Docker, Kubernetes, VMware Tanzu and RedHat OCP
11Oct2022
Day06
Minikube installation
Kubernetes Workloads I

Install Minikube all-in-one

https://minikube.sigs.k8s.io/docs/start/
Minikube Installation Prerequisites
2 vCPUs
2GB memory
30GB HDD
Internet connection

Step 1.create a VM
download ubuntu iso image
https://releases.ubuntu.com/18.04/ubuntu-18.04.6-live-server-amd64.iso
https://releases.ubuntu.com/focal/ubuntu-20.04.5-live-server-amd64.iso
create template and install ubuntu linux through iso image
config ubuntu VM
login with devops user

```
u: devops
p: ubuntu
$ sudo -i
password: ubuntu
# passwd root
set pass: ubuntu
# hostnamectl set-hostname minikube.example.com
# ifconfig
# route -n
# cat /etc/resolv.conf
# vim /etc/netplan/00-installer-config.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: false
      addresses: [192.168.29.34/24]
      gateway4: 192.168.29.1
      nameservers:
       addresses: [8.8.8.8,8.8.4.4,192.168.29.1]
:wq!
# netplan apply
# vim /etc/ssh/sshd_config
33 PermitRootLogin yes
:wq!
# systemctl restart sshd.service
# poweroff
```

NOTE: take snapshot

Step 2. Install Kubectl
Kubectl is the command line tool for Kubernetes to interact with the cluster from the command line.

```
# apt-get update && sudo apt-get install -y apt-transport-https gnupg2
# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
# echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
# apt-get update
# apt-get install -y kubectl
# kubectl version
```

Step 3. Install additional dependencies

```
# apt-get install conntrack
```

Step 4. Install minikube

```
# apt-get install -y docker.io
# curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
# chmod +x minikube
# mv minikube /usr/local/bin
# set NO_PROXY=localhost,127.0.0.1,192.168.29.0/24
```

Step 5. Install cri-dockerd

5-1. wget the file

```
# wget https://github.com/Mirantis/cri-dockerd/releases/download/v0.2.0/cri-dockerd-v0.2.0-linux-amd64.tar.gz
```

5-2. unzip the package. On Linux you can use

```
# tar xvf cri-dockerd-v0.2.0-linux-amd64.tar.gz
```

5-3. move the cri-dockerd binary to your usr/local/bin directory

```
# mv ./cri-dockerd /usr/local/bin/
```

5-4. check if it is successfully installed

```
# cri-dockerd --help
```

5-5. start the service on Linux

```
# wget https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/packaging/systemd/cri-docker.service
# wget https://raw.githubusercontent.com/Mirantis/cri-dockerd/master/packaging/systemd/cri-docker.socket
# mv cri-docker.socket cri-docker.service /etc/systemd/system/
# sed -i -e 's,/usr/bin/cri-dockerd,/usr/local/bin/cri-dockerd,' /etc/systemd/system/cri-docker.service
# systemctl daemon-reload
# systemctl enable cri-docker.service
# systemctl enable --now cri-docker.socket
# systemctl status cri-docker.socket
```

Step 6. Install CRI-O Container Runtime on Ubuntu

```
# apt update && sudo apt upgrade
# OS=xUbuntu_18.04
# CRIO_VERSION=1.23
# echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/ /"|sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
# echo "deb http://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable:/cri-o:/$CRIO_VERSION/$OS/ /"|sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable:cri-o:$CRIO_VERSION.list
# curl -L https://download.opensuse.org/repositories/devel:kubic:libcontainers:stable:cri-o:$CRIO_VERSION/$OS/Release.key | sudo apt-key add -
# curl -L https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/$OS/Release.key | sudo apt-key add -
# apt update
# apt install cri-o cri-o-runc
# systemctl enable crio.service
# systemctl start crio.service
# apt install cri-tools
```

Step 7. Start the minikube

```
# minikube start --vm-driver=none --docker-env NO_PROXY=$NO_PROXY
```

after install success

```
# sudo mv /root/.kube /root/.minikube $HOME
# sudo chown -R $USER $HOME/.kube $HOME/.minikube
# minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

```
# minikube completion bash >>~/.bashrc
# kubectl completion bash >>~/.bashrc
# source ~/.bashrc
# echo “autocmd FileType yaml setlocal ai ts=2 sw=2 et cursorcolumn” >~/.vimrc
# kubectl get all -A
```

Refrences

https://www.mirantis.com/blog/how-to-install-cri-dockerd-and-migrate-nodes-from-dockershim/
https://developer.ibm.com/tutorials/set-up-minikube-on-ubuntu-server-within-minutes/
https://computingforgeeks.com/install-cri-o-container-runtime-on-ubuntu-linux/

Workloads

https://kubernetes.io/docs/concepts/workloads/

A workload is an application running on Kubernetes.

1-Pod

Pods

https://kubernetes.io/docs/concepts/workloads/pods/

Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.

Create

cli

```
# kubectl run <pod-name> --image <image-name>
# kubectl run pod1 --image nginx --dry-run=client
# kubectl run pod1 --image nginx
```

file

```
# kubectl run  pod1 --image nginx -o yaml --dry-run=client >/tmp/pod1.yaml
# kubectl run  pod1 --image nginx -o json --dry-run=client >/tmp/pod1.json
# cat /tmp/pod1.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: nginx
    name: nginxcnt
# kubectl create -f /tmp/pod1.yaml --dry-run=client
# kubectl create -f /tmp/pod1.yaml
```

operate

```
# kubectl get pod
# kubectl describe pod pod1
# kubectl edit pod pod1
# kubectl delete pod pod1 --force --grace-period=0
```

edit pod parameters

upgrade/downgrade the image

online

```
# kubectl get pod pod1 -o yaml | grep -i "image"
  - image: nginx
# kubectl edit pod pod1
19 spec:
 20   containers:
 21   - image: nginx:1.18
:wq!
# kubectl get pod pod1 -o yaml | grep -i "image"
  - image: nginx:1.18
```

offline

```
# kubectl get pod pod1 -o yaml >/tmp/pod1running.yaml
# vim /tmp/pod1running.yaml
# kubectl replace -f /tmp/pod1running.yaml                    create and replace      apply and apply
# kubectl get pod pod1 -o yaml | grep -i "image"
  - image: nginx:1.19
```

Backup

```
kubectl get pod pod1 -o yaml >/tmp/pod1back.yaml
```

Delete

```
# kubectl delete pod pod1 --force --grace-period=0
```

Restore

```
# kubectl apply -f /tmp/pod1back.yaml
```

Log

```
# kubectl logs pods/pod1
```

get inside pod

```
# kubectl exec -it pod/pod1 -- /bin/bash
root@pod1:/# exit
```

```
exit
```

or

```
# kubectl exec -it pod/pod1 -- ls /
```

run workloads in different namespace

1-change namespace and create resource

```
# kubectl config get-contexts
# kubectl config set-context --namespace test1 –current
# kubectl config get-contexts
*        kubernetes-admin@kubernetes   kubernetes   kubernetes-admin   test1
# kubectl run pod1 --image redis
# kubectl get pods
pod1   1/1     Running   0         6s
```

2-create resource without namespace

```
# kubectl config get-contexts
*        kubernetes-admin@kubernetes   kubernetes   kubernetes-admin   test1
# kubectl run redis --image redis –namespace <namespace>
# kubectl run redis --image redis --namespace default
# kubectl get pods --namespace default
```

or

```
# kubectl run  pod3 --image alpine -o yaml --dry-run=client >alpine.yaml
# vim alpine.yaml
cat alpine.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod3
  name: pod3
  namespace: test1
spec:
  containers:
  - image: alpine
    name: alpinecnt
:wq!
# kubectl apply -f alpine.yaml
# kubectl get pods --namespace test1
```

create Pod with multiple containers

```
# kubectl run pod1 --image nginx -o yaml --dry-run=client >pod1.yaml
# vim  pod1.yaml l
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: pod1
  name: pod1
spec:
  containers:
  - image: nginx
    name: nginxcnt
  - image: redis
    name: rediscnt
:wq!
# kubectl create -f pod1.yaml --dry-run=client
# kubectl create -f pod1.yaml
# kubectl get pods -o wide
NAME   READY  STATUS    RESTARTS  AGE    IP             NODE
pod1     2/2        Running   0                106s   192.168.1.3   node01
```

get inside Redis Container

```
# kubectl exec -it pod/pod1 -c redicnt -- /bin/bash
root@pod1:/data#
exit
-c container
```

get nginx container Log

```
# kubectl logs pod/pod1 -c nginxcnt
# kubectl logs pod/pod1 -c redicnt
```

send/receive file

```
kubectl cp /tmp/f1 pod1:/tmp -c nginxcnt
kubectl exec -it pod/pod1 -c nginxcnt -- ls /tmp
```