KUBERNETES III - KUBEADM ON AWS

This is based on Installing Kubernetes on Linux with kubeadm, and it will show how to install a Kubernetes 1.6 cluster on machines running Ubuntu 16.04 using a tool called kubeadm which is part of Kubernetes.

AWS instances:

1. **master**: 54.88.23.51 (172.31.60.33)
2. **node**: 54.172.245.224 (172.31.41.29)

We'll do the following:

1. Install a secure Kubernetes cluster on our machines.
2. Install a pod network on the cluster so that application components (pods) can talk to each other.
3. Install a sample microservices application (a socks shop) on the cluster.

Kubernetes Terminology

1. **Nodes**:
   Hosts that run Kubernetes applications
2. **Containers**:
   Units of packaging
3. **Pods**:
   Units of deployment which is collection of containers
4. **Replication Controller**:
   Ensures availability and scalability
5. **Labels**:

Key-value pairs for identification
6. **Services**:
   Collection of pods exposed as an endpoint

Installing kubelet and kubeadm on all machines
We will install the following packages on all the machines:

1. **docker**: the container runtime, which Kubernetes depends on. v1.12 is recommended, but v1.10 and v1.11 are known to work as well. v1.13 and 17.03+ have not yet been tested and verified by the Kubernetes node team.
2. **kubelet**: the most core component of Kubernetes. It runs on all of the machines in our cluster and does things like starting pods and containers.
3. **kubectl**: the command to control the cluster once it's running. We will only need this on the master, but it can be useful to have on the other nodes as well.
4. **kubeadm**: the command to bootstrap the cluster.

For each host, run the following machine:

```
ubuntu@ip-172-31-60-33:~$ sudo su
root@ip-172-31-60-33:/home/ubuntu# apt-get update
&& apt-get install -y apt-transport-https
root@ip-172-31-60-33:/home/ubuntu# curl -s https://
packages.cloud.google.com/apt/doc/apt-key.gpg | apt-
key add -
root@ip-172-31-60-33:/home/ubuntu# cat <<EOF >/etc/
apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
```

EOF
root@ip-172-31-60-33:/home/ubuntu# apt-get update
root@ip-172-31-60-33:/home/ubuntu# apt-get install -y docker.io
root@ip-172-31-60-33:/home/ubuntu# apt-get install -y kubelet kubeadm kubectl kubernetes-cni

The kubelet is now restarting every few seconds, as it waits in a crashloop for kubeadm to tell it what to do.

Initializing the master
The master is the machine where the control plane components run, including etcd (the cluster database) and the API server (which the kubectl CLI communicates with).
To initialize the master, pick one of the machines we previously installed kubeadm on, and run:

ubuntu@ip-172-31-60-33:/home/ubuntu# kubeadm init
[kubeadm] WARNING: kubeadm is in beta, please do not use it for production clusters.
[init] Using Kubernetes version: v1.6.0
[init] Using Authorization mode: RBAC
[preflight] Running pre-flight checks
[certificates] Generated CA certificate and key.
[certificates] Generated API server certificate and key.
[certificates] API Server serving cert is signed for DNS names [ip-172-31-60-33 kubernetes kubernetes.default

kubernetes.default.svc
kubernetes.default.svc.cluster.local] and IPs [10.96.0.1
172.31.60.33]
[certificates] Generated API server kubelet client
certificate and key.
[certificates] Generated service account token signing
key and public key.
[certificates] Generated front-proxy CA certificate and
key.
[certificates] Generated front-proxy client certificate and
key.
[certificates] Valid certificates and keys now exist in "/etc/
kubernetes/pki"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/
kubernetes/admin.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/
kubernetes/kubelet.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/
kubernetes/controller-manager.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/
kubernetes/scheduler.conf"
[apiclient] Created API client, waiting for the control plane
to become ready
[apiclient] All control plane components are healthy after
29.327806 seconds
[apiclient] Waiting for at least one node to register
[apiclient] First node has registered after 3.502761
seconds
[token] Using token: 918ba6.e2cb11b266dced53
[apiconfig] Created RBAC rules
[addons] Created essential addon: kube-proxy

[addons] Created essential addon: kube-dns

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run (as a regular user):

```
  sudo cp /etc/kubernetes/admin.conf $HOME/
  sudo chown $(id -u):$(id -g) $HOME/admin.conf
  export KUBECONFIG=$HOME/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:
  http://kubernetes.io/docs/admin/addons/

You can now join any number of machines by running the following on each node
as root:

```
  kubeadm join --token 918ba6.e2cb11b266dced53 172.31.60.33:6443
```

kubeadm init will first run a series of prechecks to ensure that the machine is ready to run Kubernetes. It will expose warnings and exit on errors. It will then download and install the cluster database and "control plane" components. This may take several minutes.
Make a record of the kubeadm join command that kubeadm init outputs. We will need this in a moment. The token is used for mutual authentication between the

master and the joining nodes. These tokens can be listed, created and deleted with the kubeadm token command.

To start using our cluster, we need to run as a regular user:
ubuntu@ip-172-31-60-33:/home/ubuntu# exit
exit

ubuntu@ip-172-31-60-33:~$ sudo cp /etc/kubernetes/admin.conf $HOME/
ubuntu@ip-172-31-60-33:~$ sudo chown $(id -u):$(id -g) $HOME/admin.conf
ubuntu@ip-172-31-60-33:~$ export KUBECONFIG=$HOME/admin.conf

Let's check the version of kubectl:
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"6", GitVersion:"v1.6.1", GitCommit:"b0b7a323cc5a4a2019b2e9520c21c7830b7f708e", GitTreeState:"clean", BuildDate:"2017-04-03T20:44:38Z", GoVersion:"go1.7.5", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"6", GitVersion:"v1.6.0", GitCommit:"fff5156092b56e6bd60fff75aad4dc9de6b6ef37", GitTreeState:"clean", BuildDate:"2017-03-28T16:24:30Z", GoVersion:"go1.7.5", Compiler:"gc", Platform:"linux/amd64"}

Installing a pod network - master

We must install a pod network add-on so that our pods can communicate with each other.

The network must be deployed before any applications. Also, kube-dns, a helper service, will not start up before a network is installed. kubeadm only supports CNI based networks (and does not support kubenet).

We can install a pod network add-on with the following command on master:

kubectl apply -f <add-on.yaml>

Weave Net can be installed onto our CNI-enabled Kubernetes cluster with a single command:

ubuntu@ip-172-31-60-33:~$ kubectl apply -f https://git.io/weave-kube-1.6
clusterrole "weave-net" created
serviceaccount "weave-net" created
clusterrolebinding "weave-net" created
daemonset "weave-net" created

After a few seconds, a Weave Net pod should be running on each Node and any further pods we create will be automatically attached to the Weave network.

We should only install one pod network per cluster.

Once a pod network has been installed, we can confirm that it is working by checking that the kube-dns pod is Running in the output of kubectl get pods --all-namespaces:

ubuntu@ip-172-31-60-33:~$ watch kubectl get pods --all-namespaces

Every 2.0s: kubectl get pods --all-namespaces
Wed Apr 12 22:11:50 2017

```
NAMESPACE     NAME                                    READY
STATUS    RESTARTS   AGE
kube-system   etcd-ip-172-31-60-33                    1/1
Running   0          22m
kube-system   kube-apiserver-ip-172-31-60-33          1/1
Running   0          21m
kube-system   kube-controller-manager-ip-172-31-60-33
1/1       Running   0          21m
kube-system   kube-dns-3913472980-ng8td
3/3       Running   0          22m
kube-system   kube-proxy-8chjv                        1/1
Running   0          22m
kube-system   kube-proxy-8cq1c                        1/1
Running   0          3m
kube-system   kube-scheduler-ip-172-31-60-33
1/1       Running   0          22m
kube-system   weave-net-cl6gp                         2/2
Running   0          18m
kube-system   weave-net-zdh3x                         2/2
Running   0          3m
```

We can see the weave-net is running.
Once the kube-dns pod is up and running, we can continue by joining our nodes.

Joining our nodes - on the nodes

The nodes are where our workloads (containers and pods, etc) run. To add new nodes to our cluster do the following for each machine besides our master:

kubeadm join --token <token> <master-ip>:<master-port>

In our case, we can get the command from the kubeadm init outputs:

root@ip-172-31-41-29:/home/ubuntu# kubeadm join --token 918ba6.e2cb11b266dced53 172.31.60.33:6443
[kubeadm] WARNING: kubeadm is in beta, please do not use it for production clusters.
[preflight] Running pre-flight checks
[discovery] Trying to connect to API Server "172.31.60.33:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://172.31.60.33:6443"
[discovery] Cluster info signature and contents are valid, will use API Server "https://172.31.60.33:6443"
[discovery] Successfully established connection with API Server "172.31.60.33:6443"
[bootstrap] Detected server version: v1.6.0
[bootstrap] The server supports the Certificates API (certificates.k8s.io/v1beta1)
[csr] Created API client to obtain unique certificate for this node, generating keys and certificate signing request
[csr] Received signed certificate from the API server, generating KubeConfig...
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/kubelet.conf"

Node join complete:
* Certificate signing request sent to master and response
  received.
* Kubelet informed of new secure connection details.

Run 'kubectl get nodes' on the master to see this machine join.

To see this machine join, on the master, let's run:

```
ubuntu@ip-172-31-60-33:~$ kubectl get nodes
NAME              STATUS    AGE       VERSION
ip-172-31-41-29   Ready     32s       v1.6.1
ip-172-31-60-33   Ready     19m       v1.6.1
```

Installing a sample application - on the master

Now it is time to check if our new cluster is working. Sock Shop is a sample microservices application that shows how to run and connect a set of services on Kubernetes:

```
ubuntu@ip-172-31-60-33:~$ kubectl create namespace sock-shop
namespace "sock-shop" created
```

Then, apply to install all the component of Sock Shop:

```
ubuntu@ip-172-31-60-33:~$ kubectl apply -n sock-shop -f "https://github.com/microservices-demo/microservices-demo/blob/master/deploy/kubernetes/complete-demo.yaml?raw=true"
```

Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
namespace "sock-shop" configured
namespace "zipkin" created
deployment "carts-db" created
service "carts-db" created
deployment "carts" created
service "carts" created
deployment "catalogue-db" created
service "catalogue-db" created
deployment "catalogue" created
service "catalogue" created
deployment "front-end" created
service "front-end" created
deployment "orders-db" created
service "orders-db" created
deployment "orders" created
service "orders" created
deployment "payment" created
service "payment" created
deployment "queue-master" created
service "queue-master" created
deployment "rabbitmq" created
service "rabbitmq" created
deployment "shipping" created
service "shipping" created
deployment "user-db" created
service "user-db" created
deployment "user" created
service "user" created

the namespace from the provided object "zipkin" does not match the namespace "sock-shop". You must pass '--namespace=zipkin' to perform this operation.
the namespace from the provided object "zipkin" does not match the namespace "sock-shop". You must pass '--namespace=zipkin' to perform this operation.
the namespace from the provided object "zipkin" does not match the namespace "sock-shop". You must pass '--namespace=zipkin' to perform this operation.
the namespace from the provided object "zipkin" does not match the namespace "sock-shop". You must pass '--namespace=zipkin' to perform this operation.
the namespace from the provided object "zipkin" does not match the namespace "sock-shop". You must pass '--namespace=zipkin' to perform this operation.

We can then find out the port that the NodePort feature of services allocated for the front-end service by running:

```
ubuntu@ip-172-31-60-33:~$ kubectl -n sock-shop get svc front-end
NAME       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
front-end  10.97.82.211   <nodes>       80:30001/TCP     8m
```

It takes several minutes to download and start all the containers, watch the output of kubectl get pods -n sock-shop to see when they're all up and running:

```
ubuntu@ip-172-31-60-33:~$ kubectl get pods -n sock-shop
NAME                          READY     STATUS    RESTARTS   AGE
```

```
carts-153328538-dt7hc           1/1       Running   0
13m
carts-db-4256839670-cph4g       1/1       Running   0
13m
catalogue-114596073-1rt4p       1/1       Running   0
13m
catalogue-db-1956862931-v87jr   1/1       Running   0
13m
front-end-3570328172-l2m00      1/1       Running   0
13m
orders-2365168879-93fpx         1/1       Running   0
13m
orders-db-836712666-pfnmq       1/1       Running   0
13m
payment-1968871107-3pdtd        1/1       Running   0
13m
queue-master-2798459664-p8vzt   1/1       Running   0
13m
rabbitmq-3429198581-n25zq       1/1       Running   0
13m
shipping-2899287913-749pj       1/1       Running   0
13m
user-468431046-rhtq2            1/1       Running   0
13m
user-db-1166754267-f6gd2        1/1       Running   0
13m
```

Then go to the IP address of our cluster's master node in our browser, and specify the given port. So for example, http://<master_ip>:<port>. In the example above, this was 30001, but the port may be a different.