Kubernetes cluster step-by-step: ETCD

5 minute read , Jun 14, 2017

The purpose of this exercise is to create local `Kubernetes` cluster for testing deployments.
It will be deployed on 3 x VMs (Debian Jessie 8.8) nodes which will be Master and Worker nodes in same time. The nodes names will be k8s01 (192.168.0.147), k8s02 (192.168.0.148) and k8s03 (192.168.0.149). All work is done as `root` user unless otherwise specified. Each node has the IPs, short and FQDN of all the nodes set in its local hosts file.

Etcd

ETCD is a highly-available distributed KV storage that many of the K8S cluster services, addons (like Traefik for example) and network plugins (except Weave AFAIK) use for storing their configuration.

Option 1: Run as `Systemd` service

This is the option I went with. Prepare the directories and create the `etcd` user (all commands assumed being run as `root` user):

```
mkdir -p /var/lib/etcd
groupadd -f -g 1501 etcd
useradd -c "Etcd key-value store user" -d /var/lib/etcd -s /bin/false -g etcd -u 1501 etcd
chown -R etcd:etcd /var/lib/etcd
```

Download and install it on each of the nodes:

```
cd /usr/local/src
curl -ksL 'https://github.com/coreos/etcd/releases/download/v2.3.8/etcd-v2.3.8-linux-amd64.tar.gz' | tar -xzvf -
cp etcd-v2.3.8-linux-amd64/etcd* /usr/local/bin
```

Create the service on each node. On `k8s01` :

```
cat << EOF > /lib/systemd/system/etcd.service
[Unit]
Description=etcd key-value store
Documentation=https://github.com/coreos/etcd

[Service]
User=etcd
Type=notify
ExecStart=/usr/local/bin/etcd \\
 --name etcd0 \\
 --data-dir /var/lib/etcd \\
 --initial-advertise-peer-urls http://192.168.0.147:2380 \\
 --listen-peer-urls http://192.168.0.147:2380 \\
 --listen-client-urls http://192.168.0.147:2379,http://127.0.0.1:2379,http://192.168.0.147:4001 \\
 --advertise-client-urls http://192.168.0.147:2379,http://192.168.0.147:4001 \\
 --initial-cluster-token etcd-cluster-1 \\
 --initial-cluster etcd0=http://192.168.0.147:2380,etcd1=http://192.168.0.148:2380,etcd2=http://192.168.0.149:2380 \\
 --initial-cluster-state new \\
 --heartbeat-interval 1000 \\
 --election-timeout 5000
Restart=always
RestartSec=10s
LimitNOFILE=40000

[Install]
WantedBy=multi-user.target
EOF
```

On `k8s02`:

```
cat << EOF > /lib/systemd/system/etcd.service
[Unit]
Description=etcd key-value store
Documentation=https://github.com/coreos/etcd

[Service]
User=etcd
Type=notify
ExecStart=/usr/local/bin/etcd \\
 --name etcd1 \\
```

```
 --data-dir /var/lib/etcd \\
 --initial-advertise-peer-urls http://192.168.0.148:2380 \\
 --listen-peer-urls http://192.168.0.148:2380 \\
 --listen-client-urls http://192.168.0.148:2379,http://127.0.0.1:2379,http://192.168.0.148:4001 \\
 --advertise-client-urls http://192.168.0.148:2379,http://192.168.0.148:4001 \\
 --initial-cluster-token etcd-cluster-1 \\
 --initial-cluster etcd0=http://192.168.0.147:2380,etcd1=http://192.168.0.148:2380,etcd2=http://192.168.0.149:2380 \\
 --initial-cluster-state new \\
 --heartbeat-interval 1000 \\
 --election-timeout 5000
Restart=always
RestartSec=10s
LimitNOFILE=40000

[Install]
WantedBy=multi-user.target
EOF
```

On `k8s03`:

```
cat << EOF > /lib/systemd/system/etcd.service
[Unit]
Description=etcd key-value store
Documentation=https://github.com/coreos/etcd

[Service]
User=etcd
Type=notify
ExecStart=/usr/local/bin/etcd \\
 --name etcd2 \\
 --data-dir /var/lib/etcd \\
 --initial-advertise-peer-urls http://192.168.0.149:2380 \\
 --listen-peer-urls http://192.168.0.149:2380 \\
 --listen-client-urls http://192.168.0.149:2379,http://127.0.0.1:2379,http://192.168.0.149:4001 \\
 --advertise-client-urls http://192.168.0.149:2379,http://192.168.0.149:4001 \\
 --initial-cluster-token etcd-cluster-1 \\
 --initial-cluster etcd0=http://192.168.0.147:2380,etcd1=http://192.168.0.148:2380,etcd2=http://192.168.0.149:2380 \\
 --initial-cluster-state new \\
 --heartbeat-interval 1000 \\
 --election-timeout 5000
Restart=always
```

```
RestartSec=10s
LimitNOFILE=40000

[Install]
WantedBy=multi-user.target
EOF
```

Start and enable it:

```
systemctl daemon-reload
systemctl enable etcd
systemctl start etcd.service
systemctl status -l etcd.service
```

Check the cluster state:

```
root@k8s01:~# etcdctl -C http://192.168.0.147:2379,http://192.168.0.148:2379,http://192.168.0.149:2379 member list
b7c5b6db978f96c7: name=etcd2 peerURLs=http://192.168.0.149:2380 clientURLs=http://192.168.0.149:2379,http://192.168.0.149:4001 isL
ba7740fb7809ba12: name=etcd0 peerURLs=http://192.168.0.147:2380 clientURLs=http://192.168.0.147:2379,http://192.168.0.147:4001 isL
e969715163dbaf84: name=etcd1 peerURLs=http://192.168.0.148:2380 clientURLs=http://192.168.0.148:2379,http://192.168.0.148:4001 isL

root@k8s01:~# etcdctl cluster-health
member b7c5b6db978f96c7 is healthy: got healthy result from http://192.168.0.149:2379
member ba7740fb7809ba12 is healthy: got healthy result from http://192.168.0.147:2379
member e969715163dbaf84 is healthy: got healthy result from http://192.168.0.148:2379
cluster is healthy
```

Option 2: Run as Kubernetes Pod

For the services Kubernetes needs to run inside the cluster, we create manifest files under directory configured for that purpose like `/etc/kubernetes/manifests`. For `etcd` that file on `k8s01` node might look like this:

```
# /etc/kubernetes/manifests/etcd.manifest
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    k8s-app: etcd-server
  name: etcd-server
  namespace: kube-system
spec:
  containers:
  - command:
    - /bin/sh
    - -c
    - /usr/local/bin/etcd 1>>/var/log/etcd.log 2>&1
    env:
    - name: ETCD_NAME
      value: k8s01
    - name: ETCD_DATA_DIR
      value: /var/etcd/data
    - name: ETCD_LISTEN_PEER_URLS
      value: http://0.0.0.0:2380
  - name: ETCD_LISTEN_CLIENT_URLS
      value: http://0.0.0.0:4001
  - name: ETCD_ADVERTISE_CLIENT_URLS
      value: http://k8s01.virtual.local:4001
  - name: ETCD_INITIAL_ADVERTISE_PEER_URLS
      value: http://k8s01.virtual.local:2380
  - name: ETCD_INITIAL_CLUSTER_STATE
      value: new
  - name: ETCD_INITIAL_CLUSTER_TOKEN
      value: etcd-cluster-token-etcd
  - name: ETCD_INITIAL_CLUSTER
      value: etcd-a=http://k8s01.virtual.local:2380,etcd-b=http://k8s02.virtual.local:2380,etcd-c=http://k8s03.virtual.local:2380
  image: gcr.io/google_containers/etcd:2.2.1
  livenessProbe:
    httpGet:
      host: 127.0.0.1
      path: /health
      port: 4001
    initialDelaySeconds: 15
    timeoutSeconds: 15
```

```
    name: etcd-container
    ports:
    - containerPort: 2380
      hostPort: 2380
      name: serverport
    - containerPort: 4001
      hostPort: 4001
      name: clientport
    resources:
      requests:
        cpu: 200m
    volumeMounts:
    - mountPath: /var/etcd/data
      name: varetcdata
    - mountPath: /var/log/etcd.log
      name: varlogetcd
  hostNetwork: true
  volumes:
  - hostPath:
      path: /mnt/var/etcd/data
    name: varetcdata
  - hostPath:
      path: /var/log/etcd.log
    name: varlogetcd
  status: {}
```

Option 3 : Run as Docker container

`Systemd` service file in case we want to use Docker to run `etcd` :

```
[Unit]
Description=etcd container
After=docker.service

[Service]
Restart=always
RestartSec=10s
LimitNOFILE=40000
```

```
EnvironmentFile=/etc/environment
ExecStartPre=-/usr/bin/docker kill etcd
ExecStartPre=-/usr/bin/docker rm etcd
#ExecStartPre=/usr/bin/docker pull quay.io/coreos/etcd:v2.3.8
ExecStart=/usr/bin/docker run -d
 -v /usr/share/ca-certificates/:/srv/kubernetes
 -p 4001:4001 -p 2380:2380 -p 2379:2379
 --name etcd quay.io/coreos/etcd:v2.3.8
 -e ETCD_NAME=$(hostname -s)
 -e ETCD_ADVERTISE_CLIENT_URLS=http://192.168.0.149:2379,http://192.168.0.149:4001
 -e ETCD_LISTEN_CLIENT_URLS=http://0.0.0.0:2379,http://0.0.0.0:4001
 -e ETCD_INITIAL_ADVERTISE_PEER_URLS=http://192.168.0.149:2380
 -e ETCD_LISTEN_PEER_URLS=http://0.0.0.0:2380
 -e ETCD_INITIAL_CLUSTER_TOKEN=etcd-cluster-1
 -e ETCD_INITIAL_CLUSTER=k8s01=http://192.168.0.147:2380,k8s02=http://192.168.0.148:2380,k8s03=http://192.168.0.149:2380 \\
 -e ETCD_INITIAL_CLUSTER_STATE=new
 -e ETCD_HEARTBEAT_INTERVAL=1000
 -e ETCD_ELECTION_TIMEOUT=5000
ExecStop=/usr/bin/docker stop etcd

[Install]
WantedBy=multi-user.target
```

OPTIONAL: Creating `etcd` peer certificates

The procedure is for `k8s01` node, repeat the same for `k8s02` and `k8s03`. First, create one certificate signing request (CSR) for the peer certificate for each etcd node.

```
cd /srv/kubernetes
cat > openssl-etcd.cnf << "EOF"
[req]
req_extensions = v3_req
distinguished_name = req_distinguished_name
[req_distinguished_name]
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

```
extendedKeyUsage = clientAuth,serverAuth
subjectAltName = IP:$ENV::ETCD_IP, DNS:$ENV::ETCD_FQDN
EOF
```

and run openssl to create the CSR:

```
openssl genrsa -out etcd.key 2048
useradd -U -s /bin/false etcd
chmod 600 etcd.key
chown etcd:etcd etcd.key
export ETCD_IP=192.168.0.147
export ETCD_FQDN=$(hostname --fqdn)
openssl \
  req -new \
  -key etcd.key \
  -out etcd.csr \
  -subj "/CN=$(hostname -s)" \
  -extensions v3_req \
  -config openssl-etcd.cnf \
  -sha256
```

Now we sign the CSRs, I'm using the previously created K8S CA as signing CA:

```
openssl x509 -req -sha256 -CA ca.pem -CAkey ca-key.pem -CAcreateserial \
  -in etcd.csr -out etcd.crt -extensions v3_req -extfile openssl-etcd.cnf -days 7200
```

In case we decide to enable SSL and certificate authentication between the peers and for the clients, we can considered the following `etcd` parameters too:

```
  --client-cert-auth \
  --trusted-ca-file /srv/kubernetes/ca.pem \
  --cert-file /srv/kubernetes/etcd.crt \
  --key-file /srv/kubernetes/etcd.key \
  --peer-client-cert-auth \
```

```
    --peer-trusted-ca-file /srv/kubernetes/ca.pem \
    --peer-cert-file /srv/kubernetes/etcd.crt \
    --peer-key-file /srv/kubernetes/etcd.key
```

to introduce the peer and the client certificates. Now the command to verify the hosts would be:

```
sudo -u etcd etcdctl --cert-file /srv/kubernetes/etcd.crt --key-file /srv/kubernetes/etcd.key --ca-file /srv/kubernetes/ca.pem clu
```

References

- Play etcd

This article is **Part 3** in a **8-Part** Series **Kubernetes cluster step-by-step**.

- Part 1 - Kubernetes cluster step-by-step: Nodes System Setup
- Part 2 - Kubernetes cluster step-by-step: Binaries, Certificates, Kubeconfig and Tokens
- Part 3 - This Article
- Part 4 - Kubernetes cluster step-by-step: FlannelD
- Part 5 - Kubernetes cluster step-by-step: Kube-apiserver with Keepalived and HAProxy for HA
- Part 6 - Kubernetes cluster step-by-step: Kubelet, Kube-scheduler and Kube-controller-manager
- Part 7 - Kubernetes cluster step-by-step: Kubernetes Add-ons
- Part 8 - Kubernetes cluster step-by-step: Services and Load Balancing