Venkat Rahul Dantuluri – Lowe's Technical interview

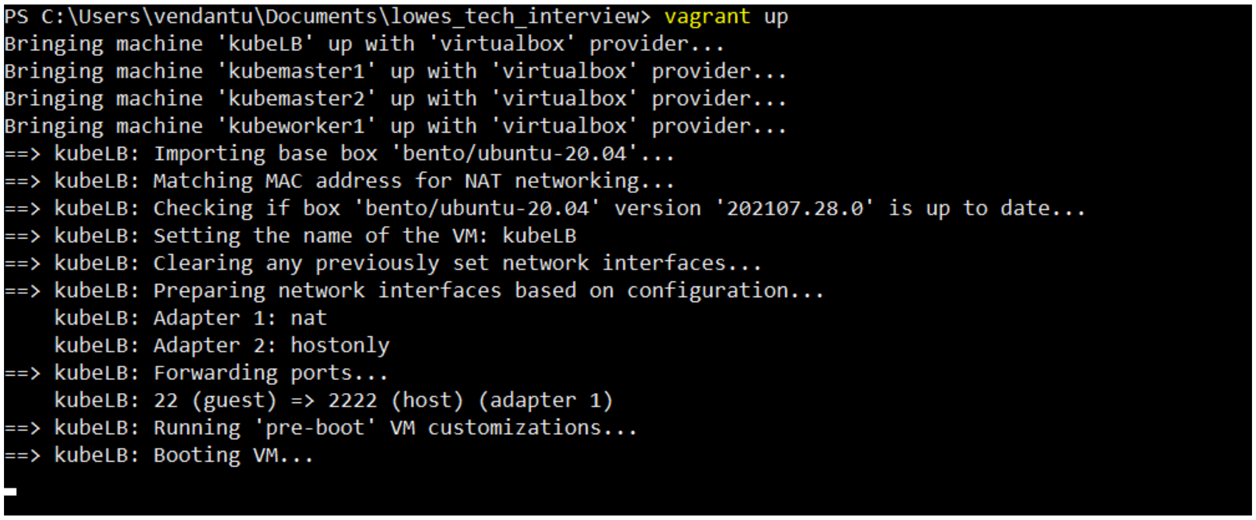
1) Create Kubernetes cluster with high availability of master nodes (if possible) on local.

   To implement this Kubernetes cluster on local, I am using virtual box and vagrant to spin up 4 virtual machines, on local, which have Ubuntu 20.04 installed on them. To demonstrate the high availability of master node, I am using 2 VMs to deploy 2 master nodes. One of the remaining nodes I am using as a worker node and another I am using as a Load balancer to load balance between the 2 master nodes. We can always increase the number of master nodes and worker nodes to a greater number but on a 8 core machine, like the one I am using, this is the maximum I could increase to.
   I am using HAProxy to act as a load balancer.

   Code is present in following GitHub repository:
   https://github.com/RahulDV/kubernetes

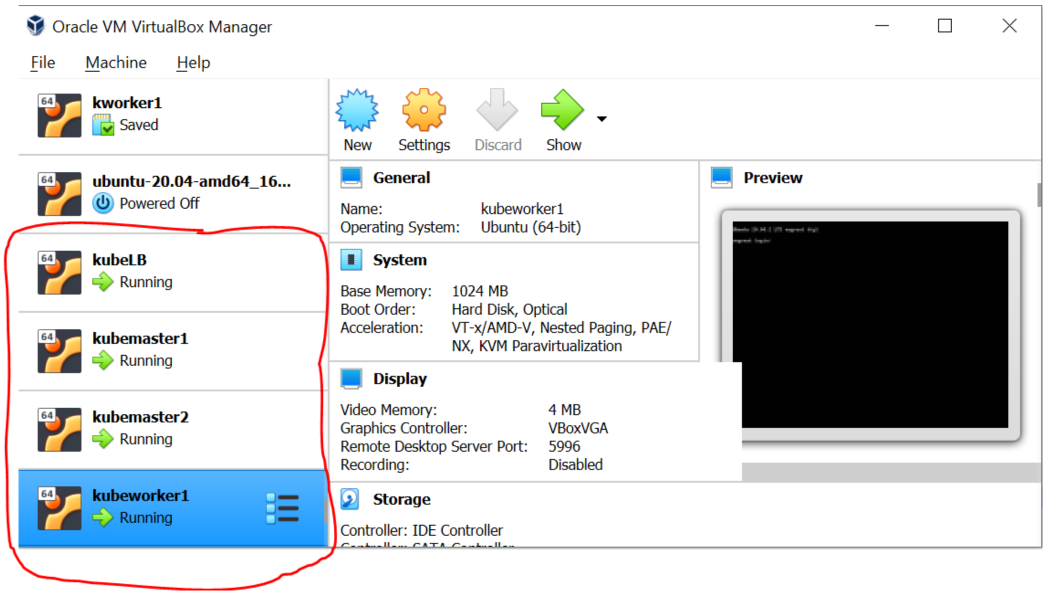
   Here is the step by step commands and execution of achieving usecase 1.

   a) Git clone the code from repo above to your local folder and run vagrant up



```
PS C:\Users\vendantu\Documents\lowes_tech_interview> vagrant up
Bringing machine 'kubeLB' up with 'virtualbox' provider...
Bringing machine 'kubemaster1' up with 'virtualbox' provider...
Bringing machine 'kubemaster2' up with 'virtualbox' provider...
Bringing machine 'kubeworker1' up with 'virtualbox' provider...
==> kubeLB: Importing base box 'bento/ubuntu-20.04'...
==> kubeLB: Matching MAC address for NAT networking...
==> kubeLB: Checking if box 'bento/ubuntu-20.04' version '202107.28.0' is up to date...
==> kubeLB: Setting the name of the VM: kubeLB
==> kubeLB: Clearing any previously set network interfaces...
==> kubeLB: Preparing network interfaces based on configuration...
    kubeLB: Adapter 1: nat
    kubeLB: Adapter 2: hostonly
==> kubeLB: Forwarding ports...
    kubeLB: 22 (guest) => 2222 (host) (adapter 1)
==> kubeLB: Running 'pre-boot' VM customizations...
==> kubeLB: Booting VM...
```

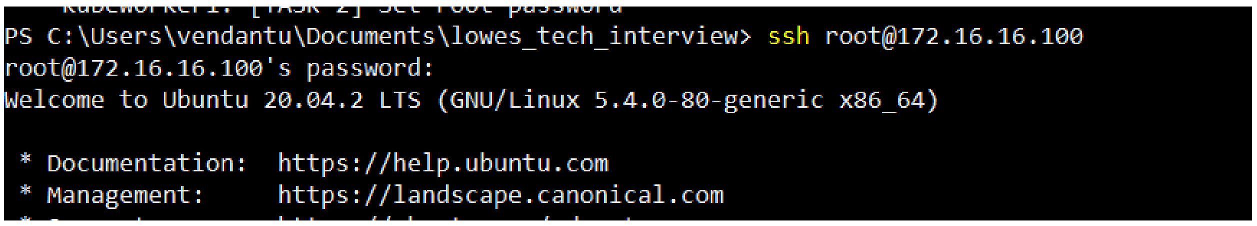
   Once all machines are up and running you can ssh to each with pwd: kubeadmin. This is set in bootstrap.sh
   As per the vagrant config file we allocated following private IPs for each VM:
   kubeLB – Contains the load balancer (HAProxy) – 172.16.16.100
   kubemaster1 – containing master node 1 – 172.16.16.101
   Kubemaster2 – containing master node 2 – 172.16.16.102
   Kubeworker1 – containing worker node 1 – 172.16.16.201



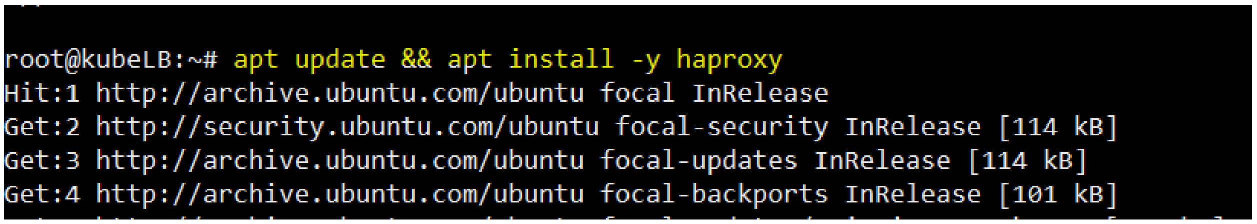
kworker1	Saved
ubuntu-20.04-amd64_16...	Powered Off
kubeLB	Running
kubemaster1	Running
kubemaster2	Running
kubeworker1	Running

```
General
Name:               kubeworker1
Operating System:   Ubuntu (64-bit)

System
Base Memory:    1024 MB
Boot Order:     Hard Disk, Optical
Acceleration:   VT-x/AMD-V, Nested Paging, PAE/
                NX, KVM Paravirtualization

Display
Video Memory:                4 MB
Graphics Controller:         VBoxVGA
Remote Desktop Server Port:  5996
Recording:                   Disabled

Storage
Controller: IDE Controller
```

b) Install HAProxy and configure load balancer to balance the 2 masters in round robin fashion: Follow commands in loadBalancer.sh

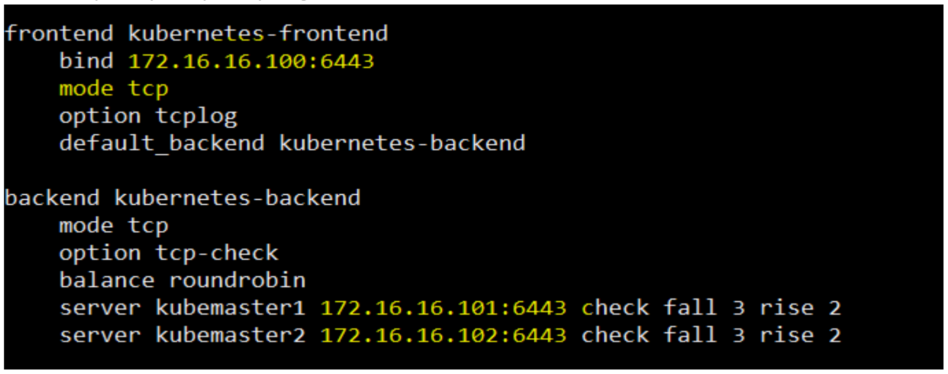


```
PS C:\Users\vendantu\Documents\lowes_tech_interview> ssh root@172.16.16.100
root@172.16.16.100's password:
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-80-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
```



```
root@kubeLB:~# apt update && apt install -y haproxy
Hit:1 http://archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
```

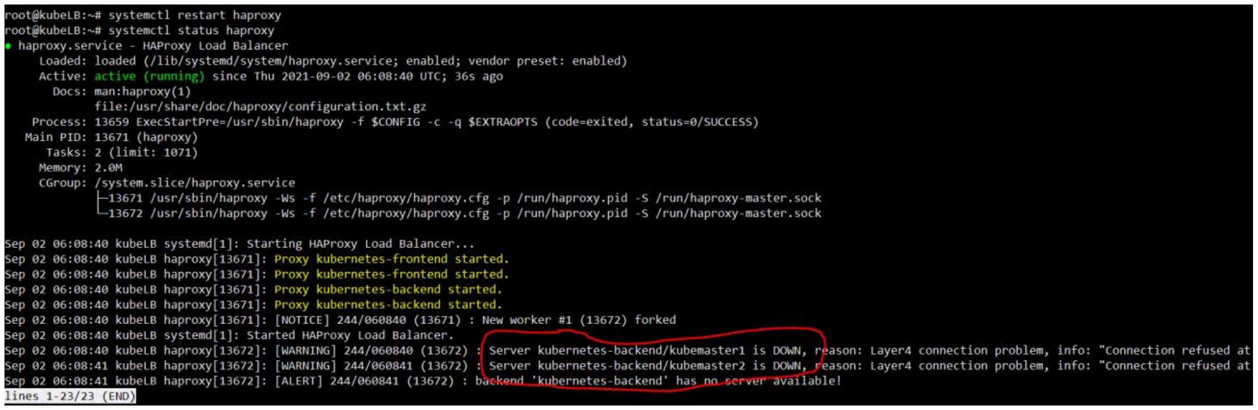
Using VIM editor configure the backend and frontend of HAproxy:File name and location: /etc/haproxy/haproxy.cfg



```
frontend kubernetes-frontend
    bind 172.16.16.100:6443
    mode tcp
    option tcplog
    default_backend kubernetes-backend

backend kubernetes-backend
    mode tcp
    option tcp-check
    balance roundrobin
    server kubemaster1 172.16.16.101:6443 check fall 3 rise 2
    server kubemaster2 172.16.16.102:6443 check fall 3 rise 2
```

Restart proxy with command: systemctl restart haproxy

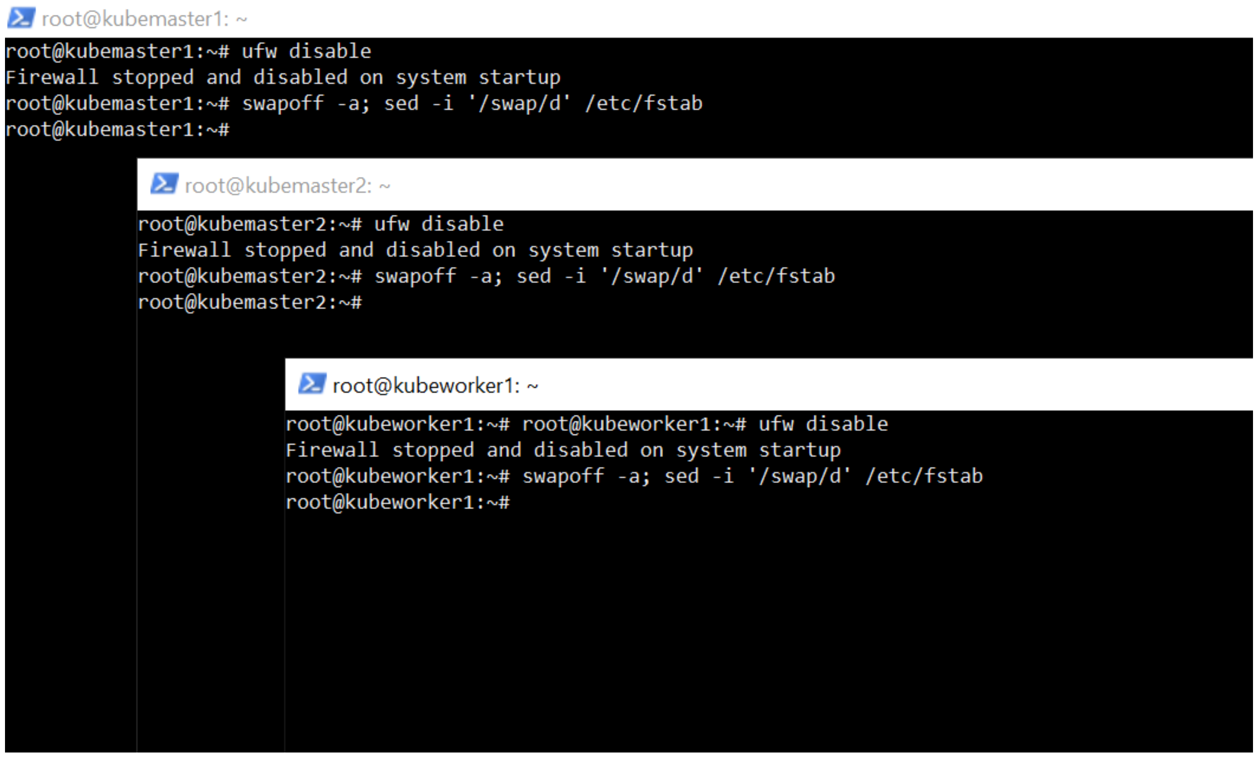
Check the status of HAProxy



```
root@kubeLB:~# systemctl restart haproxy
root@kubeLB:~# systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
     Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2021-09-02 06:08:40 UTC; 36s ago
       Docs: man:haproxy(1)
             file:/usr/share/doc/haproxy/configuration.txt.gz
    Process: 13659 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q $EXTRAOPTS (code=exited, status=0/SUCCESS)
   Main PID: 13671 (haproxy)
      Tasks: 2 (limit: 1071)
     Memory: 2.0M
     CGroup: /system.slice/haproxy.service
             ├─13671 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy-master.sock
             └─13672 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -S /run/haproxy-master.sock

Sep 02 06:08:40 kubeLB systemd[1]: Starting HAProxy Load Balancer...
Sep 02 06:08:40 kubeLB haproxy[13671]: Proxy kubernetes-frontend started.
Sep 02 06:08:40 kubeLB haproxy[13671]: Proxy kubernetes-frontend started.
Sep 02 06:08:40 kubeLB haproxy[13671]: Proxy kubernetes-backend started.
Sep 02 06:08:40 kubeLB haproxy[13671]: Proxy kubernetes-backend started.
Sep 02 06:08:40 kubeLB haproxy[13671]: [NOTICE] 244/060840 (13671) : New worker #1 (13672) forked
Sep 02 06:08:40 kubeLB systemd[1]: Started HAProxy Load Balancer.
Sep 02 06:08:40 kubeLB haproxy[13672]: [WARNING] 244/060840 (13672) : Server kubernetes-backend/kubemaster1 is DOWN, reason: Layer4 connection problem, info: "Connection refused at
Sep 02 06:08:41 kubeLB haproxy[13672]: [WARNING] 244/060841 (13672) : Server kubernetes-backend/kubemaster2 is DOWN, reason: Layer4 connection problem, info: "Connection refused at
Sep 02 06:08:41 kubeLB haproxy[13672]: [ALERT] 244/060841 (13672) : backend 'kubernetes-backend' has no server available!
lines 1-23/23 (END)
```

c) Follow these steps to install Kubernetes on all the other 3 nodes. Repeat this step on all 3 nodes. Please follow commands in commonConfig.sh in git repo.

- Disable firewall and swap:



root@kubemaster1: ~

```
root@kubemaster1:~# ufw disable
Firewall stopped and disabled on system startup
root@kubemaster1:~# swapoff -a; sed -i '/swap/d' /etc/fstab
root@kubemaster1:~#
```

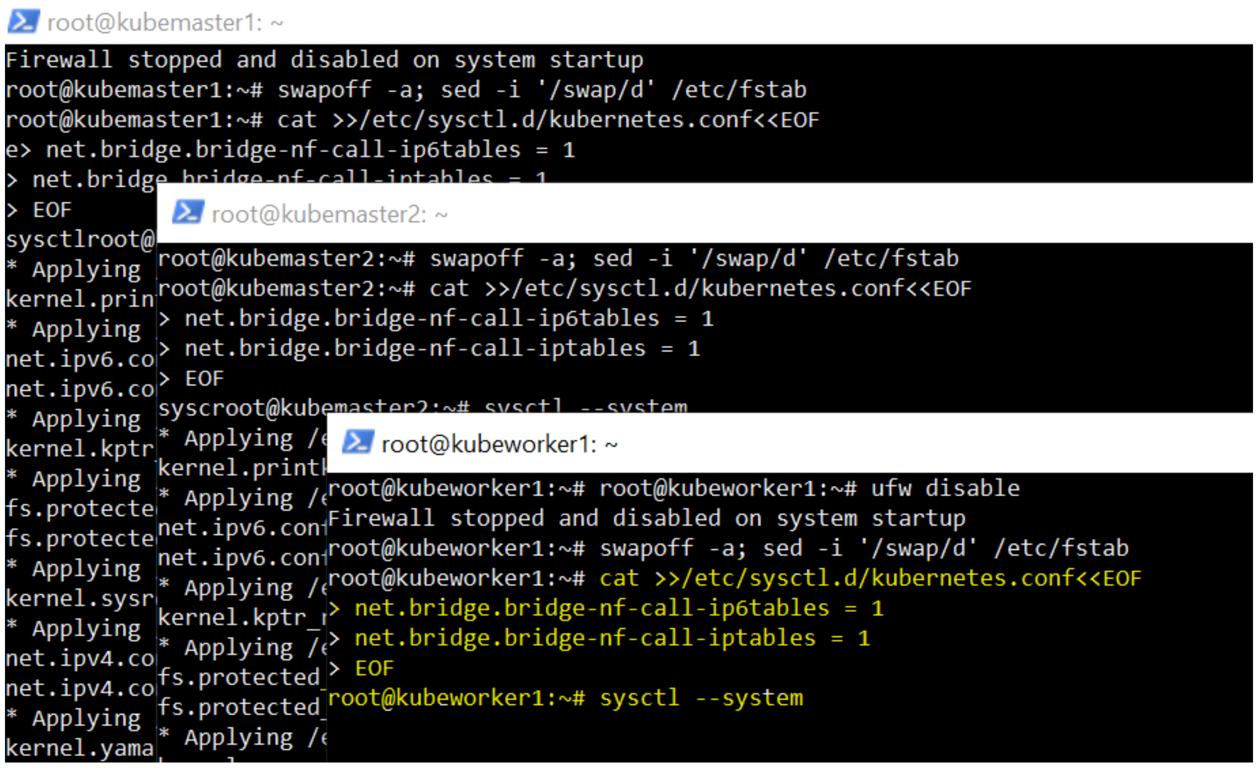
root@kubemaster2: ~

```
root@kubemaster2:~# ufw disable
Firewall stopped and disabled on system startup
root@kubemaster2:~# swapoff -a; sed -i '/swap/d' /etc/fstab
root@kubemaster2:~#
```

root@kubeworker1: ~

```
root@kubeworker1:~# root@kubeworker1:~# ufw disable
Firewall stopped and disabled on system startup
root@kubeworker1:~# swapoff -a; sed -i '/swap/d' /etc/fstab
root@kubeworker1:~#
```

- Because I using virtual machine, I need to perform a small network config update:

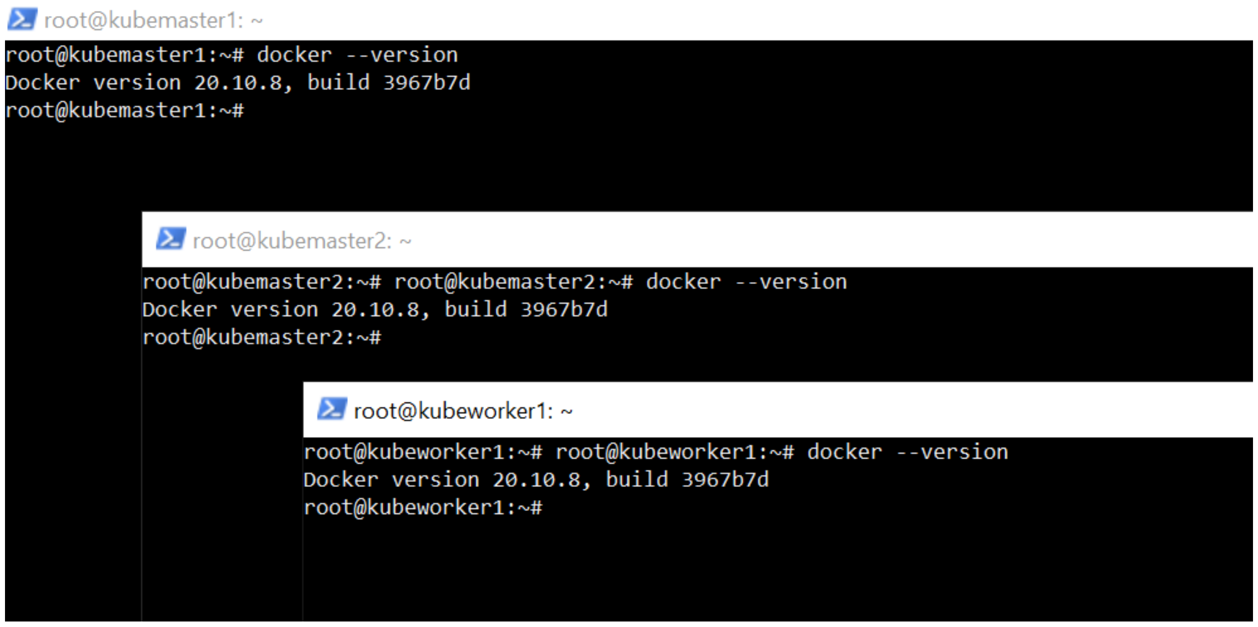


```
root@kubemaster1: ~
Firewall stopped and disabled on system startup
root@kubemaster1:~# swapoff -a; sed -i '/swap/d' /etc/fstab
root@kubemaster1:~# cat >>/etc/sysctl.d/kubernetes.conf<<EOF
e> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF
sysctlroot@
* Applying
kernel.prin
* Applying
net.ipv6.co
net.ipv6.co
* Applying
kernel.kptr
* Applying
fs.protecte
fs.protecte
* Applying
kernel.sysr
* Applying
net.ipv4.co
net.ipv4.co
* Applying
kernel.yama
```

```
root@kubemaster2: ~
root@kubemaster2:~# swapoff -a; sed -i '/swap/d' /etc/fstab
root@kubemaster2:~# cat >>/etc/sysctl.d/kubernetes.conf<<EOF
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF
syscroot@kubemaster2:~# sysctl --system
* Applying /e
kernel.printk
* Applying /e
net.ipv6.conf
net.ipv6.conf
* Applying /e
kernel.kptr_r
* Applying /e
fs.protected_
fs.protected_
* Applying /e
```

```
root@kubeworker1: ~
root@kubeworker1:~# root@kubeworker1:~# ufw disable
Firewall stopped and disabled on system startup
root@kubeworker1:~# swapoff -a; sed -i '/swap/d' /etc/fstab
root@kubeworker1:~# cat >>/etc/sysctl.d/kubernetes.conf<<EOF
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF
root@kubeworker1:~# sysctl --system
```

- Install Docker:

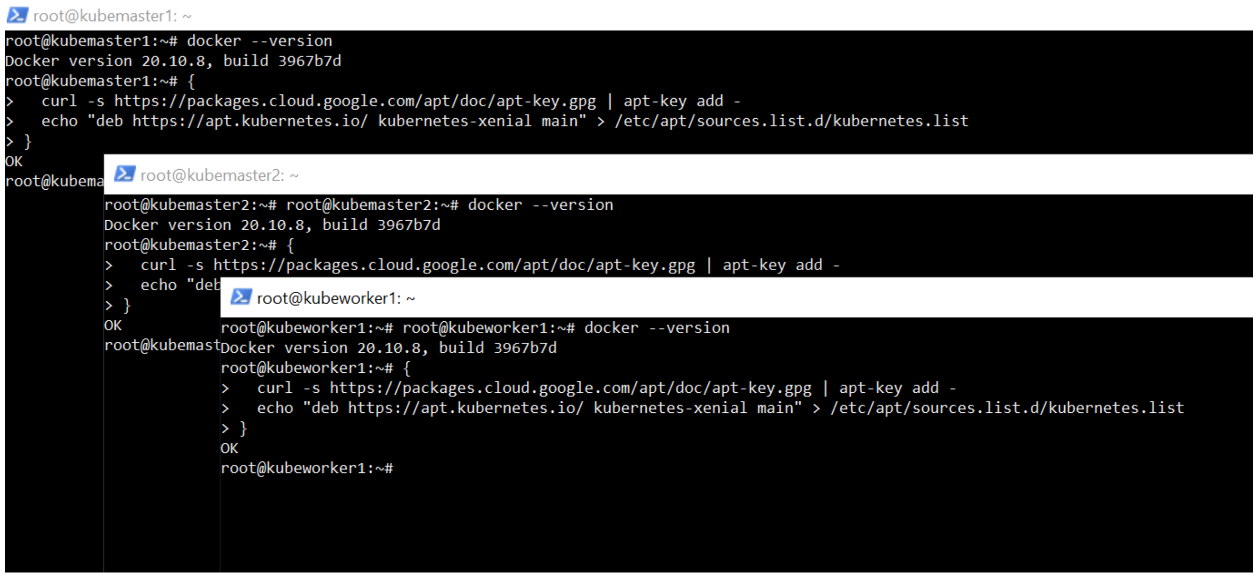


```
root@kubemaster1: ~
root@kubemaster1:~# docker --version
Docker version 20.10.8, build 3967b7d
root@kubemaster1:~#
```

```
root@kubemaster2: ~
root@kubemaster2:~# root@kubemaster2:~# docker --version
Docker version 20.10.8, build 3967b7d
root@kubemaster2:~#
```

```
root@kubeworker1: ~
root@kubeworker1:~# root@kubeworker1:~# docker --version
Docker version 20.10.8, build 3967b7d
root@kubeworker1:~#
```

- Install Kubernetes binaries:

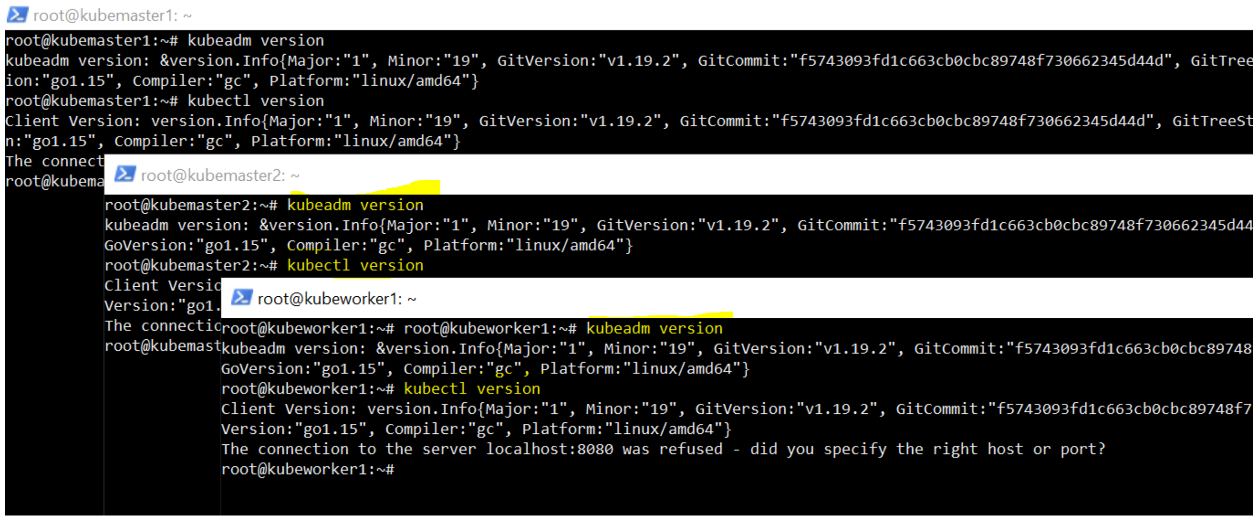


```
root@kubemaster1: ~
root@kubemaster1:~# docker --version
Docker version 20.10.8, build 3967b7d
root@kubemaster1:~# {
>   curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
>   echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" > /etc/apt/sources.list.d/kubernetes.list
> }
OK
root@kubema
```

```
root@kubemaster2: ~
root@kubemaster2:~# root@kubemaster2:~# docker --version
Docker version 20.10.8, build 3967b7d
root@kubemaster2:~# {
>   curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
>   echo "deb
> }
OK
root@kubemast
```

```
root@kubeworker1: ~
root@kubeworker1:~# root@kubeworker1:~# docker --version
Docker version 20.10.8, build 3967b7d
root@kubeworker1:~# {
>   curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
>   echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" > /etc/apt/sources.list.d/kubernetes.list
> }
OK
root@kubeworker1:~#
```

- Install kubeadm, kubectl and kubelet tools



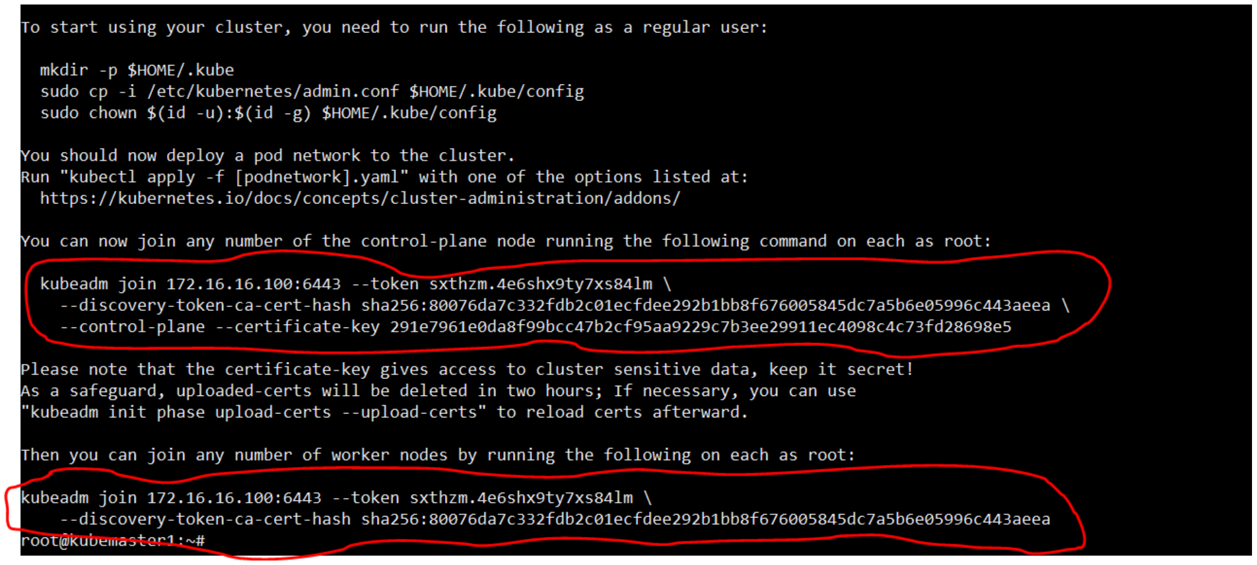
```
root@kubemaster1: ~
root@kubemaster1:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.2", GitCommit:"f5743093fd1c663cb0cbc89748f730662345d44d", GitTree
ion:"go1.15", Compiler:"gc", Platform:"linux/amd64"}
root@kubemaster1:~# kubectl version
Client Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.2", GitCommit:"f5743093fd1c663cb0cbc89748f730662345d44d", GitTreeSt
n:"go1.15", Compiler:"gc", Platform:"linux/amd64"}
The connect
root@kubema
```

```
root@kubemaster2: ~
root@kubemaster2:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.2", GitCommit:"f5743093fd1c663cb0cbc89748f730662345d44
GoVersion:"go1.15", Compiler:"gc", Platform:"linux/amd64"}
root@kubemaster2:~# kubectl version
Client Versio
Version:"go1.
The connectio
root@kubemast
```

```
root@kubeworker1: ~
root@kubeworker1:~# root@kubeworker1:~# kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.2", GitCommit:"f5743093fd1c663cb0cbc89748
GoVersion:"go1.15", Compiler:"gc", Platform:"linux/amd64"}
root@kubeworker1:~# kubectl version
Client Version: version.Info{Major:"1", Minor:"19", GitVersion:"v1.19.2", GitCommit:"f5743093fd1c663cb0cbc89748f7
Version:"go1.15", Compiler:"gc", Platform:"linux/amd64"}
The connection to the server localhost:8080 was refused - did you specify the right host or port?
root@kubeworker1:~#
```

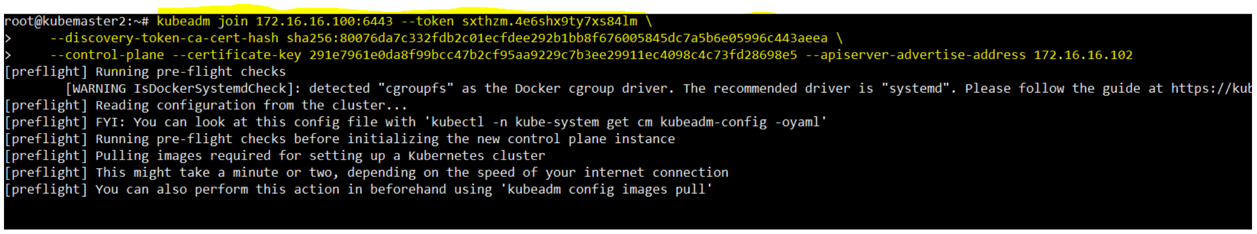
c) Now initialize k8s cluster on kubemaster1 node. All commands are present in onlyOnMaster01.sh. Once this is executed join commands will be available in log.

As we are running kubemaster2 in a VM we need to add the --apiserver-advertise-address argument to join command provided my the kubeadm init command execution.

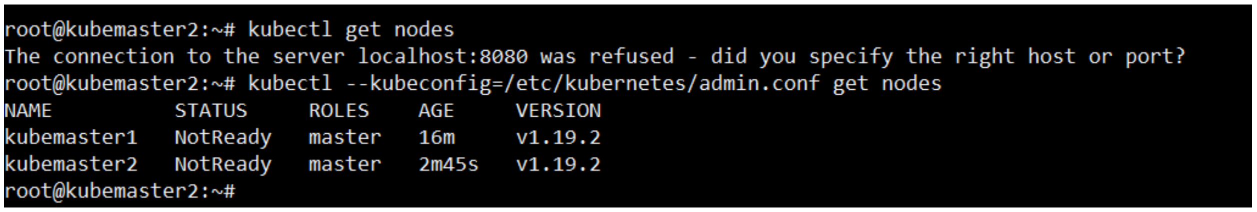


```
To start using your cluster, you need to run the following as a regular user:

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

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/

You can now join any number of the control-plane node running the following command on each as root:

  kubeadm join 172.16.16.100:6443 --token sxthzm.4e6shx9ty7xs84lm \
    --discovery-token-ca-cert-hash sha256:80076da7c332fdb2c01ecfdee292b1bb8f676005845dc7a5b6e05996c443aeea \
    --control-plane --certificate-key 291e7961e0da8f99bcc47b2cf95aa9229c7b3ee29911ec4098c4c73fd28698e5

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

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

kubeadm join 172.16.16.100:6443 --token sxthzm.4e6shx9ty7xs84lm \
    --discovery-token-ca-cert-hash sha256:80076da7c332fdb2c01ecfdee292b1bb8f676005845dc7a5b6e05996c443aeea
root@kubemaster1:~#
```

d) Run kubeadm join command on kubemaster2

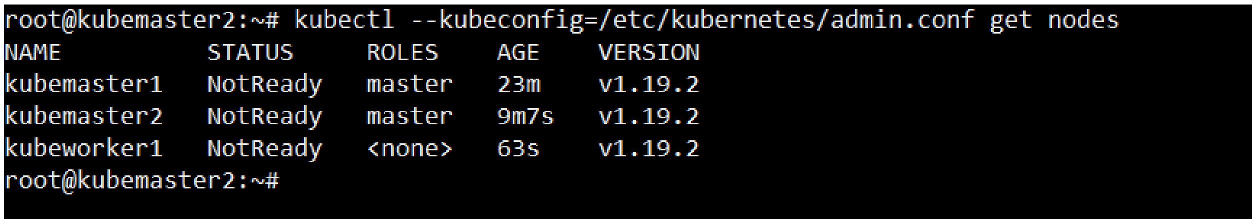


```
root@kubemaster2:~# kubeadm join 172.16.16.100:6443 --token sxthzm.4e6shx9ty7xs84lm \
>     --discovery-token-ca-cert-hash sha256:80076da7c332fdb2c01ecfdee292b1bb8f676005845dc7a5b6e05996c443aeea \
>     --control-plane --certificate-key 291e7961e0da8f99bcc47b2cf95aa9229c7b3ee29911ec4098c4c73fd28698e5 --apiserver-advertise-address 172.16.16.102
[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://kub
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
[preflight] Running pre-flight checks before initializing the new control plane instance
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
```



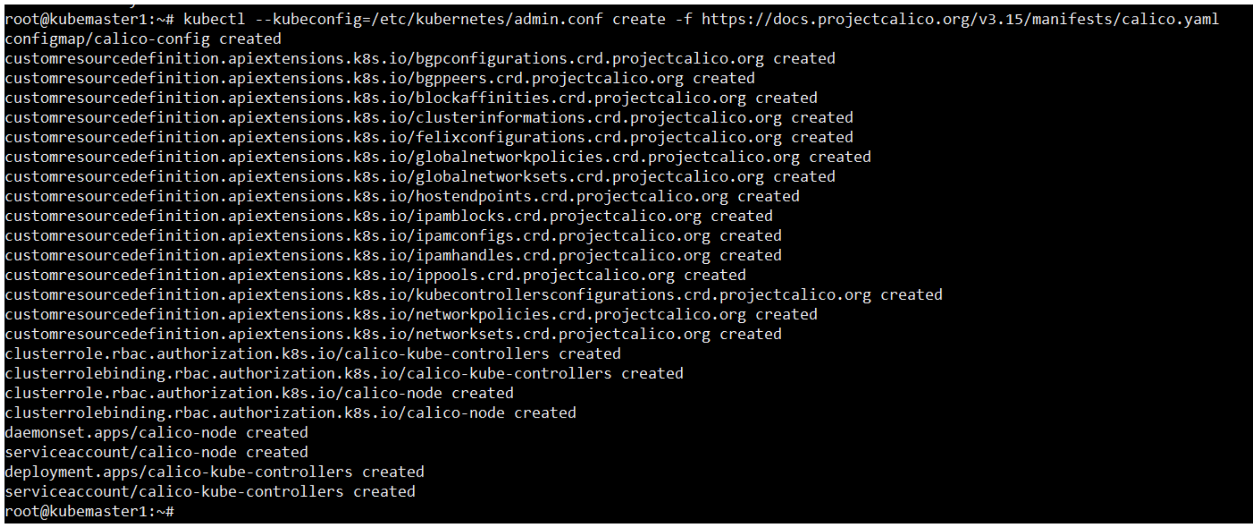
```
root@kubemaster2:~# kubectl get nodes
The connection to the server localhost:8080 was refused - did you specify the right host or port?
root@kubemaster2:~# kubectl --kubeconfig=/etc/kubernetes/admin.conf get nodes
NAME          STATUS     ROLES    AGE     VERSION
kubemaster1   NotReady   master   16m     v1.19.2
kubemaster2   NotReady   master   2m45s   v1.19.2
root@kubemaster2:~#
```

e) Run kubeadm join command on kubeworker1 node:



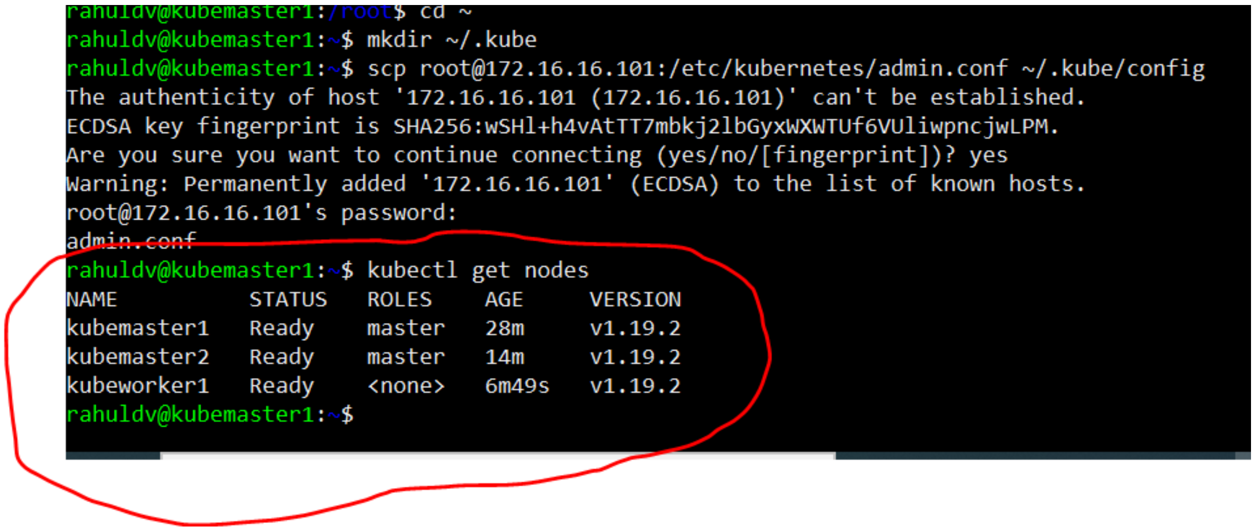
```
root@kubemaster2:~# kubectl --kubeconfig=/etc/kubernetes/admin.conf get nodes
NAME          STATUS     ROLES    AGE    VERSION
kubemaster1   NotReady   master   23m    v1.19.2
kubemaster2   NotReady   master   9m7s   v1.19.2
kubeworker1   NotReady   <none>   63s    v1.19.2
root@kubemaster2:~#
```

f) Deploy calico network on kubemaster1 node. Calico network acts as a virtual network used by Kubernetes to communicate betweek pods in a cluster



```
root@kubemaster1:~# kubectl --kubeconfig=/etc/kubernetes/admin.conf create -f https://docs.projectcalico.org/v3.15/manifests/calico.yaml
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/kubecontrollersconfigurations.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
root@kubemaster1:~#
```

Cluster is ready:



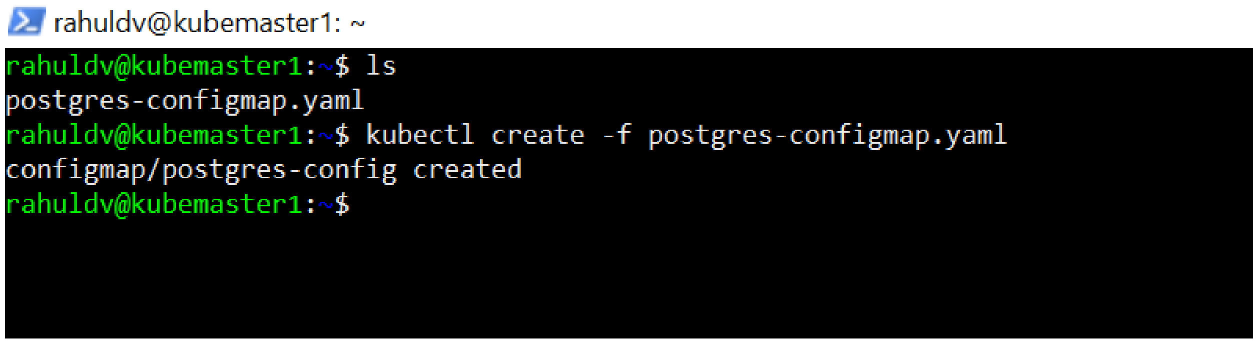
```
rahuldv@kubemaster1:/root$ cd ~
rahuldv@kubemaster1:~$ mkdir ~/.kube
rahuldv@kubemaster1:~$ scp root@172.16.16.101:/etc/kubernetes/admin.conf ~/.kube/config
The authenticity of host '172.16.16.101 (172.16.16.101)' can't be established.
ECDSA key fingerprint is SHA256:wSHl+h4vAtTT7mbkj2lbGyxWXWTUf6VUliwpncjwLPM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '172.16.16.101' (ECDSA) to the list of known hosts.
root@172.16.16.101's password:
admin.conf
rahuldv@kubemaster1:~$ kubectl get nodes
NAME          STATUS   ROLES    AGE     VERSION
kubemaster1   Ready    master   28m     v1.19.2
kubemaster2   Ready    master   14m     v1.19.2
kubeworker1   Ready    <none>   6m49s   v1.19.2
rahuldv@kubemaster1:~$
```

2) Deploy Postgres onto the k8s cluster created.

To do this we use the yamls checked in to the repository.

- Create postgres related config map using postgres-configmap.yaml and kubectl command.

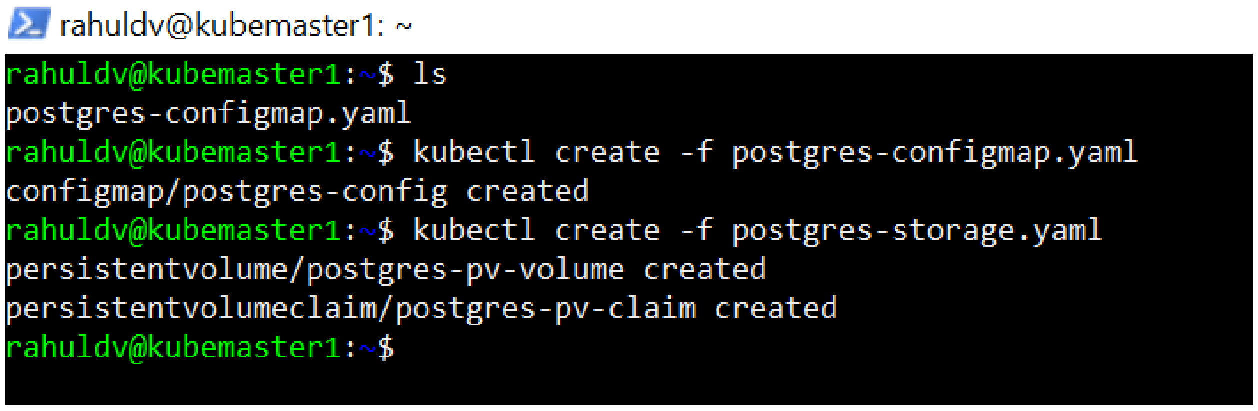
  > kubectl create -f postgres-configmap.yaml



```
rahuldv@kubemaster1:~$ ls
postgres-configmap.yaml
rahuldv@kubemaster1:~$ kubectl create -f postgres-configmap.yaml
configmap/postgres-config created
rahuldv@kubemaster1:~$
```

- Because containers are ephemeral and for databases, we need persistent stateful storage, we create a persistent volume and Persistent volume claim that will be used by database to store their data. Configuration is stored in postgres-storage.yaml

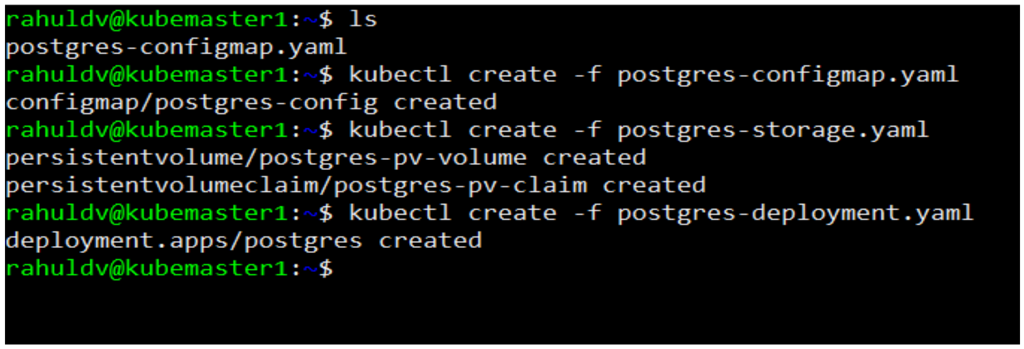
  > kubectl create -f postgres-storage.yaml



```
rahuldv@kubemaster1:~$ ls
postgres-configmap.yaml
rahuldv@kubemaster1:~$ kubectl create -f postgres-configmap.yaml
configmap/postgres-config created
rahuldv@kubemaster1:~$ kubectl create -f postgres-storage.yaml
persistentvolume/postgres-pv-volume created
persistentvolumeclaim/postgres-pv-claim created
rahuldv@kubemaster1:~$
```

- Next I create the actual deployment using postgres-deployment.yaml which specifies the replicas, the reference to config-map and reference to the persistent volume claim to be used. Due to resource limitations on my machine, I am keeping the number Of replicas as 1. But for high durability and availability and fault-tolerance, it is better to use multiple relpicas.

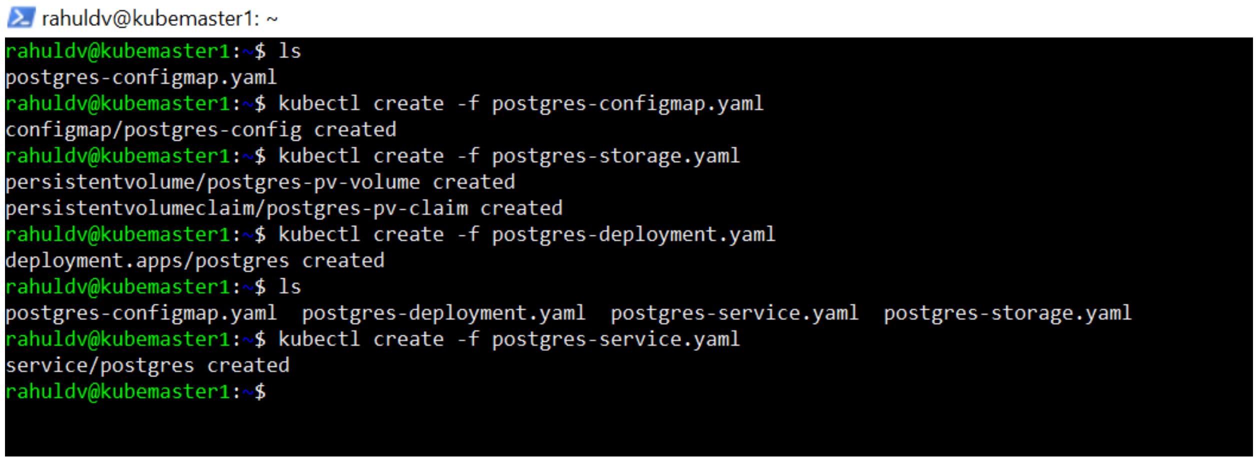
  > kubectl create -f postgres-deployment.yaml



```
rahuldv@kubemaster1:~$ ls
postgres-configmap.yaml
rahuldv@kubemaster1:~$ kubectl create -f postgres-configmap.yaml
configmap/postgres-config created
rahuldv@kubemaster1:~$ kubectl create -f postgres-storage.yaml
persistentvolume/postgres-pv-volume created
persistentvolumeclaim/postgres-pv-claim created
rahuldv@kubemaster1:~$ kubectl create -f postgres-deployment.yaml
deployment.apps/postgres created
rahuldv@kubemaster1:~$
```

- Create a NodePort service that k8s uses to enable postgres container to communicate with external world. Here I use NodePort as that enables the node I am currently on to directly access the postgres service.

  >kubectl create -f postgres-service.yaml



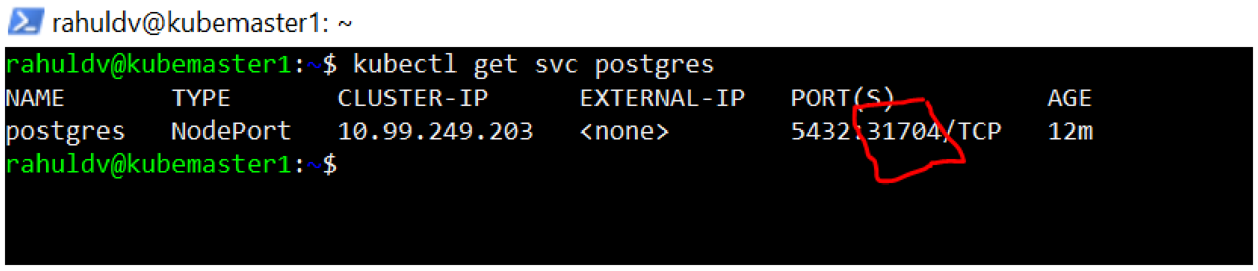
```
rahuldv@kubemaster1: ~
rahuldv@kubemaster1:~$ ls
postgres-configmap.yaml
rahuldv@kubemaster1:~$ kubectl create -f postgres-configmap.yaml
configmap/postgres-config created
rahuldv@kubemaster1:~$ kubectl create -f postgres-storage.yaml
persistentvolume/postgres-pv-volume created
persistentvolumeclaim/postgres-pv-claim created
rahuldv@kubemaster1:~$ kubectl create -f postgres-deployment.yaml
deployment.apps/postgres created
rahuldv@kubemaster1:~$ ls
postgres-configmap.yaml  postgres-deployment.yaml  postgres-service.yaml  postgres-storage.yaml
rahuldv@kubemaster1:~$ kubectl create -f postgres-service.yaml
service/postgres created
rahuldv@kubemaster1:~$
```

- Now we install postgres binaries to our host machine so that we can test connection using psql command line utility

  > sudo apt install postgresql postgresql-contrib

- To connect to postgres we need to know what port on local to use. This is provided by nodeport service.

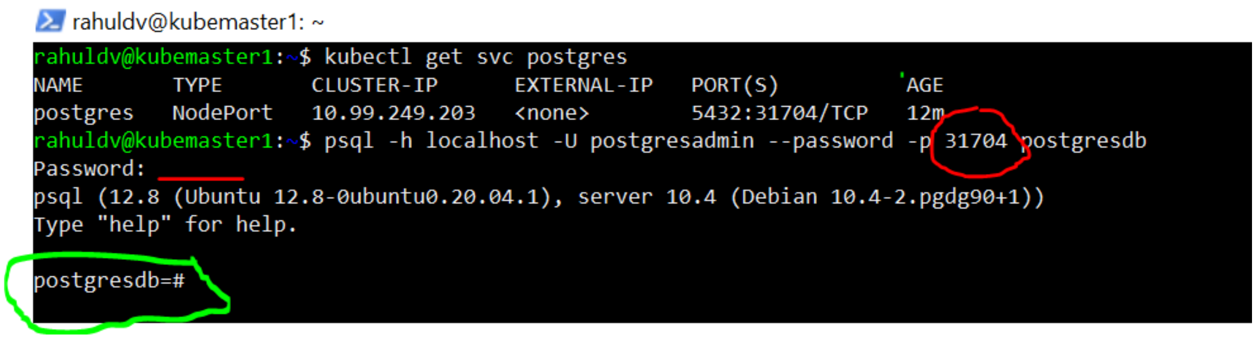
  > kubectl get svc postgres



```
rahuldv@kubemaster1: ~
rahuldv@kubemaster1:~$ kubectl get svc postgres
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
postgres   NodePort   10.99.249.203   <none>        5432:31704/TCP   12m
rahuldv@kubemaster1:~$
```

- Connect to postgres using psql

  We use above highlighted port to connect. DB credentials are found in configmap yaml.



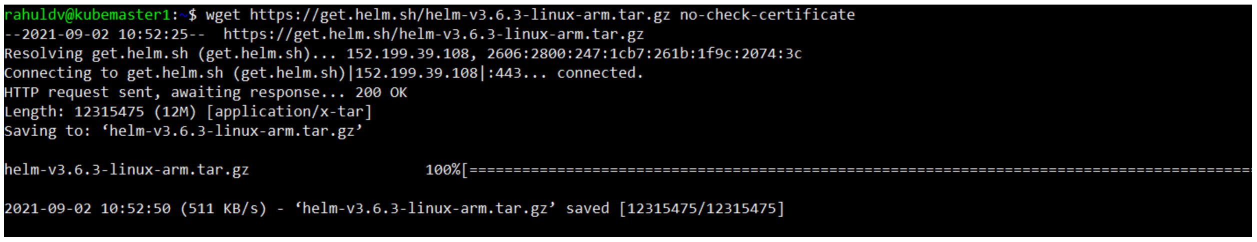
```
rahuldv@kubemaster1: ~
rahuldv@kubemaster1:~$ kubectl get svc postgres
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
postgres   NodePort   10.99.249.203   <none>        5432:31704/TCP   12m
rahuldv@kubemaster1:~$ psql -h localhost -U postgresadmin --password -p 31704 postgresdb
Password:
psql (12.8 (Ubuntu 12.8-0ubuntu0.20.04.1), server 10.4 (Debian 10.4-2.pgdg90+1))
Type "help" for help.

postgresdb=#
```

The content highlighted in green above confirms a successful connection to postgres db running on a container managed by Kubernetes cluster created.

3) Create the Postgres or any database instance deployment manifests using helm chart

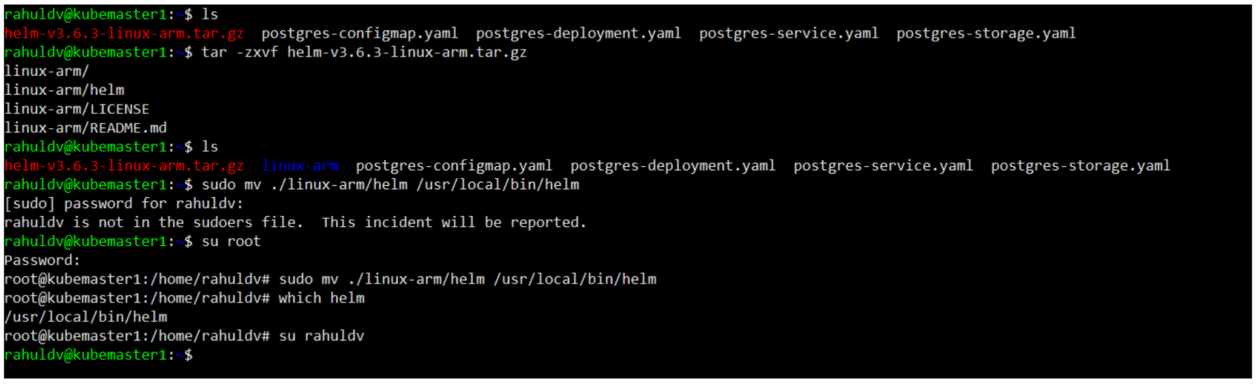
All commands are available in helmChart.sh

- Download helm binaries on the host node



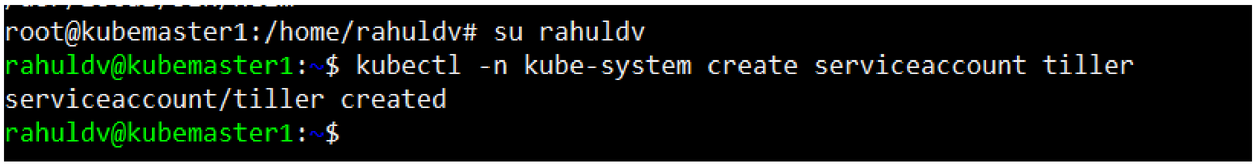
```
rahuldv@kubemaster1:~$ wget https://get.helm.sh/helm-v3.6.3-linux-arm.tar.gz no-check-certificate
--2021-09-02 10:52:25--  https://get.helm.sh/helm-v3.6.3-linux-arm.tar.gz
Resolving get.helm.sh (get.helm.sh)... 152.199.39.108, 2606:2800:247:1cb7:261b:1f9c:2074:3c
Connecting to get.helm.sh (get.helm.sh)|152.199.39.108|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 12315475 (12M) [application/x-tar]
Saving to: ‘helm-v3.6.3-linux-arm.tar.gz’

helm-v3.6.3-linux-arm.tar.gz                    100%[==========================================================================================

2021-09-02 10:52:50 (511 KB/s) - ‘helm-v3.6.3-linux-arm.tar.gz’ saved [12315475/12315475]
```

- Un-tar the tar ball



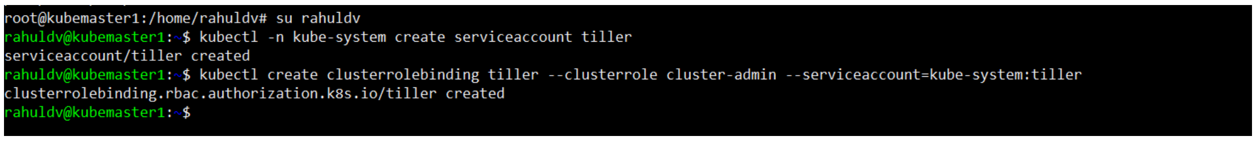
```
rahuldv@kubemaster1:~$ ls
helm-v3.6.3-linux-arm.tar.gz  postgres-configmap.yaml  postgres-deployment.yaml  postgres-service.yaml  postgres-storage.yaml
rahuldv@kubemaster1:~$ tar -zxvf helm-v3.6.3-linux-arm.tar.gz
linux-arm/
linux-arm/helm
linux-arm/LICENSE
linux-arm/README.md
rahuldv@kubemaster1:~$ ls
helm-v3.6.3-linux-arm.tar.gz  linux-arm  postgres-configmap.yaml  postgres-deployment.yaml  postgres-service.yaml  postgres-storage.yaml
rahuldv@kubemaster1:~$ sudo mv ./linux-arm/helm /usr/local/bin/helm
[sudo] password for rahuldv:
rahuldv is not in the sudoers file.  This incident will be reported.
rahuldv@kubemaster1:~$ su root
Password:
root@kubemaster1:/home/rahuldv# sudo mv ./linux-arm/helm /usr/local/bin/helm
root@kubemaster1:/home/rahuldv# which helm
/usr/local/bin/helm
root@kubemaster1:/home/rahuldv# su rahuldv
rahuldv@kubemaster1:~$
```

- Create a service account for tiller to use



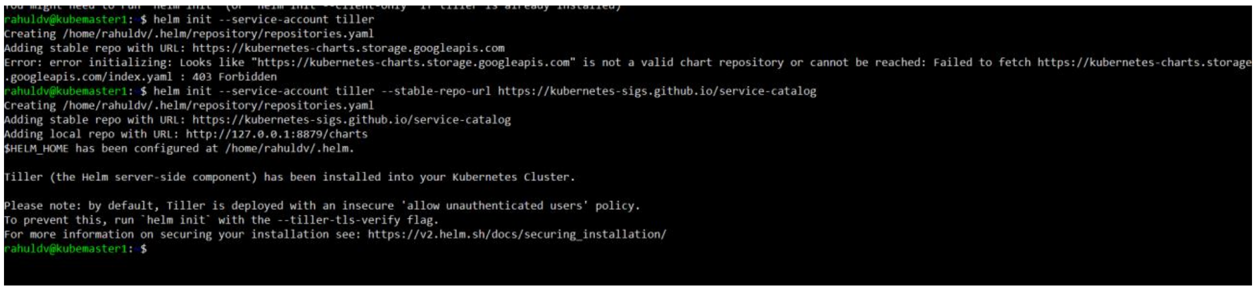
```
root@kubemaster1:/home/rahuldv# su rahuldv
rahuldv@kubemaster1:~$ kubectl -n kube-system create serviceaccount tiller
serviceaccount/tiller created
rahuldv@kubemaster1:~$
```

- Create a custom role binding to bind the role to the service account created



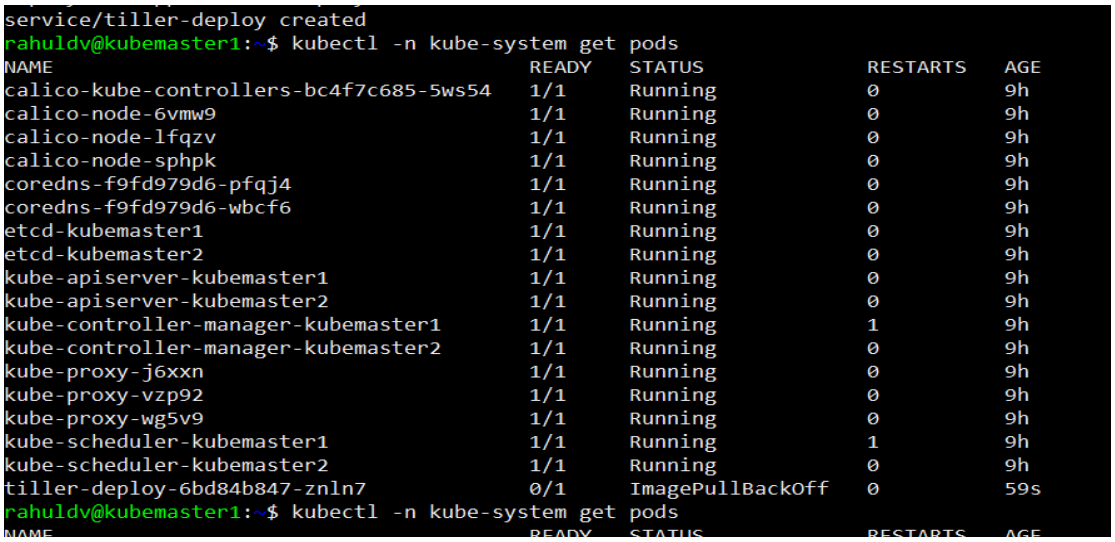
```
root@kubemaster1:/home/rahuldv# su rahuldv
rahuldv@kubemaster1:~$ kubectl -n kube-system create serviceaccount tiller
serviceaccount/tiller created
rahuldv@kubemaster1:~$ kubectl create clusterrolebinding tiller --clusterrole cluster-admin --serviceaccount=kube-system:tiller
clusterrolebinding.rbac.authorization.k8s.io/tiller created
rahuldv@kubemaster1:~$
```

- Initialize helm



```
rahuldv@kubemaster1:~$ helm init --service-account tiller
Creating /home/rahuldv/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-charts.storage.googleapis.com
Error: error initializing: Looks like "https://kubernetes-charts.storage.googleapis.com" is not a valid chart repository or cannot be reached: Failed to fetch https://kubernetes-charts.storage
.googleapis.com/index.yaml : 403 Forbidden
rahuldv@kubemaster1:~$ helm init --service-account tiller --stable-repo-url https://kubernetes-sigs.github.io/service-catalog
Creating /home/rahuldv/.helm/repository/repositories.yaml
Adding stable repo with URL: https://kubernetes-sigs.github.io/service-catalog
Adding local repo with URL: http://127.0.0.1:8879/charts
$HELM_HOME has been configured at /home/rahuldv/.helm.

Tiller (the Helm server-side component) has been installed into your Kubernetes Cluster.

Please note: by default, Tiller is deployed with an insecure 'allow unauthenticated users' policy.
To prevent this, run `helm init` with the --tiller-tls-verify flag.
For more information on securing your installation see: https://v2.helm.sh/docs/securing_installation/
rahuldv@kubemaster1:~$
```

- Unfortunately, I used Helm 2.12.2 version. The stable repository for any helm 2.* versions are not working and not accessible as a result I could install the tiller component properly.



```
service/tiller-deploy created
rahuldv@kubemaster1:~$ kubectl -n kube-system get pods
NAME                                      READY   STATUS             RESTARTS   AGE
calico-kube-controllers-bc4f7c685-5ws54   1/1     Running            0          9h
calico-node-6vmw9                         1/1     Running            0          9h
calico-node-lfqzv                         1/1     Running            0          9h
calico-node-sphpk                         1/1     Running            0          9h
coredns-f9fd979d6-pfqj4                   1/1     Running            0          9h
coredns-f9fd979d6-wbcf6                   1/1     Running            0          9h
etcd-kubemaster1                          1/1     Running            0          9h
etcd-kubemaster2                          1/1     Running            0          9h
kube-apiserver-kubemaster1                1/1     Running            0          9h
kube-apiserver-kubemaster2                1/1     Running            0          9h
kube-controller-manager-kubemaster1       1/1     Running            1          9h
kube-controller-manager-kubemaster2       1/1     Running            0          9h
kube-proxy-j6xxn                          1/1     Running            0          9h
kube-proxy-vzp92                          1/1     Running            0          9h
kube-proxy-wg5v9                          1/1     Running            0          9h
kube-scheduler-kubemaster1                1/1     Running            1          9h
kube-scheduler-kubemaster2                1/1     Running            0          9h
tiller-deploy-6bd84b847-znln7             0/1     ImagePullBackOff   0          59s
rahuldv@kubemaster1:~$ kubectl -n kube-system get pods
```

- Finally I had to uninstall the entire helm installation.

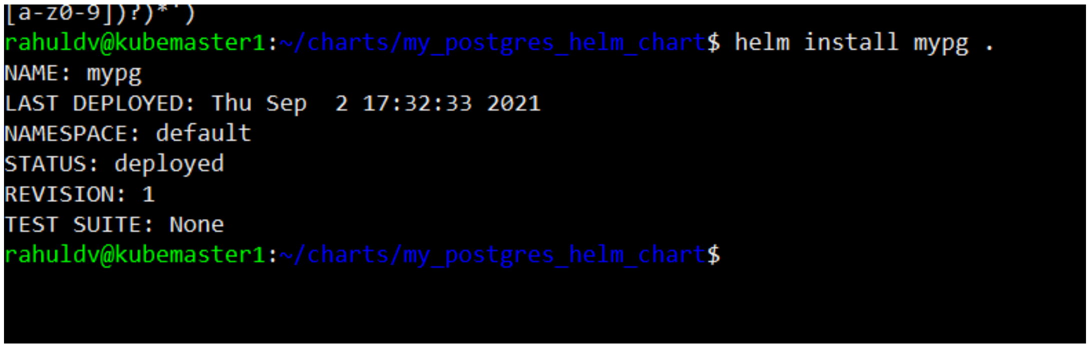
I removed using following commands:
>kubectl -n kube-system delete deployment tiller-deploy-6bd84b847-znln7
> kubectl -n kube-system delete service tiller-deploy-6bd84b847-znln7
Also removed the local .helm folder
>rm -rf .helm

- Installed helm version 3.6.3

- Created my own chart to deploy postgres just like I did in step 2. Used the same yaml in template folder. The name of the chart is my_postgres_helm_chart. Please find all the chart related code inside charts/ my_postgres_helm_chart folder.

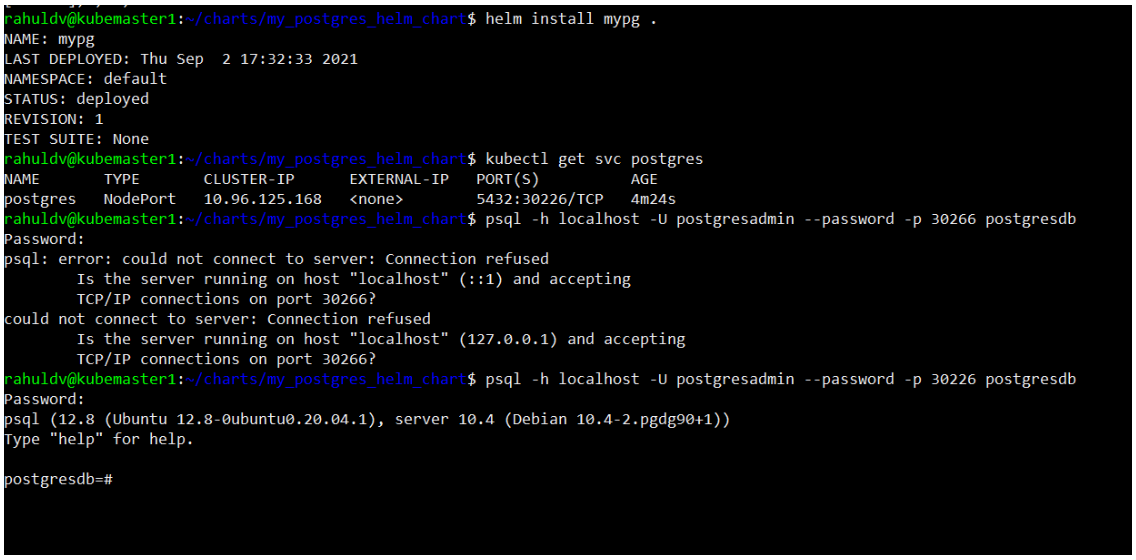
- Once I created my custom postgres chart, I installed using the following command

  >helm install mypg .
  This command created a helm installation called mypg using the current directory as local repository to find the chart



```
[a-z0-9])?)*')
rahuldv@kubemaster1:~/charts/my_postgres_helm_chart$ helm install mypg .
NAME: mypg
LAST DEPLOYED: Thu Sep  2 17:32:33 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
rahuldv@kubemaster1:~/charts/my_postgres_helm_chart$
```

- Tried connecting to postgres using psql



```
rahuldv@kubemaster1:~/charts/my_postgres_helm_chart$ helm install mypg .
NAME: mypg
LAST DEPLOYED: Thu Sep  2 17:32:33 2021
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
rahuldv@kubemaster1:~/charts/my_postgres_helm_chart$ kubectl get svc postgres
NAME       TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
postgres   NodePort   10.96.125.168   <none>        5432:30226/TCP   4m24s
rahuldv@kubemaster1:~/charts/my_postgres_helm_chart$ psql -h localhost -U postgresadmin --password -p 30266 postgresdb
Password:
psql: error: could not connect to server: Connection refused
        Is the server running on host "localhost" (::1) and accepting
        TCP/IP connections on port 30266?
could not connect to server: Connection refused
        Is the server running on host "localhost" (127.0.0.1) and accepting
        TCP/IP connections on port 30266?
rahuldv@kubemaster1:~/charts/my_postgres_helm_chart$ psql -h localhost -U postgresadmin --password -p 30226 postgresdb
Password:
psql (12.8 (Ubuntu 12.8-0ubuntu0.20.04.1), server 10.4 (Debian 10.4-2.pgdg90+1))
Type "help" for help.

postgresdb=#
```

4) Deploy the Postgres instance or any database (helm chart) using any GIT-Ops tools(FluxCD/ArgoCD)

   I could not complete this due to time crunch. I spent a lot of time debugging helm and tiller installation when using 2.12.2 version. Realized that tiller is completely removed in 3.* versions. But I know the basics of what they do.