# Kubernetes Lab

# Connect to Kubernetes Master Node using the Public IP

# Open an SSH client.

# Locate your private key file (singaporekeypair.pem)

# # chmod 400 singaporekeypair.pem

# Now you will be able to SSH using your Public DNS/Public IP

# ssh -i "singaporekeypair.pem" [ec2-user@ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com](mailto:ec2-user@ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com)

# NB:- If SSH connectivity does not work, see changing the chmod to 600 (chmod 600 singaporekeypair.pem)

# [ec2-user@ip-xx-xx-xx-xx ~]$ sudo su

# # yum install -y git [Only If GIT is not installed in your EC2 instance]

# Steps to Check out from git repository

# # git init

# # git config --global user.email "nevin.cleetus@gmail.com"

# # git config --global user.name "nevin-cleetus"

# # mkdir gitrepo

# # cd gitrepo

# # git clone https://github.com/nevin-cleetus/kubernetes.git

# Ensure

# 1. Internet is working

# 2. Nobody should be connected to vpn or any other proxy.

# 3. Disable firewall if enabled.

# 

# 

# 

# Lab Exercise 1

# Role Based Access control

# Verify the Kubernetes Master and Worker Node(s) are in running state

# [ec2-user@ip-xx-xx-xx-xx ~]$ sudo su

[root@ip- kubernetes] kubectl get nodes

Confirm both Master and Worker node is in Ready state.

NAME STATUS ROLES AGE VERSION

ip- xxxxxxx Ready master 15m v1.18.0

ip-xxxxxxx Ready <none> 10m v1.18.0

[root@ip- kubernetes] kubectl api-versions | grep -i 'rbac.authorization.k8s.io/v1'

Expected Result

bac.authorization.k8s.io/v1

rbac.authorization.k8s.io/v1beta1

1. Create a user name Dave
2. Give Dave access to a namespace called ‘devops’
3. Private Key (dave.key), Certificate Signing Request for Dave (dave.csr)

Requirements

1. Kubernetes Certificate Authority private key and certificate in order to sign Dave’s certificate request (/etc/Kubernetes/pki)

Dave would need his private key and certificate to login to the Kubernetes

Connect to Kubernetes master

1. Create the private key

[root@ip- kubernetes] openssl genrsa -out dave.key 2048

Output: - Generating RSA private key, 2048 bit long modulus

.....+++

.....+++

e is 65537 (0x10001)

1. Create a certificate signing request

[root@ip- kubernetes] openssl req -new -key dave.key -out dave.csr -subj "/CN=dave/O=devops"

[root@ip- kubernetes] ls

dave.csr dave.key

1. Sign Dave’s certificate

We need Certificate Authorities certificate and private key

[root@ip- kubernetes] cp /etc/kubernetes/pki/ca.crt /home/vagrant/rback-dave/

[root@ip- kubernetes] sudo cp /etc/kubernetes/pki/ca.key /home/vagrant/rback-dave/

[root@ip- kubernetes] openssl x509 -req -in dave.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out dave.crt -days 365

Signature ok

subject=/CN=dave/O=devops

Getting CA Private Key

1. We need to create a kube-config file for Dave to access our Kubernetes cluster.
2. Two ways we can do this.
3. Let Dave can create the kube-config file using the file we generated.

[root@k8masterMLB rback-dave]# kubectl --kubeconfig dave.kubeconfig config set-cluster kubernetes --server https://172.42.42.100:6443 --certificate-authority=ca.crt

Cluster "kubernetes" set.

[root@k8masterMLB rback-dave]# kubectl create namespace devops

Add user to the kube-config file

kubectl --kubeconfig dave.kubeconfig config set-credentials dave --client-certificate /home/vagrant/rback-dave/dave.crt --client-key /home/vagrant/rback-dave/dave.key

User "dave" set.

Set the context

[root@k8masterMLB rback-dave]# kubectl --kubeconfig dave.kubeconfig config set-context dave-kubernetes --cluster kubernetes --namespace devops --user dave

[root@k8masterMLB rback-dave]# kubectl --kubeconfig dave.kubeconfig config set-context dave-kubernetes --cluster kubernetes --namespace devops --user dave

Context "dave-kubernetes" created.

[root@k8masterMLB rback-dave]# vi dave.kubeconfig

Modify the current-context to dave-kubernetes

current-context: dave-kubernetes

Now Dave can copy the dave.kubeconfig file to ~/.kube/config

[root@k8masterMLB rback-dave]# kubectl --kubeconfig dave.kubeconfig get pods

Error from server (Forbidden): pods is forbidden: User "dave" cannot list resource "pods" in API group "" in the namespace "devops"

From the Master Admin context

[root@k8masterMLB rback-dave]# kubectl create role dave-devops --verb=get,list --resource=pods --namespace devops

role.rbac.authorization.k8s.io/dave-devops created

kubectl create rolebinding dave-devops-role-binding --role=dave-devops --user=dave --namespace devops

rolebinding.rbac.authorization.k8s.io/dave-devops-role-binding created

# Kubernetes Networking

# tcpdump on kubernetes pod using Calico as cni

Posted on [16 May 2018](https://akhileshthipparthi.wordpress.com/2018/05/16/tcpdump-on-kubernetes-pod-using-calico-as-cni/) by [akhileshthipparthi](https://akhileshthipparthi.wordpress.com/author/akhileshthipparthi/)

There might be situations during debugging where we may have to take tcpdumps on pods .Below steps help to take tcpdump on a K8s cluster with calico as cni,

1. Identify Worker node where pod is running and list down the the POD IP . You will be able to get those details using below commands,
   * kubectl describe po -n dev
2. Login to above worker node and list the interfaces using, **ip route** and filter interface matching the pod IP.root@k8s-node-0:~# ip route | grep 10.112.12.53  
   10.112.12.53 dev calixxxxxxxx scope link
3. Take tcpdump on interface cali\*\*\*\*\*\*\* which is the host side of the VETH pair connecting the container back to the root or default network namespace on the host

tcpdump -i  calixxxxxxxxx  -w /opt/capture.pcap &

Purpose

Many pods are built on a limited OS and so are missing common utilities, so we may not be able to ping the network packet capture from inside the pod. You can run it from worker node the pod is running on.

Procedure

Find the container name and node your app is running on.

kubectl get pod mypod -o json

:

"containerID": "docker://ddaaad0f556d2b1e5d4298bcc22c1701ff15e82c7a335b340334d852abe9af2e",

:

"hostIP": "10.193.90.92",

Then from the worker with the IP "10.193.90.92", find the pods unique network interface index inside it's container.

docker exec ddaaad0f556d2b1e5d4298bcc22c1701ff15e82c7a335b340334d852abe9af2e /bin/bash -c 'cat /sys/class/net/eth0/iflink'

13

Then take the result from that and locate that interface on the worker

for i in /sys/class/net/veth\*/ifindex; do grep -l 13 $i; done

/sys/class/net/veth235ab8ff/ifindex

Or

ip link |grep ^13:

13: [veth235ab8ff@flannel.1](mailto:veth235ab8ff@flannel.1): <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1450 qdisc noqueue master cni0 state UP mode DEFAULT group defaul

From this we see the virtual interface of the container is 'veth235ab8ff'.  We can use that interface to perform network packet capture's on.

tcpdump -i veth235ab8ff

Access ETCD

[root@k8masterMLB vagrant]# docker ps | grep etcd

Select the container id with ‘etcd --advertise-cl…’

[root@k8masterMLB vagrant]# docker exec -it a08243cc05b1 /bin/sh

# ETCDCTL\_API=3

# etcdctl --endpoints 127.0.0.1:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key get / --prefix --keys-only

# etcdctl --endpoints 127.0.0.1:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key get / calico

etcdctl --endpoints 127.0.0.1:2379 --cacert /etc/kubernetes/pki/etcd/ca.crt --cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key get / calico/

# Calico

kubectl get po -o json | jq .items[].status.podIP -r

Master -> kubectl get pods

Use one of the pod id

Go to worker node

[root@ip-172-31-31-122 ec2-user]# docker inspect -f '{{.State.Pid}}' 6295f124d788

1596

To enter the namespace

[root@ip-172-31-31-122 ec2-user]# nsenter -t 1596 -n ip a

Go to Master and check the interfaces

kubectl exec -it backend-deployment-7c84dfbf47-4v7gz /bin/sh

# ip a

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000

link/ipip 0.0.0.0 brd 0.0.0.0

4: eth0@if156: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1440 qdisc noqueue state UP group default

link/ether f2:33:d2:87:af:fd brd ff:ff:ff:ff:ff:ff

inet 192.168.24.88/32 scope global eth0

valid\_lft forever preferred\_lft forever

# ip route

default via 169.254.1.1 dev eth0

169.254.1.1 dev eth0 scope link

Worker 1

tcpdump -i cali0a24e7b863a@if4 -nn

ifconfig cali0a24e7b863a@if4