6. Kubernetes Deployment with MCP

In this chapter, we will go over Kubernetes deployment process.

Chapter Details	
Chapter Goal	Deploy and use a standalone Kubernetes cluster
Chapter Sections	*6.1. Prepare virtual lab* *6.2. Deploy K8s* *6.3.4. Kubernetes Hands-on*

6.1. Prepare virtual lab

You're starting from a clean VM that will act like a physical host. We will create several VMs: master node, 3 controllers, 1 compute. MCP service will be deployed over that set of VMs.

6.1.1. Get the model

Step 1 Download the model and provisioning scripts:

```
stack@lab:~$ git clone git@bitbucket.org:mirantis-training/mcp100-k8s-calico.git
```

Enter Yes if it asks about wrong fingerprints.

Change the directory;

```
stack@lab:~$ cd mcp100-k8s-calico/
```

Step 2 Install prerequisites:

```
stack@lab:~/mcp100-k8s-calico$ sudo apt-get update
stack@lab:~/mcp100-k8s-calico$ sudo apt-get install -y git mkisofs curl virtinst cpu-ch
```

6.1.2. Configure networking

We will need several virtual networks for successful deployment:

Step 1 To create networks execute provided script:

```
stack@lab:~/mcp100-k8s-calico$ ./provision-networks.sh
```

This will create *br_pxe*, *br_mgmt*, *br_ctl*, *br_ext* networks which correspond to PXE, management, control and external

6.1.3. Prepare cfg01

The deployment will start from preparing configuration ISO which will be used later to bootstrap master node. We will prepare the model, download some handy scripts and generate ISO.

Step 1 Copy the model:

```
stack@lab:~/mcp100-k8s-calico$ sudo su
root@lab:/home/stack/mcp100-k8s-calico# mkdir /root/model
root@lab:/home/stack/mcp100-k8s-calico# cp -rT /home/stack/mcp100-k8s-calico/ /root/moc
```

Step 2 Download *cfg01* image:

```
root@lab:/home/stack/mcp100-k8s-calico# wget http://repos/cfg01-day01-2018.8.0.qcow2 -O
/pool/images/cfg01.k8s-ha-calico.local.img
```

The image will be used to bootstrap master node, however before starting VM we need to generate configuration ISO.

Step 3 Download generation script and cloud-init for ISO:

Choose MCP version:

```
root@lab:/home/stack/mcp100-k8s-calico# export MCP_VERSION="2018.8.0"
```

Download ISO generation script and set permissions:

```
root@lab:/home/stack/mcp100-k8s-calico# wget -O /root/create-config-drive \
http://repos/create_config_drive_${MCP_VERSION}.sh

root@lab:/home/stack/mcp100-k8s-calico# chmod +x /root/create-config-drive
```

Download cloud-init:

```
root@lab:/home/stack/mcp100-k8s-calico# wget -O /root/user_data.yaml \
http://repos/master_config_${MCP_VERSION}.yaml
```

Step 4 Make the following changes to `/root/user_data.yaml`:

```
export SALT_MASTER_DEPLOY_IP=${SALT_MASTER_DEPLOY_IP:-"192.168.10.100"}
export SALT_MASTER_MINION_ID=${SALT_MASTER_MINION_ID:-"cfg01.k8s-ha-calico.local"}
export DEPLOY_NETWORK_GW=${DEPLOY_NETWORK_GW:-"192.168.10.1"}
export DEPLOY_NETWORK_NETMASK=${DEPLOY_NETWORK_NETMASK:-"255.255.255.0"}
```

Step 5 Download Drivetrain pipelines:

```
root@lab:/home/stack/mcp100-k8s-calico# git clone --mirror https://github.com/Mirantis/

root@lab:/home/stack/mcp100-k8s-calico# git clone --mirror https://github.com/Mirantis/
```

Step 6 Check that you have all files required:

```
root@lab:/home/stack/mcp100-k8s-calico# ls /root/
create-config-drive  mk-pipelines  model  pipeline-library  user_data.yaml
```

Step 7 Generate configuration ISO:

```
root@lab:/home/stack/mcp100-k8s-calico# /root/create-config-drive -u /root/user_data.ya
--model /root/model --mk-pipelines /root/mk-pipelines \
--pipeline-library /root/pipeline-library /pool/images/cfg01.k8s-ha-calico.local.iso
```

Configuration ISO should be ready and can be found at `/pool/images/cfg01.k8s-ha-calico.local.iso`

6.1.4. Create cfg01 VM

Step 1 Use *virt-install* to create *cfg01* VM:

```
root@lab:/home/stack/mcp100-k8s-calico# virt-install --name cfg01.k8s-ha-calico.local \
--disk path=/pool/images/cfg01.k8s-ha-calico.local.img,bus=virtio,format=qcow2,cache=nc
--disk path=/pool/images/cfg01.k8s-ha-calico.local.iso,device=cdrom \
--network network:br_pxe,model=virtio \
--network network:br_mgmt,model=virtio \
--network network:br_ctl,model=virtio \
--network network:br_ext,model=virtio \
--ram 8192 --vcpus=4 --accelerate \
--boot hd --vnc --noreboot --autostart

Starting install...
Creating domain...
Domain creation completed. You can restart your domain by running:
  virsh --connect qemu:///system start cfg01.k8s-ha-calico.local
```

The VM is created but not started yet. We will need a small update to configuration.

Step 2 Update networking:

```
root@lab:/home/stack/mcp100-k8s-calico# virsh net-update br_pxe add ip-dhcp-host "\
<host mac='$(virsh domiflist cfg01.k8s-ha-calico.local | grep br_pxe | awk '{print $5}'
" --live --config
```

We've changed the record in libvirt DHCP service to get a fixed IP address for *cfg01*.

Step 3 Start cfg01 VM:

```
root@lab:/home/stack/mcp100-k8s-calico# virsh start cfg01.k8s-ha-calico.local
```

Step 4 Login to *cfg01* and wait until cloud-init is over. The process takes about 10 min to complete. VM should perform a restart at the end:

```
root@cfg01:~# virsh console cfg01.k8s-ha-calico.local
```

Step 5 Perform initial provisioning of salt master:

```
root@cfg01:~# salt '*' saltutil.sync_all

root@cfg01:~# salt-call state.sls linux.system,linux,openssh,salt,reclass

root@cfg01:~# salt '*' saltutil.sync_all
```

Step 6 Update pipelines:

```
root@cfg01:~# pushd /home/repo/mk/mk-pipelines && \
git init && git add -A && git commit -m 'initial commit' ; popd

root@cfg01:~# pushd /home/repo/mcp-ci/pipeline-library && \
git init && git add -A && git commit -m 'initial commit' ; popd
```

Step 7 Exit to the host by pressing *CTRL +]*:

```
root@lab:/home/stack/mcp100-k8s-calico#
```

6.1.5. Create other VMs

Step 1 Create VMs:

```
./provision-cluster.sh
```

The script will spawn several new VMs from the base ubuntu cloud image. The images will be supplied with cloud init that installs salt-minion and targets to the salt-master.

6.2. Deploy K8s

At this point your environment should get master and minions ready for deployment. We will run through the group of the salt states which provision minions and bring the system to desired state.

6.2.1. Setup base infrastructure

The next states will setup infrastructure trough the cluster

Step 1 Login back to the *cfg01*:

```
root@lab:/home/stack/mcp100-k8s-calico# virsh console cfg01.k8s-ha-calico.local

root@cfg01:~#
```

Step 2 Configure Linux on the nodes:

```
root@cfg01:~# salt -C 'I@docker:host' state.sls linux.system

root@cfg01:~# salt -C 'I@kubernetes:master' state.sls  linux
```

Step 3 Bootstrap the Kubernetes Master nodes. Update salt-minion, updated grains and pillars, configure SSH and NTP:

```
root@cfg01:~# salt -C 'I@kubernetes:master' state.sls salt.minion

root@cfg01:~# salt -C 'I@kubernetes:master' state.sls openssh,ntp
```

Step 4 Install Docker:

```
root@cfg01:~# salt -C 'I@docker:host' state.sls docker.host
```

Step 5 Create and distribute SSL certificates for services using the salt state and install etcd with the SSL support:

```
root@cfg01:~# salt -C 'I@kubernetes:master' state.sls salt.minion.cert,etcd.server.serv

root@cfg01:~# salt -C 'I@etcd:server' cmd.run '. /var/lib/etcd/configenv && etcdctl clu
ctl03.k8s-ha-calico.local:
    member 3bce817bace0095a is healthy: got healthy result from https://192.168.10.102:
    member 9320de8d701d7606 is healthy: got healthy result from https://192.168.10.101:
    member e1152b408b886936 is healthy: got healthy result from https://192.168.10.103:
    cluster is healthy
ctl01.k8s-ha-calico.local:
    member 3bce817bace0095a is healthy: got healthy result from https://192.168.10.102:
    member 9320de8d701d7606 is healthy: got healthy result from https://192.168.10.101:
    member e1152b408b886936 is healthy: got healthy result from https://192.168.10.103:
    cluster is healthy
ctl02.k8s-ha-calico.local:
    member 3bce817bace0095a is healthy: got healthy result from https://192.168.10.102:
    member 9320de8d701d7606 is healthy: got healthy result from https://192.168.10.101:
    member e1152b408b886936 is healthy: got healthy result from https://192.168.10.103:
    cluster is healthy
```

Step 6 Setup HA and load balancing, install keepalived and haproxy:

```
root@cfg01:~# salt -C 'I@keepalived:cluster' state.sls keepalived -b 1

root@cfg01:~# salt -C 'I@haproxy:proxy' state.sls haproxy
```

Check the status of HAProxy:

```
root@cfg01:~# salt -C 'I@haproxy:proxy' service.status haproxy
```

Step 7 Install kubernetes:

```
root@cfg01:~# salt -C 'I@kubernetes:master' state.sls kubernetes.pool

root@cfg01:~# salt -C 'I@kubernetes:master' state.sls kubernetes.master.kube-addons
```

Step 8 Check Calico status:

```
root@cfg01:~# salt -C 'I@kubernetes:pool' cmd.run "calicoctl node status"
ctl01.k8s-ha-calico.local:
    Calico process is running.

    IPv4 BGP status
    +---------------+-------------------+-------+----------+-------------+
    |  PEER ADDRESS |     PEER TYPE     | STATE |  SINCE   |    INFO     |
    +---------------+-------------------+-------+----------+-------------+
    | 192.168.10.102 | node-to-node mesh | up    | 02:05:19 | Established |
    | 192.168.10.103 | node-to-node mesh | up    | 02:05:20 | Established |
    +---------------+-------------------+-------+----------+-------------+

    IPv6 BGP status
    No IPv6 peers found.
ctl03.k8s-ha-calico.local:
    Calico process is running.

    IPv4 BGP status
    +---------------+-------------------+-------+----------+-------------+
    |  PEER ADDRESS |     PEER TYPE     | STATE |  SINCE   |    INFO     |
    +---------------+-------------------+-------+----------+-------------+
    | 192.168.10.102 | node-to-node mesh | up    | 02:05:07 | Established |
```

```
    | 192.168.10.101 | node-to-node mesh | up    | 02:05:21 | Established |
    +----------------+-------------------+-------+----------+-------------+

    IPv6 BGP status
    No IPv6 peers found.
ctl02.k8s-ha-calico.local:
    Calico process is running.

    IPv4 BGP status
    +----------------+-------------------+-------+----------+-------------+
    |  PEER ADDRESS  |     PEER TYPE     | STATE |  SINCE   |    INFO     |
    +----------------+-------------------+-------+----------+-------------+
    | 192.168.10.103 | node-to-node mesh | up    | 02:05:08 | Established |
    | 192.168.10.101 | node-to-node mesh | up    | 02:05:20 | Established |
    +----------------+-------------------+-------+----------+-------------+

    IPv6 BGP status
    No IPv6 peers found.
```

Step 9 Update etcd settings:

```
root@cfg01:~# salt -C 'I@kubernetes:master' state.sls etcd.server.setup
```

Step 10 Configure kubernetes namespaces and run the Kubernetes Master nodes setup:

```
root@cfg01:~# salt -C 'I@kubernetes:master and *01*' state.sls kubernetes.master \
exclude=kubernetes.master.setup

root@cfg01:~# salt -C 'I@kubernetes:master' state.sls kubernetes exclude=kubernetes.mas

root@cfg01:~# salt -C 'I@kubernetes:master and *01*' state.sls kubernetes.master.setup

root@cfg01:~# salt -C 'I@kubernetes:master' service.restart kubelet
```

Repeat the command if anyone fails.

Step 11 Bootstrap kubernetes nodes:

```
root@cfg01:~# salt -C 'I@kubernetes:pool and not I@kubernetes:master' state.sls linux

root@cfg01:~# salt -C 'I@kubernetes:pool and not I@kubernetes:master' state.sls salt.mi

root@cfg01:~# salt -C 'I@kubernetes:pool and not I@kubernetes:master' state.sls openssh
```

Step 12 Create and distribute SSL certificates. Install etcd:

```
root@cfg01:~# salt -C 'I@kubernetes:pool and not I@kubernetes:master' state.sls salt.mi

root@cfg01:~# salt -C 'I@etcd:server' cmd.run '. /var/lib/etcd/configenv && etcdctl clu

ctl02.k8s-ha-calico.local:
    member 3bce817bace0095a is healthy: got healthy result from https://192.168.10.102:
    member 9320de8d701d7606 is healthy: got healthy result from https://192.168.10.101:
    member e1152b408b886936 is healthy: got healthy result from https://192.168.10.103:
    cluster is healthy
ctl01.k8s-ha-calico.local:
    member 3bce817bace0095a is healthy: got healthy result from https://192.168.10.102:
    member 9320de8d701d7606 is healthy: got healthy result from https://192.168.10.101:
    member e1152b408b886936 is healthy: got healthy result from https://192.168.10.103:
    cluster is healthy
ctl03.k8s-ha-calico.local:
    member 3bce817bace0095a is healthy: got healthy result from https://192.168.10.102:
```

```
member 9320de8d701d7606 is healthy: got healthy result from https://192.168.10.101:
member e1152b408b886936 is healthy: got healthy result from https://192.168.10.103:
cluster is healthy
```

Step 13 Install docker:

```
root@cfg01:~# salt -C 'I@docker:host' state.sls docker.host
```

Step 13 Install Kubernetes:

```
root@cfg01:~# salt -C 'I@kubernetes:pool and not I@kubernetes:master' state.sls kuberne

root@cfg01:~# salt -C 'I@kubernetes:pool and not I@kubernetes:master' service.restart k
```

Step 14 Verify installation

```
root@cfg01:~# salt -C 'I@kubernetes:master' cmd.run 'kubectl get nodes'

ctl03.k8s-ha-calico.local:
    NAME      STATUS    ROLES     AGE       VERSION
    cmp01     Ready     node      35d       v1.11.3-2+c0bd03cdec793a
    ctl01     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
    ctl02     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
    ctl03     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
ctl02.k8s-ha-calico.local:
    NAME      STATUS    ROLES     AGE       VERSION
    cmp01     Ready     node      35d       v1.11.3-2+c0bd03cdec793a
    ctl01     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
    ctl02     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
    ctl03     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
ctl01.k8s-ha-calico.local:
    NAME      STATUS    ROLES     AGE       VERSION
    cmp01     Ready     node      35d       v1.11.3-2+c0bd03cdec793a
    ctl01     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
    ctl02     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
    ctl03     Ready     master    35d       v1.11.3-2+c0bd03cdec793a
```

6.3. Operate Kubernetes

The Default model has a configuration with a single compute node and control plane deployed on 3 controllers in HA mode. In the next exercises you will be doing a basic operation tasks on the cluster: add compute node, run service etc...

6.3.1. Add new worker/minion

As MCP is following infrastructure as a code approach you will need to change the code in order to change infrastructure. The process of updating the model should typically pass through DriveTrain, including code review, pipelines etc. This environment is focused on Kubernetes and doesn't contain full DriveTrain deployment. We will have to change reclass model to use local repo on *cfg01* instead of remote.

Step 1 Login to *cfg01* and modify reclass model to convert it to local repo:

```
root@lab:/home/stack/mcp100-k8s-calico# virsh console cfg01.k8s-ha-calico.local

root@cfg01:~# cd /srv/salt/reclass
```

Initialize new git repo and commit all files:

```
root@cfg01:/srv/salt/reclass# git init
root@cfg01:/srv/salt/reclass# git add -A
root@cfg01:/srv/salt/reclass# git config --global user.email "you@example.com"
root@cfg01:/srv/salt/reclass# git commit -m 'init commit'
```

After initializing local repo we can make changes to the model without submitting code review.

Step 2 Open infrastructure configuration file:

```
root@cfg01:~# vi /srv/salt/reclass/classes/cluster/k8s-ha-calico/infra/config.yml
```

and add new *kubernetes_compute_node02* param to the *reclass.storage.node* section:

```
parameters:
  ...
  reclass:
    storage:
      node:
        kubernetes_compute_node02:
          name: ${_param:kubernetes_compute_node02_hostname}
          domain: ${_param:cluster_domain}
          classes:
          - cluster.${_param:cluster_name}.kubernetes.compute
          params:
            salt_master_host: ${_param:reclass_config_master}
            linux_system_codename: xenial
            deploy_address: ${_param:kubernetes_compute_node02_deploy_address}
            single_address: ${_param:kubernetes_compute_node02_single_address}
```

The section *reclass.storage.node* of the model does declare which node definition files will be created and used as entry points of the model. After appending to that section and applying reclass state you will get new definition file created in `/srv/reclass/nodes/_generated` directory. This file will be the entry point for new node's role.

Step 3 Open init file of the model:

```
root@cfg01:~# vi /srv/salt/reclass/classes/cluster/k8s-ha-calico/kubernetes/init.yml
```

append addresses and hostname to *_param* section:

```
parameters:
  _param:
    ...
    # addresses and hostnames
    kubernetes_compute_node02_hostname: cmp02
    kubernetes_compute_node02_deploy_address: 192.168.10.105
    kubernetes_compute_node02_single_address: 172.16.10.105
```

Add new *cmp01* host to *linux.network.host* section in the same file:

```
parameters:
  ...
  linux:
    network:
      host:
        cmp02:
          address: ${_param:kubernetes_compute_node02_single_address}
          names:
```

```
    - ${_param:kubernetes_compute_node02_hostname}
    - ${_param:kubernetes_compute_node02_hostname}.${_param:cluster_domain}
```

After all the code is ready. We will need to regenerate node definitions and run pipeline(or execute states manually)

Step 4 Run `reclass` state to generate new node definition:

```
root@cfg01:~# salt-call state.sls reclass
```

If command returns error please check your changes in the model. A wrong indent is a typical mistake.

Step 5 Exit to the host by pressing *CTRL +]*:

```
root@lab:/home/stack/mcp100-k8s-calico#
```

and create new VM using provided script:

```
stack@lab:~$ cd /home/stack/mcp100-k8s-calico && sudo k8s3/extend-cluster.sh
```

Wait for ~2 min to complete bootstrapping of the vm

Step 6 Login to *cfg01*:

```
root@lab:/home/stack/mcp100-k8s-calico# virsh console cfg01.k8s-ha-calico.local
```

and check that new *cmp01* node is discovered by salt:

```
root@cfg01:~# watch salt-key
Accepted Keys:
cfg01.k8s-ha-calico.local
cmp01.k8s-ha-calico.local
cmp02.k8s-ha-calico.local
ctl01.k8s-ha-calico.local
ctl02.k8s-ha-calico.local
ctl03.k8s-ha-calico.local
Denied Keys:
Unaccepted Keys:
Rejected Keys:
```

wait until `cmp02.k8s-ha-calico.local` appear in the list then exit using CTRL+C. In the next step please choose the way how you want to update your environment - either using pipeline or manual command execution.

6.3.2. Using DriveTrain pipelines to add a compute node

The default “Deploy - Openstack” pipeline can be reused for adding a new compute node. Choose this way if you don’t want to manually execute each deployment command. For manual execution proceed to *6.3.3. Adding a compute node using manual commands*

Step 1 Update Jenkins credentials:

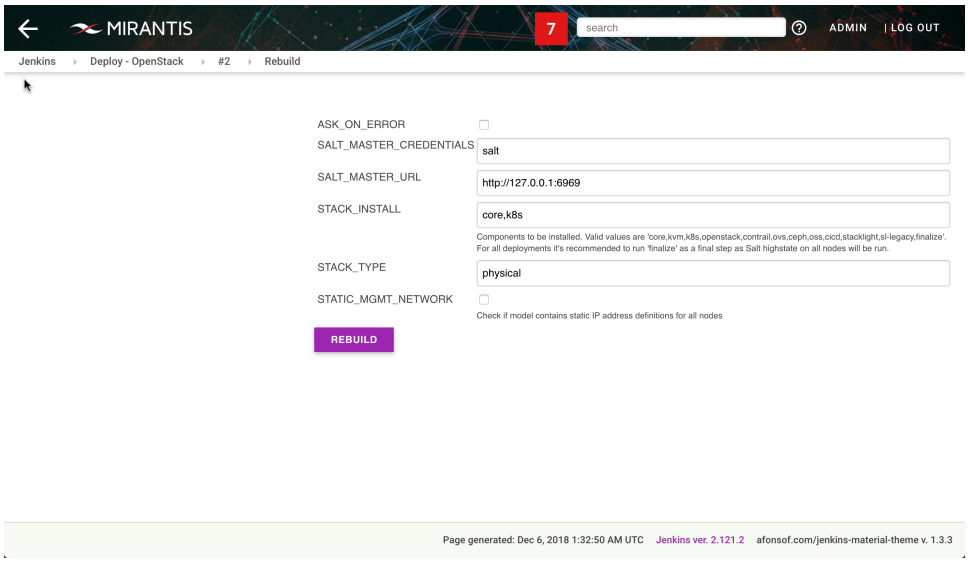
```
root@cfg01:~# salt-call state.sls jenkins
```

This will setup jenkins to be able to use Salt API.

Step 2 Open browser and go to http://192.168.10.100:8081/

Login and enter **admin** as login and **r00tme** as password.

Step 3 Launch “Deploy - Openstack” pipeline. Change the value of *STACK_INSTALL* parameter to *core,k8s* and leave other parameters as default:



“Deployment - Openstack” is a general deployment job for Kubernetes and Openstack. It can also be used to extend Kubernetes cluser.

Step 4 Wait until pipeline is done and verify kubernetes nodes:

```
root@cfg01:~# salt -C 'ctl01*' cmd.run 'kubectl get nodes'
ctl01.k8s-ha-calico.local:
    NAME      STATUS    ROLES     AGE       VERSION
    cmp01     Ready     node      29d       v1.11.3-2+c0bd03cdec793a
    cmp02     Ready     node      1m        v1.11.3-2+c0bd03cdec793a
    ctl01     Ready     master    29d       v1.11.3-2+c0bd03cdec793a
    ctl02     Ready     master    29d       v1.11.3-2+c0bd03cdec793a
    ctl03     Ready     master    29d       v1.11.3-2+c0bd03cdec793a
```

Proceed to chapter *6.3.4. Kubernetes Hands-on*

6.3.3. Adding a compute node using manual commands

You can always add a compute node manually by executing the sequence of salt commands.

Step 1 Update infrastructure on *cmp02*:

```
root@cfg01:~# salt 'cmp02*' state.apply salt
root@cfg01:~# salt 'cmp02*' state.apply linux,ntp,openssh,git
```

Step 2 Install Docker and Kubernetes on *cmp02*:

```
root@cfg01:~# salt 'cmp02*' state.sls docker.host
root@cfg01:~# salt 'cmp02*' state.sls kubernetes.pool
root@cfg01:~# salt 'cmp02*' service.restart 'kubelet'
```

Step 3 Update infrastructure on the whole cluster:

```
root@cfg01:~# salt 'cmp02*' state.apply salt
root@cfg01:~# salt '*' state.apply linux.network.host
```

Step 4 Verify new node:

```
root@cfg01:~# salt -C 'ctl01*' cmd.run 'kubectl get nodes'
ctl01.k8s-ha-calico.local:
    NAME      STATUS    ROLES     AGE       VERSION
    cmp01     Ready     node      29d       v1.11.3-2+c0bd03cdec793a
    cmp02     Ready     node      1m        v1.11.3-2+c0bd03cdec793a
    ctl01     Ready     master    29d       v1.11.3-2+c0bd03cdec793a
    ctl02     Ready     master    29d       v1.11.3-2+c0bd03cdec793a
    ctl03     Ready     master    29d       v1.11.3-2+c0bd03cdec793a
```

6.3.4. Kubernetes Hands-on

Let's test our environment using simple deployment.

Step 1 ssh to k8s master node *ctl01*:

```
root@cfg01:~# ssh ctl01
```

Step 2 Launch deployment of echoserver with 2 replicas:

```
root@ctl01:~# kubectl run echoserver \
--image=k8s.gcr.io/echoserver:1.5 \
--port=8080 \
--expose \
--replicas=2
```

Check running pods:

```
root@ctl01:~# kubectl get pods -o wide
NAME                          READY     STATUS    RESTARTS   AGE       IP
echoserver-8bd85699d-llxkc    1/1       Running   0          5s        192.168.109.130
echoserver-8bd85699d-tvwns    1/1       Running   0          5s        192.168.118.133
```

The pods are running on both *cmp01* and *cmp02*.

Step 3 Check kubernetes service:

```
root@ctl01:~# kubectl get service
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
echoserver   ClusterIP   10.254.165.146   <none>        8080/TCP   5m
kubernetes   ClusterIP   10.254.0.1       <none>        443/TCP    29d
```

Remember echoserver's ClusterIP, in our case it's `10.254.165.146`

Step 4 Test running containers using ClusterIP:

```
root@ctl01:~# curl http://<echoserver ClusterIP>:8080

Hostname: echoserver-8bd85699d-llxkc

Pod Information:
    -no pod information available-

Server values:
```

```
    server_version=nginx: 1.13.0 - lua: 10008
...
```

Step 5 Delete service and deployment:

```
root@ctl01:~# kubectl delete service,deployment echoserver
```