[토막] Network Namespace 간단한 구성

기준 : CentOS 7.6.1810
유저 : root 사용자

Network Namespace

가상화 또는 클러스터 시스템에서 자주 보이는 용어입니다.
앞서서 컨테이너를 사용하기 위한 격리 조건 중 하나로 언급이 되었었는데요.
논리적인 Network Space(공간)을 생성 후 네트워크 인터페이스, 라우팅, IP 라는 네트워크 요소들을 넣어
Host의 네트워크와 격리되는 공간을 말합니다. 다양한 환경의 네트워크들을 구분하고 통신하기 위해 활용됩니다.

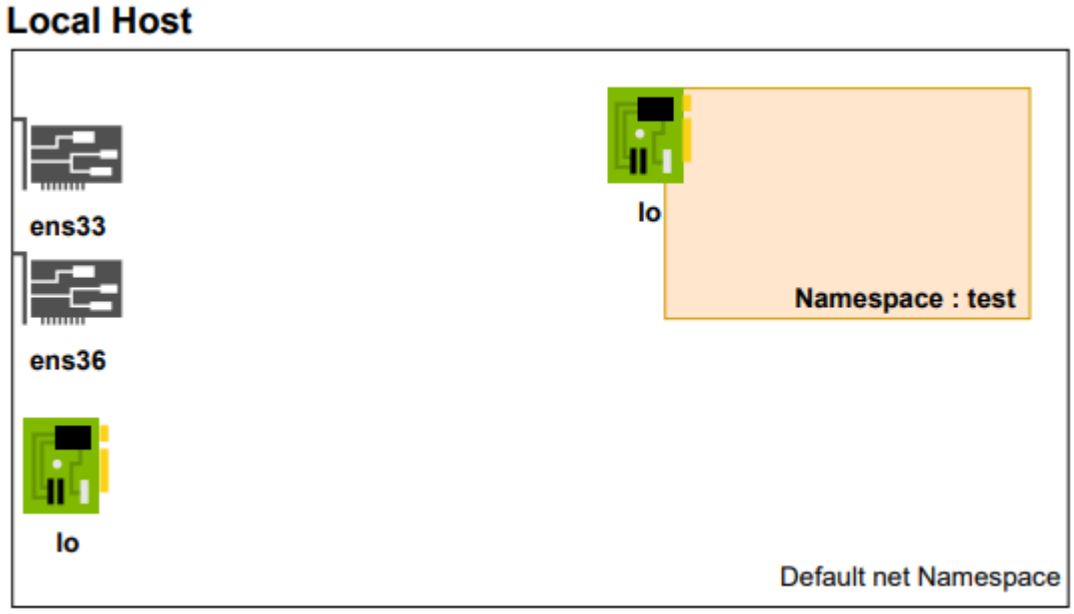
Default Network Namespace Check


```
# Host의 디폴트 Network Namespace 와 소유자를 확인
$ lsns -t net -o pid,uid,user,command
  PID   UID USER  COMMAND
    1     0 root  /sbin/init maybe-ubiquity
```

로컬 Host 의 디폴트 네트워크는 PID 1 (Init) 에서 실행되는 것을 확인할 수 있습니다.

일반적으로 우리가 보는 nic(예: eth0) 과 lo 인터페이스가 속한 기본 네임스페이스입니다.

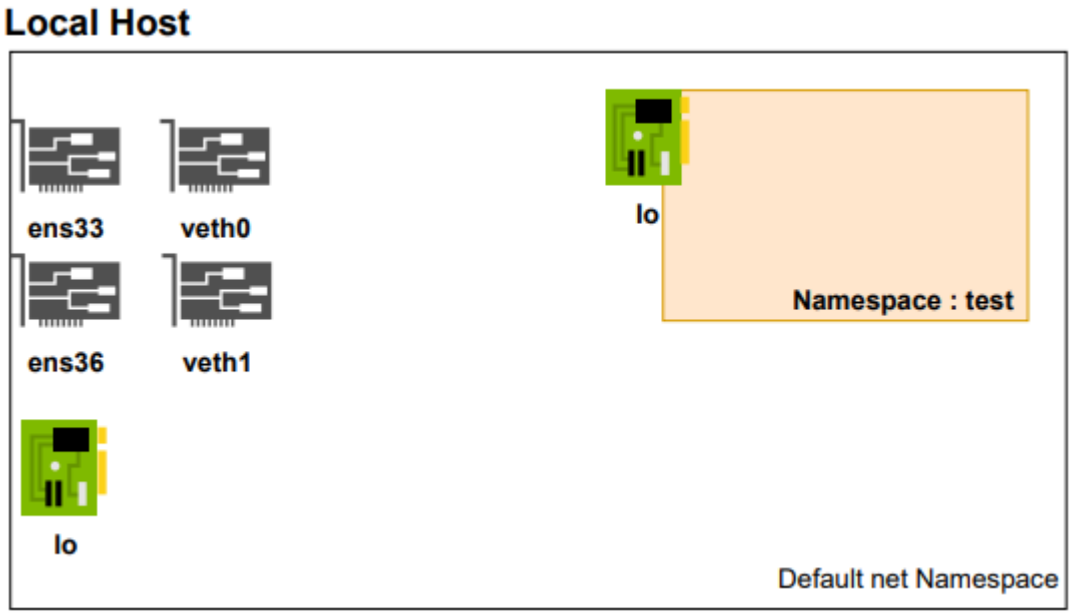
Create Network Namespace



새 네임스페이스를 생성하여, 별도의 격리된 네트워크 영역을 만들 수 있습니다.
새 네임스페이스는 lo 인터페이스만 가지고 있으며 다른 네임스페이스와 통신할 수 없는 영역입니다.

```
# test 라는 이름을 가진 Namespace 를 생성 후 리스트 확인
$ ip netns add test

$ ip netns
test

# Check
만들어진 네임스페이스를 사용하는 PID 가 없기 때문에 lsns 목록에서 보이지 않습니다
$ lsns -t net
PID USER TYPE COMMAND
   1 root net   /usr/lib/systemd/systemd --switched-root --system --deserialize 22
```

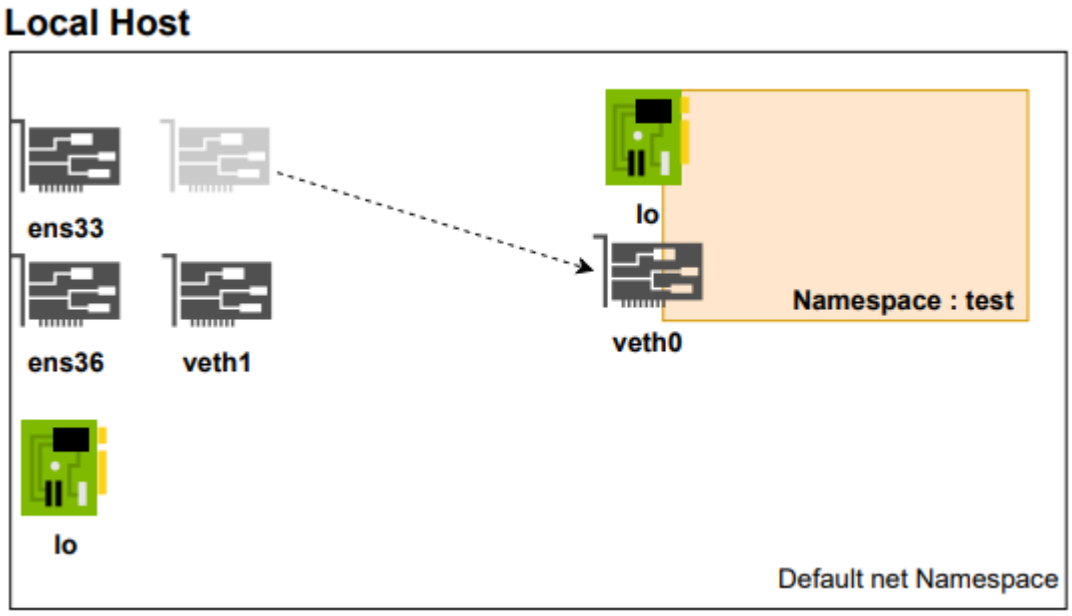
Namespace의 Network 연결 1 – 가상 인터페이스 생성



가상의 Network Namespace 간의 통신은 리눅스 커널에 존재하는 가상 인터페이스 기능인 veth 를 사용합니다.
veth 는 기본적으로 HOST <---> 다른 네임스페이스와의 연결 성립을 위한 한 쌍 단위로 구성됩니다.

```
# HOST에 가상 인터페이스를 추가 생성합니다. veth type 에 peer 로 pair 구성입니다.
$ ip link add veth0 type veth peer name veth1

# HOST에 veth0/veth1 이라는 2개의 가상 인터페이스가 생성되었습니다.
$ ip -br -c addr
lo               UNKNOWN        127.0.0.1/8
ens33            UP             211.239.150.48/23
ens36            UP             192.168.0.2/24
veth1@veth0      DOWN
veth0@veth1      DOWN
```

Namespace의 Network 연결 2 – 새 네임스페이스에 가상 인터페이스 할당



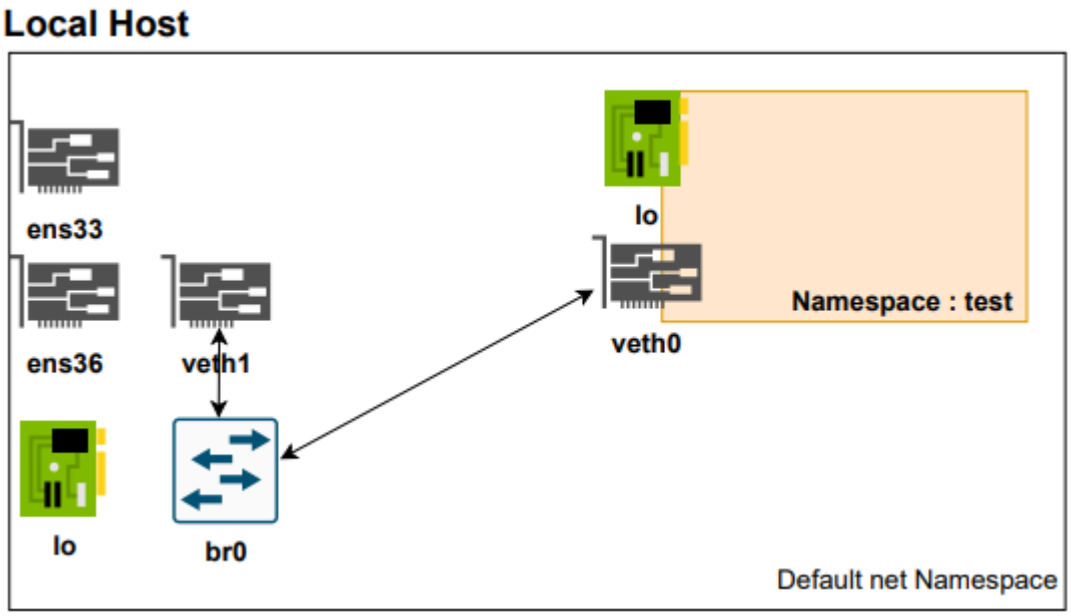
디폴트 영역에 통신을 위한 가상 인터페이스를 생성 후, 이제 새로 만든 test 네임스페이스에 달아 줄 수 있습니다.

```
# veth0 인터페이스를 test Namespace 에 Set
$ ip link set veth0 netns test

# 생성 후 HOST 에서 veth0 은 test namespace 에 장착했기 때문에 사라졌습니다.
$ ip -br -c addr
lo               UNKNOWN        127.0.0.1/8
ens33            UP             211.239.150.48/23
ens36            UP             192.168.0.2/24
veth1@if5        DOWN
```

test namespace 의 가상 인터페이스를 확인해 보면 veth0 이 장착된 것을 확인할 수 있습니다.

```
# test namespace 에 netns exec 옵션을 통한 인터페이스 조회 커맨드 실행
$ ip netns exec test ip -br addr
lo               DOWN
veth0@if4        DOWN
```

Namespace의 Network 연결 3 - bridge 연결



HOST 와 test namespace 간에 veth0과 veth1 인터페이스가 각각 세팅되었지만 DOWN 상태로 아직 통신을 할 수는 없습니다.
논리적인 가상 인터페이스 간 통신을 위해서는 IP 할당이 필요한데
이번 글에서는 HOST 영역에 브릿지 인터페이스(bridge) 를 사용해 연결해 보겠습니다.
호스트의 네트워크를 복잡하게 만들지 않으려면 브릿지라는 가상 스위치에 네임스페이스를 연결하는 것이 좋습니다.

```
# Check
$ (명령어가 없다면) yum install -y bridge-utils-1.5-9.el7.x86_64
$ btctl show
bridge name      bridge id                STP enabled
interfaces
```

현재 브릿지가 없는 것을 확인했습니다. br0 이라는 이름으로 로컬 HOST에 생성을 합니다.

```
# Bridge Create && Check
$ ip link add br0 type bridge

$ brctl show
bridge name      bridge id                STP enabled
interfaces
br0              8000.000000000000       no

# 브릿지 생성 추가 확인
$ ip -br -c addr
lo               UNKNOWN        127.0.0.1/8
```

```
ens33            UP             211.239.150.48/23
ens36            UP             192.168.0.2/24
veth1@if5        DOWN
br0              DOWN
```

하지만 여전히 br0 과 vethx 간에 어떤 연결을 설정하지 않았기 때문에 통신할 수는 없습니다.
첫 단계로 신규 네임스페이스에 설정한 인터페이스와 호스트의 브릿지를 연결해 줍니다.

```
# HOST 에 존재하는 veth1 과 Host 의 br0 브릿지를 연결합니다.
$ ip link set veth1 master br0

# check 시 bridge 에 veth1 이 추가된 것을 확인할 수 있습니다.
$ brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.46df623e69e4       no              veth1
```

가장 서두에도 말했지만, 네트워크의 통신 요소는 인터페이스, 라우팅, IP 입니다.
통신을 위해 두 번째 단계로 이제 IP 를 설정해야 합니다.
ifconfig 같이 별도 net-util 을 설정하는 것보다 ip 명령어를 사용하는 것이 수월합니다.

```
# netns exec 명령을 사용하여 test Namespace 의 veth0 인터페이스에 IP 를 할당하고 UP 합니다.
$ ip netns exec test ip addr add 10.10.10.2/24 dev veth0
$ ip netns exec test ip link set veth0 up

# 이제 host 의 veth1 인터페이스와 bridge 인터페이스도 up 해 줍니다.
$ ip link set br0 up
$ ip link set veth1 up

# UP check 시 이제 브릿지와 인터페이스가 모두 UP 된 것을 확인할 수 있습니다.
$ ip -br -c addr
lo               UNKNOWN        127.0.0.1/8
ens33            UP             211.239.150.48/23
ens36            UP             192.168.0.2/24
veth1@if5        UP
br0              UP

# test namespace 의 인터페이스 UP 도 확인됩니다.
```

```
$ ip netns exec test ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
5: veth0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default qlen 1000
     link/ether f2:1c:09:d4:47:fc brd ff:ff:ff:ff:ff:ff link-
netnsid 0
```

lo 인터페이스가 DOWN 이면 해당 네임스페이스의 내부 통신을 할 수가 없습니다. UP 이후 UNKNOWN 으로 바뀌면 됩니다.

```
$ ip netns exec test ip link set dev lo up
$ ip netns exec test ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1000
```

Check

```
$ ip netns exec test ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.058 ms
```

Check 2

Host 의 브릿지와 신규 네임스페이스 간에 통신되는지 확인하려면 브릿지에 Gateway 와 같은 동일 대역 IP할당을 합니다
브릿지에 Routing 정보가 없으면, 다른 네임스페이스의 사설 인터페이스 ip를 찾아갈 수 없습니다.

브릿지에 통신할 네임스페이스와 같은 대역의 IP 를 할당하여 라우팅 정보를 넣습니다.

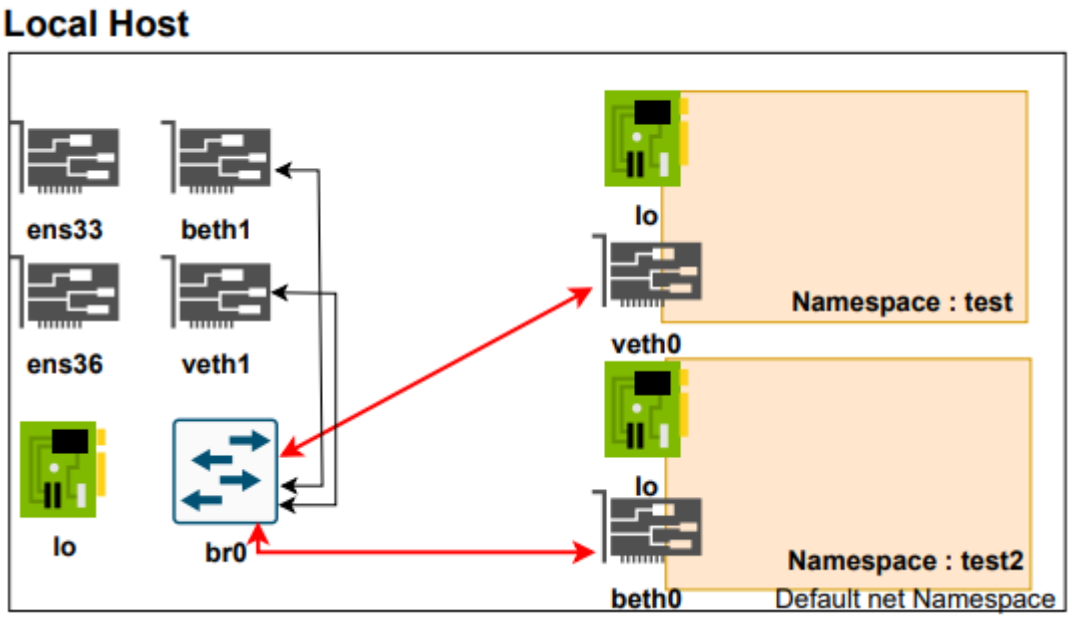
```
$ ip addr add 10.10.10.200/24 dev br0
```

test 네임스페이스의 veth0 인터페이스로 Ping 을 날려봅니다.

```
$ ping 10.10.10.2
 ping 10.10.10.2 -c 2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.073 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.071 ms

--- 10.10.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
```

```
rtt min/avg/max/mdev = 0.071/0.072/0.073/0.001 ms
```

Namespace의 Network 연결 4 – 다른 새 네임스페이스 간의 통신 연결

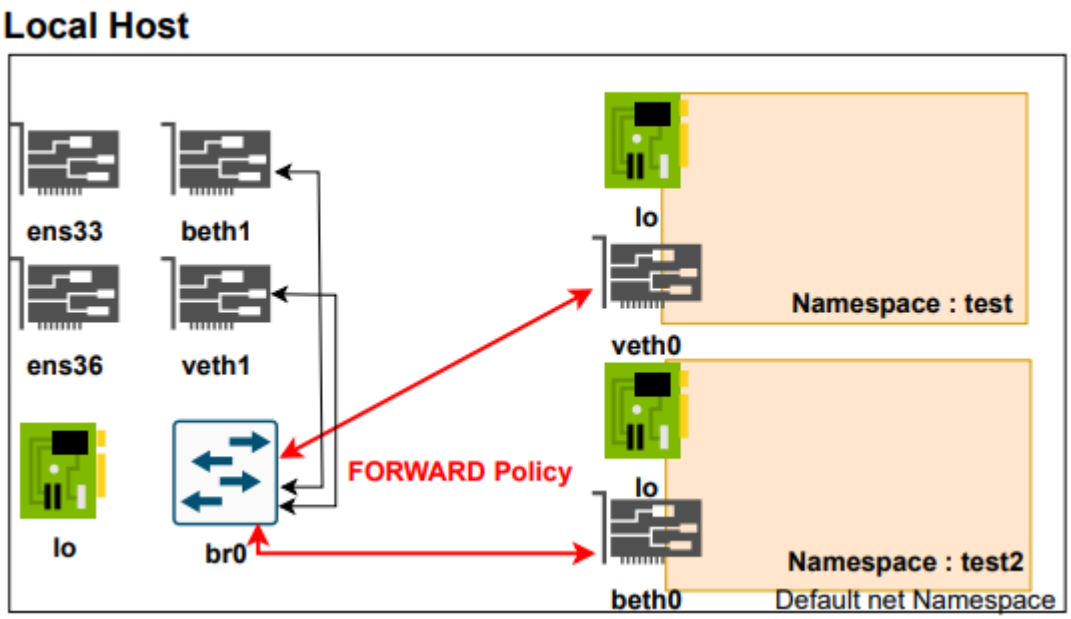


test namespace 브릿지에 추가되는 다른 네임스페이스와는 통신이 잘 되어야 합니다.
네트워크가 같다면 바로 되고, 다르다면 라우팅 설정을 추가로 해 주어야겠지요.

```
# test2 namespace 를 추가로 만들고 beth0/beth1 인터페이스를 생성 및 할당하고 브릿지에 연결합니다
# IP 세팅하고 test <---> test2 네임스페이스 간 Ping 시도시 잘 통신 됩니다.
ip netns add test2
ip link add beth0 type veth peer name beth1
ip link set beth0 netns test2
ip link set beth1 master br0
ip netns exec test2 ip addr add 10.10.10.3/24 dev beth0
ip netns exec test2 ip link set beth0 up
ip netns exec test2 ip link set dev lo up
ip link set beth1 up

# test2 namespace 확인
$ ip netns
test2 (id: 1)
test (id: 0)
```

```
$ ip -br -c addr
lo               UNKNOWN        127.0.0.1/8
ens33            UP             211.239.150.48/23
ens36            UP             192.168.0.2/24
veth1@if5        UP
br0              UP
beth1@if8        UP

$ brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.2e0e64ccb0e5       no              beth1
                                                        veth1
# test namespace 의 veth0(10.10.10.2) 에 Ping 날려보기
ip netns exec test2 ping 10.10.10.2 -c 2
PING 10.10.10.2 (10.10.10.2) 56(84) bytes of data.
64 bytes from 10.10.10.2: icmp_seq=1 ttl=64 time=0.112 ms
64 bytes from 10.10.10.2: icmp_seq=2 ttl=64 time=0.076 ms

--- 10.10.10.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.076/0.094/0.112/0.018 ms
```

트러블슈팅



다른 네임스페이스와 통신이 안 된다면?
HOST 에 생성된 브릿지 인터페이스는 lo (내부통신) 이 아니기 때문에, Host 를

거쳐
다른 네트워크에 중계해 주려면 커널의 외부 포워딩 규칙을 따르기 때문입니다.
즉 NAT 구성 때와 비슷합니다. 커널 옵션에서 정의해 주어야 합니다.
ip4 트래픽에 대해 FORWARD 설정을 HOST 의 커널 옵션에 지정하여야 합니다.

```
# 먼저 HOST iptables FORWARD 정책을 확인하고 ACCEPT 로 변경합니다.
$ iptables -nL | grep -i forward
Chain FORWARD (policy DROP)

# 변경 후 체크
$ iptables --policy FORWARD ACCEPT
$ iptables -nL | grep -i forward
Chain FORWARD (policy ACCEPT)

$ service iptables save ( 또는 저장할 수 있는 OS별 옵션을 기입합니다 )

# 커널 옵션에서 ip4v.forward 활성화 합니다.
echo 1 > /proc/sys/net/ipv4/ip_forward
sysctl --system

# check
# 다시 통신 테스트를 해 봅니다.
```

XenServer에 NVIDIA vGPU 추가하기

XenServer에 NVIDIA vGPU 추가

- 사전 구성 환경
- Xenserver 8.2 설치
- NVIDIA Tesla M60 GPU 장착
- Xenserver Update
- Grid vGPU license server(Ubuntu18.04) 생성

NVIDIA vGPU software license 발급

라이선스 발급을 위한 NVIDIA 가입

[NVIDIA 가입 주소]
https://enterpriseproductregistration.nvidia.com/?LicType=EVAL&ProductFamily=vGPU&ncid=em-news-525732

1. 가입 후 가입한 메일로 license url 접속 메일을 받음 (https://nvid.nvidia.com/dashboard/#/dashboard)
2. 위 주소로 가입한 메일로 접속

라이선스 서버 생성 및 소프트웨어, 라이선스(.bin) 파일 다운로드

NVIDIA Grid Tesla 칩셋 버전부터는 별도 라이선스 서버 구성이 필요함

1. Create License Server License Servers -> Create Server에서 Create legacy server 체크 활성화 후 Name, Description, 라이선스 서버의 사용할 MAC 주소 작성후 Next:Select features 클릭
2. 서버에 할당할 라이선스 종류 및 사용 수량 선택
3. 서버 생성
4. License Servers -> List Servers에서 생성된 서버 확인
5. 서버 Action 메뉴에서 Download로 라이선스 파일 다운로드
6. Software Downloads 에서 필수 프로그램 다운로드
    - *Citrix Xenserver용 – NVIDIA-vgpu*
    - *Grid vGPU license server용 – linux License Manager*

Xenserver에 NVIDIA-vGPU 파일 설치

```
sudo unzip NVIDIA-GRID-CitrixHypervisor-*
sudo rpm -ivh NVIDIA-vGPU-CitrixHypervisor-*
reboot
```

Grid license server(Ubuntu)에 소프트웨어 라이선스 서버 설치

1. Java, tomcat 설치 “` # java 설치 sudo apt-get install -y default-jdk sudo java -version // 버전 확인 시 OpenJDK 64-Bit 확인

tomcat 설치

sudo apt install -y tomcat8 sudo systemctl enable tomcat8.service && systemctl start tomcat8.service sudo curl http:// :8080 // 톰캣 확인

2. Linux용 License Manager 설치

sudo unzip NVIDIA-ls-linux* sudo cd NVIDIA-ls-linux˷ sudo chmod +x setup.bin sudo ./setup.bin

설치 진행

1. 설치 동의 – Enter
2. 로컬 tomcat 서버 경로 설정 – /var/lib/tomcat8 // 다른경로로 선택할 시 License Server Management interface 접속시 404 에러 발생 **중간에 서버 경로 설정을 잘못 설정하여 404 에러가 났을 시 설정 명령어
    - cp /opt/flexnetls/nvidia/ui/*.war /var/lib/tomcat8
    - jar xvf *.war
3. 방화벽 – 7070, 8080 허용 // 라이선스 관리 인터페이스에 접속할 포트 설정 (기본 설정은 7070만 허용이지만 접속이 안되는 문제가 있어 8080으로 외부 접속 가능하도록 설정)
4. License Management에 접속 http://:8080/licserver
5. License Management 에서 upload license file(.bin 파일) 업로드 **등록시에 에러발생시 라이선스 서버 구성시에 MAC 주소가 라이선스 파일과 매칭이 되지 않아서 발생함
    - Configuration에서 server host ID의 value 값을 해당 MAC주소로 변경 후 다시 license file 등록
6. Xenserver에서 GPU 확인

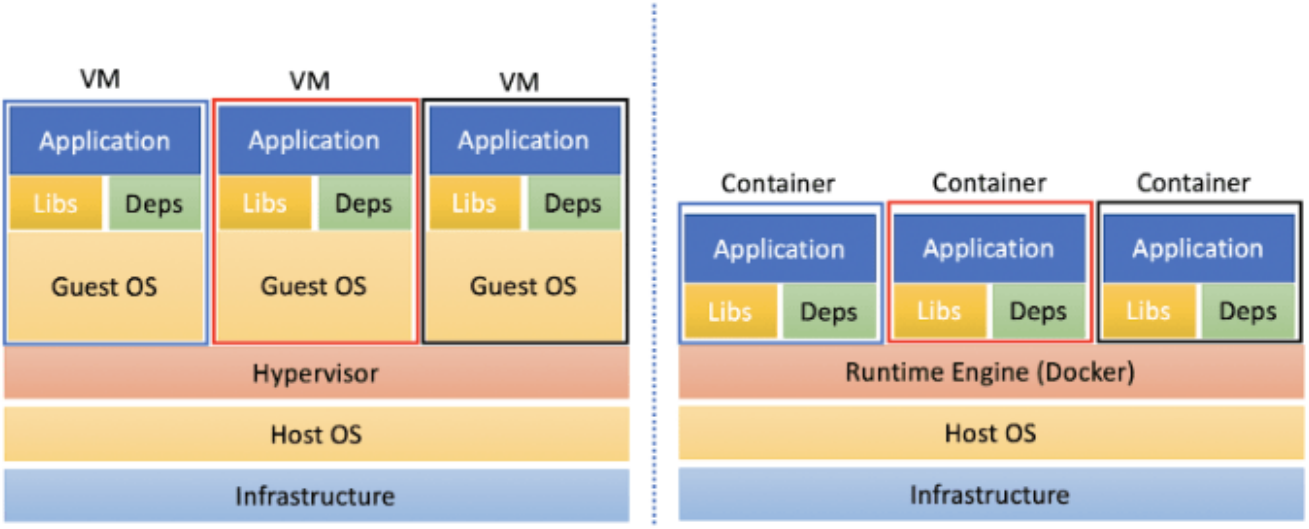
1. cli 확인 # nvidia-smi
2. GUI 확인

- Xenserver host – GPU 탭 확인

User VM에 gpu 추가

1. windows 10 VM 생성 시 NVIDIA Tesla M60 GPU 선택
2. windows 10 기본 설치 진행
3. Citrix 공식 홈페이지의 download 탭에서 XenServer 버전에 맞는 windows-Xentools 다운로드 후 설치
4. 재부팅
5. 장치 관리자에서 드라이버 설치가 필요한 추가 장치가 존재하는지 확인
6. NVIDIA Tesla M60 GPU 드라이버 다운로드 후 설치(https://www.nvidia.co.kr/Download/index.aspx?lang=kr)
7. 재부팅
8. 장치 관리자 NVIDIA gpu 장치 확인

[토막] VM과 Container 의 구분



Container

- 컨테이너는 리눅스 자체의 기술(LXC)이나 LXC를 활용하는 런타임 (예 : Docker) 등으로 HOST의 커널을 공유하여 격리 공간을 생성하는 하나의 프로세스입니다.
- 컨테이너에서 실행되는 하위 프로세스의 구분를 위해 Namespace 와 리소스 제한을 위한 cgroup 이라는 리눅스 커널 기능을 사용합니다.

#아래는 namespace 로 Host 에서 별도로 분리 가능한 자원들입니다.
HOST의 Linux 내 자원을 가상화합니다. VM 이 하드웨어 디바이스를 가상화하는 것과 구분되는 부분입니다.

```
pid
user
uts
ipc
mnt
net
```

#cgroup 으로 격리된 프로세스에 대한 Host의 컴퓨팅 리소스를 제한할 수 있습니다.

```
Memory
CPU
Network
Device
I/O
```

- 컨테이너는 해당 Host의 프로세스이기 때문에, 리눅스 커널의 컨테이너는

Windows OS 컨테이너를 만들기 어렵습니다.
- Container 는 해당 Host의 커널에서 컨트롤이 가능하기 때문에 보안에 많이 신경써야 합니다.

VM

- VM 은 Host에 올린 Hypervisor 로 가상화 된 별개의 하드웨어를 만든 후
그 위에 OS 를 올림으로써 컨테이너보다 더 높은 격리 환경이라고 볼 수 있습니다.
- Host의 커널이 아닌, 가상화 된 하드웨어 구성 위에 여러 Linux/Windows/Other Guest OS 를 설치하고
해당 OS의 커널에서 동작합니다.

[CKA] #1. 쿠버네티스 설치하기

제목 : [CKA] #1. 쿠버네티스 설치하기

Kubenertes 설치 방식

쿠버네티스는 여러가지 방식으로 설치할 수 있으나 CKA 과정을 고려한하여
공식 오픈문서의 kubeadm 방식으로 설치를 진행하였다.

구성 서버 (VM)
Controller Server : 1EA
Worker Server : 1EA

OS
Ubuntu 20.04 Server Minimal

```
# SWAP 은 기본적으로 해제하는 것을 권장하고 있다.
sudo swapoff /swap.img
sudo sed -i -e '/swap.img/d' /etc/fstab
```

호스트네임을 설정한다. 모든 작업은 일반 계정(regular user)에서 sudo 를 사용하여 권한을 빌려 실행하는 것을 권장한다.

```
sudo hostnamectl set-hostname controller
sudo hostnamectl set-hostname worker
```

Traffic Setup

컨테이너 런타임(예: Docker), kube-proxy에서 사용하는 브릿지 네트워크의 트래픽을 확인할 수 있도록 iptables 커널 옵션을 지정한다.

```
## Container / Worker
```
이 모듈을 로드하면, 브릿지를 통과하는 트래픽이 netfilter(iptables) 를 거치도록 활성화된다.
```
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF

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

Container Runtime 설치

쿠버네티스의 기본 단위인 POD 에서 컨테이너를 구동하는 런타임은 여러 종류가 존재하지만 CKA 과정 상 Docker 를 설치하였다.
쿠버네티스는 외부 추가 서비스/모듈의 일관성을 보장하지는 않는다.

```
## Controller / Worker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

## Check
sudo docker -v
sudo docker ps -a
```

cgroup 드라이버 설정

cgroup 은 런타임 프로세스의 리소스를 제약할 수 있는 관리자이다.
설치한 OS의 cgroup 기본 관리자인 systemd 와, docker, kubelet 의

cgroupfs 관리가 별개로 작동하는 혼란을 방지하기 위해 systemd 로 통합해 주는 과정이다.

```
## Controller / Worker
sudo mkdir /etc/docker
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF

## Docker enable && restart
sudo systemctl enable docker
sudo systemctl daemon-reload
sudo systemctl restart docker
```

Docker 의 cgroup driver 확인 시, 기본 cgroupfs 에서 systemd 로 변경된 것을 확인할 수 있다.

```
sudo docker info | grep -i cgroup

 Cgroup Driver: systemd
 Cgroup Version: 1
```

쿠버네티스 패키지 설치

레포지토리 구성 후 kebe 관련 관리/명령줄 패키지를 설치한다. 일반적으로 모든 서버에 설치한다.
에러 발생시 한줄씩 진행한다.

```
## Controller / Worker
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg
https://packages.cloud.google.com/apt/doc/apt-key.gpg
```

```
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
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

Kube 컨트롤 노드의 InitIalize.

Controller Node 의 시작을 위해 init 하는 과정이다. 옵션을 통해 구성을 지정할 수 있다.

--cri-socket : 지정하지 않을 경우 kubeadm이 socket 으로 내부에 설치된 컨테이너 런타임을 자동 감지하여 구성한다. 다중으로 설치되어 있으면 에러가 발생한다.
--pod-network-cidr : pod 간 통신할 수 있는 network 를 지정한다. 내부 CoreDNS Service 의 조건이다.
--apiserver-advertise-address=<ip-address> : 지정하지 않으면 Controller 의 기본 네트워크 인터페이스를 사용하여 API서버를 선언한다.

Controller. 일반적으로 컨트롤러 자신의 공인IP 를 통해 API 서버임을 알린다 (Advertise)

```
sudo kubeadm init --ignore-preflight-errors=all --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address=203.248.23.192
```

정상적으로 init 되면 하단에 아래의 메세지가 나온다.

1) 설명과 같이, 일반 유저(regular user) + sudo 권한으로 cluster를 사용하고 있다면, 아래의 환경 변수를 실행한다.

```
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
```

```
## Check
kubectl get nodes
NAME              STATUS     ROLES                  AGE
VERSION
```

```
user1-controller   NotReady   control-plane,master   6m28s
v1.23.5
```

2) pod network 를 사용하기 위해 Network Plugin 을 설치할 수 있다.

```
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/
```

Pod Network 통신이 구성되지 않으면 CoreDNS 가 작동하지 않는다(Pending)

```
kubectl get pods --all-namespaces
NAMESPACE     NAME                                          READY
STATUS    RESTARTS   AGE
kube-system   coredns-64897985d-9sj9j                         0/1
Pending   0          12m
kube-system   coredns-64897985d-zfl8q                         0/1
Pending   0          12m
kube-system   etcd-user1-controller                           1/1
Running   0          12m
kube-system   kube-apiserver-user1-controller                 1/1
Running   0          12m
kube-system   kube-controller-manager-user1-controller        1/1
Running   0          12m
kube-system   kube-proxy-g5xdv                                1/1
Running   0          12m
kube-system   kube-scheduler-user1-controller                 1/1
Running   0          12m
```

Pod Network Plugin Install

다양한 네트워크 플러그인이 있으며, 해당 CKA 과정 글에서는 Callico 의 Plugin 을 사용한다.

```
curl
https://projectcalico.docs.tigera.io/manifests/calico.yaml -O
kubectl apply -f calico.yaml
kubectl get nodes
```

Check

플러그인 설치 후, coredns의 status 가 Running 되는지 체크한다.

```
kubectl get pods --all-namespaces
NAMESPACE     NAME                                       READY
STATUS     RESTARTS   AGE
kube-system   calico-kube-controllers-56fcbf9d6b-bnxz5   0/1
Pending    0          20s
kube-system   calico-node-khp2h                          0/1
Init:2/3   0          20s
kube-system   coredns-64897985d-9sj9j                    0/1
Pending    0          22m
kube-system   coredns-64897985d-zfl8q                    0/1
Pending    0          22m
kube-system   etcd-user1-controller                      1/1
Running    0          22m
```

Multi NIC 를 가지는 환경에서 INTERNAL-IP 설정

다수의 네트워크가 존재하는 서버의 경우 K8S 에서 첫 번째 NIC 의 IP 가 INTERNAL-IP 로 자동 설정된다.
해당 INTERNAL-IP 를 Init 시 설정한 kubeadm --apiserver-advertise-address 와 동일한 IP 대역을 사용하도록 수동 설정한다.

```
# INTERNAL-IP 가 10.0.2.15 ( Calico Network Default ) 로 되어 있다
$ kubectl get nodes -o wide
NAME              STATUS   ROLES                  AGE
VERSION     INTERNAL-IP     EXTERNAL-IP     OS-IMAGE
KERNEL-VERSION      CONTAINER-RUNTIME
user-controller   Ready    control-plane,master   44h
v1.23.5   10.0.2.15     <none>        Ubuntu 20.04.1 LTS
5.4.0-64-generic   docker://20.10.14
user-worker       Ready    <none>                 44h
v1.23.5   10.0.2.15     <none>        Ubuntu 20.04.1 LTS
5.4.0-64-generic   docker://20.10.14

# Controller.  상황에 따라 수동 설정한다
cat << EOF | sudo tee /etc/default/kubelet
KUBELET_EXTRA_ARGS='--node-ip $(hostname -I | cut -d ' ' -f2)'
EOF
sudo systemctl daemon-reload
sudo systemctl restart kubelet
kubectl cluster-info
```

```
# Worker.  상황에 따라 수동 설정한다
cat << EOF | sudo tee /etc/default/kubelet
KUBELET_EXTRA_ARGS='--node-ip $(hostname -I | cut -d ' ' -f2)'
EOF
sudo systemctl daemon-reload
sudo systemctl restart kubelet

# Check 시 Internal-IP 가 advertise 인터페이스로 변경되었다.
$ kubectl get nodes -o wide
NAME               STATUS   ROLES                  AGE
VERSION    INTERNAL-IP        EXTERNAL-IP    OS-IMAGE
KERNEL-VERSION     CONTAINER-RUNTIME
user-controller    Ready     control-plane,master    45h
v1.23.5   203.248.23.214   <none>        Ubuntu 20.04.1 LTS
5.4.0-64-generic   docker://20.10.14
user-worker        Ready    <none>                  44h
v1.23.5   203.248.23.215   <none>        Ubuntu 20.04.1 LTS
5.4.0-64-generic   docker://20.10.14
```

Worker 에서 Controller 에 Join

Then you can join any number of worker nodes by running the following on each as root:
root 로 실행하라고 안내하고 있으나
일반 유저로 Worker 에서 kebeadm을 실행하고 싶다면 Controller 의 /etc/kebenertes/admin.conf 파일을 Worker 로 복제 후 동일 환경 구성을 해준다.

```
# Controller
sudo          scp          /etc/kubernetes//admin.conf
vagrant@203.248.23.193:/home/vagrant/admin.conf

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

kubeadm     join     203.248.23.192:6443     --token
wy11vq.bk2rze7g9lilg2d9 \
                        --discovery-token-ca-cert-hash
sha256:f7bc17bb974c804821b21427d500cb96615f66c1fd88cb53c023d8b
```

2c598d3f7

오류가 발생할 경우 ignore 옵션을 추가해 본다
```
sudo kubeadm join 203.248.23.192:6443 --token wy11vq.bk2rze7g9lilg2d9 --ignore-preflight-errors=all --discovery-token-ca-cert-hash sha256:f7bc17bb974c804821b21427d500cb96615f66c1fd88cb53c023d8b2c598d3f7

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the control-plane to see this node join the cluster.
```

Check
위의 환경 설정을 했을 경우 Worker 에서도 일반 유저로 pod 확인이 가능하다

```
kubectl get nodes
NAME                STATUS   ROLES                  AGE   VERSION
user1-controller    Ready    control-plane,master   33m   v1.23.5
user1-worker        Ready    <none>                 84s   v1.23.5

kubectl get pods --all-namespaces
NAMESPACE     NAME                                       READY   STATUS    RESTARTS   AGE
kube-system   calico-kube-controllers-56fcbf9d6b-bnxz5   1/1     Running   0          11m
kube-system   calico-node-khp2h                          1/1     Running   0          11m
kube-system   calico-node-skdjl                          1/1     Running   0          2m3s
kube-system   coredns-64897985d-9sj9j                    1/1     Running   0          33m
kube-system   coredns-64897985d-zfl8q                    1/1     Running   0          33m
```

```
kube-system   etcd-user1-controller                          1/1
Running   0          33m
kube-system   kube-apiserver-user1-controller                1/1
Running   0          33m
kube-system   kube-controller-manager-user1-controller       1/1
Running   0          33m
kube-system   kube-proxy-g5xdv                               1/1
Running   0          33m
kube-system   kube-proxy-m6ztf                               1/1
Running   0          2m3s
kube-system   kube-scheduler-user1-controller                1/1
Running   0          33m
```

(Trouble) 설치 중 에러 발생시 클린 삭제 후 재 설치를 진행한다

```
# All Node
sudo systemctl stop kubelet
sudo kubeadm reset -f

sudo rm -rf ~/.kube
sudo rm -rf /root/.kube
sudo rm -rf /var/lib/etcd
```

Network Plugin Status

앞서서 Pod Network 를 관리하는 애드-온 플러그인으로 Calico 를 설치하였는데, 간단한 Status 명령을 확인할 수 있다.
별도 명령어(calicoctl) 설치도 가능하고, Kubectl 의 옵션 플러그인으로 설치할 수도 있는데 전자로 진행했다.

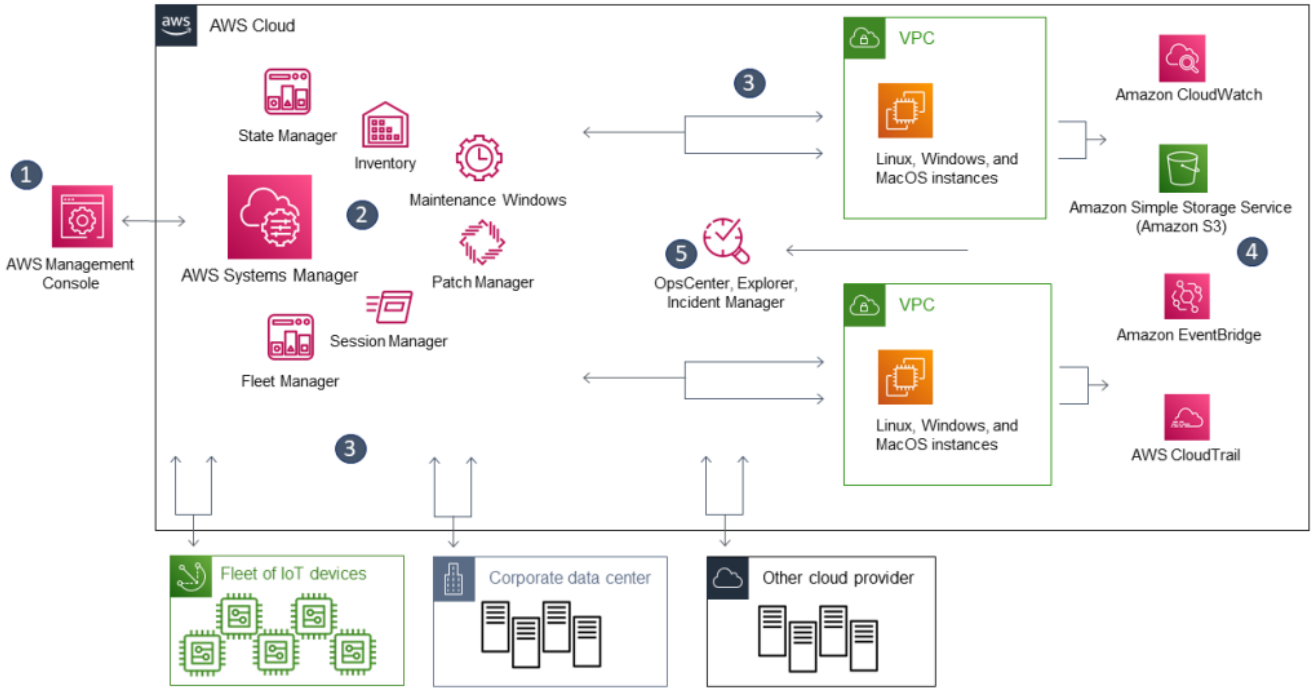
```
# Host 의 명령어 타입으로 설치
$ cd /usr/local/bin
$ sudo curl -L https://github.com/projectcalico/calico/releases/download/v3.22.1/calicoctl-linux-amd64 -o calicoctl
$ sudo chmod +x calicoctl

# Check
$ calicoctl ipam show --show-blocks
+----------+-------------------+-----------+------------+----
```

```
----------+
| GROUPING |        CIDR         | IPS TOTAL | IPS IN USE |
IPS FREE   |
+----------+---------------------+-----------+------------+----
----------+
| IP Pool  | 192.168.0.0/16      |     65536 | 8 (0%)     |
65528 (100%) |
| Block    | 192.168.136.0/26    |        64 | 3 (5%)     | 61
(95%)     |
| Block    | 192.168.153.192/26  |        64 | 5 (8%)     | 59
(92%)     |
+----------+---------------------+-----------+------------+----
----------+
```

Kubernetes Auto Complation

```
# 주요 명령어의 alias 및 Tab 키를 사용한 자동완성을 권장한다
echo '' >>~/.bashrc
echo 'source <(kubectl completion bash)' >>~/.bashrc
echo 'alias k=kubectl' >>~/.bashrc
echo 'complete -F __start_kubectl k' >>~/.bashrc

. ~/.bashrc

# Check
## Tab 자동완성 확인
k get nodes -o wide
kubectl get nodes -o wide
```

AWS SSM 을 통한 VPC Private Network 접근



목적

- VPC 내의 Private Network 에 외부 공인 대역에서 Shell 로 접근 (VPN 역할)

특징

- SSM 을 통해 AWS 연결을 하는 것으로, IGW 나 EIP 등의 VPC내 별도 제약이 필요없다
- EC2 SSH 접근 용도의 Password 나 Key-Pair 가 필요없다.
- Shell 전환으로 SSH 와 거의 동일하게 사용할 수 있다.
- AWS Client VPN 과 비교 시, 비용적으로 유리하다.

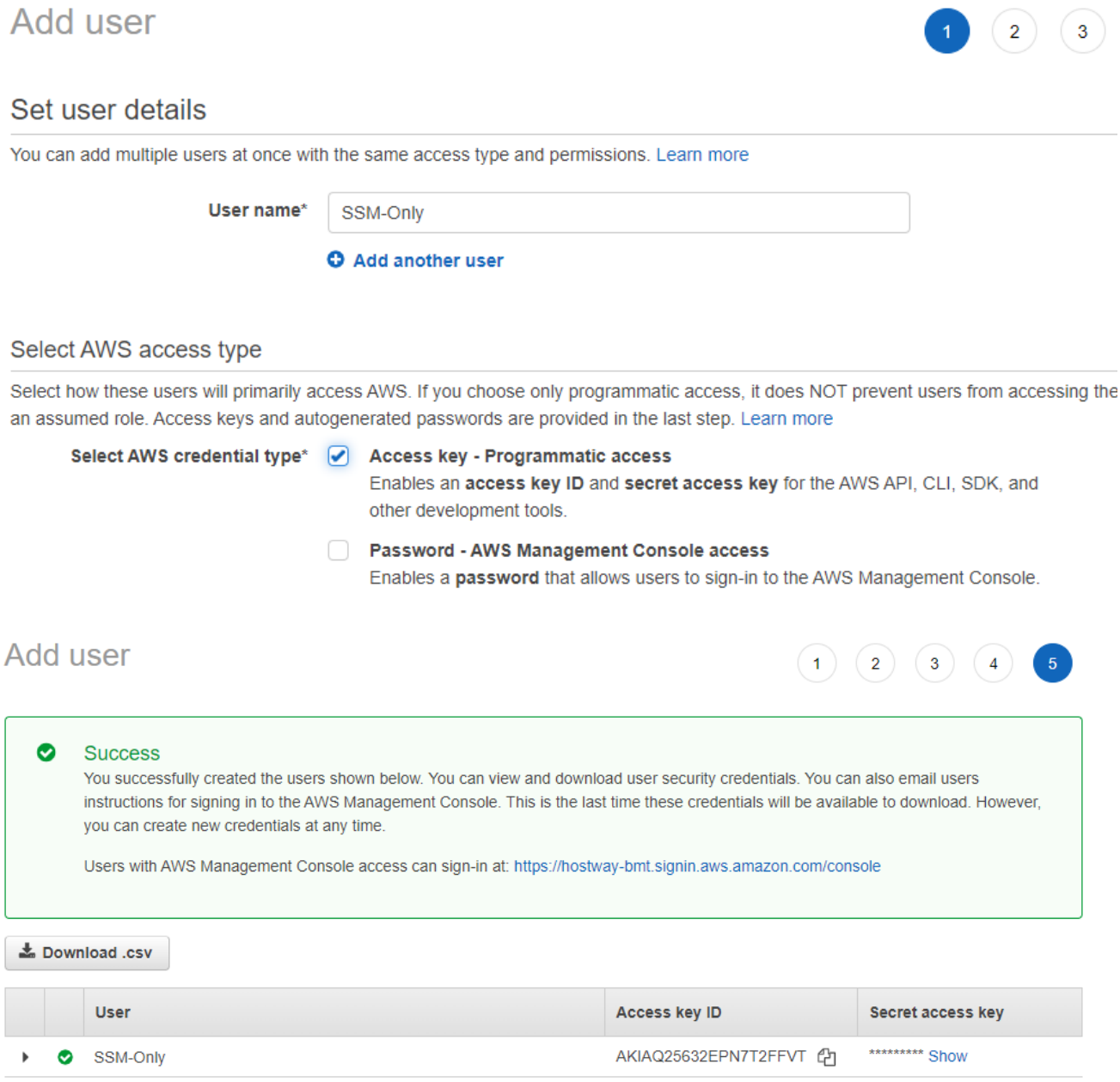
요구사항

- AWS CLi 를 설치 및 AWS 리소스에 접근할 수 있는 공인망 환경 (VM / CT / Server)
  일부 리전의 경우 AWS Console 내에서 Cloudshell 사용 가능

설치 순서

- 1) Private Network 에 접근할 수 있는 IAM 권한 설정
- 2) EC2 에 IAM Role 추가
- 3) EC2 에 SSM Agent 설치
- 4) AWS CLI 를 통한 EC2 연결

1. Key 방식 전용의 IAM 생성 후 권한 설정

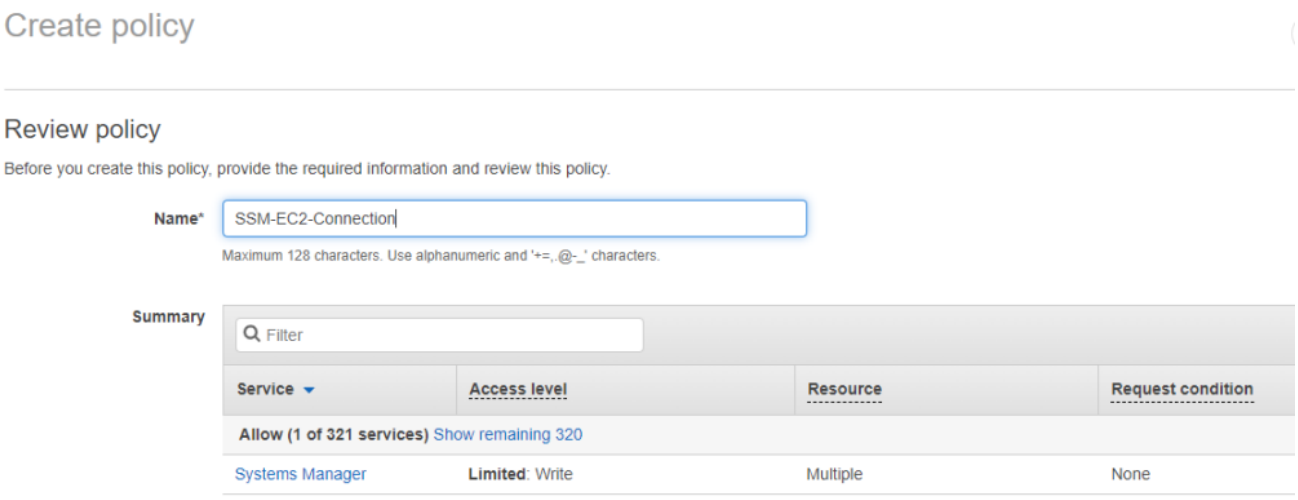


- 관리형 정책을 만들어서IAM 에 할당한다.

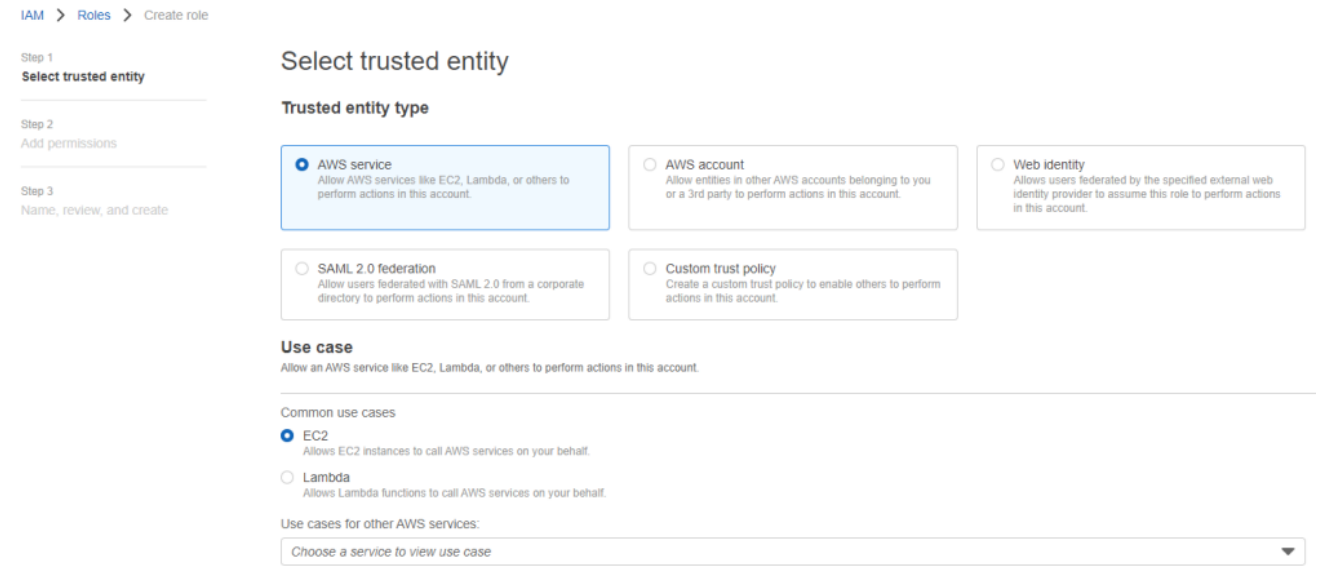
arn 은 리전과 어카운트와 접근할 EC2 ID 를 적는다

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:StartSession"
```

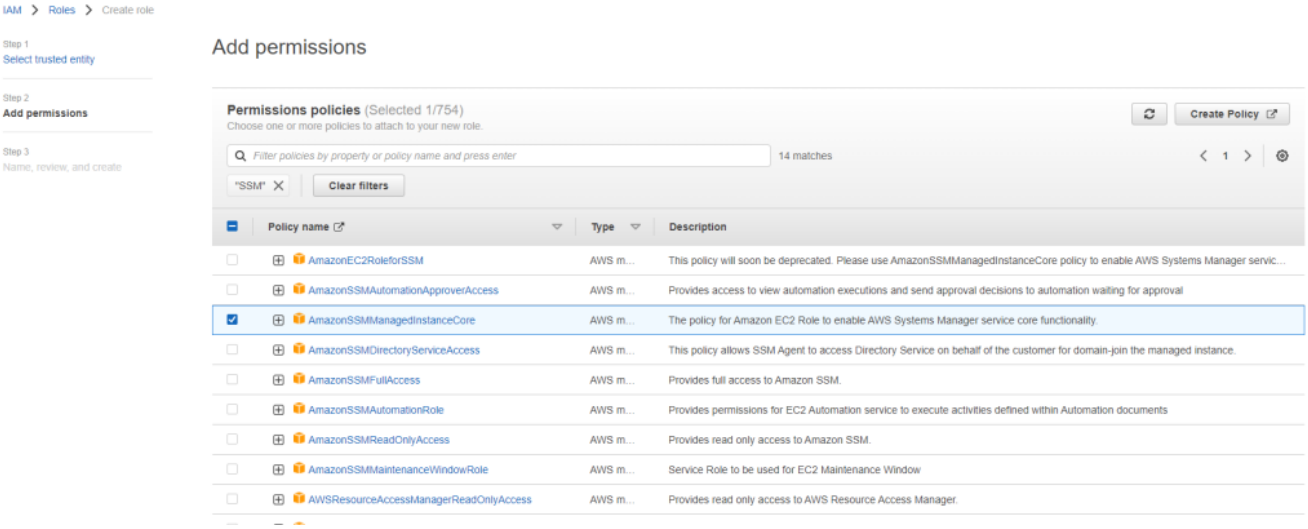
```
            ],
            "Resource": [
                "arn:aws:ec2:us-west-2:1234567890:instance/i-
ahe52134fxed6"

            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:TerminateSession"
            ],
            "Resource": [
                "arn:aws:ssm:*:*:session/${aws:username}-*"
            ]
        }
    ]
}
```



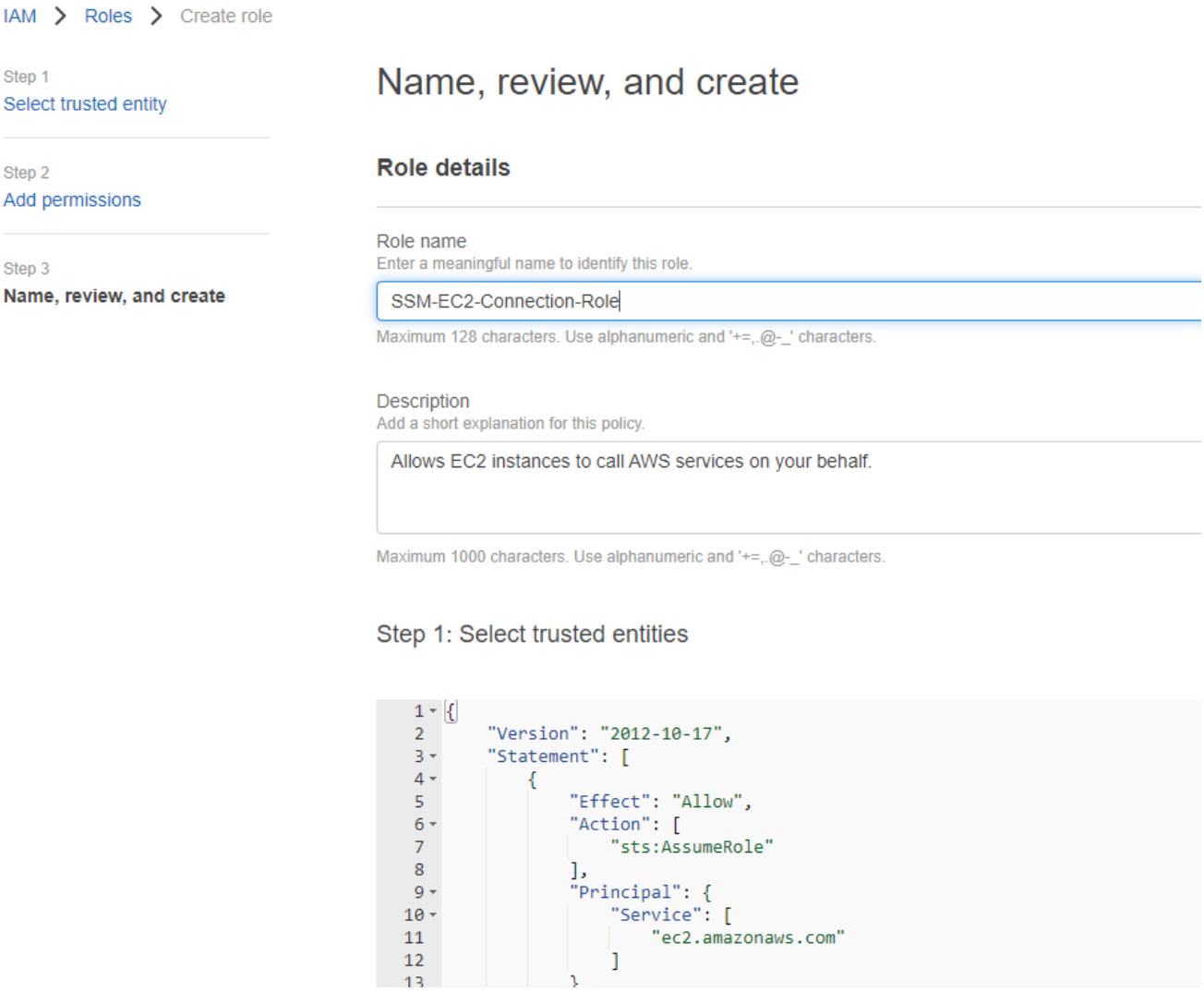
2. IAM Custom Role 을 생성한 후 VPC 내의 EC2 에 설정한다



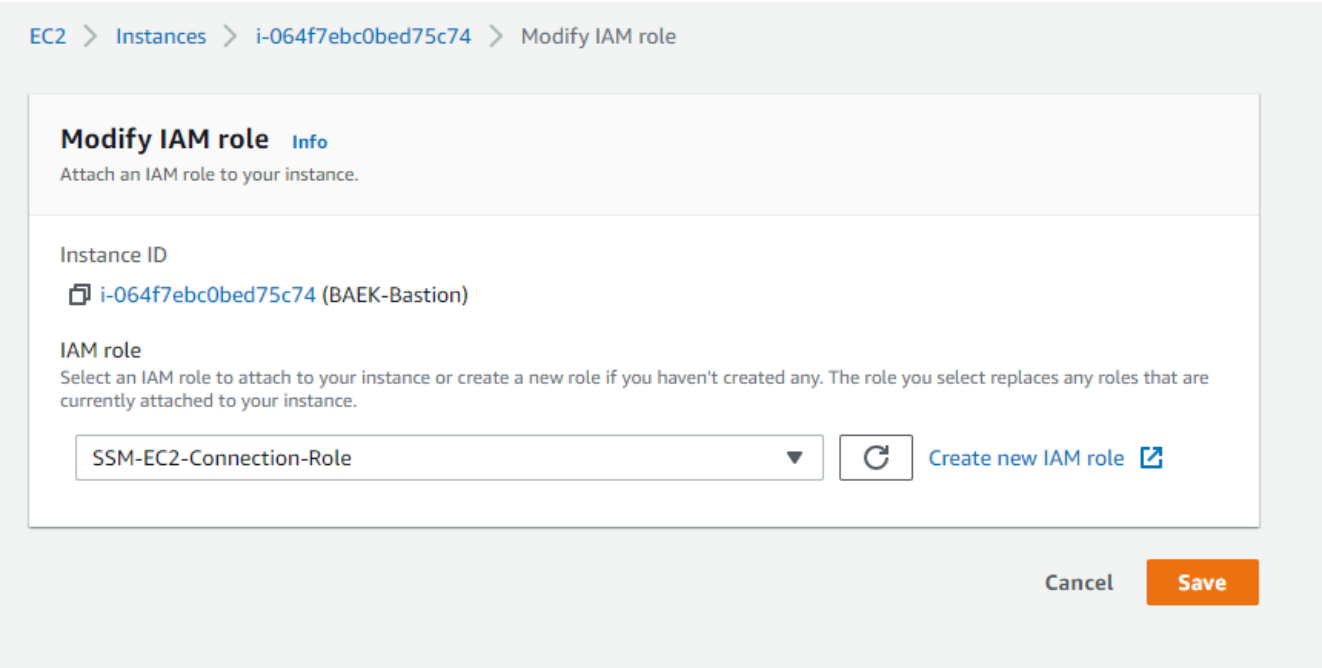
- Role 은 SSM InstanceCore 만 선택



- Role Name 지정 후 생성



EC2 의 보안 설정에서 해당 롤을 추가한다



3. EC2 에 SSM-Agent 를 설치한다

Aamazon 2 리눅스 인스턴스의 경우에는 기본 설정되어 있을 수 있다.

```
[root@ip-10-10-20-201 ~]# sudo yum install -y https://s3.region.amazonaws.com/amazon-ssm-region/latest/linux_amd64/amazon-ssm-agent.rpm
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
Cannot open: https://s3.region.amazonaws.com/amazon-ssm-region/latest/linux_amd64/amazon-ssm-agent.rpm. Skipping.
Error: Nothing to do
[root@ip-10-10-20-201 ~]# wget https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/linux_amd64/amazon-ssm-agent.rpm
--2022-03-29 02:49:06-- https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/linux_amd64/amazon-ssm-agent.rpm
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.217.196.240
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.196.240|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 26724168 (25M) [binary/octet-stream]
Saving to: ‘amazon-ssm-agent.rpm’

100%[==========================================================================================>] 26,724,168  12.7MB/s   in 2.0s

2022-03-29 02:49:08 (12.7 MB/s) - ‘amazon-ssm-agent.rpm’ saved [26724168/26724168]
```

설치 후 자동시작 활성화한다.

```
[root@ip-10-10-20-201 ~]# rpm -Uvh amazon-ssm-agent.rpm
warning: amazon-ssm-agent.rpm: Header V4 RSA/SHA1 Signature, key ID 693eca21: NOKEY
Preparing...
################################# [100%]
Updating / installing...
   1:amazon-ssm-agent-3.1.1080.0-1
################################# [100%]
Created symlink from /etc/systemd/system/multi-user.target.wants/amazon-ssm-agent.service to
```

```
/etc/systemd/system/amazon-ssm-agent.service.
[root@ip-10-10-20-201 ~]# systemctl enable amazon-ssm-agent
[root@ip-10-10-20-201 ~]# systemctl start amazon-ssm-agent



[root@ip-10-10-20-201 ~]# systemctl status amazon-ssm-agent
 -- amazon-ssm-agent.service - amazon-ssm-agent
     Loaded: loaded (/etc/systemd/system/amazon-ssm-
agent.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-03-29 02:53:54 UTC;
21s ago
 Main PID: 3355 (amazon-ssm-agen)
   CGroup: /system.slice/amazon-ssm-agent.service
           ├─3355 /usr/bin/amazon-ssm-agent
           └─3382 /usr/bin/ssm-agent-worker

Mar 29 02:53:54 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:54 INFO Agent will
take identity f...EC2
Mar 29 02:53:54 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:54 INFO [amazon-ssm-
agent] using n...IPC
Mar 29 02:53:55 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:54 INFO [amazon-ssm-
agent] using n...IPC
Mar 29 02:53:55 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:54 INFO [amazon-ssm-
agent] using n...IPC
Mar 29 02:53:55 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:54 INFO [amazon-ssm-
agent] amazon-...0.0
Mar 29 02:53:55 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:54 INFO [amazon-ssm-
agent] OS: lin...d64
Mar 29 02:53:55 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:54 INFO
[CredentialRefresher] Iden...her
Mar 29 02:53:55 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:55 INFO [amazon-ssm-
agent] [LongRu...ess
```

```
Mar 29 02:53:55 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:55 INFO [amazon-ssm-
agent] [LongRu...ted
Mar 29 02:53:55 ip-10-10-20-201.us-west-2.compute.internal
amazon-ssm-agent[3355]: 2022-03-29 02:53:55 INFO [amazon-ssm-
agent] [LongRu...nds
Hint: Some lines were ellipsized, use -l to show in full.
```

4. AWS Cli 에서 SSM 을 통한 EC2 접속

외부 CT 에 AWSCli 를 설치 후 IAM API Key 권한을 설정한다.

```
## 접속할 Client의 IP 확인
$ curl http://icanhazip.com
1.2.3.4

## AWSCli 설치
(중요) SSM의 Session-Plugin 을 사용하기 위해서는 AWSCli 1.16 버전 이상
이어야 한다.
$ curl
"https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

## Linux 용 SSM Session-Manager Plugin 설치
$ curl
"https://s3.amazonaws.com/session-manager-downloads/plugin/lat
est/linux_64bit/session-manager-plugin.rpm" -o "session-
manager-plugin.rpm"
$ rpm -Uvh session-manager-plugin.rpm
$ session-manager-plugin
The Session Manager plugin was installed successfully. Use the
AWS CLI to start a session.

## 접근 정보 설정
본문 내용의 키는 임의 생성한 내용.
$ aws configure
AWS Access Key ID [None]: AKIAQ25632EPN7T7FFVT
AWS Secret Access Key [None]:
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
Default region name [None]: us-west-2
```

```
Default output format [None]: json
```

SSM 을 통한 실제 EC2 에 접근

```
[root@node1 ~]# aws ssm start-session --target i-064f7ebc0bed75c74

Starting session with SessionId: SSM-Only-0a9041d6b13f368ce
sh-4.2$ bash
[ssm-user@ip-10-10-20-201 bin]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
        inet 10.10.20.201  netmask 255.255.255.0  broadcast
10.10.20.255
        inet6 fe80::aa:14ff:fed7:abd  prefixlen 64  scopeid
0x20<link>
        ether 02:aa:14:d7:0a:bd  txqueuelen 1000  (Ethernet)
        RX packets 31846  bytes 8338621 (7.9 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 29180  bytes 6068149 (5.7 MiB)
         TX errors 0  dropped 0 overruns 0  carrier 0
collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
         TX errors 0  dropped 0 overruns 0  carrier 0
collisions 0

[ssm-user@ip-10-10-20-201 bin]$
```

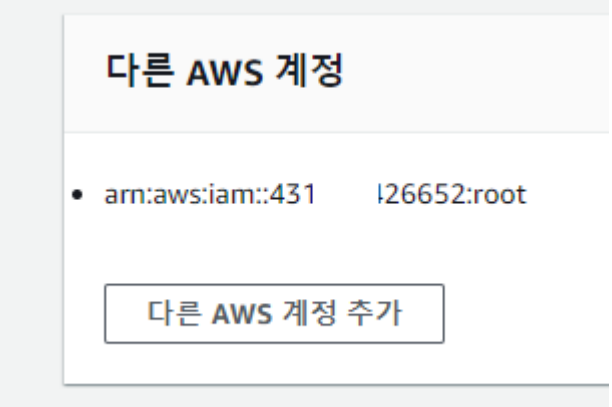
※ 참고자료
https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html
https://docs.aws.amazon.com/ko_kr/systems-manager/latest/userguide/session-manager-working-with-install-plugin.html
https://docs.aws.amazon.com/ko_kr/systems-manager/latest/userguide/session-manager-getting-started.html

암호화된 AMI 이미지를 AWS 계정간 공유하려면?

CMK 공유 설정 없이, 암호화된 볼륨이 있는 프라이빗 AMI 를 AWS 계정간에 공유하면
새 인스턴스 시작 시에 에러 메세지 없이 생성 후 그냥 인스턴스가 자동 종료(Terminated)되어 버립니다.

키 또한 권한을 공유하여 사용해야 합니다.

```
# 공유할 이미지를 지정해 주는 부분입니다.
# 원본 AMI 계정에 IAM 관리형 정책을 만들고 IAM 에 할당합니다.

{
   "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ec2:ModifyImageAttribute"
            ],
            "Resource": [
                                                "arn:aws:ec2:us-
west-2::image/<0e9fcdb7ae40e8f4c>"
## 공유할 이미지의 리전 위치와 id ( ami-xxxxxxxxxxxxxxxxx  에서 xxx 부분입니다 )
            ]
        }
 ]
}

# KMS 서비스에서 Key 를 공유 가능한 어카운트를 설정합니다.
# 원본 AMI 계정의 KMS 서비스에 가서 공유 할 AWS Account 넘버를 공유에 등록해 줍니다.
```




```
# 공유 받은 쪽에서 복호화할 CMK 를 요청하는 부분입니다.
# 공유 받은 계정의 IAM 에 관리형 정책을 등록합니다.
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms:ReEncrypt*",
                "kms:CreateGrant",
                "kms:Decrypt"
            ],
            "Resource": [
                                                "arn:aws:kms:us-
west-2:891977874274:key/bc52517a-676b-4f1a-9d1a-d50241563abc"
## 원본 AMI 의 KMS Key 권한을 요청하는 정책입니다.
            ]
        }
    ]
}
```

이제 정상적으로 공유 받은 이미지에서 EC2 가 생성되고 정상 시작되는 것을 확인할 수 있습니다.

참고 AWS Docs : https://aws.amazon.com/ko/blogs/security/how-to-share-encrypted-amis-across-accounts-to-launch-encrypted-ec2-instances/

CentOS 7에서 APM 최신 버전 yum 으로 설치하기

고정적인 버전 관리가 필요하지 않거나 간단한 설치가 필요할 경우 yum 을 이용할 수 있습니다.

Image OS : CentOS 7.6.1810 Minimal

```
# 공통사항
추가 패키지를 설치하기 위해 우선 확장 레포지토리를 설치해 줍니다
yum install -y epel-release

# Apache 2.4.52
아파치는 보통 빌드를 통해 버전 관리용 RPMs 패키지를 만듭니다.
우선 외부 레포지토리(codeit)를 사용합니다.

cd /etc/yum.repos.d/ && wget https://repo.codeit.guru/codeit.el`rpm -q --qf "%{VERSION}" $(rpm -q --whatprovides redhat-release)`.repo

# 사용가능 버전 확인
yum info httpd

Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirror.kakao.com
 * epel: hk.mirrors.thegigabit.com
 * extras: mirror.kakao.com
 * remi-safe: mirror.bebout.net
 * updates: mirror.navercorp.com
Installed Packages
Name        : httpd
Arch        : x86_64
Version     : 2.4.52
Release     : 1.codeit.el7
Size        : 4.3 M
Repo        : installed
From repo   : CodeIT
```

```
Summary     : Apache HTTP Server
URL         : https://httpd.apache.org/
License     : ASL 2.0
Description : The Apache HTTP Server is a powerful, efficient,
and extensible
            : web server.

# Apache 설치
yum --enablerepo=CodeIT install httpd mod_ssl

# PHP 7.4 설치를 위한 Remi Repository 를 설치합니다.
yum                                                    install
https://rpms.remirepo.net/enterprise/remi-release-7.rpm

# 전체 PHP 버전 리스트를 확인 후 7.4 버전 설치
yum repolist all | grep -i php
yum --enablerepo=remi-php74 install php  php-opcache php-gd
php-mysql php-xml

# MariaDB 10.3 설치를 위한 공식 레포지토리 설정
cat << EOF | tee /etc/yum.repos.d/MariaDB.repo
[mariadb]
name = MariaDB
baseurl = http://yum.mariadb.org/10.3/centos7-amd64
gpgkey=https://yum.mariadb.org/RPM-GPG-KEY-MariaDB
gpgcheck=1
enabled=0
EOF


# MariaDB 10.3 설치
yum -y install --enablerepo=mariadb MariaDB-server MariaDB-
client MariaDB-backup


# APM 버전 확인
[root@localhost ~]# php -v
PHP 7.4.28 (cli) (built: Feb 15 2022 13:23:10) ( NTS )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
     with Zend OPcache v7.4.28, Copyright (c), by Zend
Technologies
[root@localhost ~]# mysql --version
mysql  Ver 15.1 Distrib 10.3.34-MariaDB, for Linux (x86_64)
```

```
using readline 5.1
[root@localhost ~]# httpd -v
Server version: Apache/2.4.52 (codeit)
Server built:   Dec 20 2021 11:29:54
```

Windows 2012 원격 세션 늘리기

Windows Server 2012 설치 이후 고객사에서 원격 접속으로 접속하여 작업시 다른 세션의 접속으로 인하여 세션이 끊기며 서로 뺏어가는 경우가 자주 발생하게 됩니다.

Windows 2012의 경우 1개의 세션만을 공유하도록 기본으로 설정되어 있어 발생하는 문제입니다.

이에 Windows Server 2012에서 세션 설정을 변경하는 방법을 안내드리도록 하겠습니다.

Windows Server 2012의 경우 세션 설정에 대하여 로컬 컴퓨터 정책에서 설정해야 합니다.

실행창에서 gpedit.msc 를 입력하여 실행합니다.

로컬 그룹 정책 편집기가 나타납니다.

원격 세션 연결 설정 변경을 위하여 [컴퓨터 구성] – [관리 템플릿] – [Windows 구성 요소] – [터미널 서비스] – [원격 데스크톱 세션 호스트] – [연결] 메뉴를 선택 한후

[연결 개수 제한]을 항목을 선택합니다.

기본적인 [연결 개수 제한]은 구성되지 않음으로 설정되어 있습니다.

[연결 개수 제한]을 사용으로 선택하며, 옵션에서 TS 최대 허용 연결을 임의의 값으로 설정한 후 확인을 클릭 합니다.(여기서는 2개 선택)

여기 까지의 설정으로 세션 연결은 2개로 늘어났지만, 아직까지는 하나의 세션으로 접속이 됩니다.

이에 각자 별도의 세션으로 접속하기 위한 작업을 다음과 같이 추가적으로 적용해 줍니다.

[원격 데스크톱 서비스 사용자를 하나의 원격 데스크톱 서비스 세션으로 제한] 을 선택합니다.

기본적인 메뉴는 다음과 같습니다.

각각의 세션에서 작업을 하기 위하여 사용 안 함 으로 선택 후 확인을 클릭합니다.

변경된 정책이 적용 될 수 있도록 정책 업데이트를 적용 해줍니다.

CentOS 7 설치 후 할 일 – 파티셔닝 및 설정 (1)

CentOS 7 설치 후 할 일

실제로 서비스를 제공하는 서버라면 몇 가지 기본적인 설정, 패키지, 보안에 신경을 써줘야 한다.

파티셔닝 및 설정, 필수 및 권장 패키지 설치, 서버 운영을 위한 보안 설정(하드닝)

세가지로 분류해 작성한다.

파티셔닝

LVM : Default로 설정되어 있으며 스토리지 확장 시 유연하게 변경 가능, 관리 난이도가 높아 일반적으로 **표준 파티션** 사용 권장

표준 파티션 : 디스크 성능이 좋아짐에 따라 속도 면에서 **LVM**과 크게 차이가 없음, xfs 형식 사용하여 이전보다 파티션 조절 수월함

1. 네트워크 설정

(OS 설치 과정에서 네트워크 기설정되어 있는 경우 해당 내용 생략)

```
# ip addr
# vi /etc/sysconfig/network-scripts/ifcfg-eth0

. . .
BOOTPROTO=none
. . .
IPV6INIT=no
IPV6_AUTOCONF=no
IPV6_DEFROUTE=no
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
. . .
ONBOOT=yes                                  # 부팅/네트워크 재시작 시  자동으로 인터페이스 활성화를 위해  yes  설정
IPV6_PRIVACY=no
IPADDR=192.168.122.243
NETMASK=255.255.255.0
GATEWAY=192.168.122.1
DNS1=8.8.8.8
DNS2=8.8.4.4

# systemctl restart network
```

```
# ip addr
eth0 영역 IP 확인



# ping -c 4 google.com                # 네트워크 상태 확인
--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3094ms
rtt min/avg/max/mdev = 82.692/83.010/83.554/0.492 ms
```

2. 바이오스 타이머 조정

CentOS 7 부터 timedatectl 명령어로 시간, 타임존 설정을 할 수 있다.

```
# timedatectl

      RTC time : 하드웨어 시간
      NTP enabled : NTP 활성화 여부
      NTP synchronized : NTP 동기화 여부
      RTC in local TZ : RTC 와 Time zone 시간 동기화 여부

# timedatectl list-timezones | grep -i Asia*

# timedatectl set-timezone Asia/Seoul

# timedatectl
```

3. Hostname 설정

CentOS 7 설치 후 기본적으로 hostname이 localhost.localdomain 으로 설정되어있으므로 변경해주어야 함

```
# hostnamectl
Static hostname: localhost.localdomain

# hostnamectl set-hostname newhostname
```

```
# hostnamectl
Static hostname: newhostname
```

4. SELinux 비활성화

SELinux는 기본적으로 활성화 되어있으나, 해당 메뉴얼에서는 disabled(=미사용) 설정한다.

```
# vi /etc/sysconfig/selinux

. . .
SELINUX=disabled     . . .

# shutdown -r now

# getenforce
Disabled
```

5. root 계정 원격 접근 불가 설정

보안을 위해 root로의 원격 접속은 거부하고 타 계정을 사용하여 su 명령어를 통해 사용하도록 한다.

```
# ps -ef | grep sshd
# systemctl enable sshd

# vi /etc/ssh/sshd_config
. . .
PermitRootLogin=no
. . .

# systemctl restart sshd
```

6. 자동 로그아웃 설정

보안을 위해 일정 시간 응답이 없으면 자동으로 로그아웃 하도록 설정한다.

```
# vi /etc/profile.d/timeout.sh
TMOUT=600
export TMOUT

chmod +x /etc/profile.d/timeout.sh

# source /etc/profile
# echo $TMOUT
600
```

7. 히스토리 포맷 설정

명령어의 내역을 출력하는 명령어인 history의 날짜 및 시간을 확인하기 위해 설정한다.

```
# vi /etc/profile.d/history.sh

HISTTIMEFORMAT="%F %T -- "
export HISTTIMEFORMAT


# chmod 644 /etc/profile.d/history.sh
# source /etc/profile.d/history.sh

# hisotry
999  2022-04-06 14:50:10 -- vi /etc/profile.d/history.sh
1000       2022-04-06    14:50:19    --    chmod    644
/etc/profile.d/history.sh
1001  2022-04-06 14:50:28 -- source /etc/profile.d/history.sh
1002  2022-04-06 14:50:30 -- history
```

8. 문자셋 설정

설치시 언어와 문자셋을 잘못 선택했을 경우에 설정한다.

```
# localectl
  System Locale: LANG=en_US.UTF-8
      VC Keymap: us
     X11 Layout: us

# localectl list-locales | grep -i kr
ko_KR
ko_KR.euckr
ko_KR.utf8

# localectl set-locale LANG=ko_KR.UTF-8
# localectl set-keymap kr
# localectl set-x11-keymap kr

# localectl
   System Locale: LANG=ko_KR.UTF-8
       VC Keymap: kr
      X11 Layout: kr
```

다음 장에선 필수적으로 설치해야할 패키지 및 권장 패키지들에 대해서 설명한다.