Install and Deploy Kubernetes on Ubuntu 16.04

By **Mohamed Ettayeb** - January 19, 2019

> *As a part of the Kubernetes tutorials for beginners series, my goal is to offer a full and solid list of articles that goes through the very basics definitions, the history and the need to use Kubernetes and containers until i reach the deep parts, so regardless of your technical background, i will offer you everything you need here to master Kubernetes step by step.*

If you aren't coming from the last article on this series, i would like to recommend that you go back and start from the first article rather than diving directly into the install and deploy of Kubernetes on Ubuntu 16.04 cause i will not re explain terms and concepts that we have already discussed again, otherwise, you can start with the requirements section.

Requirements

- Two machines or VMs with Ubuntu 16.04 installed (for me i am using my own computer as the master node, so i have prepared a VM with VirtualBox to be the worker. If you want to do the same you can refer to this article : Install Ubuntu 16.04 on VirtualBox)

- Two static IPs configured: the address 192.168.1.105 is configured on the first machine which is the Master and the adress 192.168.1.106 is configured on the second machine which is the Worker.

- 2GB RAM per machine.
- The root user or a user with root permissions

Initial setup

Before we start, we need to configure hosts file and hostname on each machine, so each one can communicate with each other using the hostname.

First of all, you need to open the /etc/hosts file on the first machine (the master) with this command:

```
sudo nano /etc/hosts
```

And add the following lines, which are the static IPs and the matched hostnames:

```
192.168.1.105 master-node
192.168.1.106 worker-node
```

You can save and close the file when you are finished, then change the hostname by running the following command:

```
sudo hostnamectl set-hostname master-node
```

On the second machine, open /etc/hosts file:

```
sudo nano /etc/hosts
```

And add the following lines:

```
192.168.1.105 master-node
192.168.1.106 worker-node
```

That's it, you can save and close the file when you are done, then setup hostname by running the following command:

```
sudo hostnamectl set-hostname worker-node
```

The last step which is very important, is that we need to disable swap memory on each machine. Because kubelets do not support swap memory and will not work if swap is active or even present in your /etc/fstab file.

Just run this command to disable swap memory usage:

```
sudo swapoff -a
```

Install Docker

Another important step, which is installing Docker on both the master and worker machines. By default, the latest version of the Docker is not available in Ubuntu 16.04 repository, so you have to add the Docker repository to your system.

First of all, we need to install the required packages to add Docker repository with the following command:

```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common -y
```

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

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

Then, just add Docker repository to your list with the following command:

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

Last but not least, we should update the repository and install Docker with the following command:

```
sudo apt-get update -y
sudo apt-get install docker-ce -y
```

Remember you should install Docker on both machines.

Install Kubernetes

After installing Docker we should install kubeadm, kubectl and kubelet tools on both machines.

Note:
Kubeadm is a tool which is new and part of the Kubernetes distribution 1.4.0 and it makes the setup and install of Kubernetes more easier.

First, we should download and add the GPG key with the following command:

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

Secondly, we need to add Kubernetes repository with the following command:

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

Finally, just update the repository and launch the install command for the Kubernetes:

```
sudo apt-get update -y
sudo apt-get install kubelet kubeadm kubectl -y
```

Master Node Configuration

Right now, All the required packages are installed on both machines. So, it's time to configure the Kubernetes Master Node.

So first, we need to initialize our cluster using its private IP address with the following command:

```
sudo kubeadm init  --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=192.168.1.105
```

After executing this command, you should see the following output:

```
Your Kubernetes master 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

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 machines by running the following on each node
as root:

kubeadm join 192.168.1.105:6443 --token fwldpz.bgejrgr1cqviuoeo --discovery-token-ca-cert-hash sha25
```

Note : Copy somewhere the token from your output, cause it will be used to join Worker Node to the Master Node in the next step.

Secondly, we should execute the first three commands that you can find on the output above which are used to configure the **kubectl** tool:

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

Thirdly, we should check the status of the Master Node by running the following command:

```
kubectl get nodes
```

You should see the following output:

```
NAME          STATUS     ROLES    AGE   VERSION
master-node   NotReady   master   14m   v1.13.2
```

On the last output, you should see that the Master Node have the Status **NotReady** and this is because that the cluster does not have a Container Networking Interface (CNI).

Let's deploy the Calico CNI for the Master Node, but first we need to install an etcd instance with the following command:

```
kubectl apply -f \
https://docs.projectcalico.org/v3.4/getting-started/kubernetes/installation/hosted/etcd.yaml
```

You should see the following output:

```
daemonset.extensions/calico-etcd created
service/calico-etcd created
```

Then to install the Calico CNI, run this command:

```
kubectl apply -f \
https://docs.projectcalico.org/v3.4/getting-started/kubernetes/installation/hosted/calico.yaml
```

You should see the following output:

```
configmap/calico-config created
secret/calico-etcd-secrets created
daemonset.extensions/calico-node created
serviceaccount/calico-node created
deployment.extensions/calico-kube-controllers created
serviceaccount/calico-kube-controllers 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
```

Confirm that all of the pods are running with the following command.

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

Just keep waiting until each pod has the STATUS of Running like this:

```
NAMESPACE     NAME                                    READY   STATUS    RESTARTS   AGE
kube-system   calico-etcd-xt8mt                       1/1     Running   0          75s
kube-system   calico-kube-controllers-5d94b577bb-wljc9   1/1     Running   0          87s
kube-system   calico-node-p79ln                       1/1     Running   1          87s
kube-system   coredns-86c58d9df4-jjwcz                1/1     Running   0          113s
kube-system   coredns-86c58d9df4-t9sxv                1/1     Running   0          113s
kube-system   etcd-master-node                        1/1     Running   0          67s
kube-system   kube-apiserver-master-node              1/1     Running   0          54s
```

```
kube-system   kube-controller-manager-master-node        1/1     Running   0          60s
kube-system   kube-proxy-wdlqm                           1/1     Running   0          113s
kube-system   kube-scheduler-master-node                 1/1     Running   0          69s
```

Press CTRL+C to exit watch

Note: if you got a problem with the coredns pods (not running) then open the file **on both machines** (/etc/resolv.conf) and change the nameserver to 8.8.8.8 then execute the command " sudo kubeadm reset " and restart the process.

if you got any problem with other pods, run this command " kubectl logs -n kube-system POD_NAME " then comment here the output and i will help you to solve it.

Now, run the `kubectl get nodes` command again, and you will see that the Master Node is listed as Ready.

```
kubectl get nodes
```

Output:

```
NAME          STATUS   ROLES    AGE     VERSION
master-node   Ready    master   6m44s   v1.13.2
```

Add Worker Node to the Kubernetes Cluster

Next, we have to log in to the Worker Node and add it to the Cluster. Remember the join command in the output from the Master Node initialization command and run it on the Worker Node as shown below:

```
kubeadm join 192.168.1.105:6443 --token fwldpz.bgejrgr1cqviuoeo --discovery-token-ca-cert-hash sha25
```

When the Node is joined successfully, you should see this output:

```
[discovery] Trying to connect to API Server "192.168.1.105:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.168.1.105:6443"
[discovery] Requesting info from "https://192.168.1.105:6443" again to validate TLS against the pinn
[discovery] Cluster info signature and contents are valid and TLS certificate validates against pinn
[discovery] Successfully established connection with API Server "192.168.1.105:6443"
[join] Reading configuration from the cluster...
[join] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyar
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.13" ConfigMap in the
[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.
[kubelet-start] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "

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 master to see this node join the cluster.
```

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

```
kubectl get nodes
```

Output:

```
NAME          STATUS   ROLES    AGE     VERSION
master-node   Ready    master   21m     v1.13.2
worker-node    Ready    <none>   5m29s   v1.13.2
```

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 an Nginx deployment:

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

Output:

```
deployment.apps/nginx created
```

You can list out the deployments with the following command:

```
kubectl get deployments
```

Output :

```
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
nginx   0/1     1            0           24s
```

You can see more information about Nginx deployment with the following command:

```
kubectl describe deployment nginx
```

Output:

```
Name:                   nginx
Namespace:              default
CreationTimestamp:      Thu, 17 Jan 2019 16:13:10 +0100
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               app=nginx
Replicas:               1 desired | 1 updated | 1 total | 0 available | 1 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>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      False   MinimumReplicasUnavailable
  Progressing    True    ReplicaSetUpdated
OldReplicaSets:  <none>
```

```
NewReplicaSet:   nginx-5c7588df (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  28s   deployment-controller  Scaled up replica set nginx-5c7588df to 1
```

You can run this command and watch the Nginx pod status until it becomes “Running”:

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

After, we will need to make the Nginx container available to the network with this command:

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

Now, list out all the services by running the following command:

```
kubectl get svc
```

You should see the Nginx service with assigned port 30784:

```
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
nginx        NodePort    10.102.166.47   <none>        80:30784/TCP   31s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        25m
```

Finally, just open your web browser and type the URL http://192.168.1.106:30784 (Worker Node IP:The port number from your output). You should see the default Nginx Welcome page:

nginx-kubernetes-ubuntu-16.04

Congratulations! Your Nginx container has been deployed on your Kubernetes Cluster.

On our next article we will take a look on how to install and deploy Kubernetes on CentOs7.

See you!

Mohamed Ettayeb

I m a Tunisian geek who start digging on technology information since 8 years ,i am a fullstack javascript web developer and a pentester. There is nothing that i cannot learn.