Deploying production grade Kubernetes Cluster on bare metal

You may have heard about Docker and Kubernetes. These two technologies are making a lot of buzz recently. In a nutshell, Docker is a light weight virtual machine providing isolated environment without the complexity of managing a full fledged virtual machine. And Kubernetes complements Docker by providing complete environment for managing and orchestrating containerised applications at scale. There are lot of articles on the internet about Docker and Kubernetes and so I'm not going to go in details but I'd like to take on the deployment of Kubernetes which is overly complex to setup in a production environment.

For cloud deployments, there are very good tools such as kops for AWS and for GCP, it provides an option to launch managed/hosted Kubernetes. Similarly, Azure provides AKS (Azure Container Service) for managed kubernetes cluster as a service.

However, on bare metal there are limited options for production grade setup. While Kubernetes provides a very good utility called kubeadm, it can't be used to bring up production cluster due to few limitations. After researching several options, I ended up settling down with kubespray at work. It is a set of tools designed for easily deploying production-ready Kubernetes clusters and it heavily depends on Ansible. This article provides details on how to bootstrap a production grade kubernetes cluster on VMs/baremetal.

1. Make sure you've latest version of ansible (≥ 2.4) and python-netaddr
2. Ensure you've Cgroup memory controller enabled on all the nodes. If not, do:

> *a. Open /etc/default/grub in your favourite text editor*

> *b. Add cgroup_enable=memory swapaccount=1 to GRUB_CMDLINE_LINUX*

> *c. Run sudo update-grub followed by sudo reboot*

3. Disable swap as kubelet won't start if swap is enabled

> *cat /proc/swaps (if swap is enabled, it shows entries here)*

> *swapoff -a*

4. Clone the kubespray repo: https://github.com/kubernetes-incubator/kubespray.git

5. Prepare your inventory file for ansible to suit your deployment. In my case, I had 5 nodes; I've selected 3 nodes for master which also run etcd cluster and remaining 2 nodes exclusively as worker nodes. It looks like below

> *tcdocker2 ansible_ssh_host=x.x.x.x ip=x.x.x.x*
>
> *tcdocker3 ansible_ssh_host=y.y.y.y ip=y.y.y.y*
>
> *tcdocker4 ansible_ssh_host=z.z.z.z ip=z.z.z.z*
>
> *tcdocker5 ansible_ssh_host=a.a.a.a ip=a.a.a.a*
>
> *tcdocker6 ansible_ssh_host=b.b.b.b ip=b.b.b.b*
>
> *[kube-master]*
>
> *tcdocker2*
>
> *tcdocker3*
>
> *tcdocker4*
>
> *[etcd]*
>
> *tcdocker2*
>
> *tcdocker3*

> *tcdocker4*
>
> *[kube-node]*
>
> *tcdocker5*
>
> *tcdocker6*
>
> *[k8s-cluster:children]*
>
> *kube-node*
>
> *kube-master*

6. Refer inventory/group_vars/k8s-cluster.yml and inventory/group_vars/all.yml

You can change the network plugin, kubernetes internal network for services and internal network for pods as per your needs. In my case, I had to make below changes:

```
diff --git a/inventory/group_vars/all.yml
b/inventory/group_vars/all.yml
```

```
index dd67969..14a68b1 100644
```

```
--- a/inventory/group_vars/all.yml
```

```
+++ b/inventory/group_vars/all.yml
```

```
@@ -38,7 +38,7 @@ bin_dir: /usr/local/bin
```

```
## for mounting persistent volumes into containers. These may not be
loaded by preinstall kubernetes
```

```
 ## processes. For example, ceph and rbd backed volumes. Set to true to
allow kubelet to load kernel
```

```
 ## modules.
```

```
 -# kubelet_load_modules: false
```

```
 +# kubelet_load_modules: true
```

```
 ## Internal network total size. This is the prefix of the
```

```
 ## entire network. Must be unused in your environment.
```

```
 @@ -102,7 +102,7 @@ bin_dir: /usr/local/bin
```

```
 #docker_storage_options: -s overlay2
```

```
 # Uncomment this if you have more than 3 nameservers, then we'll only
use the first 3.
```

```
 -#docker_dns_servers_strict: false
```

```
 +docker_dns_servers_strict: false
```

```
 ## Default packages to install within the cluster, f.e:
```

```
 #kpm_packages:
```

```
 @@ -122,3 +122,4 @@ bin_dir: /usr/local/bin
```

```
## Set level of detail for etcd exported metrics, specify 'extensive'
to include histogram metrics.

#etcd_metrics: basic

+

diff --git a/inventory/group_vars/k8s-cluster.yml
b/inventory/group_vars/k8s-cluster.yml

index a400d05..d2601e8 100644

--- a/inventory/group_vars/k8s-cluster.yml

+++ b/inventory/group_vars/k8s-cluster.yml

@@ -40,7 +40,8 @@ kube_log_level: 2

# Users to create for basic auth in Kubernetes API via HTTP

# Optionally add groups for user

-kube_api_pwd: "{{ lookup('password', 'credentials/kube_user length=15
chars=ascii_letters,digits') }}"

+#kube_api_pwd: "{{ lookup('password', 'credentials/kube_user length=15
chars=ascii_letters,digits') }}"

+kube_api_pwd: "supersecret"

kube_users:
```

```
kube:
```

```
pass: "{{kube_api_pwd}}"
```

```
@@ -117,7 +118,7 @@ dns_mode: kubedns
```

```
# Can be docker_dns, host_resolvconf or none
```

```
resolvconf_mode: docker_dns
```

```
# Deploy netchecker app to verify DNS resolve as an HTTP service
```

```
-deploy_netchecker: false
```

```
+deploy_netchecker: true
```

```
# Ip address of the kubernetes skydns service
```

```
skydns_server: "{{
kube_service_addresses|ipaddr('net')|ipaddr(3)|ipaddr('address') }}"
```

```
dnsmasq_dns_server: "{{
kube_service_addresses|ipaddr('net')|ipaddr(2)|ipaddr('address') }}"
```

```
@@ -147,7 +148,7 @@ k8s_image_pull_policy: IfNotPresent
```

```
dashboard_enabled: true
```

```
# Monitoring apps for k8s
```

```
-efk_enabled: false
```

```
+efk_enabled: true
```

7. Once you are satisfied with the options, do the deployment the usual ansible way:

ansible-playbook -i inventory/inventory cluster.yml -u root -v

It usually takes about 15–20 mins for it to complete the deployment. Towards the end, you will see a summary section for all the components deployed along with the execution time for each section.

8. How to administer and verify your cluster?

During the cluster bootstrapping process, it generates various SSL certificates which we can use to authenticate ourselves. They are located on the master node under /etc/kubernetes/ssl. Also the master node, it would have generated a config file under ~/.kube . This can be copied to your laptop from where you'd like to administer the cluster and placed under `/home/user/.kube/.`

krishna:~ krishnapmv$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

tcdocker2 Ready master 3d v1.8.2+coreos.0

tcdocker3 Ready master 3d v1.8.2+coreos.0

tcdocker4 Ready master 3d v1.8.2+coreos.0

tcdocker5 Ready node 3d v1.8.2+coreos.0

tcdocker6 Ready node 3d v1.8.2+coreos.0

krishna:~ krishnapmv$ kubectl cluster-info

Kubernetes master is running at https://x.x.x.x:6443

> *Elasticsearch is running at https://x.x.x.x:6443/api/v1/namespaces/kube-system/services/elasticsearch-logging/proxy*

> *Heapster is running at https://x.x.x.x:6443/api/v1/namespaces/kube-system/services/heapster/proxy*

> *Kibana is running at https://x.x.x.x:6443/api/v1/namespaces/kube-system/services/kibana-logging/proxy*

> *KubeDNS is running at https://x.x.x.x:6443/api/v1/namespaces/kube-system/services/kube-dns/proxy*

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

9. Kubernetes dashboard

If you've enabled the deployment of kubernetes dashboard in the configuration options (dashboard_enabled: true), it would have deployed a shiny cool dashboard for your kubernetes cluster which you can view via browser. Check that you've dashboard deployed as a service:

> *kubectl describe svc kubernetes-dashboard—namespace=kube-system*

> *Name: kubernetes-dashboard*

> *Namespace: kube-system*

> *Labels: k8s-app=kubernetes-dashboard*

> *Annotations: kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"k8s-app":"kubernetes-dashboard-head"},"name":"kubernetes-dashboard-head","n…*

> *Selector: k8s-app=kubernetes-dashboard-head*

> *Type: ClusterIP*
>
> *IP: z.z.z.z*
>
> *Port: <unset> 443/TCP*
>
> *TargetPort: 8443/TCP*
>
> *Endpoints: <none>*
>
> *Session Affinity: None*
>
> *Events: <none>*

You need to use kubectl to proxy the dashboard from the cluster on your laptop:

> *kubectl proxy*
>
> *Starting to serve on 127.0.0.1:8001*

Now, you can open http://localhost:8001/ui to view the dashboard!!

Alternatively, you can modify the kubernetes deployment to change from ClusterIP to NodePort and expose the dashboard via any of the nodes.

> *kubectl -n kube-system edit service kubernetes-dashboard*
>
> *(and change type: ClusterIP to type:NodePort)*
>
> *kubectl describe services kubernetes-dashboard—namespace=kube-system|grep NodePort*
>
> *Type: NodePort*

> *NodePort: <unset> 30871/TCP*

Now you can access the dashboard using any node eg:
http://tcdocker2:30871/ui

Thats it!! You can as well deploy several services such as heapster for resource monitoring, ELK stack and prometheus/grafana for detailed monitoring of the cluster and containerised apps.

If you find my article informative and detailed enough for an hands on deployment, please clap so it encourages me to write more often :)

UPDATE: To integrate kubernetes with LDAP, refer my new post:
https://medium.com/@pmvk/step-by-step-guide-to-integrate-ldap-with-kubernetes-1f3fe1ec644e