

Build the kubernetes master slave architecture and establish the communication

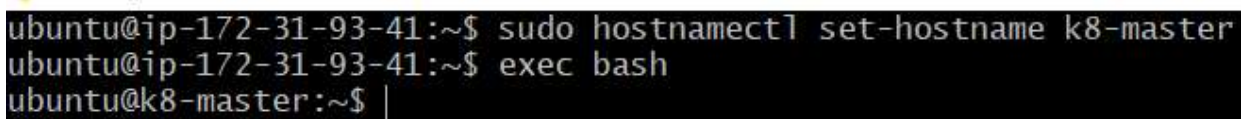
Step 1: Create Instance for Master

The first step is to create EC2 Ubuntu instances, one for master with t2 medium. Once the instance is available, the next step is to deploy kubeadm.

To start with, Ubuntu EC2 instance was given meaningful name i.e. k8-master using the command below:.

```
sudo hostnamectl set-hostname k8-master
```

```
exec bash
```

A terminal window showing the command 'exec bash' being executed on the k8-master instance. The prompt changes from 'ubuntu@ip-172-31-93-41:~\$' to 'ubuntu@k8-master:~\$'.A terminal window showing the commands 'sudo hostnamectl set-hostname k8-master' and 'exec bash' being executed. The prompt changes from 'ubuntu@ip-172-31-93-41:~\$' to 'ubuntu@k8-master:~\$'.

Step 2. Install Docker

In the second step, Docker needs to be installed on the instance and the same needs to be verified. For this, the following set of commands needs to be executed:

To update packages:

```
sudo apt-get update
```

To install Docker:

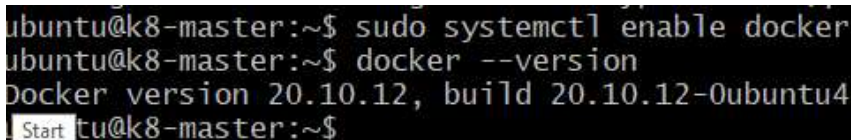
```
sudo apt-get install docker.io -y
```

Start Docker on the node using the following command:

```
sudo systemctl enable docker
```

Verify if Docker is installed using the following command:

```
docker --version
```

A terminal window showing the commands 'sudo systemctl enable docker' and 'docker --version' being executed. The output of the second command is 'Docker version 20.10.12, build 20.10.12-0ubuntu4'.

Step 3: Install Kubeadm

Once docker gets installed the next step will be to install kubeadm

We also need to install a transparent HTTPS package to allow the clusters to communicate with each other:

```
sudo apt-get install apt-transport-https curl -y
```

Add Kubernetes package repository key:

```
wget https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

```
sudo apt-key add apt-key.gpg
```

```
ubuntu@k8-master:~$ sudo apt-key add apt-key.gpg
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
ubuntu@k8-master:~$ |
```

Disable swap temporary (as recommended):

```
sudo swapoff -a
```

To configure Kubernetes repository

```
sudo apt-add-repository deb http://apt.kubernetes.io/ kubernetes-xenial main
```

Install Kubeadm package:

```
sudo apt-get install kubeadm -y
```

Verify kubeadm installation:

```
kubeadm version
```

```
ubuntu@k8-master:~$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.2", GitCommit:"5835544ca568b757a8ecae5c153f317e5736700e",
reeState:"clean", BuildDate:"2022-09-21T14:32:18Z", GoVersion:"go1.19.1", Compiler:"gc", Platform:"linux/amd64"}
ubuntu@k8-master:~$
```

Step 4: Initialize Kubernetes

Note: A minimum of two vCPUs are required for the master node and hence we should select t2.medium.

The next step of the Kubernetes cluster configuration is to initialize Kubernetes on the master node using the kubeadm init command.

After successful execution of the kubeadm init command, one can acquire all the information to configure the Kubernetes client as well as how the worker node can join the master node. This is how the worker node can

join the master node.

```
sudo kubeadm init --pod-network-cidr=pvt ip
```

Step 5: Set Up kubeconfig

To set up kubeconfig locally, execute the following command on master node:

```
mkdir -p $HOME/.kube
```

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

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

```
ubuntu@k8-master:~$ mkdir -p $HOME/.kube
ubuntu@k8-master:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
ubuntu@k8-master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Step 6: Apply Flannel CNI network overlay by running the following command on the master node:

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

```
ubuntu@k8-master:~$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
namespace/kube-flannel created
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
configmap/kube-flannel-cfg created
daemonset.apps/kube-flannel-ds created
ubuntu@k8-master:~$ |
```

Step 7 : Creating Worker Nodes

Next step is to create EC2 Ubuntu instances for worker nodes.

To start with three Ubuntu ec2 instances were given meaningful names i.e. , k8-worker-node1, k8-worker-node2, using the command below:.

```
sudo hostnamectl set-hostname k8-Arun-node 1
```

```
exec bash
```

```
sudo hostnamectl set-hostname k8-Arun-node 2
```

```
exec bash
```

```
ubuntu@k8-worker-node1:~$ sudo hostnamectl set-hostname k8-Arun-node-1
ubuntu@k8-worker-node1:~$ exec bash
ubuntu@k8-Arun-node-1:~$ |
```

```
ubuntu@k8-worker-node2:~$ sudo hostnamectl set-hostname k8-Arun-node-2
ubuntu@k8-worker-node2:~$ exec bash
ubuntu@k8-Arun-node-2:~$ |
```

Step 8 : Once all the instances are available, the next step is to deploy docker and kubeadm on all instances

Install Docker on all machines

```
sudo apt-get update
```

```
sudo apt-get install docker.io -y
```

```
sudo systemctl enable docker
```

```
docker --version
```

```
ubuntu@k8-Arun-node-1:~$ sudo systemctl enable docker
ubuntu@k8-Arun-node-1:~$ docker --version
Docker version 20.10.12, build 20.10.12-0ubuntu4
ubuntu@k8-Arun-node-1:~$
```

```
ubuntu@k8-Arun-node-2:~$ sudo systemctl enable docker
ubuntu@k8-Arun-node-2:~$ docker --version
Docker version 20.10.12, build 20.10.12-0ubuntu4
ubuntu@k8-Arun-node-2:~$ |
```

Install Kubeadm

```
sudo apt-get install apt-transport-https curl -y
```

```
wget https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

```
sudo apt-key add apt-key.gpg
```

```
sudo swapoff -a
```

```
sudo apt-add-repository deb http://apt.kubernetes.io/ kubernetes-xenial main
```

```
sudo apt-get install kubeadm -y
```

```
kubeadm version
```

```
ubuntu@k8-Arun-node-1:~$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.2", GitCommit:"5835544ca568b757a8ecae5c153f317e5736700e", GitTreeState:"clean", BuildDate:"2022-09-21T14:32:18Z", GoVersion:"go1.19.1", Compiler:"gc", Platform:"linux/amd64"}
```

```
ubuntu@k8-Arun-node-2:~$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"25", GitVersion:"v1.25.2", GitCommit:"5835544ca568b757a8ecae5c153f317e5736700e", GitTreeState:"clean", BuildDate:"2022-09-21T14:32:18Z", GoVersion:"go1.19.1", Compiler:"gc", Platform:"linux/amd64"}
```

Step 9: Joining the worker nodes to form a cluster

we must join the worker nodes to form a cluster by executing the following command on worker Nodes:

```
sudo kubeadm join 172.31.93.41:6443 --token r44guc.b6cx0i1h7nwyagph \ --discovery-token-ca-cert-hash sha256:f30a6feb502c44c9b0b7bad69a6b7243a9a8c945c14229196de7f12e2dadcf6f
```

To verify that the worker nodes have joined the cluster successfully, the following command needs to be executed on the master node:

```
kubectl get nodes
```

This will get your simple multi-cluster Kubernetes architecture with one master and two nodes up and running:

```
ubuntu@k8-master:~$ kubectl get nodes
NAME           STATUS    ROLES    AGE   VERSION
k8-master      Ready    control-plane   3m37s   v1.25.2
Arun-node-1    Ready    <none>        1m50s   v1.25.2
Arun-node-2    Ready    <none>        1m50s   v1.25.2
```

Conclusion

A multi-cluster Kubernetes architecture definitely helps you manage and scale your workloads in better ways with minimal additional investments and no cloud vendor lock-in.

Kubernetes is leading the way to a better, more efficient, and secure IT environment. The Kubernetes Multi-Cluster architecture has been designed to create multiple isolated clusters that each have their own sets of resources, ensuring high availability and reliability. The multi-cluster architecture helps in creating highly available, self-healing, and elastic pods.