Kubernetes on Ubuntu OS cluster deployment

NAME: SUTRAPU SHARAN KUMAR REDDY

SRN: R18BS055

CLASS: BSc(R) CC & BD

PROJECT NAME: Kubernetes on Ubuntu OS cluster deployment

Introduction:

Kubernetes is an open-source container management system for automating deployment, scaling, and management of containerized applications. Kubernetes is absolutely free and open source, so it gives you the freedom to take advantage of on-premises, hybrid, or public cloud infrastructure, letting you effortlessly move workloads to where it matters to you. Kubernetes was originally designed by Google and maintained by the Cloud Native Computing Foundation. Kubernetes is quickly becoming the new standard for deploying and managing software in the cloud, and provides a platform for automating deployment, scaling, and operations of application containers across clusters of hosts. It helps you to run containerized applications whenever you want. Kubernetes follows the master-slave architecture, where it has a master that provides centralized control for an all agents. Kubernetes has several components, including: etcd, flannel, kube-apiserver, kube-controller-manager, kube-scheduler, kubelet, kube-proxy, docker and much more.

we are going to set up multi-node Kubernetes Cluster on Ubuntu 16.04 VM's

Requirements:

Two fresh Virtual Machines on VM ware with Ubuntu 16.04 ISO installed.

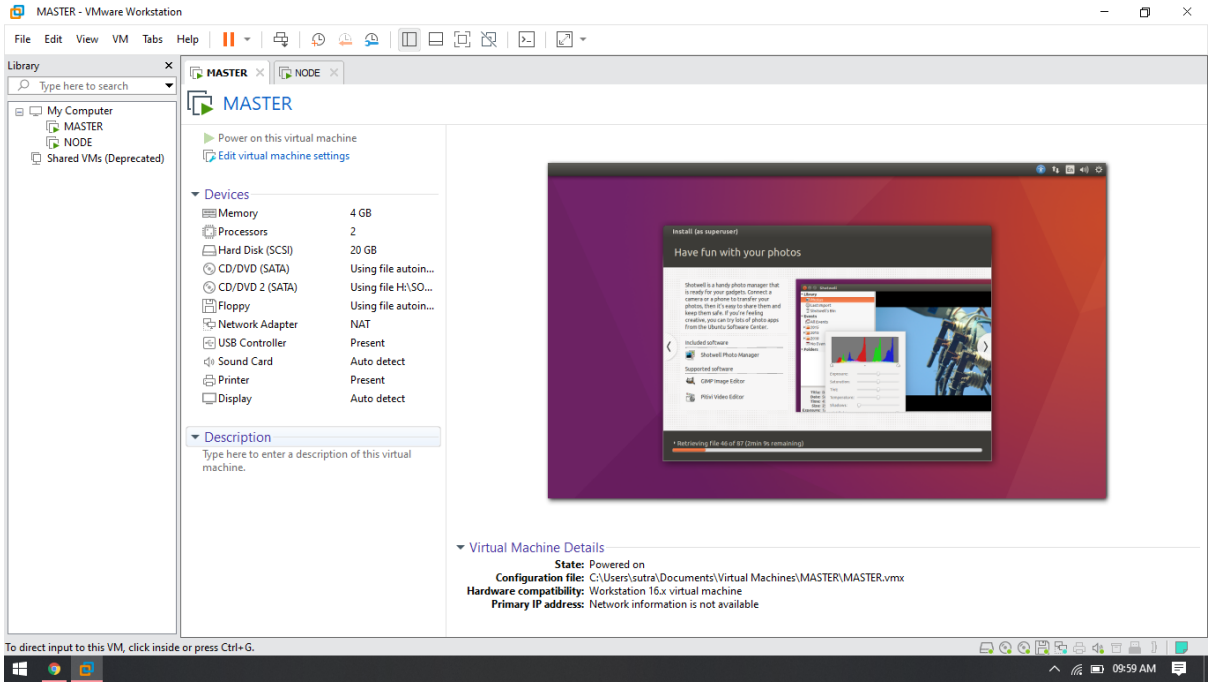
A static IP address is configured on the both Virtual Machines (Master, Slave).

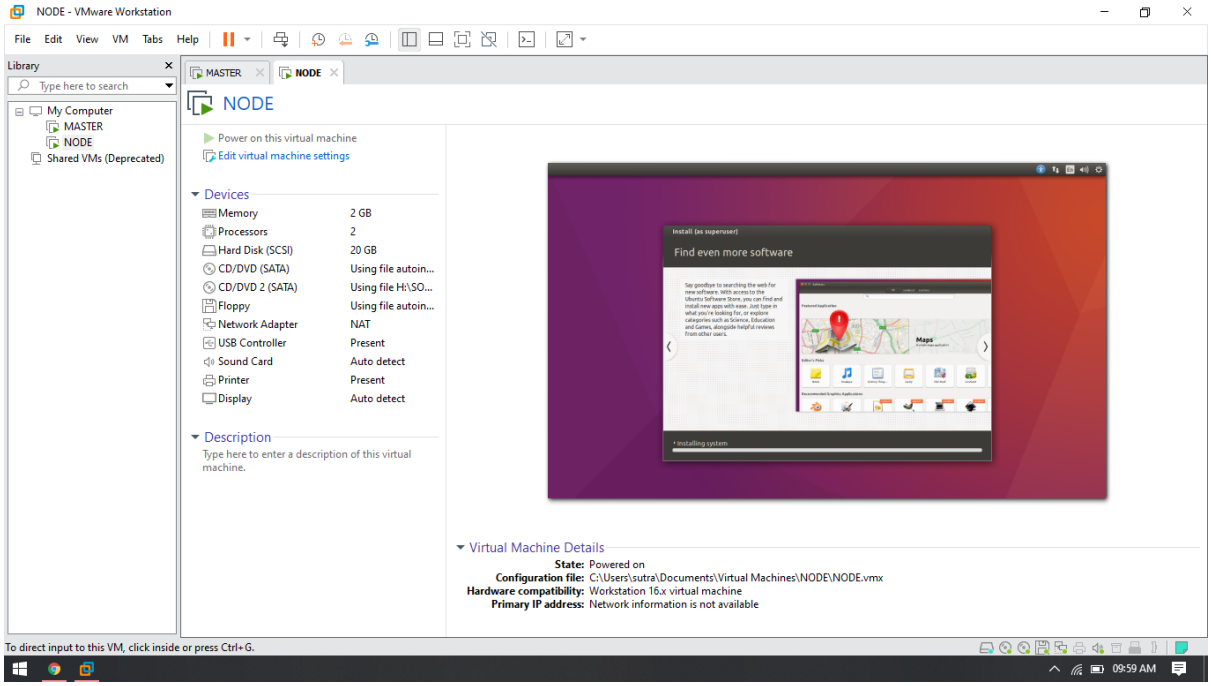
Minimum 2GB RAM per instance and 2 Core CPU Is Required.

A Root password is set up on each Virtual Machines.

Launch Virtual Machines on the VM Ware:

First, open VM ware Application. Create a new VM Ware Virtual Machines, choosing Ubuntu 16.04 as the operating system with at least 2GB RAM and 2 core CPU. Open to your VM Machines and log in as the root user.





Once you are logged into your Ubuntu 16.04 instance, run the following command to become a root user.

$ sudo su

For Password give your root Password.

Then run the following command to update your base system with the latest available packages.

$ sudo apt-get update

Then run the following command to install VIM text editor to edit the configuration files.

$ sudo apt-get install vim –y

Getting Started:

Before starting, you will need to configure hosts file and hostname on each Machines, so each server can communicate with each other using the hostname.

First, open /etc/hosts file on the first Machine:

$ vim /etc/hosts

Add the following lines:

192.168.248.150 master-node

192.168.248.151 slave-node

Save and close the file when you are finished, then setup hostname by running the following command:

$ hostnamectl set-hostname master-node

Next, open /etc/hosts file on the second server:

$ vim /etc/hosts

Add the following lines:

192.168.248.150 master-node

192.168.248.151 slave-node

Save and close the file when you are finished, then setup hostname by running the following command:

$ hostnamectl set-hostname slave-node

Next, you will need to disable swap memory on each server. Because kubelets do not support swap memory and will not work if swap is active or even present in your /etc/fstab file.

You can disable swap memory usage with the following command:

$ swapoff -a

You can disable this permanent by commenting out the swap file in /etc/fstab:

$ vim /etc/fstab

Comment out the swap line as shown below:

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
# / was on /dev/sda1 during installation
UUID=5b6dc434-fb24-4aa2-a02d-e5ecef72bf2a /               ext4    errors=remount-ro 0       1
# swap was on /dev/sda5 during installation
#UUID=be1341b6-911c-48bc-96fb-e508940ff604 none            swap    sw              0       0
/dev/fd0        /media/floppy0  auto    rw,user,noauto,exec,utf8 0       0
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                                                          11,2          All
```

Save and close the file, when you are finished.

Install Docker:

Before starting, you will need to install Docker on both the master and slave Virtual Machines. By default, the latest version of the Docker is not available in Ubuntu 16.04 repository, so you will need to add Docker repository to your system.

First, install required packages to add Docker repository with the following command:

$ apt-get install apt-transport-https ca-certificates curl software-properties-common –y

Next, download and add Docker's GPG key with the following command:

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

Next, add Docker repository with the following command:

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

Next, update the repository and install Docker with the following command:

$ apt-get update -y

$ apt-get install docker-ce -y

Install Kubernetes:

Next, you will need to install kubeadm, kubectl and kubelet on both the server. First, download and GPG key with the following command:

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

Next, add Kubernetes repository with the following command:

$ echo 'deb http://apt.kubernetes.io/ kubernetes-xenial main' | sudo tee /etc/apt/sources.list.d/kubernetes.list

Finally, update the repository and install Kubernetes with the following command:

$ apt-get update –y

$ apt-get install kubelet kubeadm kubectl –y

Configure Master Node:

All the required packages are installed on both servers. Now, it's time to configure Kubernetes Master Node.

First, initialize your cluster using its private IP address with the following command:

$ kubeadm init --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=192.168.248.150

You should see the following output:

```
root@ubuntu: /home/master

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 as
 root:

kubeadm join 192.168.248.150:6443 --token mvdhdj.v4fabwuu3x1oh2h5 \
        --discovery-token-ca-cert-hash sha256:2aad75b48a8f360d8efe81961fce727724
2624986ccbeef180f41c68ce1c2216
root@ubuntu:/home/master#
```

Note : Note down the token from the above output. This will be used to join Slave Node to the Master Node in the next step. (You can use this command also to see the token later):

$ kubeadm token create --print-join-command

kubeadm join 192.168.248.150:6443 --token t4dbpf.tx2ltxpn86lz45zn --discovery-token-ca-cert-hash sha256:2aad75b48a8f360d8efe81961fce7277242624986ccbeef180f41c68ce1c2216

Next, you will need to run the following command to configure kubectl tool:

$ mkdir -p $HOME/.kube

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

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

Next, check the status of the Master Node by running the following command:

$ kubectl get nodes

You should see the following output:

```
root@ubuntu: /home/master
root@ubuntu:/home/master# kubectl get nodes
NAME          STATUS     ROLES                  AGE   VERSION
master-node   NotReady   control-plane,master   18m   v1.21.1
root@ubuntu:/home/master#
```

In the above output, you should see that Master Node is listed as not ready. Because the cluster does not have a Container Networking Interface (CNI).

The below command can be run on the leader node to install the CNI plugin:

$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

Make sure CNI was deployed correctly by running the following command:

$ kubectl get pods --all-namespaces

You should see the following output:

```
root@ubuntu:/home/master# kubectl get pods --all-namespaces
NAMESPACE     NAME                                     READY   STATUS             RESTARTS   AGE
kube-system   coredns-558bd4d5db-4spw5                 0/1     CrashLoopBackOff   3          23m
kube-system   coredns-558bd4d5db-6tvzk                 0/1     CrashLoopBackOff   3          23m
kube-system   etcd-master-node                         1/1     Running            0          23m
kube-system   kube-apiserver-master-node               1/1     Running            0          23m
kube-system   kube-controller-manager-master-node      1/1     Running            0          23m
kube-system   kube-flannel-ds-l9dsw                    1/1     Running            0          84s
kube-system   kube-proxy-wpnzh                         1/1     Running            0          23m
kube-system   kube-scheduler-master-node               1/1     Running            0          23m
root@ubuntu:/home/master#
```

Now, run the kubectl get nodes command again, and you should see the Master Node is now listed as Ready.

$ kubectl get nodes

Output:

```
root@ubuntu: /home/master
root@ubuntu:/home/master# kubectl get nodes
NAME          STATUS   ROLES                  AGE   VERSION
master-node   Ready    control-plane,master   25m   v1.21.1
root@ubuntu:/home/master#
```

Add Slave Node to the Kubernetes Cluster:

Next, you will need to log in to the Slave Node and add it to the Cluster. Remember the join command in the output from the Master Node initialization command and issue it on the Slave Node as shown below:

$ kubeadm join 192.168.248.150:6443 --token t4dbpf.tx2ltxpn86lz45zn --discovery-token-ca-cert-hash sha256:2aad75b48a8f360d8efe81961fce7277242624986ccbeef180f41c68ce1c2216

Once the Node is joined successfully, you should see the following output:

```
root@ubuntu: /home/node
root@ubuntu:/home/node# kubeadm join 192.168.248.150:6443 --token t4dbpf.tx2ltxp
n86lz45zn --discovery-token-ca-cert-hash sha256:2aad75b48a8f360d8efe81961fce7277
242624986ccbeef180f41c68ce1c2216
[preflight] Running pre-flight checks
        [WARNING IsDockerSystemdCheck]: detected "cgroupfs" as the Docker cgroup
 driver. The recommended driver is "systemd". Please follow the guide at https:/
/kubernetes.io/docs/setup/cri/
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system g
et cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.y
aml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/ku
belet/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.

root@ubuntu:/home/node#
```

Now, go back to the Master Node and issue the command kubectl get nodes to see that the slave node is now ready:

$ kubectl get nodes

Output:

```
root@ubuntu: /home/master
root@ubuntu:/home/master# kubectl get nodes
NAME          STATUS   ROLES                  AGE     VERSION
master-node   Ready    control-plane,master   30m     v1.21.1
slave-node    Ready    <none>                 2m48s   v1.21.1
root@ubuntu:/home/master#
```

Deploy the Nginx container to the Cluster:

Kubernetes Cluster is now ready, it's time to deploy the Nginx container.

On the Master Node, run the following command to create a Nginx deployment:

$ kubectl create deployment nginx --image nginx

Output:

deployment.apps/nginx created

You can expose out the deployments with the following command:

$ kubectl expose deployment nginx --type NodePort --port 80

Output:

service/nginx exposed

You can list out the deployments with the following command:

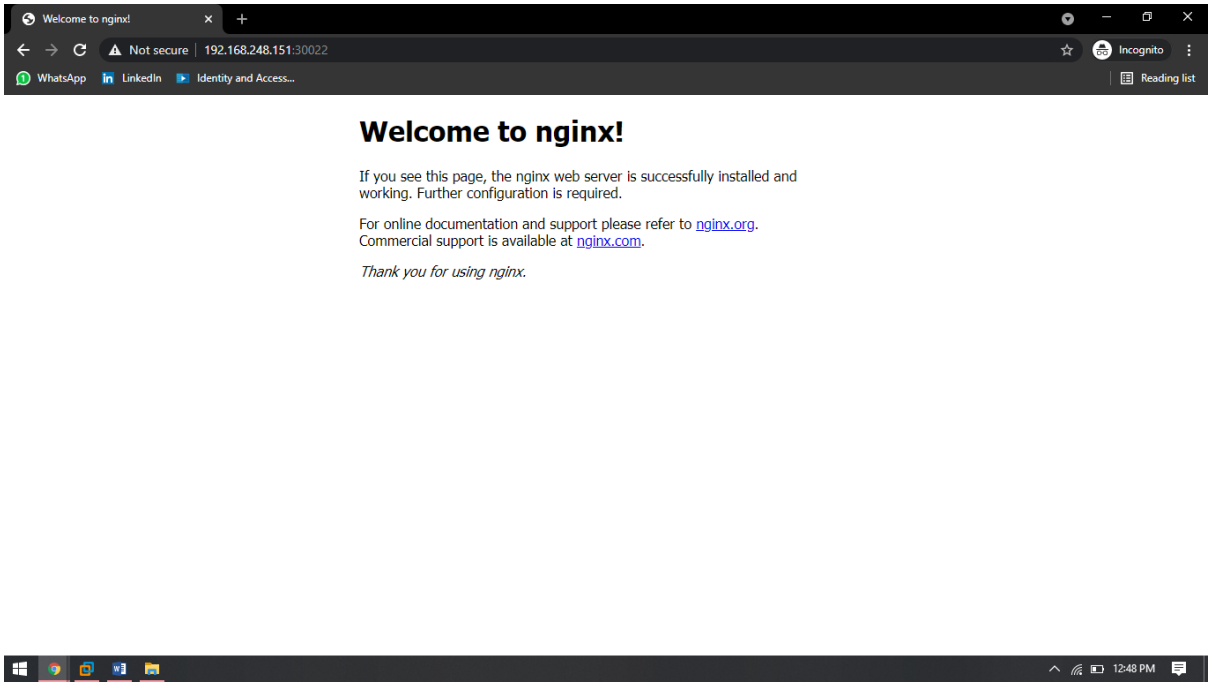
$ kubectl get all

Output:

```
root@ubuntu: /home/master
root@ubuntu:/home/master# kubectl get all
NAME                          READY   STATUS    RESTARTS   AGE
pod/nginx-6799fc88d8-7tm27    1/1     Running   0          3m23s

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        39m
service/nginx        NodePort    10.108.251.31   <none>        80:30022/TCP   82s

NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx   1/1     1            1           3m23s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-6799fc88d8   1         1         1       3m23s
root@ubuntu:/home/master#
```

Now go to browser and type the node IP address with the port number which you can find in the above output.

192.168.248.151: 30022


Welcome to nginx!
Not secure
192.168.248.151:30022
Incognito
WhatsApp
LinkedIn
Identity and Access...
Reading list
Welcome to nginx!
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.
For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.
Thank you for using nginx.
12:48 PM