Deploy Kubernetes Cluster with Ansible & Kubespray | ComputingForGeeks

Josphat Mutai

16-20 minutes

There are varying ways of deploying a Production ready Kubernetes cluster. In this article, we will focus on deployment of Production grade Kubernetes Cluster with Ansible and Kubespray. Kubespray is a composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks.

With Kubespray you can quickly deploy a highly available Kubernetes Cluster on *AWS*, *GCE*, *Azure*, *OpenStack*, *vSphere*, *Packet* (bare metal), or *Baremetal*. It has support for most popular Linux distributions, such as Debian, Ubuntu, CentOS, RHEL, Fedora, CoreOS, openSUSE and Oracle Linux 7.

Want a different deployment way, check out:

Install Production Kubernetes Cluster with Rancher RKE

For semi manual deployment, check:

Deploy Kubernetes Cluster on CentOS 7 / CentOS 8 With Ansible and Calico CNI

For a lightweight Kubernetes cluster fit for IoT and Edge, try: How To Deploy Lightweight Kubernetes Cluster in 5 minutes with K3s

Step 1: Infrastructure Preparation

You need to start by creating Virtual Machines / servers used during the deployment of Kubernetes cluster. This involves choosing the Linux distribution of your preference. In my setup, I'll go with CentOS 7 as the base OS for all deployments.

My masters / workers / etcd nodes will use the **m1.medium** flavor. This will have more resources if you expect huge workload in your cluster.

```
$ openstack flavor list
list+----+-----------+-------+------+-----------+-------+-----------+
| ID | Name      |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+----+-----------+-------+------+-----------+-------+-----------+
| 0  | m1.tiny   |  1024 |   10 |         0 |     1 | True      |
| 1  | m1.small  |  2048 |   20 |         0 |     1 | True      |
| 2  | m1.medium |  4096 |   20 |         0 |     2 | True      |
| 3  | m1.large  |  8192 |   40 |         0 |     4 | True      |
| 4  | m1.xlarge | 16384 |   40 |         0 |     4 | True      |
+----+-----------+-------+------+-----------+-------+-----------+
```

I'll create my VMs using the openstack CLI. Three controller/etcd nodes and two worker nodes.

```
for i in master0 master1 master2 worker0 worker1; do
 openstack server create \
 --image CentOS-7 \
 --key-name jmutai \
 --flavor m1.medium \
 --security-group  7fffea2a-b756-473a-a13a-219dd0f1913a  \
 --network private  \
 $i
done
```

All the controller nodes will also run *etcd* service. Here are my servers created.

```
$ openstack server list
+--------------------------------------+------------------+--------+----------------------------------+----------+---
| ID                                   | Name             | Status | Networks
| Image    | Flavor    |
+--------------------------------------+------------------+--------+----------------------------------+----------+---
| 5eba57c8-859c-4edb-92d3-ba76d38b56d0 | worker1          | ACTIVE | private=10.10.1.122
| CentOS-7 | m1.medium |
| 72a63616-2ba0-4542-82eb-a64acb093216 | worker0          | ACTIVE | private=10.10.1.146
| CentOS-7 | m1.medium |
| b445424c-364f-4667-9de1-559282e23ce1 | master2          | ACTIVE | private=10.10.1.134
| CentOS-7 | m1.medium |
| 6a20fa48-8ae8-4a30-a301-af32dbb67277 | master1          | ACTIVE | private=10.10.1.194
| CentOS-7 | m1.medium |
| 29ad13aa-261f-47e8-8ba5-9350f8c09847 | master0          | ACTIVE | private=10.10.1.126
| CentOS-7 | m1.medium |
+--------------------------------------+------------------+--------+----------------------------------+----------+---
```

Step 2: Clone kubespray project

Clone Project repository:

```
$ git clone https://github.com/kubernetes-sigs/kubespray.git
Cloning into 'kubespray'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 38488 (delta 2), reused 2 (delta 0), pack-reused 38471
Receiving objects: 100% (38488/38488), 11.06 MiB | 548.00 KiB/s, done.
Resolving deltas: 100% (21473/21473), done.
```

Change to the project directory:

```
$ cd kubespray
```

This directory contains the inventory files and playbooks used to deploy Kubernetes.

Step 3: Prepare Local machine

On the Local machine where you'll run deployment from, you need to install pip Python package manager.

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
python get-pip.py --user
```

Step 4: Create Kubernetes Cluster inventory file and Install dependencies

The inventory is composed of 3 groups:

- **kube-node** : list of kubernetes nodes where the pods will run.
- **kube-master** : list of servers where kubernetes master components (apiserver, scheduler, controller) will run.
- **etcd**: list of servers to compose the etcd server. You should have at least 3 servers for failover purpose.

There are also two special groups:

- **calico-rr** : explained for advanced Calico networking cases
- **bastion** : configure a bastion host if your nodes are not directly reachable

Create an inventory file:

```
cp -rfp inventory/sample inventory/mycluster
```

Define your inventory with your server's IP addresses and map to correct node purpose.

```
$ vim inventory/mycluster/inventory.ini

master0   ansible_host=10.10.1.126 ip=10.10.1.126
master1   ansible_host=10.10.1.194 ip=10.10.1.194
master2   ansible_host=10.10.1.134 ip=10.10.1.134
worker0   ansible_host=10.10.1.146 ip=10.10.1.146
worker1   ansible_host=10.10.1.122 ip=10.10.1.122

# ## configure a bastion host if your nodes are not directly reachable
# bastion ansible_host=x.x.x.x ansible_user=some_user

[kube-master]
master0
master1
master2

[etcd]
master0
master1
master2

[kube-node]
worker0
worker1

[calico-rr]

[k8s-cluster:children]
kube-master
kube-node
calico-rr
```

Add A records to */etc/hosts* on your workstation.

```
$ sudo tee -a /etc/hosts <<EOF
10.10.1.126 master0
10.10.1.194 master1
10.10.1.134 master2
10.10.1.146 worker0
10.10.1.122 worker1
EOF
```

If your private ssh key has passphrase, save it before starting deployment.

```
$ eval `ssh-agent -s` && ssh-add
Agent pid 4516
Enter passphrase for /home/centos/.ssh/id_rsa:
Identity added: /home/centos/.ssh/id_rsa (/home/centos/.ssh/id_rsa)
```

Install dependencies from `requirements.txt`

```
pip install --user -r requirements.txt
```

Confirm ansible installation.

```
$ ansible --version
ansible 2.7.12
  config file = /home/centos/kubespray/ansible.cfg
  configured module search path = [u'/home/centos/kubespray/library']
  ansible python module location = /home/centos/.local/lib/python2.7/site-packages/ansible
  executable location = /home/centos/.local/bin/ansible
  python version = 2.7.5 (default, Jun 20 2019, 20:27:34) [GCC 4.8.5 20150623 (Red Hat 4.8.5-36)]
```

Review and change parameters under `inventory/mycluster/group_vars`

```
cat inventory/mycluster/group_vars/all/all.yml
cat inventory/mycluster/group_vars/k8s-cluster/k8s-cluster.yml
```

Step 5: Deploy Kubernetes Cluster with Kubespray Ansible Playbook

Now execute the playbook to deploy Production ready Kubernetes with Ansible. Please note that the target servers must have **access to the Internet** in order to pull docker images.

Start new tmux session.

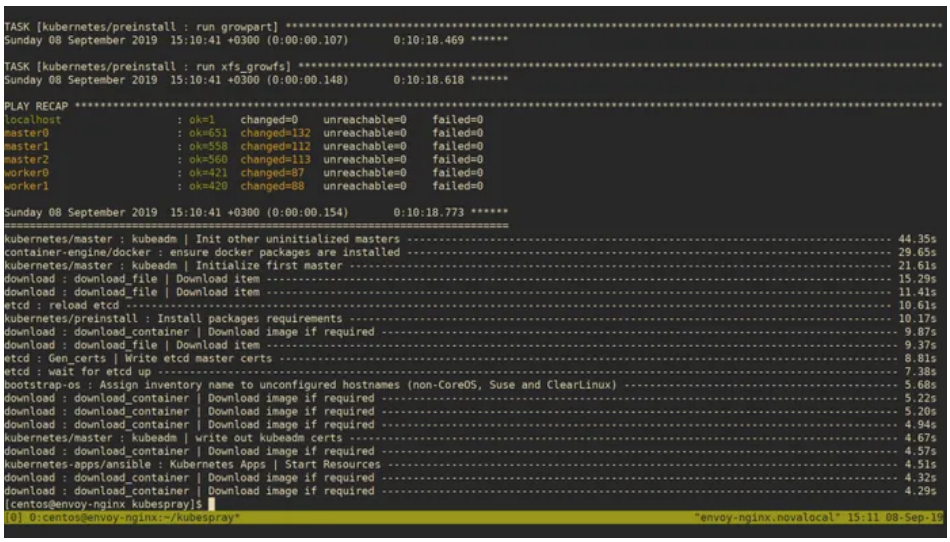
```
tmux new -s kubespray
```

Start the deployment by running the command:

```
ansible-playbook -i inventory/mycluster/inventory.ini --become \
```

```
--user=centos --become-user=root cluster.yml
```

Replace **centos** with the remote user ansible will connect to the nodes as. You should not get failed task in execution.



Login to one of the master nodes and check cluster status.

```
$ sudo su -

# kubectl config get-clusters
NAME
cluster.local

# kubectl cluster-info
Kubernetes master is running at https://10.10.1.126:6443
coredns is running at https://10.10.1.126:6443/api/v1/namespaces/kube-system/services/coredns:dns/proxy
kubernetes-dashboard is running at https://10.10.1.126:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

# kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://10.10.1.126:6443
  name: cluster.local
contexts:
- context:
    cluster: cluster.local
    user: kubernetes-admin
  name: kubernetes-admin@cluster.local
current-context: kubernetes-admin@cluster.local
kind: Config
preferences: {}
users:
- name: kubernetes-admin
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED

# kubectl get  nodes
 NAME      STATUS   ROLES    AGE   VERSION
 master0   Ready    master   23m   v1.15.3
 master1   Ready    master   22m   v1.15.3
 master2   Ready    master   22m   v1.15.3
 worker0   Ready       22m   v1.15.3
 worker1   Ready       22m   v1.15.3

# kubectl get endpoints -n kube-system
 NAME                     ENDPOINTS                                                     AGE
 coredns                  10.233.97.1:53,10.233.98.2:53,10.233.97.1:53 + 3 more...   78m
 kube-controller-manager                                                            80m
 kube-scheduler                                                                     80m
 kubernetes-dashboard     10.233.110.1:8443                                             78m
```

You can also check running Pods in the cluster under *kube-system* namespace.

```
# kubectl get pods -n kube-system
NAME                                    READY   STATUS    RESTARTS   AGE
calico-kube-controllers-55c59dd474-fn7fj   1/1     Running   0          69m
calico-node-5fjcp                       1/1     Running   1          69m
calico-node-9rt6v                       1/1     Running   1          69m
calico-node-cx472                       1/1     Running   1          69m
calico-node-v7db8                       1/1     Running   0          69m
calico-node-x2cwz                       1/1     Running   1          69m
coredns-74c9d4d795-bsqk5                1/1     Running   0          68m
coredns-74c9d4d795-bv5qh                1/1     Running   0          69m
dns-autoscaler-7d95989447-ccpf4         1/1     Running   0          69m
kube-apiserver-master0                  1/1     Running   0          70m
kube-apiserver-master1                  1/1     Running   0          70m
kube-apiserver-master2                  1/1     Running   0          70m
```

```
kube-controller-manager-master0            1/1     Running   0          70m
kube-controller-manager-master1            1/1     Running   0          70m
kube-controller-manager-master2            1/1     Running   0          70m
kube-proxy-6mvwq                           1/1     Running   0          70m
kube-proxy-cp7f9                           1/1     Running   0          70m
kube-proxy-fkmqk                           1/1     Running   0          70m
kube-proxy-nlmsk                           1/1     Running   0          70m
kube-proxy-pzwjh                           1/1     Running   0          70m
kube-scheduler-master0                     1/1     Running   0          70m
kube-scheduler-master1                     1/1     Running   0          70m
kube-scheduler-master2                     1/1     Running   0          70m
kubernetes-dashboard-7c547b4c64-q92qk      1/1     Running   0          69m
nginx-proxy-worker0                        1/1     Running   0          70m
nginx-proxy-worker1                        1/1     Running   0          70m
nodelocaldns-6pjn8                         1/1     Running   0          69m
nodelocaldns-74lwl                         1/1     Running   0          69m
nodelocaldns-95ztp                         1/1     Running   0          69m
nodelocaldns-mx26s                         1/1     Running   0          69m
nodelocaldns-nmqbq                         1/1     Running   0          69m
```

Step 6: Configure HAProxy Load Balancer

Let's configure an external loadbalancer (LB) to provide access for external clients, while the internal LB accepts client connections only to the localhost. Install HAProxy package on the server you're using as Load balancer.

```
sudo yum -y install haproxy
```

Configure backend servers for API.

```
listen k8s-apiserver-https
  bind *:6443
  option ssl-hello-chk
  mode tcp
  balance roundrobin
  timeout client 3h
  timeout server 3h
  server master0 10.10.1.126:6443
  server master1 10.10.1.194:6443
  server master2 10.10.1.134:6443
```

Start and enable haproxy service.

```
sudo systemctl enable --now haproxy
```

Get service status.

```
$ systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/usr/lib/systemd/system/haproxy.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2019-09-08 15:47:44 EAT; 37s ago
 Main PID: 23051 (haproxy-systemd)
   CGroup: /system.slice/haproxy.service
           ├─23051 /usr/sbin/haproxy-systemd-wrapper -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
           ├─23052 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds
           └─23053 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid -Ds

Sep 08 15:47:44 envoy-nginx.novalocal systemd[1]: Started HAProxy Load Balancer.
Sep 08 15:47:44 envoy-nginx.novalocal haproxy-systemd-wrapper[23051]: haproxy-systemd-wrapper:
executing /usr/sbin/haproxy -f /etc/haproxy/...d -Ds
Sep 08 15:47:44 envoy-nginx.novalocal haproxy-systemd-wrapper[23051]: [WARNING] 250/154744 (23052) :
parsing [/etc/haproxy/haproxy.cfg:45] ...log'.
Sep 08 15:47:44 envoy-nginx.novalocal haproxy-systemd-wrapper[23051]: [WARNING] 250/154744 (23052) :
config : 'option forwardfor' ignored f...mode.
Hint: Some lines were ellipsized, use -l to show in full.
```

Allow service port on the firewall.

```
$ sudo firewall-cmd --add-port=6443/tcp --permanent
$ sudo firewall-cmd --reload
```

To connect to the API Server, the external clients can then go through a load balancer we configured.

Get a kube config file from the */etc/kubernetes/admin.conf* location on a master

```
$ scp root@master0_IP:/etc/kubernetes/admin.conf kubespray.conf
```

We can then configure the kubectl client to use downloaded configuration file through the ***KUBECONFIG*** environment variable:

```
$ export KUBECONFIG=./kubespray.conf
$ kubectl --insecure-skip-tls-verify get nodes
NAME      STATUS   ROLES    AGE   VERSION
master0   Ready    master   92m   v1.15.3
master1   Ready    master   91m   v1.15.3
master2   Ready    master   91m   v1.15.3
worker0   Ready    <none>   90m   v1.15.3
worker1   Ready    <none>   90m   v1.15.3
```

Scaling Kubernetes Cluster

You may want to add worker, master or etcd nodes to your existing cluster. This can be done by re-running the `cluster.yml` playbook, or you can target the bare minimum needed to get kubelet installed on the worker and talking to your masters.

1. Add the new worker node to your inventory in the appropriate group
2. Run the ansible-playbook command:

```
ansible-playbook -i inventory/mycluster/inventory.ini --become \
--user=centos --become-user=root -v cluster.yml
```

Kubernetes Mastery courses:

Accessing Kubernetes Dashboard

If the variable `dashboard_enabled` is set (default is true), then you can access the Kubernetes Dashboard at the following URL, You

will be prompted for credentials: https://first_master:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login

Or use `kubectl proxy` command to create proxy server between your machine and Kubernetes API server. By default it is only accessible locally (from the machine that started it).

First let's check if *kubectl* is properly configured and has access to the cluster.

```
$ kubectl cluster-info
```

Start local proxy server.

```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Access the dashboard locally in your browser from: http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login

Setup Dynamic Volume Provisioning

If you need dynamic provisioning of Persistent Volumes, then check:

Setup How To Setup Kubernetes / OpenShift Dynamic Persistent Volume Provisioning with GlusterFS and

Check more Kubernetes articles:

How To run Local Kubernetes clusters in Docker

Best Kubernetes Study books 2019

Best Storage Solutions for Kubernetes & Docker Containers

Deploy Lightweight Kubernetes with MicroK8s and Snap

How To Deploy Lightweight Kubernetes Cluster in 5 minutes with K3s

How to run Local Openshift Cluster with Minishift

We really appreciate you supporting our efforts by buying us Coffee:

Coming up with fresh, high quality content takes time. Sometimes working late at night building labs and then doing the writing. We appreciate if you consider supporting our efforts with a cup of coffee to keep us awake and always deliver.

No contribution is small. We are greatful for any amount you support us with. Thank you!