubernetes recommends (https://kubernetes.io/docs/setup/production-environment/container-runtimes/#docker) that Docker should `run` with `"systemd"` as the driver. If you skip this step and try to initialize the

kubeadm in the next step, you will get the following warning in your terminal:

```
[preflight] Running pre-flight checks
    [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup driver.
```

On both master and worker nodes, update the `cgroupdriver` with the following commands:

```
sudo mkdir /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{ "exec-opts": ["native.cgroupdriver=systemd"],
"log-driver": "json-file",
"log-opts":
{ "max-size": "100m" },
"storage-driver": "overlay2"
}
EOF
```

Then, execute the following commands to `restart` and `enable` Docker on system boot-up:

```
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

Once that is set, we can proceed to the fun stuff, deploying the Kubernetes cluster!

Step 6: Initializing the Kubernetes Master Node

The first step in deploying a Kubernetes cluster is to fire up the master node. While on the terminal of your master node, execute the following command to initialize the `kubernetes-master`:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

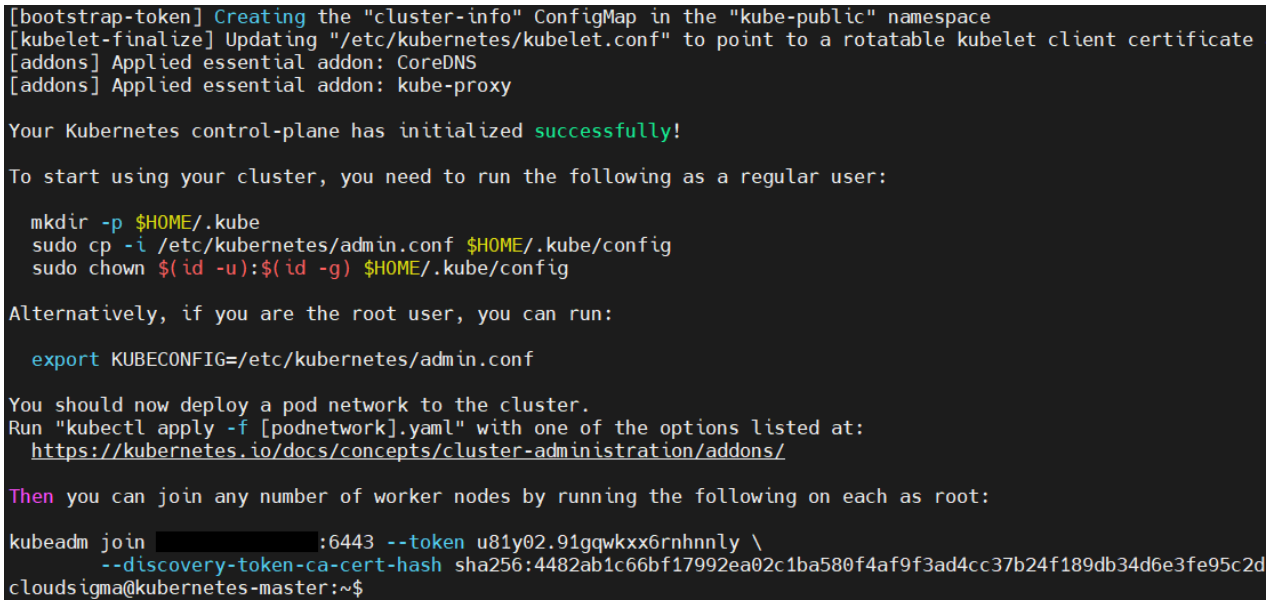
If you execute the above command and your system doesn't match the expected requirements, such as minimum RAM or CPU as explained in the Prerequisites section, you will get a warning and the cluster will not start:

```
cloudsigma@kubernetes-master:~$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
[init] Using Kubernetes version: v1.22.2
[preflight] Running pre-flight checks
error execution phase preflight: [preflight] Some fatal errors occurred:
        [ERROR NumCPU]: the number of available CPUs 1 is less than the required 2
[preflight] If you know what you are doing, you can make a check non-fatal with `--ignore-preflight-errors=...`
To see the stack trace of this error execute with --v=5 or higher
cloudsigma@kubernetes-master:~$
```

Note: If you are building for production, it's a good idea to always meet the minimum requirements for Kubernetes to run smoothly. However, if you are doing this tutorial for learning purposes, then you can add the following flag to the kubeadm init command to ignore the error warnings:

```
sudo kubeadm init --ignore-preflight-errors=NumCPU,Mem --pod-network-cidr=10.244.0.0/16
```

The screenshot below shows that the initialization was successful. We have also added a flag to specify the pod network with the IP 10.244.0.0, It's the default IP that the kube-flannel uses (https://blog.laputa.io/kubernetes-flannel-networking-6a1cb1f8ec7c). We will discuss more on the pod network in the next step.



In the output, you can see the `kubeadm join` command (we've hidden our IP address) and a unique token that you will run on the worker node and all other worker nodes that you want to join onto this cluster. Next, copy-paste this command as you will use it later in the worker node.

In the output, Kubernetes also displays some additional commands that you should run as a regular user on the master node before you start to use the cluster. Let's run these commands:

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

We have now initialized the master node. However, we also have to set up the pod network on the master node before we join the worker nodes.

Step 7: Deploying a Pod Network

A pod network facilitates communication between servers and it's necessary for the proper functioning of the Kubernetes cluster. You can read more about Kubernetes Cluster Networking (https://kubernetes.io/docs/concepts/cluster-administration/networking/) from the official docs.

We will be using the Flannel (https://kubernetes.io/docs/concepts/cluster-administration/networking/#flannel) pod network for this tutorial. Flannel is a simple overlay network that satisfies the Kubernetes requirements.

Before we deploy the pod network, we need to check on the firewall status. If you have enabled the firewall after following step 5 of the tutorial on setting up your Ubuntu server (https://www.cloudsigma.com/how-to-set-up-your-ubuntu-18-04-server/), you must first add a `firewall rule` to create exceptions for port `6443` (the default port for Kubernetes). Run the following ufw (https://wiki.ubuntu.com/UncomplicatedFirewall) commands on both master and worker nodes:

```
sudo ufw allow 6443
sudo ufw allow 6443/tcp
```

After that, you can run the following two commands to deploy the pod network on the master node:

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentati
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentati
```

This may take a couple of seconds to a minute depending on your environment to load up the flannel network. Run the following command to confirm that everything is fired up:

```
kubectl get pods --all-namespaces
```

The output of the command should show all services status as running if everything was successful:

```
cloudsigma@kubernetes-master:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                      READY   STATUS    RESTARTS   AGE
kube-system   coredns-78fcd69978-hrbg5                  1/1     Running   0          10m
kube-system   coredns-78fcd69978-zn94f                  1/1     Running   0          10m
kube-system   etcd-kubernetes-master                    1/1     Running   0          10m
kube-system   kube-apiserver-kubernetes-master          1/1     Running   0          10m
kube-system   kube-controller-manager-kubernetes-master 1/1     Running   0          10m
kube-system   kube-flannel-ds-2xk6h                     1/1     Running   0          114s
kube-system   kube-proxy-d2x5v                          1/1     Running   0          10m
kube-system   kube-scheduler-kubernetes-master          1/1     Running   0          10m
```

You can also view the `health` of the components using the get component status command:

```
kubectl get componentstatus
```

```
cloudsigma@kubernetes-master:~$ kubectl get componentstatus
Warning: v1 ComponentStatus is deprecated in v1.19+
NAME                 STATUS    MESSAGE                         ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health":"true","reason":""}
```

This command has a short form es:

```
kubectl get cs
```

```
cloudsigma@kubernetes-master:~$ kubectl get cs
Warning: v1 ComponentStatus is deprecated in v1.19+
NAME                 STATUS    MESSAGE                         ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health":"true","reason":""}
```

If you see the unhealthy status, modify the following files and delete the line at (spec->containers->command) containing this phrase `- --port=0` :

```
sudo nano /etc/kubernetes/manifests/kube-scheduler.yaml
```

Do the same for this file:

```
sudo nano /etc/kubernetes/manifests/kube-controller-manager.yaml
```

Finally, restart the Kubernetes service:

```
sudo systemctl restart kubelet.service
```

Step 8: Joining Worker Nodes to the Kubernetes Cluster

With the `kubernetes-master` node up and the pod network ready, we can join our worker nodes to the cluster. In this tutorial, we only have one worker node, so we will be working with that. If you have more worker nodes, you can always follow the same steps as we will explain below to join the cluster.

First, log into your worker node on a separate terminal session. You will use your `kubeadm join` command that was shown in your terminal when we initialized the master node in Step 6. Execute the command:

```
sudo kubeadm join 127.0.0.188:6443 --token u81y02.91gqwkxx6rnhnnly --discovery-token
```

You should see similar output like the screenshot below when it completes joining the cluster:

```
cloudsigma@kubernetes-worker:~$ sudo kubeadm join [redacted]:6443 --token u81y02.91gqwkxx6rnhnnly --discovery-token-ca
a02c1ba580f4af9f3ad4cc37b24f189db34d6e3fe95c2d
[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.
```

Once the joining process completes, switch the master node terminal and execute the following command to confirm that your worker node has joined the cluster:

```
kubectl get nodes
```

In the screenshot from the output of the command above, we can see that the worker node has joined the cluster:

```
cloudsigma@kubernetes-master:~$ kubectl get nodes
NAME                STATUS   ROLES                  AGE   VERSION
kubernetes-master   Ready    control-plane,master   41m   v1.22.2
kubernetes-worker   Ready    <none>                 85s   v1.22.2
cloudsigma@kubernetes-master:~$
```

Step 9: Deploying an Application to the Kubernetes Cluster

At this point, you have successfully set up a Kubernetes cluster. Let's make the cluster usable by deploying a service to it. Nginx is a popular web server boasting incredible speeds even with thousands of connections. We will deploy the Nginx webserver to the cluster to prove that you can use this setup in a real-life application.

Execute the following command on the master node to create a Kubernetes deployment (https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) for Nginx:

```
kubectl create deployment nginx --image=nginx
```

You can view the created `deployment` by using the `describe deployment` command:

```
kubectl describe deployment nginx
```

```
cloudsigma@kubernetes-master:~$ kubectl describe deployment nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Tue, 19 Oct 2021 05:12:33 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
```


To make the `nginx service` accessible via the internet, run the following command:

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

```
cloudsigma@kubernetes-master:~$ kubectl create service nodeport nginx --tcp=80:80
service/nginx created
cloudsigma@kubernetes-master:~$
```

The command above will create a public-facing service for the Nginx deployment. This being a nodeport (https://kubernetes.io/docs/concepts/services-networking/service/) deployment, Kubernetes assigns the service a port in the range of `32000+`.

You can get the current services by issuing the command:

```
kubectl get svc
```

```
cloudsigma@kubernetes-master:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        59m
nginx        NodePort    10.97.107.34   <none>        80:32264/TCP   80s
cloudsigma@kubernetes-master:~$
```

You can see that our assigned port is `32264`. Take note of the port displayed in your terminal to use in the next step.

To verify that the Nginx service deployment is successful, issue a `curl` call to the worker node from the master. Replace your worker node IP and the port you got from the above command:

```
curl your-kubernetes-worker-ip:32264
```

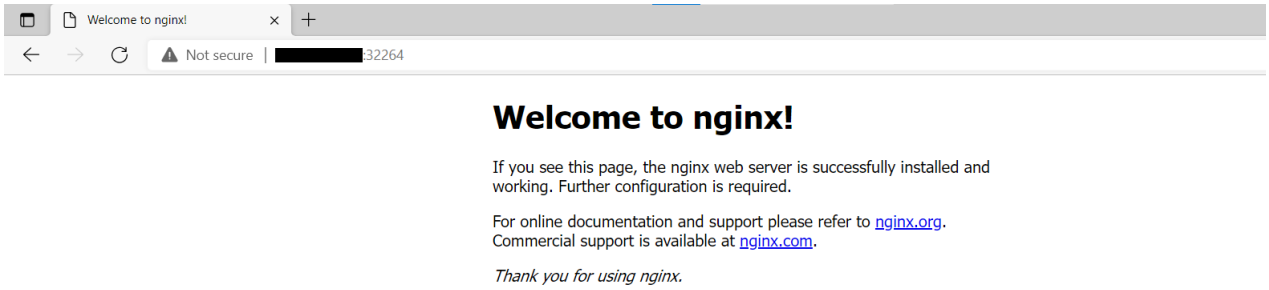
You should see the output of the default Nginx `index.html`:

```
cloudsigma@kubernetes-master:~$ curl [redacted]:32264
<!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>
cloudsigma@kubernetes-master:~$
```

Optionally, you can visit the worker node IP address and port combination in your browser and view the default Nginx index page:



You can `delete` a deployment by specifying the name of the deployment. For example, this command will delete our deployment:

```
1 kubectl delete deployment nginx
```

We have now successfully tested our cluster!

Conclusion

In this tutorial, you have learned how to install a Kubernetes cluster on Ubuntu 20.04. You set up a cluster consisting of a master and worker node. You were able to install the Kubernetes toolset, created a pod network, and joined the worker node to the master node. We also tested our concept by doing a basic deployment of an Nginx webserver to the cluster. This should work as a foundation to working with Kubernetes clusters on Ubuntu.

While we only used one worker node, you can extend your cluster with as many nodes as you wish. If you would like to get deeper into DevOps with automation tools like Ansible (https://www.ansible.com/), we have a tutorial that delves into provisioning Kubernetes cluster deployments with Ansible and Kubeadm (https://www.cloudsigma.com/how-to-create-a-kubernetes-cluster-using-kubeadm-on-ubuntu-18-04/), check it out. If you want to learn how to deploy a PHP application on a Kubernetes cluster check this tutorial (https://www.cloudsigma.com/deploy-a-php-application-on-a-kubernetes-cluster-with-ubuntu-18-04/).

Happy Computing!

About | Latest

About Pranay Kapgate

Software Engineer | Open Source Software aficionado

CLOUD (HTTPS://WWW.CLOUDSIGMA.COM/TAG/CLOUD/)

CONTAINERIZATION (HTTPS://WWW.CLOUDSIGMA.COM/TAG/CONTAINERIZATION/)

CONTAINERIZED APPLICATIONS (HTTPS://WWW.CLOUDSIGMA.COM/TAG/CONTAINERIZED-APPLICATIONS/)

CONTAINERS (HTTPS://WWW.CLOUDSIGMA.COM/TAG/CONTAINERS/)

DEVOPS (HTTPS://WWW.CLOUDSIGMA.COM/TAG/DEVOPS/) DOCKER (HTTPS://WWW.CLOUDSIGMA.COM/TAG/DOCKER/)

INSTALL KUBERNETES (HTTPS://WWW.CLOUDSIGMA.COM/TAG/INSTALL-KUBERNETES/)

KUBERNETES (HTTPS://WWW.CLOUDSIGMA.COM/TAG/KUBERNETES/)

KUBERNETES CLUSTER (HTTPS://WWW.CLOUDSIGMA.COM/TAG/KUBERNETES-CLUSTER/)

KUBERNETES CLUSTERS (HTTPS://WWW.CLOUDSIGMA.COM/TAG/KUBERNETES-CLUSTERS/)

MASTER NODE (HTTPS://WWW.CLOUDSIGMA.COM/TAG/MASTER-NODE/)

NGINX (HTTPS://WWW.CLOUDSIGMA.COM/TAG/NGINX/) PAAS (HTTPS://WWW.CLOUDSIGMA.COM/TAG/PAAS/)

PLATFORM AS A SERVICE (HTTPS://WWW.CLOUDSIGMA.COM/TAG/PLATFORM-AS-A-SERVICE/)

TUTORIAL (HTTPS://WWW.CLOUDSIGMA.COM/TAG/TUTORIAL/)

UBUNTU (HTTPS://WWW.CLOUDSIGMA.COM/TAG/UBUNTU/)

WORKER NODE (HTTPS://WWW.CLOUDSIGMA.COM/TAG/WORKER-NODE/)

(https://marketplace.intel.com/s/partner/a5S3b0000002kE0EAI/cloudsigma?language=en_US&wapkw=cloudsigma)

(https://www.cloudsigma.com/cloudsigma-with-hpe-delivers-innovative-cloud-services-around-the-globe/)

ISO 27001 CERTIFIED Information Security (https://www.cloudsigma.com/iso-27001-information-security-certified-cloud/)

ISO 27017 CERTIFIED Cloud Security (https://www.cloudsigma.com/iso-27017-cloud-security-certified-cloud/)

ISO 27018 CERTIFIED Privacy Protection (https://www.cloudsigma.com/iso-27018-privacy-protection-certified-cloud/)

GÉANT preferred cloud partner (https://www.cloudsigma.com/geant-preferred-cloud-partner//)

EU GDPR COMPLIANT (https://www.cloudsigma.com/eu-gdpr-compliant-cloud/)

PCI DSS COMPLIANT (https://www.cloudsigma.com/pci-dss-compliant-cloud/)

STAR LEVEL ONE: SELF-ASSESSMENT SECURITY·TRUST ASSURANCE & RISK (https://www.cloudsigma.com/star-level-1-registered-public-cloud/)

EBA The European Banking Authority (https://www.cloudsigma.com/eba-recommendations-compliant-cloud/)

Proud member of

gaia-x (https://www.gaia-x.eu/)

DE-CIX (https://www.de-cix.net/)

eco (https://international.eco.de/)

RIPE NCC RIPE NETWORK COORDINATION CENTRE (https://www.ripe.net)

ARIN American Registry for Internet Numbers (https://www.arin.net/)

OCRE (https://www.ocre-project.eu/)

First Name

Your first name

Email address:

Your email address

SUBSCRIBE TO OUR LATEST BLOGS