Kubespray Advanced Configuration For A Production Cluster

/ Kubernetes, tech

In this post, we will see some advanced options provided by kubespray for production-grade cluster creation. In most places on the internet, you will see the basic instructions to create a simple(Hello World) Kubernetes cluster using a bunch of virtual machines. However, kubespray offer much more than that. For example:

1. You can install several add-ons required in the daily operations of the cluster.
2. Kubespray provides capabilities to expose control plan nodes via pre-existing load balancers.

We will explore some of the critical configurations. However, there would always be more in the official documents.

You might want to see this page if you intend to create a simple cluster that works fine for development or learning.

If you are unfamiliar with load-balancing, consider reading the basics here to make more sense of the information provided in this post.

The entire procedure described here is automated and available; check this page for details.

Use case:

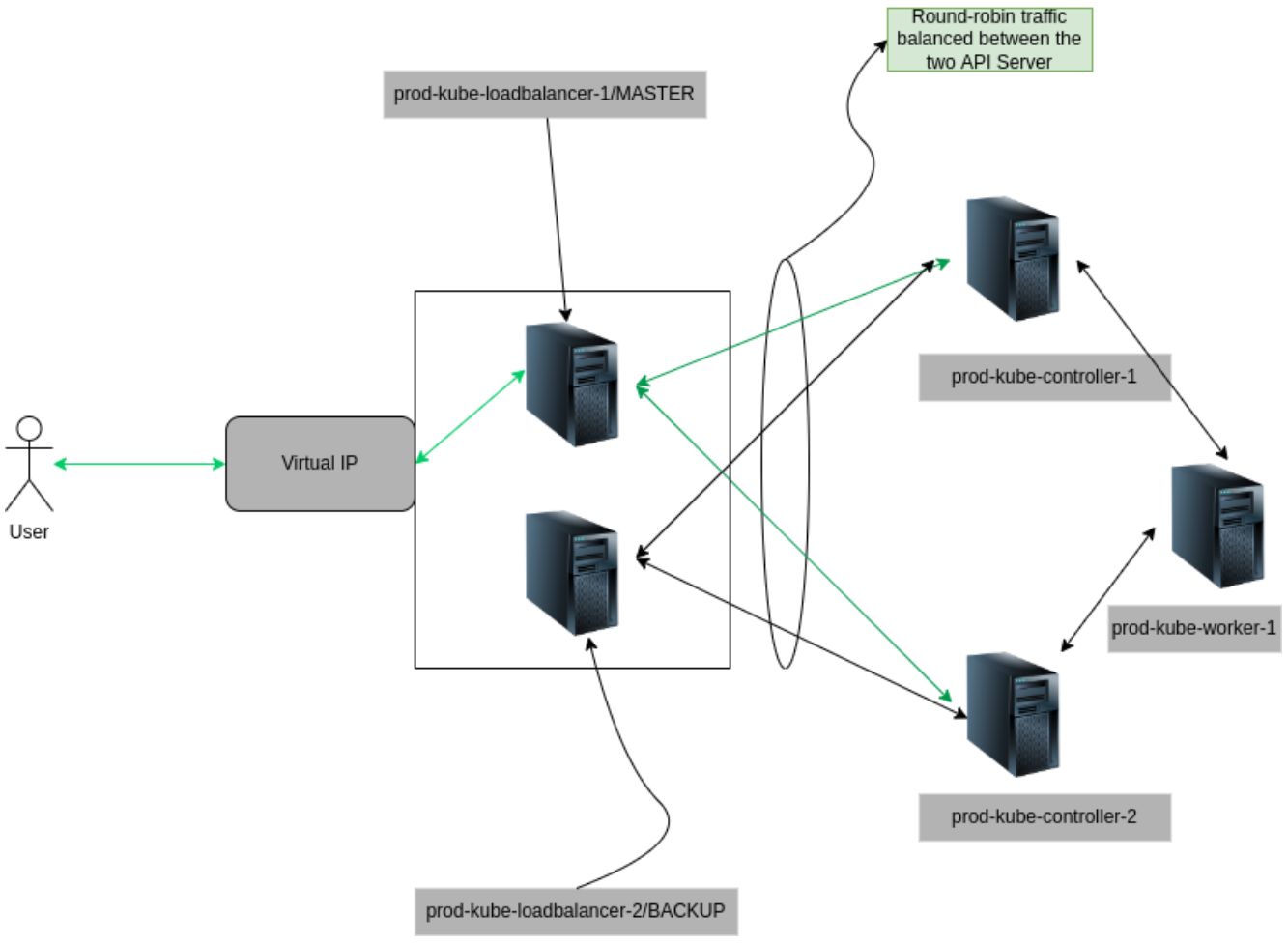
The following is the objective we will achieve in this post. These actions will make the cluster resilient to a single point of failure and provide several add-ons ready to use after cluster instantiation.

1. Create a High-available cluster, using multiple(two) Kube-API servers and one worker node(You can have more).
2. Create multiple frontend external load balancers nodes(2).
3. Configure the external load balancers, and expose the Kube-API server via Virtual IP.
4. Using kubespray addons, Install helm, Kubernetes dashboard, metric server, and a few more addons.

The Setup: I have already created the following virtual machines

```
Name        MAC address           Protocol     Address
-------------------------------------------------------------
 vnet20     52:54:00:97:cb:dd     ipv4         192.168.122.1
 vnet21     52:54:00:19:d9:6e     ipv4         192.168.122.1
 vnet22     52:54:00:90:40:35     ipv4         192.168.122.2
 vnet23     52:54:00:ca:4c:d1     ipv4         192.168.122.2
```

```
vnet24      52:54:00:5d:d7:13    ipv4          192.168.122.4
```



Traffic path in Green, when LB-1 is VRRP Master

Start of the Procedure:

Step-1: Install git and Pip3

```
sudo apt update && sudo apt-get -y install python3-pip git
```

Step-2: Clone The Git Repo

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

Step-3: Install the requirements

```
cd kubespray/
sudo pip3 install -r requirements.txt
```

Step-4: Copy the blank inventory template

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

step-5: Make a list/array of hostname and IP for all controller and worker nodes

```
declare -a IPS=(controller-vm-hostname-1,IP-1 controller-vm
```

```
declare -a IPS=(prod-kube-controller-1,192.168.122.105 prod
```

Step-6: Build the inventory

```
CONFIG_FILE=inventory/mycluster/hosts.yaml python3 contrib/
```

```
CONFIG_FILE=inventory/mycluster/hosts.yaml python3 contrib/
DEBUG: Adding group all
DEBUG: Adding group kube_control_plane
```

```
DEBUG: Adding group kube_node
DEBUG: Adding group etcd
DEBUG: Adding group k8s_cluster
DEBUG: Adding group calico_rr
DEBUG: adding host rod-kube-controller-1 to group all
DEBUG: adding host prod-kube-controller-2 to group all
DEBUG: adding host prod-kube-worker-1 to group all
DEBUG: adding host rod-kube-controller-1 to group etcd
DEBUG: adding host prod-kube-controller-2 to group etcd
DEBUG: adding host prod-kube-worker-1 to group etcd
DEBUG: adding host rod-kube-controller-1 to group kube_cont
DEBUG: adding host prod-kube-controller-2 to group kube_con
DEBUG: adding host rod-kube-controller-1 to group kube_node
DEBUG: adding host prod-kube-controller-2 to group kube_nod
DEBUG: adding host prod-kube-worker-1 to group kube_node
```

Step-7: Understand, Validate and visualize the generated inventory file

In the inventory file, there are three host groups created.
1. **kube_control_plane:** the nodes under this host group will be configured as controller nodes(API server will run here)
2. **kube_node:** The nodes under this host group will be configured as worker nodes. (payload nodes)
3. **etcd:** The nodes under this host group will be running the etcd database for the cluster. The number of nodes under this host group must be odd. To meet, within the etcd quorum.

```
cat inventory/mycluster/hosts.yaml
```

```
all:
  hosts:
    rod-kube-controller-1:
      ansible_host: 192.168.122.105
      ip: 192.168.122.105
      access_ip: 192.168.122.105
    prod-kube-controller-2:
      ansible_host: 192.168.122.137
      ip: 192.168.122.137
      access_ip: 192.168.122.137
    prod-kube-worker-1:
      ansible_host: 192.168.122.253
      ip: 192.168.122.253
      access_ip: 192.168.122.253
  children:
    kube_control_plane:
      hosts:
        rod-kube-controller-1:
        prod-kube-controller-2:
    kube_node:
      hosts:
        rod-kube-controller-1:
        prod-kube-controller-2:
        prod-kube-worker-1:
    etcd:
      hosts:
        rod-kube-controller-1:
        prod-kube-controller-2:
        prod-kube-worker-1:
    k8s_cluster:
      children:
        kube_control_plane:
        kube_node:
    calico_rr:
```

```
    hosts: {}
```

Now the inventory file is reNext, we next; we can work towards more advanced configuration options. Most of these options are mutually exclusive from each other, so if you skip anyone, likely, it won't trouble you.

File-1: Tweaking the cluster configuration

The first file we would want to tweak is **inventory/mycluster/group_vars/k8s_cluster/k8s-cluster.yml**. These are the bare minimum configurations I would like to highlight. Feel free to explore the rest of the file and tweak it if required.

The last variable, "**supplementary_addresses_in_ssl_keys," is critical** for load balancers to work; in this step, we tell kubeadm to add VIP while creating the certificates.

```
kube_version: v1.24.3
```

```
container_manager: containerd
```

```
cluster_name: prod.local
```

```
kube_network_plugin: calico
```

```
k8s_image_pull_policy: IfNotPresent
```

```
kubernetes_audit: true
```

```
kube_encrypt_secret_data: true
```

```
kubeconfig_localhost: true
```

```
kubectl_localhost: true
```

```
supplementary_addresses_in_ssl_keys: [10.0.0.1, 10.0.0.2, 1
```

File-2: adding the external load balancer VIP

In this section, we will provide the external load balancer Virtual IP and port in **inventory/mycluster/group_vars/all/all.yml**

```
loadbalancer_apiserver:
  address: "192.168.122.211"
  port: "8443"
```

File-3: Manage the addons

I am limiting this section to the dashboard, registry, helm, metrics server, an ingress controller. However, many more add-ons are present in the inventory/mycluster/group_vars/k8s_cluster/addons.yml fi—for example, metallb, cert-manager, etc.
Consider going through the file and choosing what you need.

```
dashboard_enabled: true
```

```
helm_enabled: true
```

```
registry_enabled: true
registry_namespace: kube-system
registry_storage_class: ""
registry_disk_size: "10Gi"
```

```
metrics_server_enabled: true
metrics_server_container_port: 4443
metrics_server_kubelet_insecure_tls: true
metrics_server_metric_resolution: 15s
metrics_server_kubelet_preferred_address_types: "InternalIP
```

```
ingress_nginx_enabled: true
ingress_nginx_host_network: true
ingress_publish_status_address: ""
ingress_nginx_nodeselector:
  kubernetes.io/os: "linux"
ingress_nginx_tolerations:
  - key: "node-role.kubernetes.io/master"
    operator: "Equal"
    value: ""
    effect: "NoSchedule"
  - key: "node-role.kubernetes.io/control-plane"
    operator: "Equal"
    value: ""
    effect: "NoSchedule"
ingress_nginx_namespace: "ingress-nginx"
ingress_nginx_insecure_port: 80
ingress_nginx_secure_port: 443
ingress_nginx_class: nginx
```

Now we have desired configuration done, and all add-ons are set up. We will now set up the load balancers. The kubespray does not cover this process. We will use keepalived and haproxy for the configuration. In this post, we will superficially go through the installation. However, You can read more about load balancers in great detail HERE. If you already have load balancers configured, you may skip all the following steps(LB-step-xx)

LB-step-1: Install keepalived and haproxy by running the following commands on all load balancers

```
sudo apt-get update
sudo apt-get upgrade
sudo apt install haproxy -y


sudo apt install keepalived -y
```

LB-Step-2: Create the keepalived config file on all load balancer nodes with minor changes described below

Add the following text in **/etc/keepalived/keepalived.conf** present on all load balancer nodes. Note that I have added the Virtual IP here. Copy the same content to all the load balancer nodes, **except** change the state to BACKUP and decrease the priority to 254.
IMPORTANT NOTE: My load balancer nodes are running

ubuntu22.04, which has ‘enp1s0’ as a default interface. If you have some other interface, you might need to replace it with the right one.

```
global_defs {
router_id LVS_DEVEL
script_user root
enable_script_security
}
vrrp_script check_apiserver {
  script "/etc/keepalived/check_apiserver.sh"
  interval 3
  weight -2
  fall 10
  rise 2
}

vrrp_instance VI_1 {
    state MASTER
    interface enp1s0
    virtual_router_id 51
    priority 255
    authentication {
    auth_type PASS
    auth_pass mypass
    }
    virtual_ipaddress {
        192.168.122.211/24
    }
    track_script {
        check_apiserver
    }
    notify_master "/etc/keepalived/status_capture.sh MASTER
```

```
    notify_backup "/etc/keepalived/status_capture.sh BACKUP
    notify_fault  "/etc/keepalived/status_capture.sh FAULT"
}
```

LB-Step-3: Create an identical Keepalived check script on all the load balancer node

Add the following content to **/etc/keepalived/check_apiserver.sh**

```
errorExit() {
    echo "*** $*" 1>&2
    exit 1
}

curl --silent --max-time 2 --insecure https://localhost:844

if ip addr | grep -q 192.168.122.211; then
    curl --silent --max-time 2 --insecure https://192.168.1
fi
```

LB-Step-4: Create an identical keepalived notify script on both load balancer node

Create a notify script at **/etc/keepalived/status_capture.sh**; this is optional but

very helpful in real-world scenarios to trigger actions based on state change.

```
#
echo "$(date): The loadbalancer instance running on $(hostn
chmod 755 /tmp/load-balancer-status || true
```

LB-Step-5: Create an identical haproxy config file on both load balancer node

Create the haproxy config file at **/etc/haproxy/haproxy.cfg**

```
defaults
  mode tcp
  timeout connect 10s
  timeout client 30s
  timeout server 30s

frontend apiserver
  bind *:8443
  mode tcp
  option tcplog
  log 127.0.0.1 local0
  default_backend apiserver

backend apiserver
  option httpchk GET /healthz
  http-check expect status 200
  mode tcp
  option ssl-hello-chk
```

```
  balance     roundrobin
  server kube-controller-1 192.168.122.105:6443 check
  server kube-controller-2 192.168.122.137:6443 check
```

LB-Step-6: Make the keepalive check and notify script executable on both load balancer nodes

```
sudo chmod u+x  /etc/keepalived/check_apiserver.sh
sudo chmod u+x  /etc/keepalived/status_capture.sh
```

LB-Step-7: Enable non-local binding on both load balancer nodes

```
echo 'net.ipv4.ip_nonlocal_bind=1'|sudo tee -a /etc/sysctl.
sudo sysctl -p
```

LB-Step-8: Enable keepalived and haproxy on both load balancer nodes

```
sudo service keepalived start
sudo service keepalived status
sudo service haproxy start
sudo service haproxy status
```

We have created the load balancer nodes, and the kubespray configuration is ready. Trigger the playbook for

cluster creation.

Trigger the cluster creation

```
ansible-playbook -i inventory/mycluster/hosts.yaml --become
```

Once the playbook completes, you will notice zero failure, indicating that whatever we have done so far is working.

```
PLAY RECAP *********************************************
localhost                  : ok=3    changed=0    unreachab
prod-kube-controller-1     : ok=813  changed=136  unreachab
prod-kube-controller-2     : ok=711  changed=125  unreachab
prod-kube-worker-1         : ok=560  changed=92   unreachab
```

Locate the kubeconfig file

Remember, we set up a variable called **kubeconfig_localhost** in earlier steps. That caused the kubeconfig to be copied to **inventory/mycluster/artifacts/admin.conf**

```
 ls -lrt inventory/mycluster/artifacts/admin.conf
-rw------- 1 technekey technekey 5683 Jul 21 14:06 inventor
```

Checking the addons and cluster state

```
kubectl get pod -A --kubeconfig inventory/mycluster/artifac
NAMESPACE       NAME
ingress-nginx   ingress-nginx-controller-59g57
ingress-nginx   ingress-nginx-controller-bnxvf
ingress-nginx   ingress-nginx-controller-djv4p
kube-system     calico-node-5pxff
kube-system     calico-node-gnkhj
kube-system     calico-node-k6xjs
kube-system     coredns-74d6c5659f-d9n87
kube-system     coredns-74d6c5659f-pqd4n
kube-system     dns-autoscaler-59b8867c86-z7dhx
kube-system     kube-apiserver-prod-kube-controller-1
kube-system     kube-apiserver-prod-kube-controller-2
kube-system     kube-controller-manager-prod-kube-controlle
kube-system     kube-controller-manager-prod-kube-controlle
kube-system     kube-proxy-6h96q
kube-system     kube-proxy-9hpg2
kube-system     kube-proxy-q72x6
kube-system     kube-scheduler-prod-kube-controller-1
kube-system     kube-scheduler-prod-kube-controller-2
kube-system     kubernetes-dashboard-55bf5db569-m4sqm
kube-system     kubernetes-metrics-scraper-84bbbc8b75-28zr9
kube-system     metrics-server-68b8967c9f-4fprw
kube-system     nodelocaldns-9mlxt
kube-system     nodelocaldns-k2qqg
kube-system     nodelocaldns-zrbg6
kube-system     registry-mw4b6
```

```
kubectl get svc -n kube-system --kubeconfig inventory/myclu
NAME                        TYPE        CLUSTER-IP      EXT
coredns                     ClusterIP   10.233.0.3      <no
```

```
dashboard-metrics-scraper   ClusterIP   10.233.58.178   <no
kubernetes-dashboard        ClusterIP   10.233.57.157   <no
metrics-server              ClusterIP   10.233.24.49    <no
registry                    ClusterIP   10.233.18.18    <no
```

```
kubectl top node --kubeconfig inventory/mycluster/artifacts
NAME                     CPU(cores)   CPU%   MEMORY(bytes)
prod-kube-controller-1   184m         10%    1503Mi
prod-kube-controller-2   176m         9%     1407Mi
prod-kube-worker-1       114m         6%     1197Mi
```

```
kubectl get pod  -n ingress-nginx --kubeconfig inventory/my
NAME                                READY   STATUS    RESTARTS
ingress-nginx-controller-59g57   1/1     Running   0
ingress-nginx-controller-bnxvf   1/1     Running   0
ingress-nginx-controller-djv4p   1/1     Running   0
```

Related reading and references:

- https://github.com/kubernetes/kubeadm/blob/main/docs/ha-considerations.md
- https://github.com/kubernetes-sigs/kubespray/blob/master/docs/ha-mode.md
- https://technekey.com/ha-kubernetes-cluster-using-keepalived-and-haproxy/