K8S Overlay Network with High Availability

High Availability K8S Cluster Architecture

#	Available Feature		Used Tools	Version	Status
1	Environment setup	Environment setting	3 Ubuntu VMs: spec for each vCPUs: 4 vRAM: 8GB Disk: 100GB	18.04	Done
		Container runtime	Containerd://1.6.14 Docker	20.10.22	Done
		Container orchestrator	Kubernetes cluster	v1.26.0	Done
		Container Network Interface (CNI)	Antrea	latest	Done
2	IP security protocol (IPsec)		Antrea CNI enabled IPsec protocol	latest	Done
3	High availability support	Virtual IP	Pacemaker	1.1.18	Done
		Cluster membership	Corosync	2.4.3	Done
		Load balancer	HA proxy	1.8.8	Done
4	Distributed storages		Longhorn	1.4.0	Done

Contents

All commands run in **root** mode

I. Install Corosync and Pacemaker

In this section, please pay attention on the command runs on which machine

Now run these commands in three master nodes

Add all master nodes IP to **/etc/hosts** file
root# vim /etc/hosts
192.168.122.243 master1
192.168.122.180 master2
192.168.122.107 master3

The hosts file in master1 should be like

```
127.0.0.1       localhost
127.0.1.1       master1

192.168.122.243 master1
192.168.122.180 master2
192.168.122.107 master3

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

允許作為ssh server讓外部機器連進(若沒有這個folder表示尚未安裝openssh-server)
Enable root ssh, public key authentication for ssh in three machines
root# vim /etc/ssh/sshd_config
PermitRootLogin yes
PubkeyAuthentication yes
PasswordAuthentication yes

Enable three lines as the figure bellow

```
#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile     .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
#PermitEmptyPasswords no
```

Restart ssh sshd service
root# systemctl restart ssh sshd

Run these commands in only master1
因為ssh是用非對稱方式把訊息加密，兩方溝通之前要有對方的key
Generate ssh public key
root@master1# ssh-keygen -t rsa -b 4096

Just press Enter until done with the follow output

```
root@master1:~# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Created directory '/root/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:hSBqp9idSgrYf/nsTbKaByxhwdXhocBvmgH4qD/shww root@master1
The key's randomart image is:
+---[RSA 4096]----+
| . oo.o.o.       |
|. ..+o +.o       |
| oo..o. o .      |
|o=.++.o  .       |
|= =.oB  S        |
|Eo o+ o.         |
|.=....o.. .      |
|  * .. +.=       |
| ..o  o+= .      |
+----[SHA256]-----+
```

Copy this public key to the rest two machines master2 and master3: This command runs in master1 machine.

root@master1:~# ssh-copy-id root@192.168.122.180
root@master1:~# ssh-copy-id root@192.168.122.107

The output of these 2 commands as bellow

```
root@master1:~# ssh-copy-id root@192.168.122.180
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '192.168.122.180 (192.168.122.180)' can't be established.
ECDSA key fingerprint is SHA256:wBqozWD+vLG5A2YdqcRJ46lS5Ej+JATby05rssz9wys.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@192.168.122.180's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@192.168.122.180'"
and check to make sure that only the key(s) you wanted were added.

root@master1:~#
root@master1:~# ssh-copy-id root@192.168.122.107
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host '192.168.122.107 (192.168.122.107)' can't be established.
ECDSA key fingerprint is SHA256:SpEnJ1HROhKKL+x7G/nsiEpArEebNWX52vEq8mLBp+E.
Are you sure you want to continue connecting (yes/no)? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@192.168.122.107's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@192.168.122.107'"
and check to make sure that only the key(s) you wanted were added.
```

為甚麼不需要passwd了，是因為有對方的public key嗎？？？

Verify that success SSH to others master nodes **without password** as figure bellow

```
root@master1:~# ssh root@192.168.122.107
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-84-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

8 updates can be applied immediately.
8 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

New release '20.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2023.
*** System restart required ***
Last login: Thu Dec 29 14:32:03 2022 from 192.168.122.243
root@master3:~# WE ALREADY SSH TO MASTER3 WITHOUT ANY PASSWORD :)))))))))))
```

Now run these commands in three master nodes

root# apt-get update -y
root# apt-get -y install ntp
root# apt-get install pacemaker -y

Run these commands in only master1

root@master1:~# apt-get install haveged -y
root@master1:~# corosync-keygen

Copy the corosync key file to the others two master node

root@master1:~# scp /etc/corosync/authkey root@192.168.122.180:/etc/corosync/
root@master1:~# scp /etc/corosync/authkey root@192.168.122.107:/etc/corosync/

The output should be like

```
root@master1:~# scp /etc/corosync/authkey root@192.168.122.180:/etc/corosync/
authkey                                                                                100%  128    33.3KB/s   00
root@master1:~# scp /etc/corosync/authkey root@192.168.122.107:/etc/corosync/
authkey                                                                                100%  128   136.2KB/s   00
```

Now run these commands in master2 and master3

root# chown root: /etc/corosync/authkey
root# chmod 400 /etc/corosync/authkey

Now run these commands in three master nodes

root# mv /etc/corosync/corosync.conf /etc/corosync/corosync.conf.bk

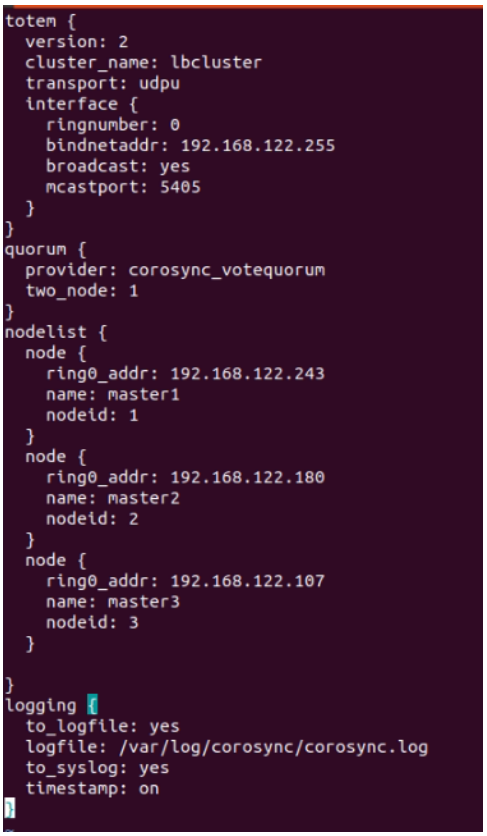
Add bellow content to /etc/corosync/corosync.conf file, please change the IP with your three master nodes's IP
root# vim /etc/corosync/corosync.conf

```
totem {
  version: 2
  cluster_name: lbcluster
  transport: udpu
  interface {
    ringnumber: 0
    bindnetaddr: 192.168.122.255
    broadcast: yes
    mcastport: 5405
  }
}
quorum {
  provider: corosync_votequorum
  two_node: 1
}
nodelist {
  node {
    ring0_addr: 192.168.122.243
    name: master1
    nodeid: 1
  }
  node {
    ring0_addr: 192.168.122.180
    name: master2
    nodeid: 2
  }
  node {
    ring0_addr: 192.168.122.107
    name: master3
    nodeid: 3
  }

}
logging {
```

```
  to_logfile: yes
  logfile: /var/log/corosync/corosync.log
  to_syslog: yes
  timestamp: on
}
```

The file's content should be like



root# mkdir -p /etc/corosync/service.d

Create and add bellow content to /etc/corosync/service.d/pcmk
root# vim /etc/corosync/service.d/pcmk

```
service {
  name: pacemaker
  ver: 1
}
```

The file content should be like

```
root@master1: ~
service {
  name: pacemaker
  ver: 1
}
```

Create and add bellow content to /etc/default/corosync
root# vim /etc/default/corosync

START=yes

The file content should be like

```
# Corosync runtime directory
#COROSYNC_RUN_DIR=/var/lib/corosync

# Path to corosync.conf
#COROSYNC_MAIN_CONFIG_FILE=/etc/corosync/corosync.conf

# Path to authfile
#COROSYNC_TOTEM_AUTHKEY_FILE=/etc/corosync/authkey

# Command line options
#OPTIONS=""
START=yes
~
```

Restart corosync service
root# service corosync restart

Verify successful installation, the correct installation should show three master nodes joined to the corosync group as bellow
root# corosync-cmapctl | grep members

authkey沒有用到???????

The output should be like

```
root@master1:~# corosync-cmapctl | grep members
runtime.totem.pg.mrp.srp.members.1.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.1.ip (str) = r(0) ip(192.168.122.243)
runtime.totem.pg.mrp.srp.members.1.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.1.status (str) = joined
runtime.totem.pg.mrp.srp.members.2.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.2.ip (str) = r(0) ip(192.168.122.180)
runtime.totem.pg.mrp.srp.members.2.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.2.status (str) = joined
runtime.totem.pg.mrp.srp.members.3.config_version (u64) = 0
runtime.totem.pg.mrp.srp.members.3.ip (str) = r(0) ip(192.168.122.107)
runtime.totem.pg.mrp.srp.members.3.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.3.status (str) = joined
```

Update Pacemaker service
root# update-rc.d pacemaker defaults 20 01
root# service pacemaker restart
root# apt install crmsh -y

重新啟動各節點

Verify successful install Pacemaker, there are three nodes Online. This command should be able to run on three nodes with the same output.
root# crm status

The output should be like

```
root@master1:~# crm status
Stack: corosync
Current DC: master1 (version 1.1.18-2b07d5c5a9) - partition with quorum
Last updated: Thu Dec 29 16:39:31 2022
Last change: Thu Dec 29 16:32:16 2022 by hacluster via crmd on master1

3 nodes configured
0 resources configured

Online: [ master1 master2 master3 ]

No resources
```

Run these commands in only master1

root@master1:~# crm configure property stonith-enabled=false
root@master1:~# crm configure property no-quorum-policy=ignore

Create virtual IP (vIP) resource for the cluster, which is **192.168.122.250**. The IP should be in the same subnet with the primary interface. It will be added as the secondary interface to the primary interface.
root@master1:~# crm configure primitive virtual_public_ip ocf:heartbeat:IPaddr2 params ip="192.168.122.250" cidr_netmask="24" op monitor interval="10s" meta migration-threshold="2" failure-timeout="60s" resource-stickiness="100"

Verify successful create vIP. The vIP is added to **enp1s0** interface as the secondary. The output should be like
root@master1:~# ip a

試想這個VIP的目的是？就是故障轉移？
我想就是

(我的是ens33)雖有不同但我想沒差

```
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 100
    link/ether 52:54:00:26:8a:20 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.243/24 brd 192.168.122.255 scope global dynamic noprefixroute enp1s0
       valid_lft 2535sec preferred_lft 2535sec
    inet 192.168.122.250/24 brd 192.168.122.255 scope global secondary enp1s0
       valid_lft forever preferred_lft forever
    inet6 fe80::284f:703b:4d07:d0cc/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Verify resource with crm command. The vIP now is placed at master1.

```
root@master1:~# crm status
Stack: corosync
Current DC: master1 (version 1.1.18-2b07d5c5a9) - partition with quorum
Last updated: Thu Dec 29 16:50:37 2022
Last change: Thu Dec 29 16:48:09 2022 by root via cibadmin on master1

3 nodes configured
1 resource configured

Online: [ master1 master2 master3 ]

Full list of resources:

 virtual_public_ip      (ocf::heartbeat:IPaddr2):       Started master1
```

其實我這邊多了一個
OFFLINE: [ubuntu]
不確定有沒有問題？

Run these commands in only master2

Verify High Availability: Because vIP is placed at master1 now. We can disable network at master1 to see where the vIP will be migrated.

master1 is offline now, the vIP is migrated to master2
root@master2:~# crm status

```
root@master2:~# crm status
Stack: corosync
Current DC: master3 (version 1.1.18-2b07d5c5a9) - partition with quorum
Last updated: Thu Dec 29 16:53:28 2022
Last change: Thu Dec 29 16:48:09 2022 by root via cibadmin on master1

3 nodes configured
1 resource configured

Online: [ master2 master3 ]
OFFLINE: [ master1 ]

Full list of resources:

 virtual_public_ip      (ocf::heartbeat:IPaddr2):       Started master2
```

Verify master2 network interface. The vIP 192.168.122.250 is added to the primary interface enp1s0.

```
root@master2:~# ip a | grep enp1s0
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 100
    inet 192.168.122.180/24 brd 192.168.122.255 scope global dynamic noprefixroute enp1s0
    inet 192.168.122.250/24 brd 192.168.122.255 scope global secondary enp1s0
```

第一次看下來不知道這功能是啥？

II. Install HAproxy

Now run these commands in three master nodes

root# apt install haproxy psmisc -y

Backup current config file
root# mv /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.bk

Create new config file with the following content, replace with your three master nodes IP
root# vim /etc/haproxy/haproxy.cfg

```
global
    log /dev/log  local0 warning
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon

   stats socket /var/lib/haproxy/stats

defaults
  log global
  option  httplog
  option  dontlognull
        timeout connect 5000
        timeout client 50000
        timeout server 50000

frontend kube-apiserver
  bind *:8443
  mode tcp
  option tcplog
  default_backend kube-apiserver
```

backend kube-apiserver
    mode tcp
    option tcplog
    option tcp-check
    balance roundrobin
    default-server inter 10s downinter 5s rise 2 fall 2 slowstart 60s maxconn 250 maxqueue 256 weight 100
    server kube-apiserver-1 192.168.122.243:6443 check # Replace the IP address with your own.
    server kube-apiserver-2 192.168.122.180:6443 check # Replace the IP address with your own.
    server kube-apiserver-3 192.168.122.107:6443 check # Replace the IP address with your own.

The file content should be like

```
global
    log /dev/log  local0 warning
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon

   stats socket /var/lib/haproxy/stats

defaults
  log global
  option  httplog
  option  dontlognull
        timeout connect 5000
        timeout client 50000
        timeout server 50000

frontend kube-apiserver
  bind *:8443
  mode tcp
  option tcplog
  default_backend kube-apiserver

backend kube-apiserver
    mode tcp
    option tcplog
    option tcp-check
    balance roundrobin
    default-server inter 10s downinter 5s rise 2 fall 2 slowstart 60s maxconn 250 maxqueue 256 weight 100
    server kube-apiserver-1 192.168.122.243:6443 check # Replace the IP address with your own.
    server kube-apiserver-2 192.168.122.180:6443 check # Replace the IP address with your own.
    server kube-apiserver-3 192.168.122.107:6443 check # Replace the IP address with your own.
~
```

user access IP:8443 (HAproxy server)，HAproxy server再去找kube api server執行服務

配置前端聆聽的埠號

配置後端伺服器資訊

Enable HAproxy service
root# systemctl restart haproxy
root# systemctl enable haproxy

The output should be like

```
root@master1:~# systemctl restart haproxy
root@master1:~# systemctl enable haproxy
Synchronizing state of haproxy.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable haproxy
```

Verify successful install HAproxy
root# systemctl status haproxy.service

The output should be like

```
root@master1:~# systemctl status haproxy.service
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2022-12-30 05:18:10 CST; 32s ago
     Docs: man:haproxy(1)
           file:/usr/share/doc/haproxy/configuration.txt.gz
 Main PID: 11891 (haproxy)
    Tasks: 2 (limit: 4915)
   CGroup: /system.slice/haproxy.service
           ├─11891 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
           └─11899 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
```

III. Install Docker

Now run these commands in three master nodes

root# cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF

這裡在幹嘛？

Output of these command should be like

```
root@master3:~# cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
> net.bridge.bridge-nf-call-ip6tables = 1
> net.bridge.bridge-nf-call-iptables = 1
> EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

Install Docker prerequisites packages
root# apt-get update
root# apt-get upgrade -y
root# apt-get install linux-image-extra-virtual -y
root# apt-get remove docker docker-engine docker.i
root# apt-get install apt-transport-https ca-certificates curl software-properties-common
root# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

Output of this command should be like

```
root@master1:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
OK
```

Add the follow line to /etc/apt/sources.list
root# vim /etc/apt/sources.list
deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable

The /etc/apt/sources.list file should be like

```
# deb-src http://security.ubuntu.com/ubuntu bionic-security universe
deb http://security.ubuntu.com/ubuntu bionic-security multiverse
# deb-src http://security.ubuntu.com/ubuntu bionic-security multiverse
#deb [arch=amd64] https://download.docker.com/linux/ubuntu xenial stable
# deb-src [arch=amd64] https://download.docker.com/linux/ubuntu xenial stable
deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable
# deb-src [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable
```

root# apt-get update

Run install Docker command
root# apt-get install docker-ce docker-ce-cli containerd.io

Check Docker installation status

root# systemctl status docker.service
The output should be like

```
root@master3:~# systemctl status docker.service
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2022-12-29 13:03:11 CST; 4min 15s ago
     Docs: https://docs.docker.com
 Main PID: 16223 (dockerd)
    Tasks: 10
   CGroup: /system.slice/docker.service
           └─16223 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

IV. Install K8S cluster

Now run these commands in three master nodes

Install prerequisites packages for K8S
root# apt-get install -y apt-transport-https ca-certificates curl

换成自己的pub key

sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys B53DC80D13EDEF05

Install K8S
root# curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg
root# echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
root# apt-get update
root# apt-get install -y kubelet kubeadm kubectl
root# apt-mark hold kubelet kubeadm kubectl

echo "deb [signed-by=/etc/apt/trusted.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

用下面兩行取代
sudo swapoff -a
sudo sed -i '/ swap / s/^/#/' /etc/fstab

The output of this command should be like

```
root@master1:~# apt-mark hold kubelet kubeadm kubectl
kubelet set on hold.
kubeadm set on hold.
kubectl set on hold.
```

Turn off the swap
root# swapoff -a

Disable this line in /etc/fstab file: /dev/mapper/ubuntu--vg-swap_1
root# vim /etc/fstab
comment out:
#/dev/mapper/ubuntu--vg-swap_1

The file content should be like

```
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust way to name devices
# that works even if disks are added and removed. See fstab(5).
#
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
/dev/mapper/ubuntu--vg-root /               ext4    errors=remount-ro 0       1
#/dev/mapper/ubuntu--vg-swap_1 none            swap    sw              0       0
~
```

Add the follow lines to **/etc/docker/daemon.json** file
root# vim /etc/docker/daemon.json

```
{
   "exec-opts": ["native.cgroupdriver=systemd"]
}
```

The content of the file should be like

```
{
    "exec-opts": ["native.cgroupdriver=systemd"]
}
```

Restart Docker service
root# systemctl enable docker
root# systemctl restart docker
root# systemctl daemon-reload

V. Install K8S Continue

Now run these commands in three master nodes

root# rm /etc/containerd/config.toml
root# systemctl restart containerd

Run these commands in only master1

Deploy K8S master1, **please replace endpoint IP with your virtual IP (vIP)**

root@master1:~# kubeadm init --control-plane-endpoint "192.168.122.250:8443" --pod-network-cidr=192.168.0.0/16 --upload-certs

Congratulation 😊**!!!!** If you successfully install your K8S, then the output should be similar as bellow. I would explain a bit about this output for you now.

Please pay attention on two lines **kubeadm join** in bellow output **.** The first join line with **--control-plane** option means that you want to join the other nodes as the control-plane rule (master rule). While the second join line **without --control-plane** means you want to join the other nodes as worker rule. In our system, we want all the nodes are the control-plane rule, so that we copy the first **kubeadm join** line, and paste in the other nodes terminal to join to the K8S cluster.

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

  export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of the control-plane node running the following command on each as root:

  kubeadm join 192.168.122.250:8443 --token ywsjut.f2axhr5g48c7j7rm \
        --discovery-token-ca-cert-hash sha256:678344fc4678b3db338458abc8cf071339ae320f9471202cace3eca07656eee4 \
        --control-plane --certificate-key ff791a58ca67948d8cb4401dfd71f558ecc6a2653482f320461ad38d8063ffe4

Please note that the certificate-key gives access to cluster sensitive data, keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary, you can use
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 192.168.122.250:8443 --token ywsjut.f2axhr5g48c7j7rm \
        --discovery-token-ca-cert-hash sha256:678344fc4678b3db338458abc8cf071339ae320f9471202cace3eca07656eee4
```

Run these commands to start your K8S cluster

root@master1:~# mkdir -p $HOME/.kube

root@master1:~# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root@master1:~# sudo chown $(id -u):$(id -g) $HOME/.kube/config
root@master1:~# source <(kubectl completion bash)
root@master1:~# echo "source <(kubectl completion bash)" >> ~/.bashrc

Run these commands in master2 and master3

Copy the **kubeadm join** with **--control-plane** option in the output of the right previous step, and paste in master2 and master3 terminal to join to the K8S cluster. Please don't copy my command bellow because the token is totally different with your machine.
Pay attention!! this token is only preserved for 24 hours. If later you want to join more nodes to K8S cluster, then you need to create another token as reference here:
https://facsiaginsa.com/kubernetes/join-existing-kubernetes-cluster

root# kubeadm join 192.168.122.250:8443 --token ywsjut.f2axhr5g48c7j7rm \
--discovery-token-ca-cert-hash sha256:678344fc4678b3db338458abc8cf071339ae320f9471202cace3eca07656eee4 \
--control-plane --certificate-key ff791a58ca67948d8cb4401dfd71f558ecc6a2653482f320461ad38d8063ffe4

--apiserver-advertise-address $MASTER_IP
我這裡還要加上這個flag才可以運作

Output of this command should be like

```
This node has joined the cluster and a new control plane instance was created:

* Certificate signing request was sent to apiserver and approval was received.
* The Kubelet was informed of the new secure connection details.
* Control plane label and taint were applied to the new node.
* The Kubernetes control plane instances scaled up.
* A new etcd member was added to the local/stacked etcd cluster.

To start administering your cluster from this node, you need to run the following as a regular user

        mkdir -p $HOME/.kube
        sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
        sudo chown $(id -u):$(id -g) $HOME/.kube/config

Run 'kubectl get nodes' to see this node join the cluster.
```

Run these commands to start your K8S cluster
root# mkdir -p $HOME/.kube
root# sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
root# sudo chown $(id -u):$(id -g) $HOME/.kube/config
root# source <(kubectl completion bash)
root# echo "source <(kubectl completion bash)" >> ~/.bashrc

By default, K8S disables deploy application pod in the K8S master node. We need to run these commands to enable it.
root# kubectl taint nodes --all node-role.kubernetes.io/control-plane:NoSchedule-

Verify your K8S cluster. You can clearly observe that 3 master nodes in the cluster. The status are **NotReady**, and coredns pods **Pending** mean that three master nodes now can not find each other. You need to deploy CNI (Container Network Interface) to enable the communication between these nodes.

```
root@master1:~# kubectl get nodes -o wide
NAME      STATUS     ROLES           AGE   VERSION   INTERNAL-IP       EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION     CONTAINER-RUNTIME
master1   NotReady   control-plane   39m   v1.26.0   192.168.122.243   <none>        Ubuntu 18.04.6 LTS   5.4.0-84-generic   containerd://1.6.14
master2   NotReady   control-plane   13m   v1.26.0   192.168.122.180   <none>        Ubuntu 18.04.6 LTS   5.4.0-84-generic   containerd://1.6.14
master3   NotReady   control-plane   13m   v1.26.0   192.168.122.107   <none>        Ubuntu 18.04.6 LTS   5.4.0-84-generic   containerd://1.6.14
root@master1:~# kubectl get pod --all-namespaces -o wide
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE   IP                NODE      NOMINATED NODE   READINESS GATES
kube-system   coredns-787d4945fb-5dcz7            0/1     Pending   0          39m   <none>            <none>    <none>           <none>
kube-system   coredns-787d4945fb-7bbpj            0/1     Pending   0          39m   <none>            <none>    <none>           <none>
kube-system   etcd-master1                        1/1     Running   0          39m   192.168.122.243   master1   <none>           <none>
kube-system   etcd-master2                        1/1     Running   0          14m   192.168.122.180   master2   <none>           <none>
kube-system   etcd-master3                        1/1     Running   0          13m   192.168.122.107   master3   <none>           <none>
kube-system   kube-apiserver-master1              1/1     Running   0          39m   192.168.122.243   master1   <none>           <none>
kube-system   kube-apiserver-master2              1/1     Running   0          13m   192.168.122.180   master2   <none>           <none>
kube-system   kube-apiserver-master3              1/1     Running   0          13m   192.168.122.107   master3   <none>           <none>
kube-system   kube-controller-manager-master1     1/1     Running   0          39m   192.168.122.243   master1   <none>           <none>
kube-system   kube-controller-manager-master2     1/1     Running   0          13m   192.168.122.180   master2   <none>           <none>
kube-system   kube-controller-manager-master3     1/1     Running   0          12m   192.168.122.107   master3   <none>           <none>
kube-system   kube-proxy-8szlr                    1/1     Running   0          14m   192.168.122.107   master3   <none>           <none>
kube-system   kube-proxy-jlnnp                    1/1     Running   0          14m   192.168.122.180   master2   <none>           <none>
kube-system   kube-proxy-r9dzj                    1/1     Running   0          39m   192.168.122.243   master1   <none>           <none>
kube-system   kube-scheduler-master1              1/1     Running   0          39m   192.168.122.243   master1   <none>           <none>
kube-system   kube-scheduler-master2              1/1     Running   0          14m   192.168.122.180   master2   <none>           <none>
kube-system   kube-scheduler-master3              1/1     Running   0          13m   192.168.122.107   master3   <none>           <none>
root@master1:~#
```

VI. Install Antrea CNI with IPsec supported

Run these commands in only master1

Copy the content of the manifest Antrea-ipsec file at https://raw.githubusercontent.com/antrea-io/antrea/main/build/yamls/antrea-ipsec.yml to antrea-cni-with-ipsec.yaml

root@master1:~# vim antrea-cni-with-ipsec.yaml

Enable the following lines
tunnelType: “gre”
trafficEncryptionMode: “ipsec”

and change your **preshared key**
psk: “whateveryouwant”

```
# Name of the interface antrea-agent will create and use for host <--> pod communication.
# Make sure it doesn't conflict with your existing interfaces.
hostGateway: "antrea-gw0"

# Determines how traffic is encapsulated. It has the following options:
# encap(default):    Inter-node Pod traffic is always encapsulated and Pod to external network
#                    traffic is SNAT'd.
# noEncap:           Inter-node Pod traffic is not encapsulated; Pod to external network traffic is
#                    SNAT'd if noSNAT is not set to true. Underlying network must be capable of
#                    supporting Pod traffic across IP subnets.
# hybrid:            noEncap if source and destination Nodes are on the same subnet, otherwise encap.
# networkPolicyOnly: Antrea enforces NetworkPolicy only, and utilizes CNI chaining and delegates Pod
#                    IPAM and connectivity to the primary CNI.
#
trafficEncapMode: "encap"

# Whether or not to SNAT (using the Node IP) the egress traffic from a Pod to the external network.
# This option is for the noEncap traffic mode only, and the default value is false. In the noEncap
# mode, if the cluster's Pod CIDR is reachable from the external network, then the Pod traffic to
# the external network needs not be SNAT'd. In the networkPolicyOnly mode, antrea-agent never
# performs SNAT and this option will be ignored; for other modes it must be set to false.
noSNAT: false

# Tunnel protocols used for encapsulating traffic across Nodes. If WireGuard is enabled in trafficEncryptionMode,
# this option will not take effect. Supported values:
# - geneve (default)
# - vxlan
# - gre
# - stt
# Note that "gre" is not supported for IPv6 clusters (IPv6-only or dual-stack clusters).
tunnelType: "gre"

# Determines how tunnel traffic is encrypted. Currently encryption only works with encap mode.
# It has the following options:
# - none (default):  Inter-node Pod traffic will not be encrypted.
# - ipsec:           Enable IPsec (ESP) encryption for Pod traffic across Nodes. Antrea uses
#                    Preshared Key (PSK) for IKE authentication. When IPsec tunnel is enabled,
#                    the PSK value must be passed to Antrea Agent through an environment
#                    variable: ANTREA_IPSEC_PSK.
# - wireGuard:       Enable WireGuard for tunnel traffic encryption.
trafficEncryptionMode: "ipsec"
```

```
---
# Source: antrea/templates/agent/ipsec-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: antrea-ipsec
  namespace: kube-system
  labels:
    app: antrea
type: Opaque
stringData:
  # Preshared Key used by IKE for authentication with peers.
  psk: "nemstest"
---
```

Choose one of two options to deploy Antrea CNI with IPsec

Option 1. Deploy Antrea CNI with IPsec supported by kubectl command

root@master1:~# kubectl apply -f antrea-cni-with-ipsec.yaml

Option 2. Alternatively, you can directly deploy by below command option, but you are not be able to modify the preshared key.

root@master1:~# kubectl apply -f https://raw.githubusercontent.com/antrea-io/antrea/main/build/yamls/antrea-ipsec.yml

Output should be like

```
root@master1:~# kubectl apply -f antrea-cni-with-ipsec.yaml

customresourcedefinition.apiextensions.k8s.io/antreaagentinfos.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/antreacontrollerinfos.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/clustergroups.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/clusternetworkpolicies.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/egresses.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/externalentities.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/externalippools.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/externalnodes.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/supportbundlecollections.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/tiers.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/traceflows.crd.antrea.io created
customresourcedefinition.apiextensions.k8s.io/trafficcontrols.crd.antrea.io created
serviceaccount/antrea-agent created
serviceaccount/antctl created
serviceaccount/antrea-controller created
secret/antrea-ipsec created
secret/antrea-agent-service-account-token created
secret/antctl-service-account-token created
configmap/antrea-config created
customresourcedefinition.apiextensions.k8s.io/groups.crd.antrea.io created
clusterrole.rbac.authorization.k8s.io/antrea-agent created
clusterrole.rbac.authorization.k8s.io/antctl created
clusterrole.rbac.authorization.k8s.io/antrea-cluster-identity-reader created
clusterrole.rbac.authorization.k8s.io/antrea-controller created
clusterrole.rbac.authorization.k8s.io/aggregate-antrea-policies-edit created
clusterrole.rbac.authorization.k8s.io/aggregate-antrea-policies-view created
clusterrole.rbac.authorization.k8s.io/aggregate-traceflows-edit created
clusterrole.rbac.authorization.k8s.io/aggregate-traceflows-view created
clusterrole.rbac.authorization.k8s.io/aggregate-antrea-clustergroups-edit created
clusterrole.rbac.authorization.k8s.io/aggregate-antrea-clustergroups-view created
clusterrolebinding.rbac.authorization.k8s.io/antrea-agent created
clusterrolebinding.rbac.authorization.k8s.io/antctl created
clusterrolebinding.rbac.authorization.k8s.io/antrea-controller created
service/antrea created
daemonset.apps/antrea-agent created
deployment.apps/antrea-controller created
apiservice.apiregistration.k8s.io/v1beta2.controlplane.antrea.io created
apiservice.apiregistration.k8s.io/v1beta1.system.antrea.io created
apiservice.apiregistration.k8s.io/v1alpha1.stats.antrea.io created
mutatingwebhookconfiguration.admissionregistration.k8s.io/crdmutator.antrea.io created
validatingwebhookconfiguration.admissionregistration.k8s.io/crdvalidator.antrea.io created
```

After install CNI, your K8S cluster should work well as the bellow. The status of all nodes is Ready

```
root@master1:~# kubectl get pod --all-namespaces -o wide
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE   IP                NODE
kube-system   antrea-agent-cnh82                  3/3     Running   0          76s   192.168.122.180   master2
kube-system   antrea-agent-g6xd9                  3/3     Running   0          76s   192.168.122.107   master3
kube-system   antrea-agent-qmr6g                  3/3     Running   0          76s   192.168.122.243   master1
kube-system   antrea-controller-6c84dc7b4b-qjt8s  1/1     Running   0          76s   192.168.122.243   master1
kube-system   coredns-787d4945fb-5dcz7            1/1     Running   0          14h   192.168.1.2       master2
kube-system   coredns-787d4945fb-7bbpj            1/1     Running   0          14h   192.168.1.3       master2
kube-system   etcd-master1                        1/1     Running   0          14h   192.168.122.243   master1
kube-system   etcd-master2                        1/1     Running   0          13h   192.168.122.180   master2
kube-system   etcd-master3                        1/1     Running   0          13h   192.168.122.107   master3
kube-system   kube-apiserver-master1              1/1     Running   0          14h   192.168.122.243   master1
kube-system   kube-apiserver-master2              1/1     Running   0          13h   192.168.122.180   master2
kube-system   kube-apiserver-master3              1/1     Running   0          13h   192.168.122.107   master3
kube-system   kube-controller-manager-master1     1/1     Running   0          14h   192.168.122.243   master1
kube-system   kube-controller-manager-master2     1/1     Running   0          13h   192.168.122.180   master2
kube-system   kube-controller-manager-master3     1/1     Running   0          13h   192.168.122.107   master3
kube-system   kube-proxy-8szlr                    1/1     Running   0          13h   192.168.122.107   master3
kube-system   kube-proxy-jlnnp                    1/1     Running   0          13h   192.168.122.180   master2
kube-system   kube-proxy-r9dzj                    1/1     Running   0          14h   192.168.122.243   master1
kube-system   kube-scheduler-master1              1/1     Running   0          14h   192.168.122.243   master1
kube-system   kube-scheduler-master2              1/1     Running   0          13h   192.168.122.180   master2
kube-system   kube-scheduler-master3              1/1     Running   0          13h   192.168.122.107   master3
root@master1:~# kubectl get nodes  -o wide
NAME      STATUS   ROLES           AGE   VERSION   INTERNAL-IP       EXTERNAL-IP   OS-IMAGE             KERNEL-
master1   Ready    control-plane   14h   v1.26.0   192.168.122.243   <none>        Ubuntu 18.04.6 LTS   5.4.0-8
master2   Ready    control-plane   13h   v1.26.0   192.168.122.180   <none>        Ubuntu 18.04.6 LTS   5.4.0-8
master3   Ready    control-plane   13h   v1.26.0   192.168.122.107   <none>        Ubuntu 18.04.6 LTS   5.4.0-8
```

VII. Test communication between master nodes

Run these commands in only one node. I use master1 to run it.

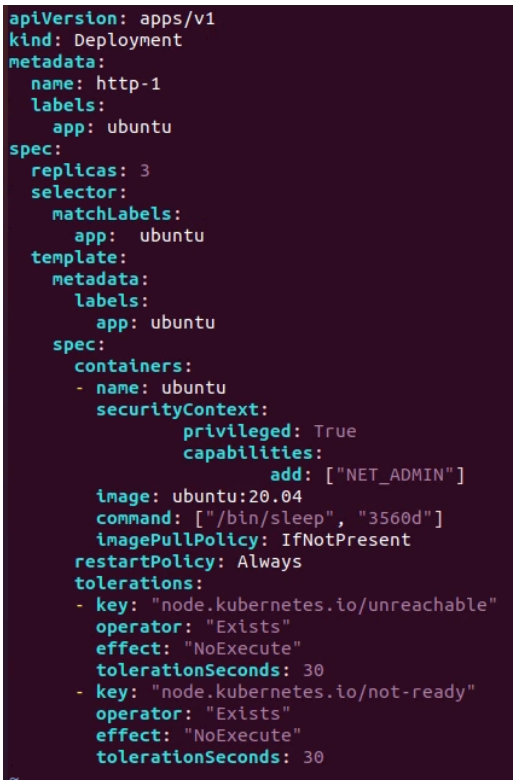
Your K8S Overlay network has been successfully setup up to this step. We would deploy some pods and verify the network communication between pods, and nodes.

Create and add the bellow content to a deployment yaml file.
root@master1:~# vim ubuntu.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: http-1
  labels:
    app: ubuntu
spec:
  replicas: 3
  selector:
    matchLabels:
      app:  ubuntu
  template:
    metadata:
      labels:
        app: ubuntu
    spec:
```

```
containers:
- name: ubuntu
  securityContext:
        privileged: True
        capabilities:
              add: ["NET_ADMIN"]
  image: ubuntu:20.04
  command: ["/bin/sleep", "3560d"]
  imagePullPolicy: IfNotPresent
restartPolicy: Always
tolerations:
- key: "node.kubernetes.io/unreachable"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 30
- key: "node.kubernetes.io/not-ready"
  operator: "Exists"
  effect: "NoExecute"
  tolerationSeconds: 30
```

The file content should be like



Create deployment with three ubuntu pods by kubectl command

root@master1:~# kubectl apply -f ubuntu.yaml

甚麼是pod？？？？？？？？？？

Three pods are deployed at three nodes in default namepsace

```
root@master1:~# kubectl get pod --all-namespaces -o wide
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE   IP               NODE
default       http-1-747cf89dff-8phts             1/1     Running   0          12s   192.168.1.4      master2
default       http-1-747cf89dff-ll7p7             1/1     Running   0          12s   192.168.0.2      master1
default       http-1-747cf89dff-zqgrz             1/1     Running   0          12s   192.168.2.2      master3
```

SSH to a pod, and try to ping to the other pods
root@master1:~# kubectl exec -it -n default http-1-747cf89dff-ll7p7 -- /bin/bash

```
root@master1:~# kubectl exec -it -n default http-1-747cf89dff-ll7p7 -- /bin/bash
root@http-1-747cf89dff-ll7p7:/# apt update
Get:1 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:3 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [1779 kB
```

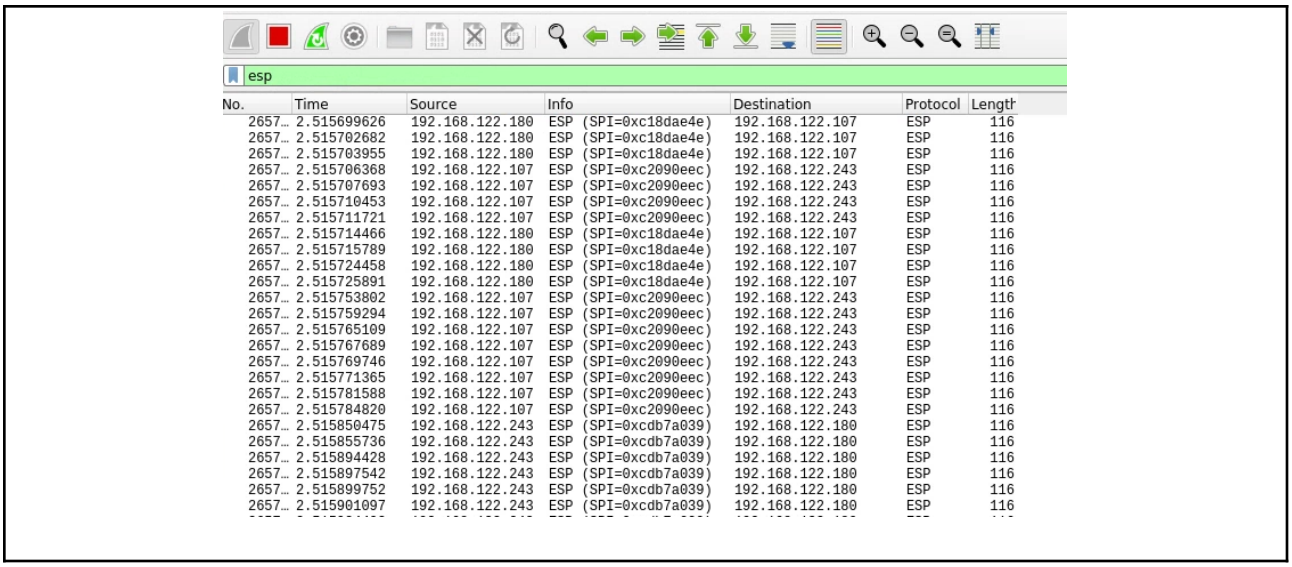
Install ping package
root@http-1-747cf89dff-ll7p7:/# apt install iputils-ping

```
root@http-1-747cf89dff-ll7p7:/# apt install  iputils-ping
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

Ping to the ubuntu pod that located in master2
root@http-1-747cf89dff-ll7p7:/# ping 192.168.1.4

```
root@http-1-747cf89dff-ll7p7:/# ping 192.168.1.4
PING 192.168.1.4 (192.168.1.4) 56(84) bytes of data.
64 bytes from 192.168.1.4: icmp_seq=1 ttl=62 time=2.86 ms
64 bytes from 192.168.1.4: icmp_seq=2 ttl=62 time=0.683 ms
64 bytes from 192.168.1.4: icmp_seq=3 ttl=62 time=0.335 ms
64 bytes from 192.168.1.4: icmp_seq=4 ttl=62 time=0.802 ms
64 bytes from 192.168.1.4: icmp_seq=5 ttl=62 time=1.02 ms
64 bytes from 192.168.1.4: icmp_seq=6 ttl=62 time=0.588 ms
```

Verify IPsec of the overlay network between master nodes. We can clearly observe that all the message between master nodes are encrypted



VIII. Install Longhorn

Now run these commands in three master nodes

root# apt-get install open-iscsi
root# apt-get install nfs-common

Run these commands in only one node. I use master1 to run it.

root@master1:~# kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/v1.4.0/deploy/longhorn.yaml

The output should be like

```
root@k8s:~# kubectl apply -f https://raw.githubusercontent.com/longhorn/longhorn/v1.4.0/deploy/longhorn.yaml
namespace/longhorn-system created
serviceaccount/longhorn-service-account created
serviceaccount/longhorn-support-bundle created
configmap/longhorn-default-setting created
configmap/longhorn-storageclass created
customresourcedefinition.apiextensions.k8s.io/backingimagedatasources.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/backingimagemanagers.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/backingimages.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/backups.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/backuptargets.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/backupvolumes.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/engineimages.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/engines.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/instancemanagers.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/nodes.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/orphans.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/recurringjobs.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/replicas.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/settings.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/sharemanagers.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/snapshots.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/supportbundles.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/systembackups.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/systemrestores.longhorn.io created
customresourcedefinition.apiextensions.k8s.io/volumes.longhorn.io created
clusterrole.rbac.authorization.k8s.io/longhorn-role created
clusterrolebinding.rbac.authorization.k8s.io/longhorn-bind created
clusterrolebinding.rbac.authorization.k8s.io/longhorn-support-bundle created
service/longhorn-backend created
service/longhorn-frontend created
service/longhorn-conversion-webhook created
service/longhorn-admission-webhook created
service/longhorn-recovery-backend created
service/longhorn-engine-manager created
service/longhorn-replica-manager created
daemonset.apps/longhorn-manager created
deployment.apps/longhorn-driver-deployer created
deployment.apps/longhorn-recovery-backend created
deployment.apps/longhorn-ui created
deployment.apps/longhorn-conversion-webhook created
deployment.apps/longhorn-admission-webhook created
```

Verify Longhorn deployment pods
root@master1:~# kubectl get pod --all-namespaces -o wide

The output should be like. In this part, the Master nodes IP has been changed because I did this part in another cluster, so the functionality should be the same.

```
root@master1:~# kubectl get pod -n longhorn-system -o wide
NAME                                                READY   STATUS    RESTARTS       AGE   IP              NODE
csi-attacher-7b5d5ff865-kr7bb                       1/1     Running   0              12m   192.168.0.11    master1
csi-attacher-7b5d5ff865-qbldj                       1/1     Running   0              12m   192.168.2.8     master3
csi-attacher-7b5d5ff865-zdvq7                       1/1     Running   0              12m   192.168.1.11    master2
csi-provisioner-68d785644d-jckrh                    1/1     Running   2 (2m15s ago)  12m   192.168.0.12    master1
csi-provisioner-68d785644d-jfvbf                    1/1     Running   0              12m   192.168.1.12    master2
csi-provisioner-68d785644d-wx6k6                    1/1     Running   0              12m   192.168.2.9     master3
csi-resizer-8558456d59-bqxng                        1/1     Running   0              12m   192.168.1.13    master2
csi-resizer-8558456d59-lb9wc                        1/1     Running   0              12m   192.168.0.13    master1
csi-resizer-8558456d59-z47s2                        1/1     Running   0              12m   192.168.2.10    master3
csi-snapshotter-588bd6f85d-fqsct                    1/1     Running   0              12m   192.168.0.14    master1
csi-snapshotter-588bd6f85d-sjrh7                    1/1     Running   0              12m   192.168.1.14    master2
csi-snapshotter-588bd6f85d-xjrp9                    1/1     Running   0              12m   192.168.2.11    master3
engine-image-ei-fc06c6fb-fp4rs                      1/1     Running   0              12m   192.168.0.8     master1
engine-image-ei-fc06c6fb-wwtc5                      1/1     Running   0              12m   192.168.1.8     master2
engine-image-ei-fc06c6fb-xndpb                      1/1     Running   0              12m   192.168.2.7     master3
instance-manager-e-731bc4d8df1b46a57e88e0951cc1fad4 1/1     Running   0              12m   192.168.0.9     master1
instance-manager-e-88c6ee91de31318db8013bd6151f5e65 1/1     Running   0              12m   192.168.1.7     master2
instance-manager-e-cb917883d85f8acdf9dbaeaacddd67af 1/1     Running   0              12m   192.168.2.5     master3
instance-manager-r-731bc4d8df1b46a57e88e0951cc1fad4 1/1     Running   0              12m   192.168.0.10    master1
instance-manager-r-88c6ee91de31318db8013bd6151f5e65 1/1     Running   0              12m   192.168.1.9     master2
instance-manager-r-cb917883d85f8acdf9dbaeaacddd67af 1/1     Running   0              12m   192.168.2.6     master3
longhorn-admission-webhook-ffd8997d5-s2t86          1/1     Running   0              13m   192.168.1.6     master2
longhorn-admission-webhook-ffd8997d5-sgjgn          1/1     Running   0              13m   192.168.0.7     master1
longhorn-conversion-webhook-59d556bb8d-lvszw        1/1     Running   0              13m   192.168.0.6     master1
longhorn-conversion-webhook-59d556bb8d-ttclv        1/1     Running   0              13m   192.168.1.5     master2
longhorn-csi-plugin-2wdhr                           3/3     Running   4 (9m20s ago)  12m   192.168.2.12    master3
longhorn-csi-plugin-dzw8l                           3/3     Running   0              12m   192.168.1.15    master2
longhorn-csi-plugin-p2bj9                           3/3     Running   0              12m   192.168.0.15    master1
longhorn-driver-deployer-767d8c6df7-ptdsn           1/1     Running   0              13m   192.168.0.2     master1
longhorn-manager-9t6d7                              1/1     Running   0              13m   192.168.0.4     master1
longhorn-manager-gp8cp                              1/1     Running   1 (12m ago)    13m   192.168.1.2     master2
longhorn-manager-pntd4                              1/1     Running   0              13m   192.168.2.4     master3
longhorn-recovery-backend-6c779c5db9-7c2g6          1/1     Running   0              13m   192.168.0.3     master1
longhorn-recovery-backend-6c779c5db9-bqsbr          1/1     Running   0              13m   192.168.1.3     master2
longhorn-ui-54d99fbf-6g6dm                          1/1     Running   0              13m   192.168.1.4     master2
longhorn-ui-54d99fbf-k986l                          1/1     Running   0              13m   192.168.0.5     master1
```

Enable UI access

By default, longhorn-frontend enables ClusterIP service for UI feature, it means that we can only access the UI from the internal cluster. It is not convenient to manage the cluster.
We would like to modify some code to expose the UI service to external. Nodeport service in K8S allows to expose the service inside the cluster to outside.

```
root@master1:~# kubectl get service -n longhorn-system
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)     AGE
csi-attacher                  ClusterIP   10.110.28.82     <none>        12345/TCP   20m
csi-provisioner               ClusterIP   10.97.195.20     <none>        12345/TCP   20m
csi-resizer                   ClusterIP   10.103.192.138   <none>        12345/TCP   20m
csi-snapshotter               ClusterIP   10.104.82.196    <none>        12345/TCP   20m
longhorn-admission-webhook    ClusterIP   10.100.20.160    <none>        9443/TCP    21m
longhorn-backend              ClusterIP   10.99.196.56     <none>        9500/TCP    21m
longhorn-conversion-webhook   ClusterIP   10.97.241.102    <none>        9443/TCP    21m
longhorn-engine-manager       ClusterIP   None             <none>        <none>      21m
longhorn-frontend             ClusterIP   10.98.194.166    <none>        80/TCP      21m
longhorn-recovery-backend     ClusterIP   10.108.227.194   <none>        9600/TCP    21m
longhorn-replica-manager      ClusterIP   None             <none>        <none>      21m
```

Use kubectl edit to edit the longhorn-frontend from ClusterIP to Nodeport service. Modify the **longhorn-frontend** based on the red line content bellow

root@master1:~# kubectl edit service -n longhorn-system longhorn-frontend

spec:

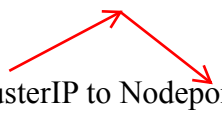
```
  clusterIP: 10.98.194.166
  clusterIPs:
  - 10.98.194.166
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    nodePort: 31724
    port: 80
    protocol: TCP
    targetPort: http
  selector:
    app: longhorn-ui
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

ClusterIP service

```
spec:
  clusterIP: 10.98.194.166
  clusterIPs:
  - 10.98.194.166
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: http
  selector:
    app: longhorn-ui
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

Nodeport service

```
spec:
  clusterIP: 10.98.194.166
  clusterIPs:
  - 10.98.194.166
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    nodePort: 31724
    port: 80
    protocol: TCP
    targetPort: http
  selector:
    app: longhorn-ui
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```



The longhorn-frontend service changed from ClusterIP to Nodeport as bellow figture. K8S cluster will expose UI service via port 31724.

```
root@master1:~# kubectl get service -n longhorn-system
NAME                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
csi-attacher                  ClusterIP   10.110.28.82     <none>        12345/TCP        103m
csi-provisioner               ClusterIP   10.97.195.20     <none>        12345/TCP        103m
csi-resizer                   ClusterIP   10.103.192.138   <none>        12345/TCP        103m
csi-snapshotter               ClusterIP   10.104.82.196    <none>        12345/TCP        103m
longhorn-admission-webhook    ClusterIP   10.100.20.160    <none>        9443/TCP         104m
longhorn-backend              ClusterIP   10.99.196.56     <none>        9500/TCP         104m
longhorn-conversion-webhook   ClusterIP   10.97.241.102    <none>        9443/TCP         104m
longhorn-engine-manager       ClusterIP   None             <none>        <none>           104m
longhorn-frontend             NodePort    10.98.194.166    <none>        80:31724/TCP     104m
longhorn-recovery-backend     ClusterIP   10.108.227.194   <none>        9600/TCP         104m
longhorn-replica-manager      ClusterIP   None             <none>        <none>           104m
```

Verify dashboard install. Open browser, and enter URL: http//:192.168.122.200:31724/#/dashboard . 192.168.122.200 is the virtual IP, it should be 192.168.122.250 if you follow the topology in page 1. I have some change because I install it on another cluster.
31724 is the Nodeport service.

Longhorn UI dashboard should be like bellow.