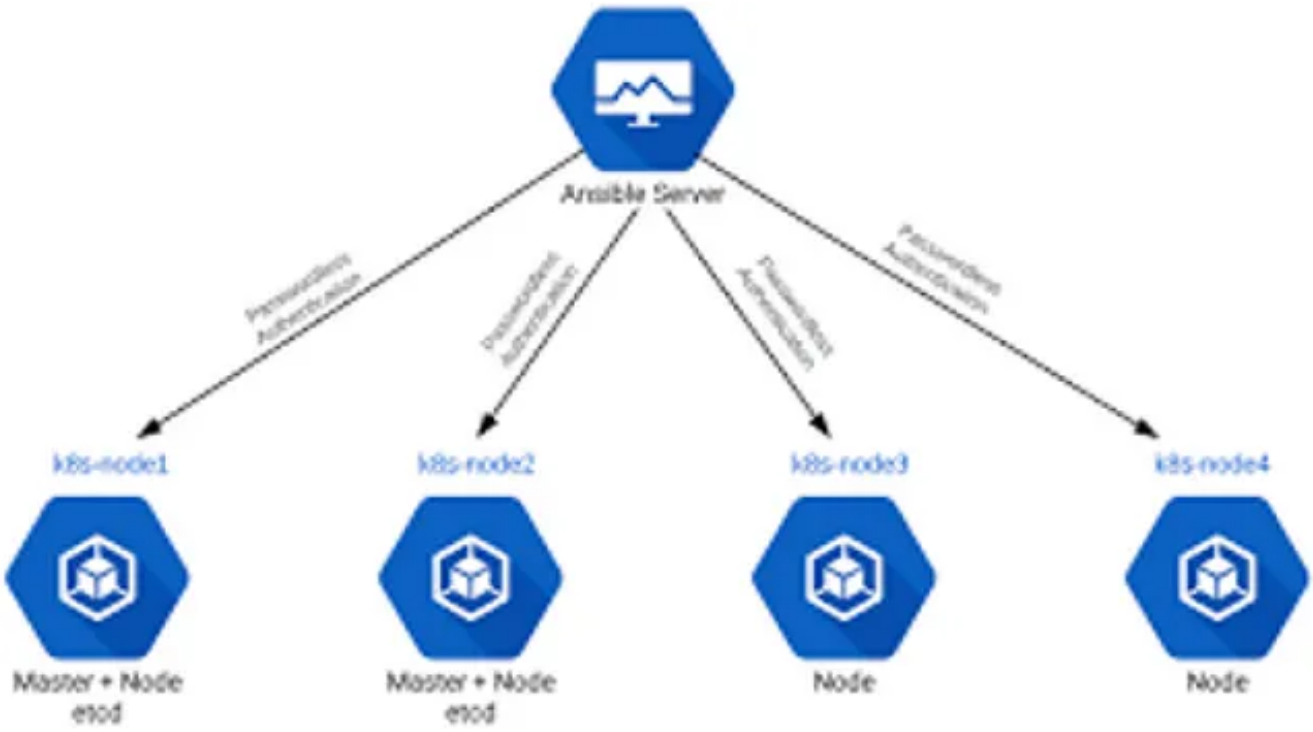
Multi-Node Kubernetes Cluster Deployment with Kubespray and Ansible

Ritik Agrawal



We will be deploying a cluster with one master node and 2 worker nodes. The master node is the server that controls all the schedulers for applications as well as has the admin configurations. The worker node will be the server that is only responsible for hosting applications provided by the master.

Steps:

Each step can be executed separately using commands for that OS. I will list commands for Ubuntu 22.04 LTS.

Step 1 — Enable password-less sudo privilege for your user:

By default when we execute `sudo user ...` it asks for a password. But ansible cannot put in a password. So we need to disable the password prompt. For this we edit the /etc/sudoers file:

- `sudo nano /etc/sudoers`
- Add `username ALL=(ALL) NOPASSWD: ALL` after the `#includedir /etc/sudoers.d` line.

Repeat this step for each server. Execute `sudo echo "it works"` . If you are not prompted for a password it works.

Step 2 — Login to the machine where you will execute kubespray:

You can set up the whole thing using your own machine or one of the servers. I am going to use the master node for the kubespray and ansible setup.

Step 3 — Setting up keyless SSH login:

Ansible uses ssh to access the servers. So we need to make sure that the machine in which ansible runs has ssh authentication into each server. The below commands will be executed in the machine where ansible will run.

- `ssh-keygen -t rsa` generates an RSA key for our

machine
- Our public key is now saved as `$HOME/.ssh/id_rsa.pub` . We need to add this key as an authorized key to all the servers.
- you can either do `ssh-copy-id -i login_key username@hostip` which will copy the public key automatically or login to each server and copy the `id_rsa.pub` in `$HOME/.ssh/authorized_keys` the file of each server. Append the key in a newline in `authorized_keys` file. Repeat this for each server.
- Check if you can ssh into these servers without the key. Execute `ssh -o StrickHostChecking=no server-public-ip`. If it works you are all set!

Step 4 — Setup the environment to execute ansible playbook:

Login to the machine that will setup kubespray and ansible. Ansible requires `python3.8` , `pip3` and`git` installed. If not installed you can do so separately for your OS.

To check which python version is `pip` using, run `pip3 -version`
If you get something like`pip 21.0.1 from /usr/lib/python3.8/site-packages/pip (python 3.8)` then it's fine.

- Clone the kubespray repository:

```
git clone https://github.com/kubernetes-sigs/kube
cd kubespray
```

- Now install dependencies for Ansible run the Kubespray playbook:

```
pip3 install -r requirements.txt
```

If you run `ansible` you might get `command not found`. In that case, add `export PATH=$HOME/.local/bin:$PATH` to the end of `~/.bashrc`. Then run `source ~/.bashrc`. This will add the path where ansible scripts and loaded in your PATH env variable. Now you should be able to execute `ansible`.

- Copy the sample inventory:

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

- Now we need to declare a variable which will contain the privateip_addresses of each machine:

```
declare -a IPS=(172.31.31.253 172.31.29.187 172.3
```

- Generate the inventory file for ansible:

```
# Replace python3.8 with your python version
CONFIG_FILE=inventory/mycluster/hosts.yaml pythor
```

- Configure the generated `inventory/mycluster/hosts.yaml` so that it uses the ip of master node as a `kube_control_plane` and the other two as `kube_node` .

This is my origional hosts.yaml:

```
all:
  hosts:
    node1:
      ansible_host: 172.31.31.253
      ip: 172.31.31.253
      access_ip: 172.31.31.253
    node2:
      ansible_host: 172.31.29.187
      ip: 172.31.29.187
      access_ip: 172.31.29.187
    node3:
      ansible_host: 172.31.28.68
      ip: 172.31.28.68
      access_ip: 172.31.28.68
  children:
    kube_control_plane:
      hosts:
        node1:
        node2:
    kube_node:
      hosts:
```

```
        node1:
        node2:
        node3:
    etcd:
      hosts:
        node1:
        node2:
        node3:
    k8s_cluster:
      children:
        kube_control_plane:
        kube_node:
    calico_rr:
      hosts: {}
```

After editing:

```
all:
  hosts:
    master:
      ansible_host: 172.31.31.253
      ip: 172.31.31.253
      access_ip: 172.31.31.253
    worker1:
      ansible_host: 172.31.29.187
      ip: 172.31.29.187
      access_ip: 172.31.29.187
    worker2:
      ansible_host: 172.31.28.68
      ip: 172.31.28.68
      access_ip: 172.31.28.68
  children:
```

```
    kube_control_plane:
      hosts:
       master:
    kube_node:
      hosts:
       worker1:
       worker2:
    etcd:
      hosts:
       master:
       worker1:
       worker2:
    k8s_cluster:
      children:
        kube_control_plane:
        kube_node:
    calico_rr:
      hosts: {}
```

By default ansible will prompt to add the hosts to the known_hosts file when it tries to ssh into them. To avoid this, create a file `$HOME/.ansible.cfg` and put the following init:

```
[defaults]
host_key_checking = False
```

Now go to the file `inventory/mycluster/group_vars/all/all.yaml` and uncomment the line `# kube_read_only_port: 10255` . This

is a YAML file so make sure there is no space at the beginning of the line after removing the “#”.

Now open the file `inventory/mycluster/group_vars/k8s_cluster/k8s-cluster.yml` and check the `kube_network_plugin: calico` line. This sets the network plugin to `calico` . We can change this but `calico` is secure so we can leave it like that.

Optional: We can enable the `metrics_server` for our cluster which will provide some default metrics about the nodes like CPU Usage and Memory. In the second part of this blog we will use Prometheus for monitoring but if you want to enable this then edit `inventory/mycluster/group_vars/k8s_cluster/addons.yml` and make `metrics_server_enabled` property from `false` to `true` .

Step 5 — Run the ansible playbook:

```
ansible-playbook -i inventory/mycluster/hosts.yar
```

Replace `username` with the username in all servers. The `-b` flag will make ansible use `sudo` to execute commands. Since we enabled passwordless sudo in Step 1, this shouldn’t cause any problems.

Run this command in the `kubespray` directory. If all network configurations are right and the hardware meets minimum requirements, this should execute without issue or else you can get “assertion failed” erros.

This will take around 20 minutes to execute.

Step 6 — Configure kubectl to use admin.conf:

Now ssh into the master node.

If you notice, we should now have a file `/etc/kubernetes/admin.conf` . Also the command `kubectl` should now be executable. This is the CLI tool that we use to communicate with kubernetes.

If you execute `kubectl get all` you should receive this:

```
The connection to the server localhost:8080 was r
```

This is because we need to tell kubectl to use that admin.conf. We will also need to modify permissions on it so our user can access it.

```
# Copy admin.conf in user directory so it's acces
mkdir $HOME/.kubernetes
sudo cp /etc/kubernetes/admin.conf $HOME/.kuberne
USERNAME=$(whoami)
sudo chown -R $USERNAME:$USERNAME $HOME/.kubernet
```

Now test the cluster access using `kubeconfig` :

```
kubectl get nodes --kubeconfig=$HOME/.kubernetes/
```

```
NAME      STATUS   ROLES           AGE   VERSION
master    Ready    control-plane   20m   v1.25.5
worker1   Ready    <none>          19m   v1.25.5
worker2   Ready    <none>          19m   v1.25.5
```

To not have to type – `kubeconfig` every time, we will edit our `~/.bashrc` :

```
nano ~/.bashrc
#Add this at the end of your ~/.bashrc
export KUBECONFIG=$HOME/.kubernetes/admin.conf
#reload shell
source ~/.bashrc
```

Now run `kubectl get nodes` and you should get the same output.

```
master    Ready    control-plane   20m   v1.25.5
worker1   Ready    <none>          19m   v1.25.5
worker2   Ready    <none>          19m   v1.25.5
```

It might take some time for all nodes to reach`Ready` state.

To check the status of everything running run this:

```
kubectl get all --all-namespaces
NAMESPACE     NAME
kube-system   pod/calico-kube-controllers-75748c
kube-system   pod/calico-node-96928
kube-system   pod/calico-node-msklc
kube-system   pod/calico-node-p9gpk
kube-system   pod/coredns-588bb58b94-mnbnp
kube-system   pod/coredns-588bb58b94-vgbkr
kube-system   pod/dns-autoscaler-5b9959d7fc-89dp
kube-system   pod/kube-apiserver-master
kube-system   pod/kube-controller-manager-master
kube-system   pod/kube-proxy-8ls9x
kube-system   pod/kube-proxy-bggfl
kube-system   pod/kube-proxy-n8npw
kube-system   pod/kube-scheduler-master
kube-system   pod/metrics-server-6bd8d699c5-k597
kube-system   pod/nginx-proxy-worker1
kube-system   pod/nginx-proxy-worker2
kube-system   pod/nodelocaldns-k6wck
kube-system   pod/nodelocaldns-l6689
kube-system   pod/nodelocaldns-s5bjw

NAMESPACE     NAME                      TYPE
default       service/kubernetes        ClusterIP
kube-system   service/coredns           ClusterIP
kube-system   service/metrics-server    ClusterIP

NAMESPACE     NAME                           DESI
kube-system   daemonset.apps/calico-node     3
kube-system   daemonset.apps/kube-proxy      3
kube-system   daemonset.apps/nodelocaldns    3
```

```
NAMESPACE     NAME
kube-system   deployment.apps/calico-kube-contro
kube-system   deployment.apps/coredns
kube-system   deployment.apps/dns-autoscaler
kube-system   deployment.apps/metrics-server

NAMESPACE     NAME
kube-system   replicaset.apps/calico-kube-contro
kube-system   replicaset.apps/coredns-588bb58b94
kube-system   replicaset.apps/dns-autoscaler-5b9
kube-system   replicaset.apps/metrics-server-6bd
```

This will list everything and their status. As you can see everything is in `Running` state.

To access this cluster from some other machine, we need to use this admin.conf file. Let's see how we can do that from one of our worker nodes.

```
#copy admin.conf to our worker node
scp ~/.kubernetes/admin.conf username@worker-ip:
```

Now ssh into the worker node. We first need to install `kubectl` on the worker node as this is not done by kubespray. Follow this to install `kubectl`

```
# ssh into worker node
ssh worker-ip
```

```
# install kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s
sudo install -o root -g root -m 0755 kubectl /usr

# Execute this on worker
kubectl get nodes --kubeconfig=admin.conf

# Output
NAME      STATUS   ROLES           AGE   VERSION
master    Ready    control-plane   46m   v1.25.5
worker1   Ready    <none>          44m   v1.25.5
worker2   Ready    <none>          44m   v1.25.5
```

Thus now we can access the cluster.

This gives us a working multi-node Kubernetes cluster.

Next, we should run a test application on our master node, and monitor them using Prometheus and Grafana and auto-scale based on their metrics. This has been done in *this blog* which is a continuation of our current setup! Please let me know if you face any issues!

LinkedIn: https://www.linkedin.com/in/ritik-agrawal-b14886191/
Email: ritikagrawal1292000@gmail.com