1.1 인프라 아키텍쳐

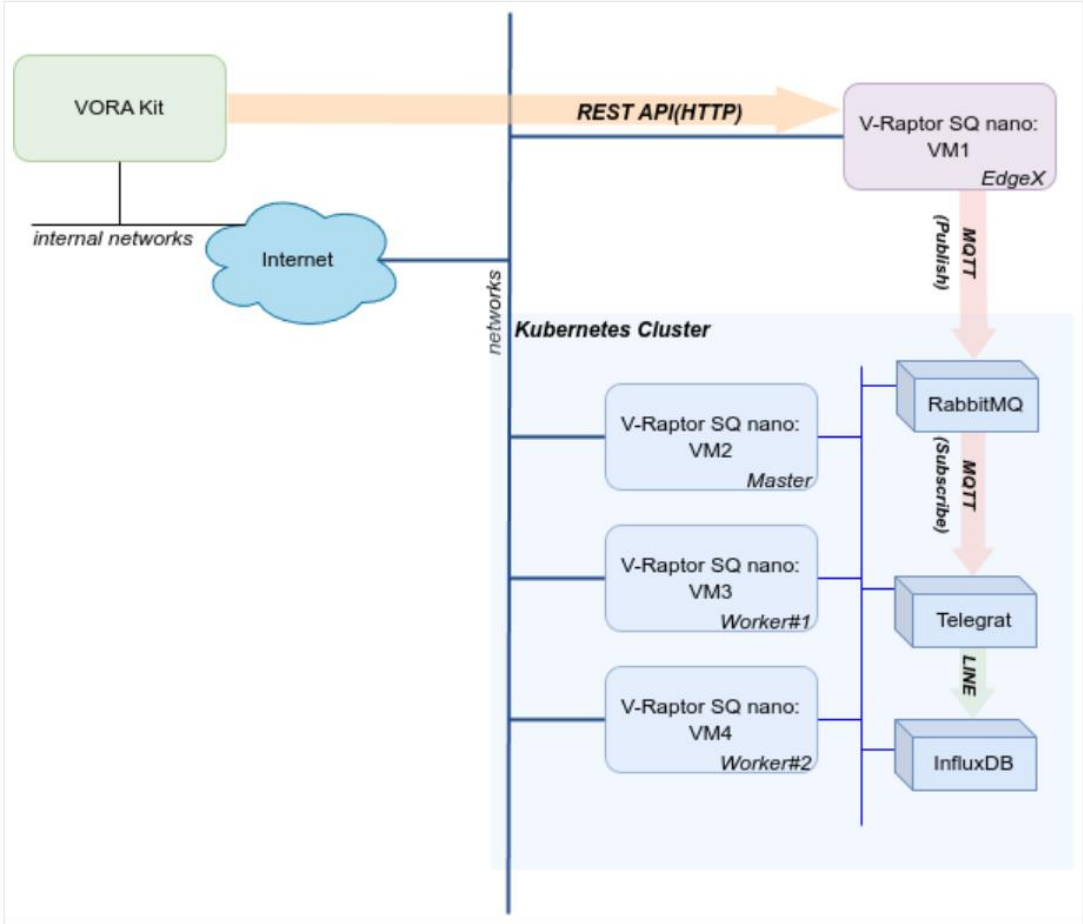
이 가이드에서는 Vora Kit 및 V-Raptor SQ nano를 사용합니다.

- Vora Kit은 온/습도 센서를 이용해 데이터를 측정하고,
- V-Raptor SQ nano은 사용한 총 4개의 가상머신을 이용하여, 엣지서버 및 클라우드서버(Kubernetes 인프라환경)를 구축함

인프라 환경은 다음과 같습니다.

VM	서버	설명
Vora Kit	센서 서버	센서가 장착된 서버활용
SQnano VM1	엣지 서버	Edge 컴퓨팅 서비스 서버활용
SQnano VM2	클라우드 서버	K8S 마스터 서버활용
SQnano VM3	클라우드 서버	K8S 워커 서버활용
SQnano VM4	클라우드 서버	K8S 워커 서버활용

1.2 데이터 흐름도



3.3. V-Raptor SQ nano에서 VM의 환경설정

- 총 4개의 VM을 만들고, 그 정보는 아래와 같습니다.

Node	Info.	IP	vCPU	Memory(GB)	Storage(GB)
VM1	Edge Server	192.168.1.191	4	4096	25
vm2	K8S Master	192.168.1.192	4	4096	25
vm3	K8S Worker#1	192.168.1.193	4	4096	25
vm4	K8S Worker#2	192.168.1.194	4	4096	25

```
vraptor@vraptor:~$ sudo nmcli device wifi connect "XSLAB_5G" password 'xslab7238!!'
Device 'wlx14ebb6553772' successfully activated with 'a53cfdd5-b197-458b-98d7-61e1d5a65c9e'.
```

```
vraptor@vraptor:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 50:62:55:40:b0:38 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.56/22 brd 192.168.3.255 scope global dynamic eth0
       valid_lft 525521sec preferred_lft 525521sec
    inet6 fe80::5262:55ff:fe40:b038/64 scope link
       valid_lft forever preferred_lft forever
3: wlx14ebb6553772: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 14:eb:b6:55:37:72 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.141/22 brd 192.168.3.255 scope global dynamic noprefixroute wlx14ebb6553772
       valid_lft 604789sec preferred_lft 604789sec
    inet6 fe80::67a7:df23:139b:ae5f/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
vraptor@vraptor:~$
```

```
vraptor@vraptor:~$ ifconfig -a
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet6 fe80::5262:55ff:fe40:b038  prefixlen 64  scopeid 0x20<link>
        ether 50:62:55:40:b0:38  txqueuelen 1000  (Ethernet)
        RX packets 1813893  bytes 236314201 (236.3 MB)
        RX errors 0  dropped 43668  overruns 0  frame 0
        TX packets 93330  bytes 4712146 (4.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 10

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 51702  bytes 4549594 (4.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 51702  bytes 4549594 (4.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlx14ebb6553772: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.141  netmask 255.255.252.0  broadcast 192.168.3.255
        inet6 fe80::67a7:df23:139b:ae5f  prefixlen 64  scopeid 0x20<link>
        ether 14:eb:b6:55:37:72  txqueuelen 1000  (Ethernet)
        RX packets 5107  bytes 1048254 (1.0 MB)
        RX errors 0  dropped 159  overruns 0  frame 0
        TX packets 552  bytes 81379 (81.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vraptor@vraptor:~$
```

```
# Cgroup를 systemd로 설정
cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
```

왜 cgroup 를 systemd 로 설정해야 하지?

3. cgroup에 대한 쿠버네티스와 도커 선호도 차이는 왜?

이론적으로는 직접 cgroup을 마운트해서 사용하는 cgroupfs를 이용하거나 systemd와 같이 계층적인 구조로 만들어 접근하거나 **결과적으로는 동일**해야 합니다.
그럼에도 쿠버네티스에서는 systemd를 선호하고 도커에서는 cgroupfs를 왜 선호하는지 알아보았지만, 정확한 결론을 도출하진 못했습니다.
다만 쿠버네티스 공식 문서에서는 도커 18.03 + 커널 4.17.17 이하의 RHEL 에서는 cgroupfs 에서 메모리 leaking issue가 발생하여 systemd로 변경하라고 권고하는 것이 있으며,
도커에서는 cgroupfs를 cgroup v1에서만 사용하고 cgroup v2에서는 systemd로 변경할 계획이 있음을 밝히고 있습니다. 그리고 카카오 엔터프라이즈에서는 2개의 차이는 경로 관리 정책의 차이 일뿐 지금 당장은 시스템 성능에 영향을 주는 것은 아니라고 판단하여 cgroupfs 으로 진행하기로 하였다고 밝힌바 있습니다.
이를 종합적으로 본다면, 결과적으로는 cgroup을 구조화 하는 **방법론적인 차이**가 있는 수준으로 추정해 볼 수 있습니다.

https://www.slideshare.net/slideshow/systemd-cgroup/250042237

왜 쿠버네티스는 systemd로 cgroup을 관리하려고 할까요?

This document provides who want to know about cgroupfs in k8s.

- Email: pagaia@hotmail.com, geunwoo.j.shim@gmail.com
- Github: https://github.com/sysnet4admin, https://github.com/gnu-gnu

왜 cgroup 를 systemd 로 설정해야 하지?

3. cgroup에 대한 쿠버네티스와 도커 선호도 차이는 왜?

이론적으로는 직접 cgroup을 마운트해서 사용하는 cgroupfs를 이용하거나 systemd와 같이 계층적인 구조로 만들어 접근하거나 **결과적으로는 동일**해야 합니다.
그럼에도 쿠버네티스에서는 systemd를 선호하고 도커에서는 cgroupfs를 왜 선호하는지 알아보았지만, 정확한 결론을 도출하진 못했습니다.
다만 쿠버네티스 공식 문서에서는 도커 18.03 + 커널 4.17.17 이하의 RHEL 에서는 cgroupfs 에서 메모리 leaking issue가 발생하여 systemd로 변경하라고 권고하는 것이 있으며,
도커에서는 cgroupfs를 cgroup v1에서만 사용하고 cgroup v2에서는 systemd로 변경할 계획이 있음을 밝히고 있습니다. 그리고 카카오 엔터프라이즈에서는 2개의 차이는 경로 관리 정책의 차이 일뿐 지금 당장은 시스템 성능에 영향을 주는 것은 아니라고 판단하여 cgroupfs 으로 진행하기로 하였다고 밝힌바 있습니다.
이를 종합적으로 본다면, 결과적으로는 cgroup을 구조화 하는 **방법론적인 차이**가 있는 수준으로 추정해 볼 수 있습니다.

https://www.slideshare.net/slideshow/systemd-cgroup/250042237

cgroup drivers

On Linux, control groups are used to constrain resources that are allocated to processes.

Both the kubelet and the underlying container runtime need to interface with control groups to enforce resource management for pods and containers and set resources such as cpu/memory requests and limits. To interface with control groups, the kubelet and the container runtime need to use a *cgroup driver*. It's critical that the kubelet and the container runtime use the same cgroup driver and are configured the same.

There are two cgroup drivers available:

- `cgroupfs`
- `systemd`

cgroupfs driver

The `cgroupfs` driver is the default cgroup driver in the kubelet. When the `cgroupfs` driver is used, the kubelet and the container runtime directly interface with the cgroup filesystem to configure cgroups.

The `cgroupfs` driver is **not** recommended when systemd is the init system because systemd expects a single cgroup manager on the system. Additionally, if you use cgroup v2, use the `systemd` cgroup driver instead of `cgroupfs`.

systemd cgroup driver

When systemd is chosen as the init system for a Linux distribution, the init process generates and consumes a root control group (`cgroup`) and acts as a cgroup manager.

systemd has a tight integration with cgroups and allocates a cgroup per systemd unit. As a result, if you use `systemd` as the init system with the `cgroupfs` driver, the system gets two different cgroup managers.

https://kubernetes.io/docs/setup/production-environment/container-runtimes/

왜 cgroup 를 systemd 로 설정해야 하지?

3. cgroup에 대한 쿠버네티스와 도커 선호도 차이는 왜?

이론적으로는 직접 cgroup을 마운트해서 사용하는 cgroupfs를 이용하거나 systemd와 같이 계층적인 구조로 만들어 접근하거나 **결과적으로는 동일**해야 합니다.
그럼에도 쿠버네티스에서는 systemd를 선호하고 도커에서는 cgroupfs를 왜 선호하는지 알아보았지만, 정확한 결론을 도출하진 못했습니다.
다만 쿠버네티스 공식 문서에서는 도커 18.03 + 커널 4.17.17 이하의 RHEL 에서는 cgroupfs 에서 메모리 leaking issue가 발생하여 systemd로 변경하라고 권고하는 것이 있으며,
도커에서는 cgroupfs를 cgroup v1에서만 사용하고 cgroup v2에서는 systemd로 변경할 계획이 있음을 밝히고 있습니다. 그리고 카카오 엔터프라이즈에서는 2개의 차이는 경로 관리 정책의 차이 일뿐 지금 당장은 시스템 성능에 영향을 주는 것은 아니라고 판단하여 cgroupfs 으로 진행하기로 하였다고 밝힌바 있습니다.
이를 종합적으로 본다면, 결과적으로는 cgroup을 구조화 하는 **방법론적인 차이**가 있는 수준으로 추정해 볼 수 있습니다.

https://www.slideshare.net/slideshow/systemd-cgroup/250042237

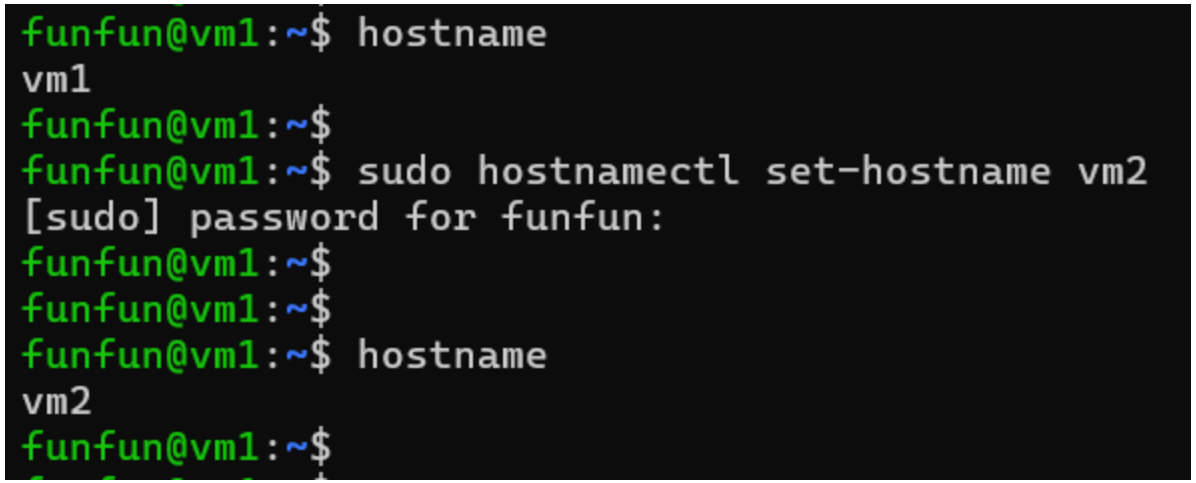
```
funfun@vm1:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2024-06-29 12:06:28 UTC; 8min ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 45454 (dockerd)
      Tasks: 10
     Memory: 22.4M
     CGroup: /system.slice/docker.service
             └─45454 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Jun 29 12:06:26 vm1 dockerd[45454]: time="2024-06-29T12:06:26.076473834Z" level=info msg="Starting up"
Jun 29 12:06:26 vm1 dockerd[45454]: time="2024-06-29T12:06:26.084986166Z" level=info msg="detected 127.0.0.53 nameserve>
Jun 29 12:06:26 vm1 dockerd[45454]: time="2024-06-29T12:06:26.471388724Z" level=info msg="[graphdriver] trying configur>
Jun 29 12:06:26 vm1 dockerd[45454]: time="2024-06-29T12:06:26.530896399Z" level=info msg="Loading containers: start."
Jun 29 12:06:27 vm1 dockerd[45454]: time="2024-06-29T12:06:27.796339932Z" level=info msg="Loading containers: done."
Jun 29 12:06:27 vm1 dockerd[45454]: time="2024-06-29T12:06:27.865518440Z" level=warning msg="WARNING: No swap limit sup>
Jun 29 12:06:27 vm1 dockerd[45454]: time="2024-06-29T12:06:27.865782411Z" level=info msg="Docker daemon" commit=e953d76>
Jun 29 12:06:27 vm1 dockerd[45454]: time="2024-06-29T12:06:27.866352532Z" level=info msg="Daemon has completed initiali>
Jun 29 12:06:28 vm1 systemd[1]: Started Docker Application Container Engine.
Jun 29 12:06:28 vm1 dockerd[45454]: time="2024-06-29T12:06:28.072852974Z" level=info msg="API listen on /run/docker.soc>
lines 1-21/21 (END)
```

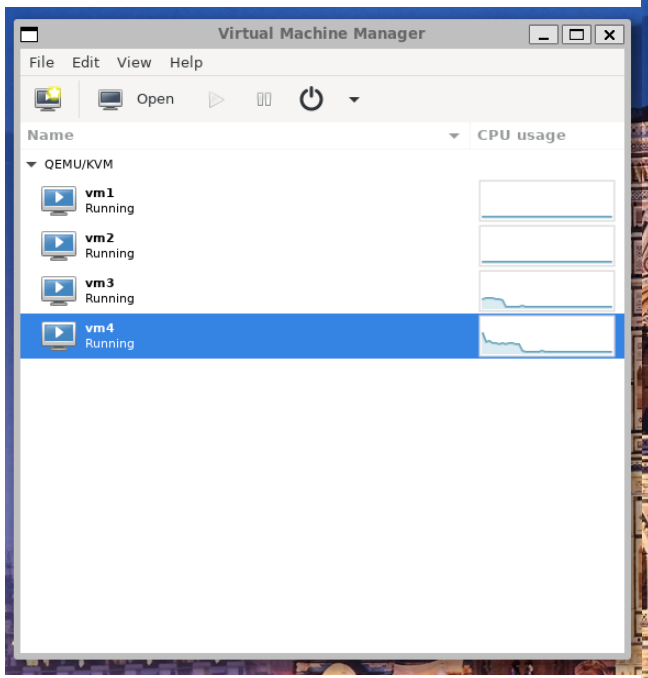
$ sudo systemctl status docker

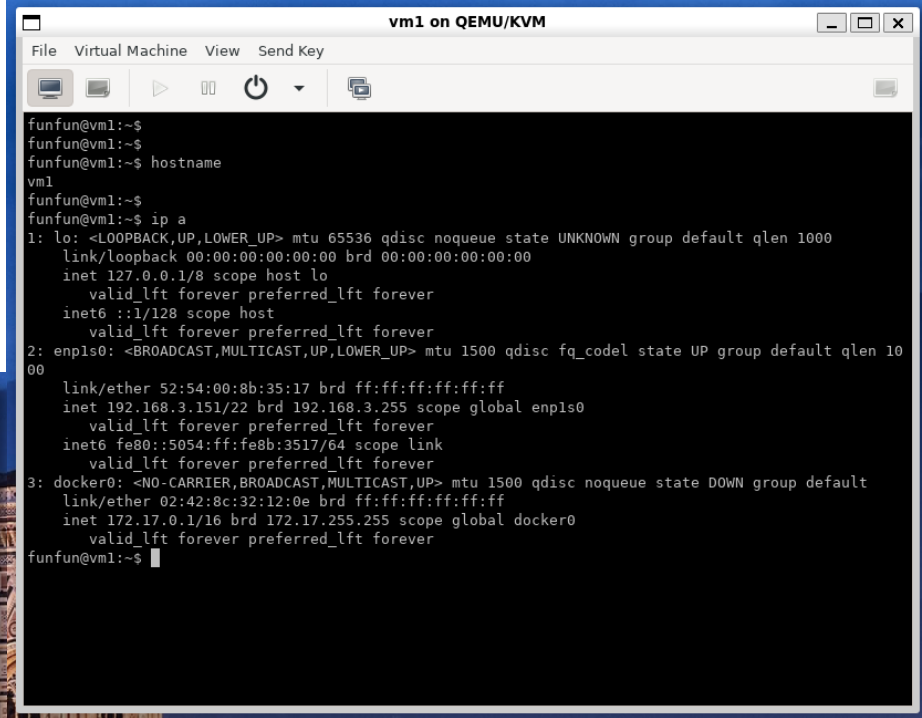
```
funfun@vm1:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
478afc919002: Pull complete
Digest: sha256:94323f3e5e09a8b9515d74337010375a456c909543e1ff1538f5116d38ab3989
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

funfun@vm1:~$
```

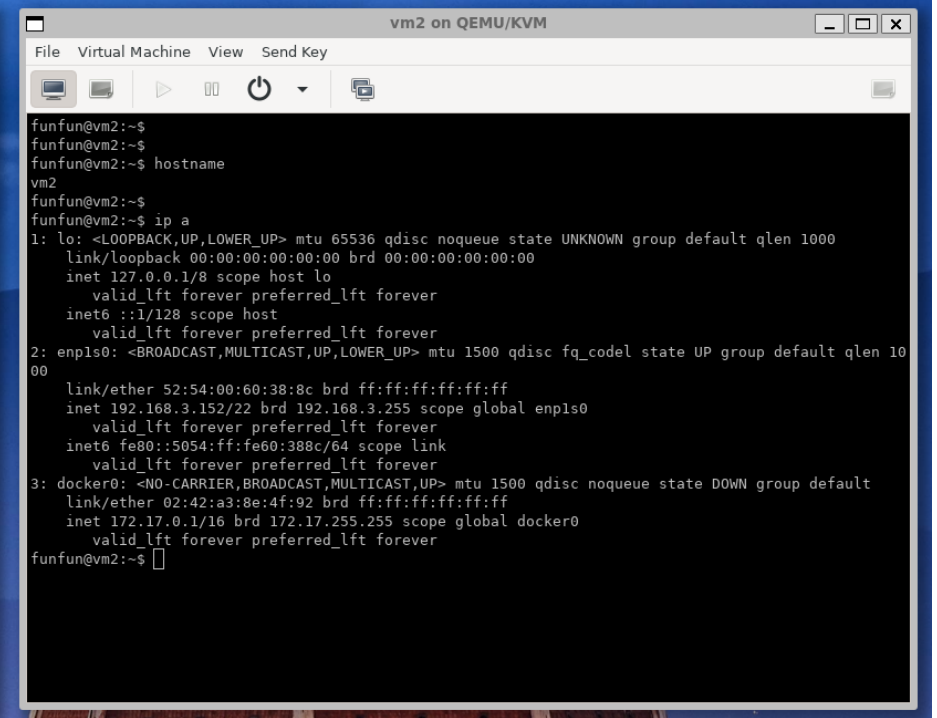
$ sudo docker run hello-world

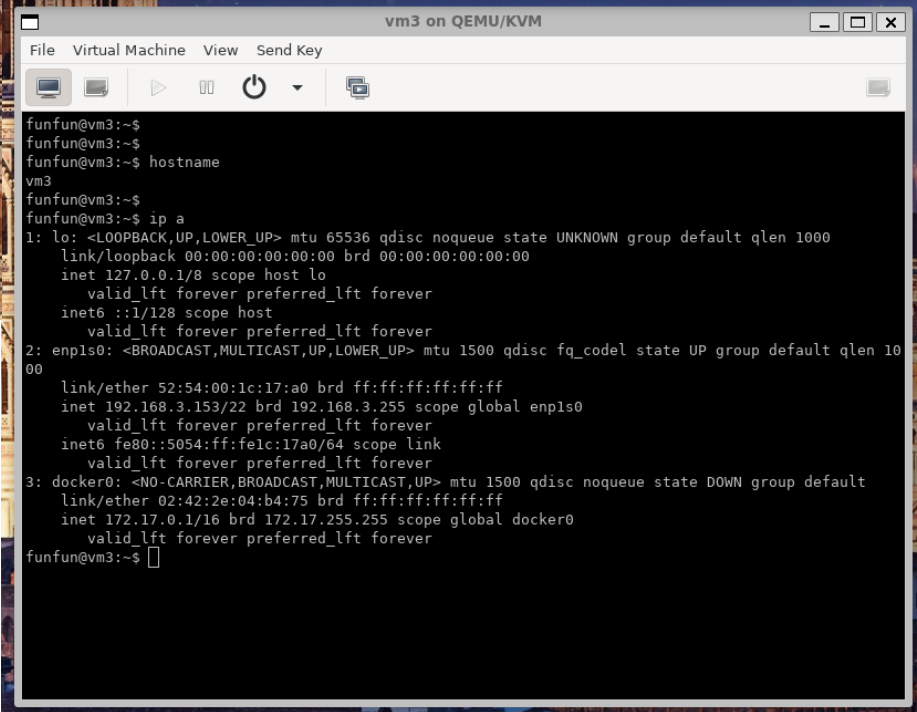
```
funfun@vm1:~$ docker ps
permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Get "http://%2Fvar
%2Frun%2Fdocker.sock/v1.46/containers/json": dial unix /var/run/docker.sock: connect: permission denied
funfun@vm1:~$
funfun@vm1:~$
funfun@vm1:~$ ls -lF /var/run/docker.sock
srw-rw---- 1 root docker 0 Jun 29 12:06 /var/run/docker.sock=
funfun@vm1:~$
funfun@vm1:~$ sudo groupadd docker
groupadd: group 'docker' already exists
funfun@vm1:~$
funfun@vm1:~$ sudo usermod -aG docker $USER
funfun@vm1:~$
funfun@vm1:~$ sudo chown root:docker /var/run/docker.sock
funfun@vm1:~$
funfun@vm1:~$ sudo service docker restart
funfun@vm1:~$
funfun@vm1:~$ sudo chmod 666 /var/run/docker.sock
funfun@vm1:~$
funfun@vm1:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
funfun@vm1:~$
funfun@vm1:~$
```

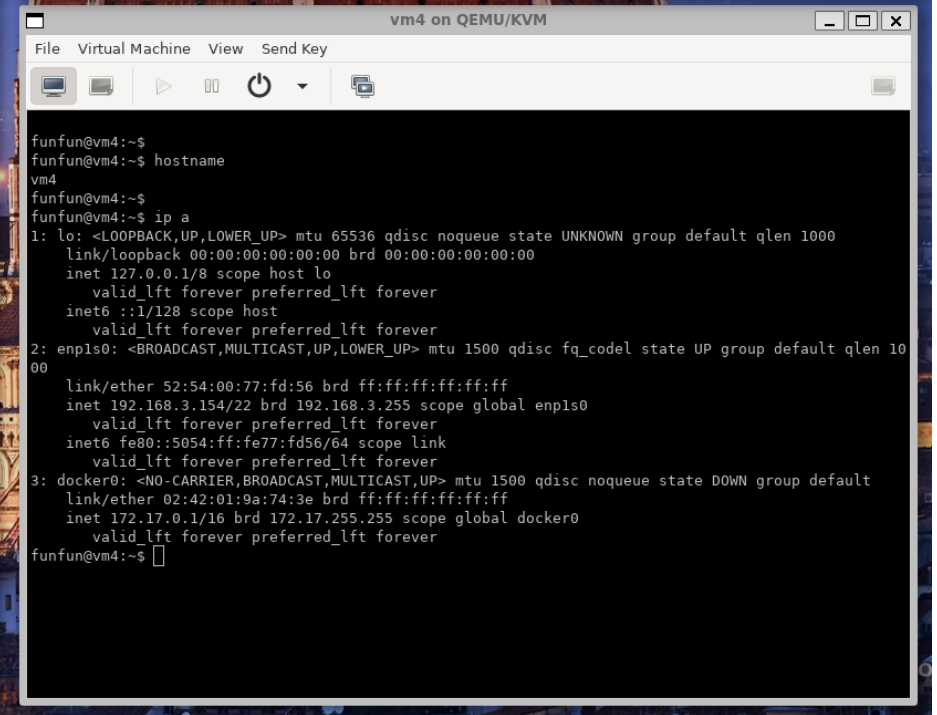

funfun@vm1:~$ hostname
vm1
funfun@vm1:~$
funfun@vm1:~$ sudo hostnamectl set-hostname vm2
[sudo] password for funfun:
funfun@vm1:~$
funfun@vm1:~$
funfun@vm1:~$ hostname
vm2
funfun@vm1:~$


Virtual Machine Manager
File Edit View Help
Open
Name
CPU usage
QEMU/KVM
vm1
Running
vm2
Running
vm3
Running
vm4
Running


vm1 on QEMU/KVM
File Virtual Machine View Send Key
funfun@vm1:~$
funfun@vm1:~$
funfun@vm1:~$ hostname
vm1
funfun@vm1:~$
funfun@vm1:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether 52:54:00:8b:35:17 brd ff:ff:ff:ff:ff:ff
inet 192.168.3.151/22 brd 192.168.3.255 scope global enp1s0
valid_lft forever preferred_lft forever
inet6 fe80::5054:ff:fe8b:3517/64 scope link
valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
link/ether 02:42:8c:32:12:0e brd ff:ff:ff:ff:ff:ff
inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
valid_lft forever preferred_lft forever
funfun@vm1:~$


vm2 on QEMU/KVM
File Virtual Machine View Send Key
funfun@vm2:~$
funfun@vm2:~$
funfun@vm2:~$ hostname
vm2
funfun@vm2:~$
funfun@vm2:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether 52:54:00:60:38:8c brd ff:ff:ff:ff:ff:ff
inet 192.168.3.152/22 brd 192.168.3.255 scope global enp1s0
valid_lft forever preferred_lft forever
inet6 fe80::5054:ff:fe60:388c/64 scope link
valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
link/ether 02:42:a3:8e:4f:92 brd ff:ff:ff:ff:ff:ff
inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
valid_lft forever preferred_lft forever
funfun@vm2:~$


vm3 on QEMU/KVM
File Virtual Machine View Send Key
funfun@vm3:~$
funfun@vm3:~$
funfun@vm3:~$ hostname
vm3
funfun@vm3:~$
funfun@vm3:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether 52:54:00:1c:17:a0 brd ff:ff:ff:ff:ff:ff
inet 192.168.3.153/22 brd 192.168.3.255 scope global enp1s0
valid_lft forever preferred_lft forever
inet6 fe80::5054:ff:fe1c:17a0/64 scope link
valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
link/ether 02:42:2e:04:b4:75 brd ff:ff:ff:ff:ff:ff
inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
valid_lft forever preferred_lft forever
funfun@vm3:~$


vm4 on QEMU/KVM
File Virtual Machine View Send Key
funfun@vm4:~$
funfun@vm4:~$ hostname
vm4
funfun@vm4:~$
funfun@vm4:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether 52:54:00:77:fd:56 brd ff:ff:ff:ff:ff:ff
inet 192.168.3.154/22 brd 192.168.3.255 scope global enp1s0
valid_lft forever preferred_lft forever
inet6 fe80::5054:ff:fe77:fd56/64 scope link
valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
link/ether 02:42:01:9a:74:3e brd ff:ff:ff:ff:ff:ff
inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
valid_lft forever preferred_lft forever
funfun@vm4:~$

생성된 모든 인스턴스에 아래와 같이 도커 및 쿠버네티스를 설치합니다.

- 실습 데이터 다운로드

```
sudo apt update; sudo apt upgrade -y
sudo su
git clone https://yona.xslab.co.kr/Programmers/1st-KDT-Linux
cd 1st-KDT-Linux/k8s
```

- 실행파일 권한설정

```
chmod +x docker-install.sh
chmod +x k8s-install-setup.sh
```

왜 스왑을 꺼놔야 할까?

- 스왑 영역 삭제

```
sudo swapoff -a
sudo vim /etc/fstab
> # swap
```

- 도커 및 쿠버네티스 설치

```
sudo ./docker-install.sh
sudo ./k8s-install-setup.sh
```

swap 메모리란

우선 swap 메모리가 무엇인지부터 간단하게 살펴보자. 스왑(swap)이란 물리 메모리(RAM)의 용량이 부족할 때 하드 디스크의 일부 공간을 메모리처럼 사용하는 것이다.

이러한 동작은 swap in과 swap out으로 구분할 수 있다. 현재 메모리에 최대 100개의 프로세스가 올라갈수 있는데 이때, 101번째 프로세스가 추가로 올라가야 할 경우를 생각해보자.

swap out은 100개의 실행된 프로세스중 특정 프로세스를 잠시 Swap Partition으로 이동시켜 놓는 것을 말한다. 반대로 swap in은 스왑으로 이동했던 프로세스에서 이벤트가 올 경우, 다시 메모리 영역으로 이동시키는 것을 말한다.

왜 스왑 메모리를 꺼놓아야 할까

설명만으로는 스왑 메모리가 굉장히 좋은 기능을 하는 것 같은데 왜 쿠버네티스를 구축할 때는 이 기능을 꺼놓아야할까? 이유는 쿠버네티스의 kubelet이 이러한 상황을 처리하도록 만들어지지 않았기 때문이다.

쿠버네티스는 Pod를 생성할 때, 필요한 만큼의 리소스를 할당 받아서 사용하는 구조이다. 따라서 메모리 Swap을 고려하지 않고 설계되었기 때문에, 쿠버네티스 클러스터 Node들은 모두 Swap 메모리를 비활성화 해줘야 한다.

정리해보면, Swap 기능은 본래 가용된 메모리보다 더 큰 메모리 할당을 가능하도록 하기 위함인데, 쿠버네티스 철학은 주어진 인스턴스의 자원을 100% 가깝게 사용하는 것이 목표인데, 스왑 메모리를 켜놓으면 인스턴스 자원이 일관되지 않아 이러한 철학에 부합되지 않는 것이다.

https://kgw7401.tistory.com/50

```
vraptor@vraptor:~$ sudo nmtui
```

```
┌──┤ NetworkManager TUI ├──┐
│                          │
│ Please select an option  │
│                          │
│ Edit a connection        │
│ Activate a connection    │
│ Set system hostname      │
│                          │
│ Quit                     │
│                          │
│                   <OK>   │
└──────────────────────────┘
```

```
┌──┤ Edit Connection ├──┐
│                                                    │
│        Profile name XSLAB_5G                       │
│              Device wlx40ae309b984b (40:AE:30:9B:98:4B)
│                                                    │
│ = WI-FI                                    <Hide>  │
│ │            SSID XSLAB_5G                         │
│ │            Mode <Client>                         │
│ │                                                  │
│ │        Security <WPA & WPA2 Personal>            │
│ │        Password **********                       │
│ │                 [ ] Show password                │
│ │                                                  │
│ │           BSSID                                  │
│ │ Cloned MAC address                               │
│ │             MTU           (default)              │
│ └                                                  │
│                                                    │
│ = IPv4 CONFIGURATION <Automatic>           <Show>  │
│ = IPv6 CONFIGURATION <Automatic>           <Show>  │
│                                                    │
│ [X] Automatically connect                          │
│ [X] Available to all users                         │
│                                                    │
│                                  <Cancel> <OK>     │
└────────────────────────────────────────────────────┘
```

https://www.postman.com/downloads/

오늘 (1)

Download Postman

Download the app to get started using the Postman API Platform today. Or, if you prefer a browser experience, you can try the web version of Postman.

The Postman app

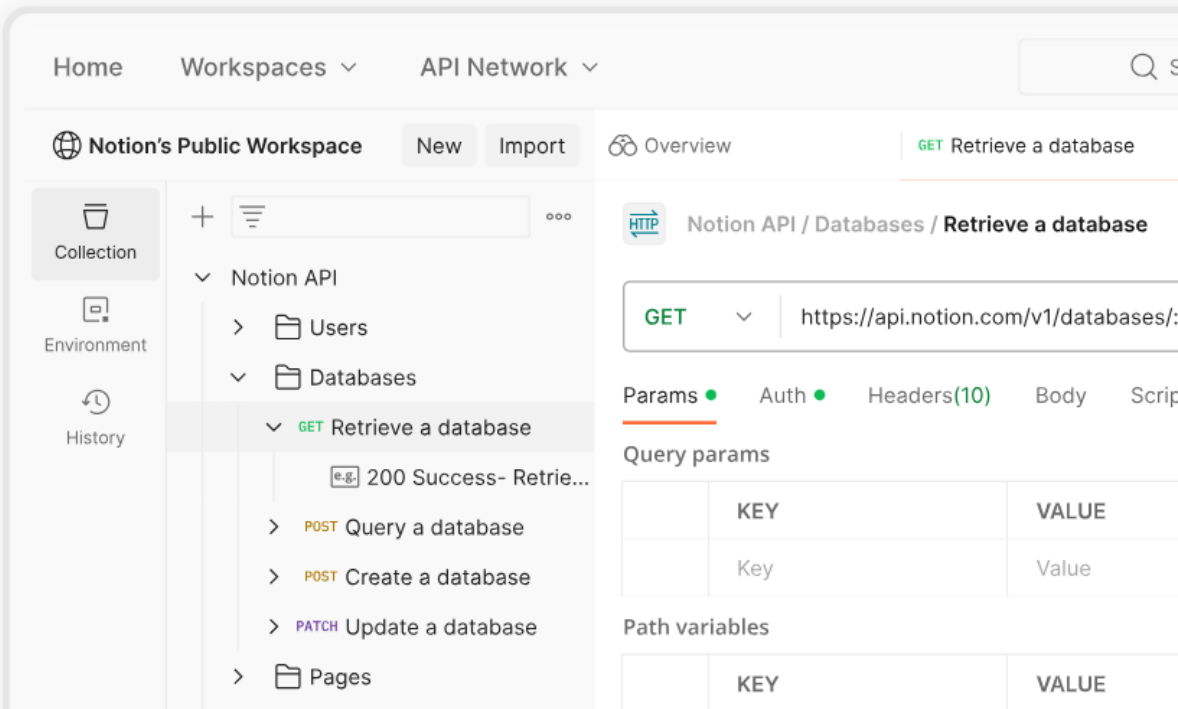
Download the app to get started with the Postman API Platform.

Windows 64-bit

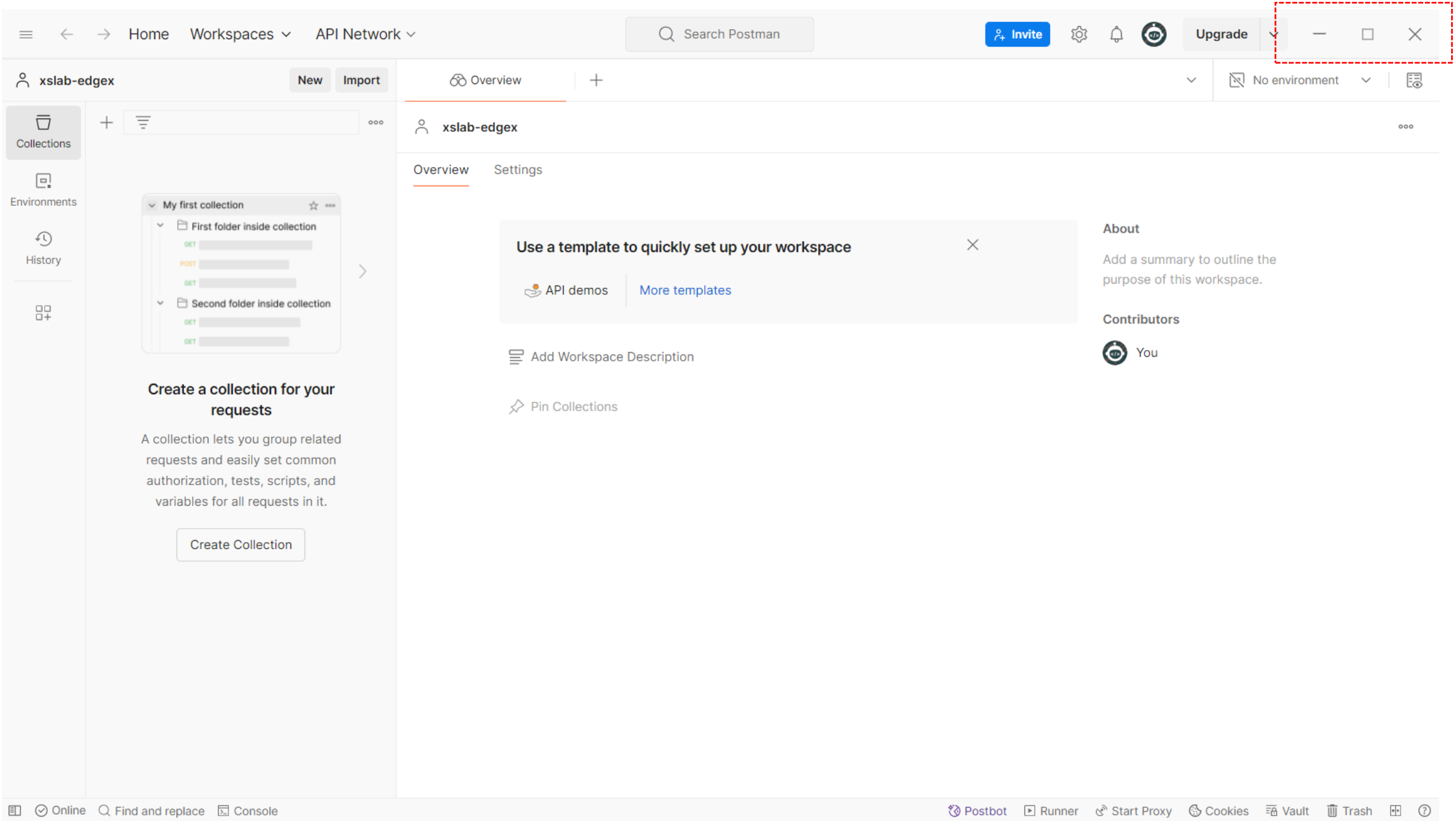
By downloading and using Postman, I agree to the Privacy Policy and Terms.

Release Notes →

Not your OS? Download for Mac (Intel Chip, Apple Chip) or Linux (x64, arm64)



윈도우용 postman

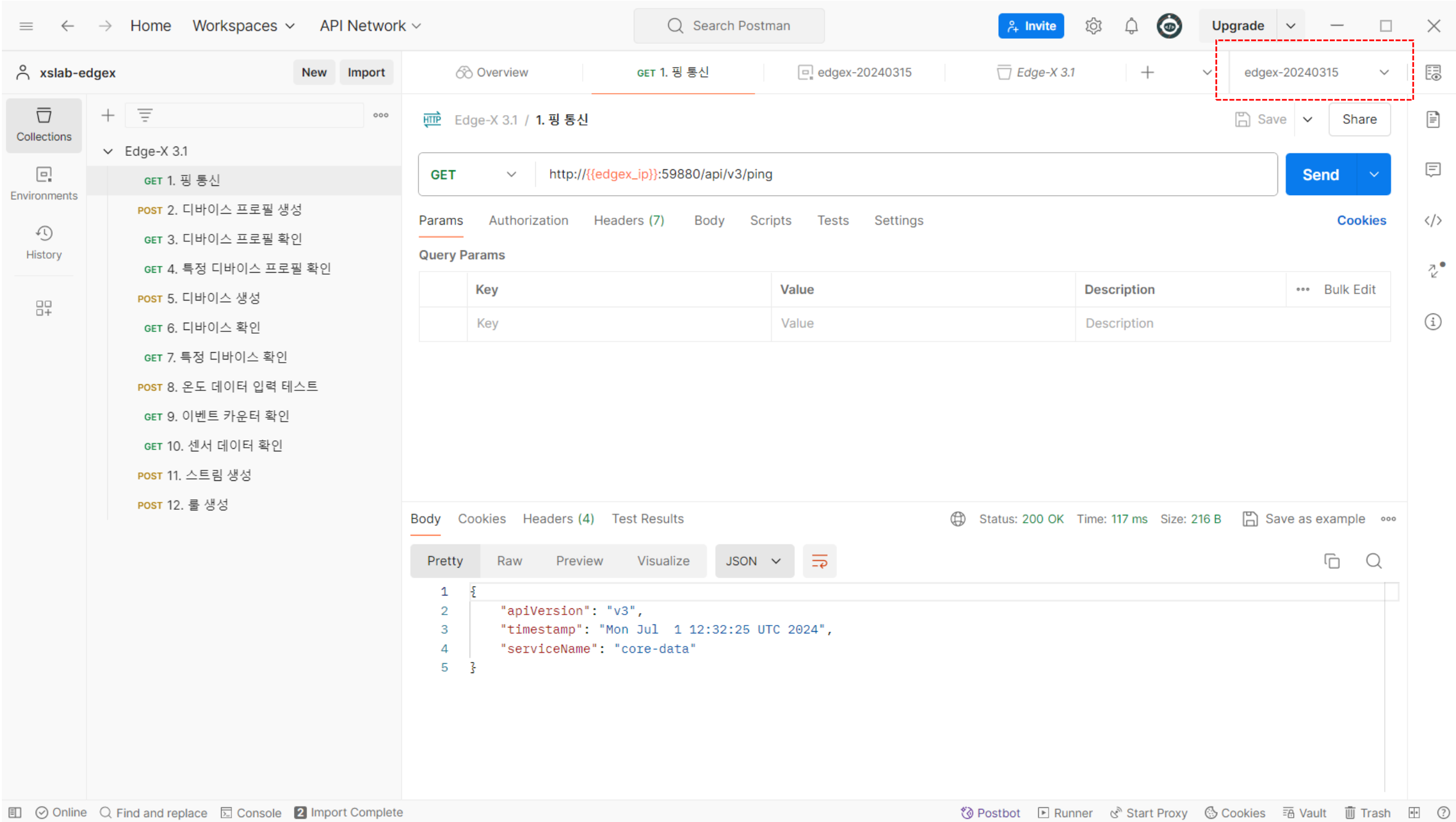


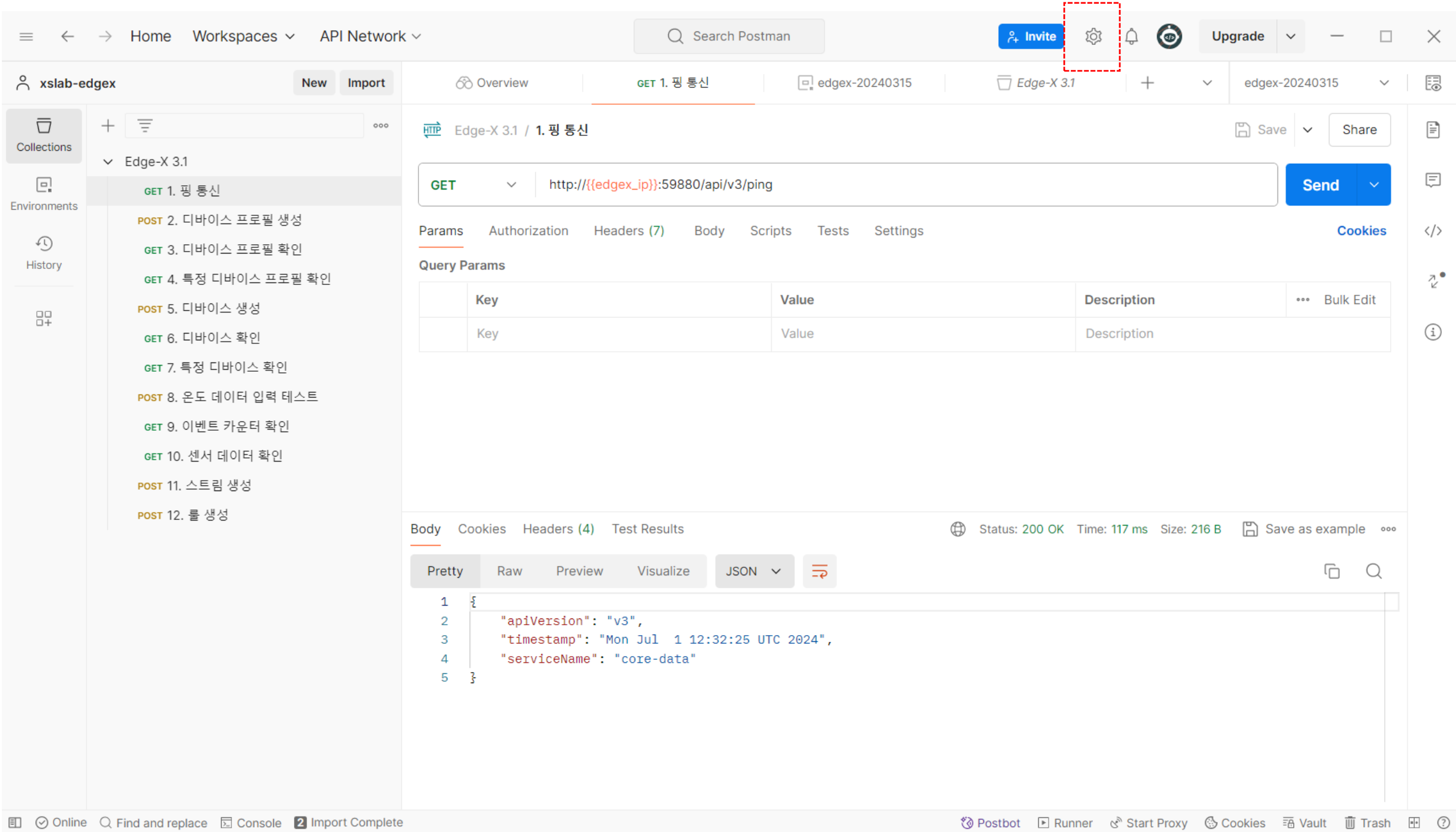
```
/edgex$ sudo install -m 0755 -d /etc/apt/keyrings
```

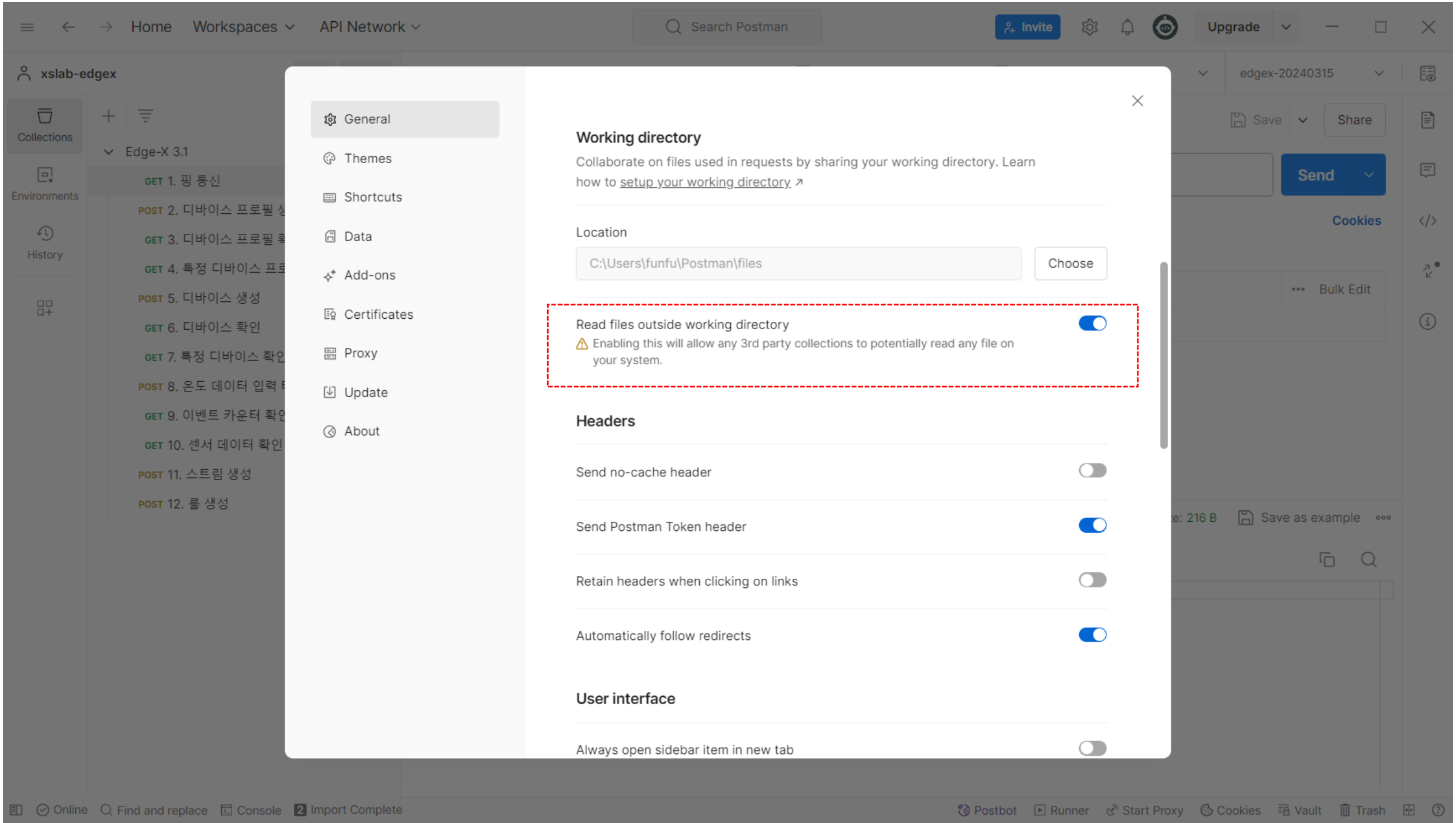
```
INSTALL(1)                                  User Commands                                  INSTALL(1)

NAME
       install - copy files and set attributes

SYNOPSIS
       install [OPTION]... [-T] SOURCE DEST
       install [OPTION]... SOURCE... DIRECTORY
       install [OPTION]... -t DIRECTORY SOURCE...
       install [OPTION]... -d DIRECTORY...

DESCRIPTION
       This install program copies files (often just compiled) into destination locations you choose.  If you want to
       download and install a ready-to-use package on a GNU/Linux system, you should instead be using a package  man-
       ager like yum(1) or apt-get(1).

       In  the  first three forms, copy SOURCE to DEST or multiple SOURCE(s) to the existing DIRECTORY, while setting
       permission modes and owner/group.  In the 4th form, create all components of the given DIRECTORY(ies).

       Mandatory arguments to long options are mandatory for short options too.
```

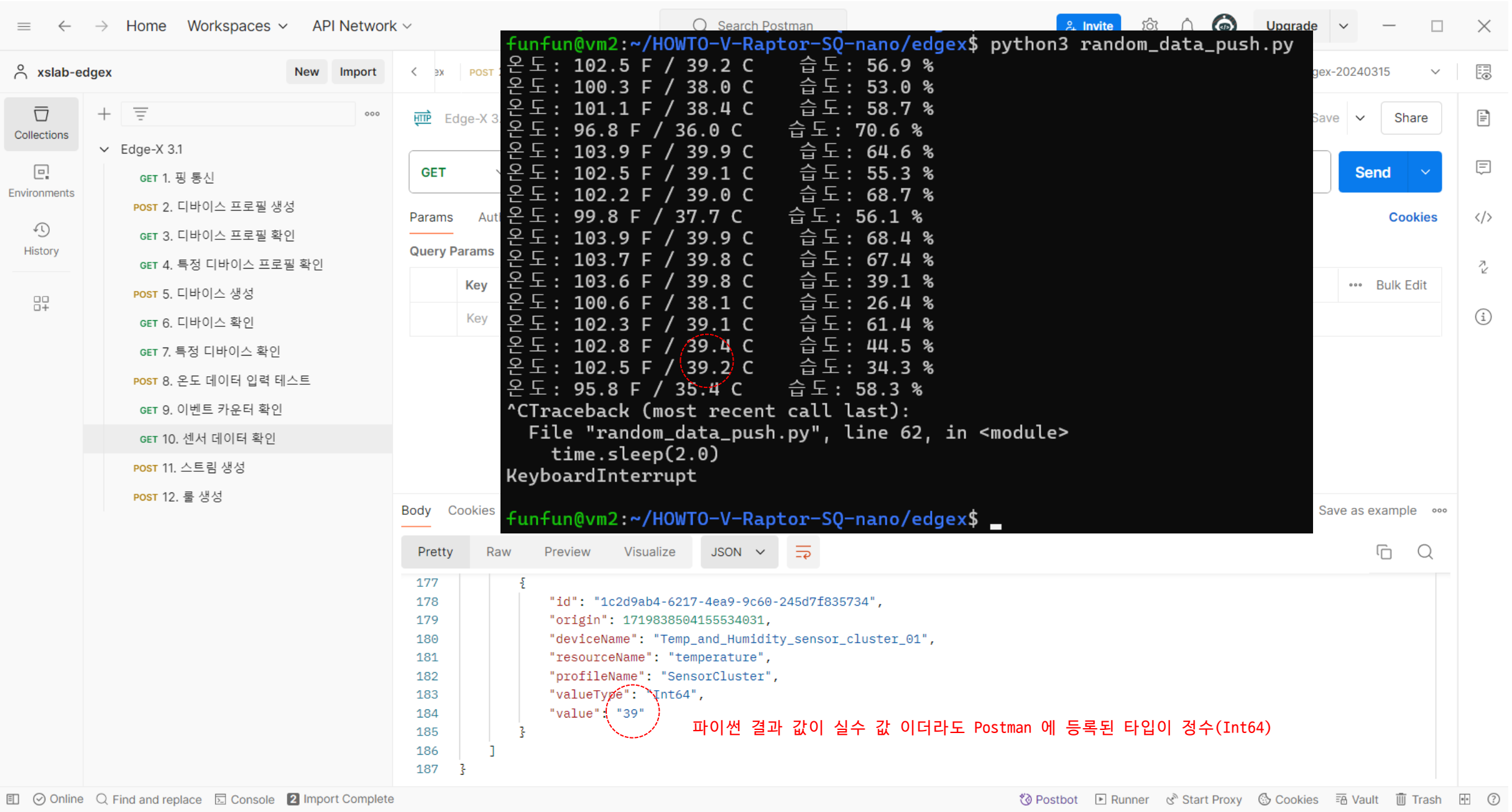
mkdir 명령어가 아닌 install 명령어를 사용하는 이유는, mkdir 의 경우 디렉토리 생성시 시스템 디폴트 권한(644)으로 생성되기에 권한을 변경하려면

디렉토리 생성 후 다시 권한 변경 명령어를 실행해야 되는 두번의 명령어 실행이 필요하지만, install 명령어는 옵션으로 한번의 실행으로 가능하기 때문에!


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
xslab-edgex
New
Import
Overview
GET 1. 핑 통신
edgex-20240315
Edge-X 3.1
edgex-20240315
Collections
Environments
History
Edge-X 3.1
GET 1. 핑 통신
POST 2. 디바이스 프로필 생성
GET 3. 디바이스 프로필 확인
GET 4. 특정 디바이스 프로필 확인
POST 5. 디바이스 생성
GET 6. 디바이스 확인
GET 7. 특정 디바이스 확인
POST 8. 온도 데이터 입력 테스트
GET 9. 이벤트 카운터 확인
GET 10. 센서 데이터 확인
POST 11. 스트림 생성
POST 12. 룰 생성
Edge-X 3.1 / 1. 핑 통신
Save
Share
GET
http://{{edgex_ip}}:59880/api/v3/ping
Send
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 117 ms
Size: 216 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"apiVersion": "v3",
"timestamp": "Mon Jul 1 12:32:25 UTC 2024",
"serviceName": "core-data"
}
Online
Find and replace
Console
Import Complete
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
xslab-edgex
New
Import
Overview
GET 1. 핑 통신
edgex-20240315
Edge-X 3.1
edgex-20240315
Collections
Environments
History
Edge-X 3.1
GET 1. 핑 통신
POST 2. 디바이스 프로필 생성
GET 3. 디바이스 프로필 확인
GET 4. 특정 디바이스 프로필 확인
POST 5. 디바이스 생성
GET 6. 디바이스 확인
GET 7. 특정 디바이스 확인
POST 8. 온도 데이터 입력 테스트
GET 9. 이벤트 카운터 확인
GET 10. 센서 데이터 확인
POST 11. 스트림 생성
POST 12. 룰 생성
Edge-X 3.1 / 1. 핑 통신
Save
Share
GET
http://{{edgex_ip}}:59880/api/v3/ping
Send
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (4)
Test Results
Status: 200 OK
Time: 117 ms
Size: 216 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"apiVersion": "v3",
"timestamp": "Mon Jul 1 12:32:25 UTC 2024",
"serviceName": "core-data"
}
Online
Find and replace
Console
Import Complete
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
xslab-edgex
edgex-20240315
Collections
Environments
History
Edge-X 3.1
GET 1. 핑 통신
POST 2. 디바이스 프로필
GET 3. 디바이스 프로필
GET 4. 특정 디바이스 프로
POST 5. 디바이스 생성
GET 6. 디바이스 확인
GET 7. 특정 디바이스 확인
POST 8. 온도 데이터 입력
GET 9. 이벤트 카운터 확인
GET 10. 센서 데이터 확인
POST 11. 스트림 생성
POST 12. 룰 생성
Save
Share
Send
Cookies
Bulk Edit
Save as example
General
Themes
Shortcuts
Data
Add-ons
Certificates
Proxy
Update
About
Working directory
Collaborate on files used in requests by sharing your working directory. Learn how to setup your working directory
Location
C:\Users\funfu\Postman\files
Choose
Read files outside working directory
Enabling this will allow any 3rd party collections to potentially read any file on your system.
Headers
Send no-cache header
Send Postman Token header
Retain headers when clicking on links
Automatically follow redirects
User interface
Always open sidebar item in new tab
Online
Find and replace
Console
Import Complete
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
xslab-edgex
New
Import
Collections
Environments
History
Edge-X 3.1
GET 1. 핑 통신
POST 2. 디바이스 프로필 생성
GET 3. 디바이스 프로필 확인
GET 4. 특정 디바이스 프로필 확인
POST 5. 디바이스 생성
GET 6. 디바이스 확인
GET 7. 특정 디바이스 확인
POST 8. 온도 데이터 입력 테스트
GET 9. 이벤트 카운터 확인
GET 10. 센서 데이터 확인
POST 11. 스트림 생성
POST 12. 룰 생성
GET
Send
Save
Share
Params
Query Params
Key
Bulk Edit
Cookies
Body
Cookies
Save as example
Pretty
Raw
Preview
Visualize
JSON
funfun@vm2:~/HOWTO-V-Raptor-SQ-nano/edgex$ python3 random_data_push.py
온도: 102.5 F / 39.2 C 습도: 56.9 %
온도: 100.3 F / 38.0 C 습도: 53.0 %
온도: 101.1 F / 38.4 C 습도: 58.7 %
온도: 96.8 F / 36.0 C 습도: 70.6 %
온도: 103.9 F / 39.9 C 습도: 64.6 %
온도: 102.5 F / 39.1 C 습도: 55.3 %
온도: 102.2 F / 39.0 C 습도: 68.7 %
온도: 99.8 F / 37.7 C 습도: 56.1 %
온도: 103.9 F / 39.9 C 습도: 68.4 %
온도: 103.7 F / 39.8 C 습도: 67.4 %
온도: 103.6 F / 39.8 C 습도: 39.1 %
온도: 100.6 F / 38.1 C 습도: 26.4 %
온도: 102.3 F / 39.1 C 습도: 61.4 %
온도: 102.8 F / 39.4 C 습도: 44.5 %
온도: 102.5 F / 39.2 C 습도: 34.3 %
온도: 95.8 F / 35.4 C 습도: 58.3 %
^CTraceback (most recent call last):
File "random_data_push.py", line 62, in <module>
time.sleep(2.0)
KeyboardInterrupt
funfun@vm2:~/HOWTO-V-Raptor-SQ-nano/edgex$
177 {
178 "id": "1c2d9ab4-6217-4ea9-9c60-245d7f835734",
179 "origin": 1719838504155534031,
180 "deviceName": "Temp_and_Humidity_sensor_cluster_01",
181 "resourceName": "temperature",
182 "profileName": "SensorCluster",
183 "valueType": "Int64",
184 "value": "39"
185 }
186]
187 }
파이썬 결과 값이 실수 값 이더라도 Postman 에 등록된 타입이 정수(Int64)
Online
Find and replace
Console
Import Complete
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

```
funfun@vm2:~$ MASTER_IP=`hostname -I | awk '{print $1}'`
funfun@vm2:~$ echo $MASTER_IP
192.168.3.182
funfun@vm2:~$ sudo kubeadm init --apiserver-advertise-address="${MASTER_IP}" --pod-network-cidr "10.244.0.0/16"
[sudo] password for funfun:
[init] Using Kubernetes version: v1.30.2
[preflight] Running pre-flight checks
[preflight] Pulling images required for setting up a Kubernetes cluster
[preflight] This might take a minute or two, depending on the speed of your internet connection
[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'
W0701 13:26:40.708741    1911 checks.go:844] detected that the sandbox image "registry.k8s.io/pause:3.8" of the containe
r runtime is inconsistent with that used by kubeadm.It is recommended to use "registry.k8s.io/pause:3.9" as the CRI sand
box image.
```

```
[addons] Applied essential addon: kube-proxy

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/

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

kubeadm join 192.168.3.182:6443 --token ojhe3v.q1sxtu0eesenajon \
        --discovery-token-ca-cert-hash sha256:f72c9f0f90737b2ed21bb9af723599a68f2f05b2a8216d0a32bf83e77975363d
funfun@vm2:~$
funfun@vm2:~$ mkdir -p $HOME/.kube
funfun@vm2:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
funfun@vm2:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

```
funfun@vm3:~$ sudo kubeadm join 192.168.3.182:6443 --token ojhe3v.q1sxtu0eesenajon \
>         --discovery-token-ca-cert-hash sha256:f72c9f0f90737b2ed21bb9af723599a68f2f05b2a8216d0a32bf83e77975363d
[sudo] password for funfun:
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 3.507373296s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

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.

funfun@vm3:~$
```

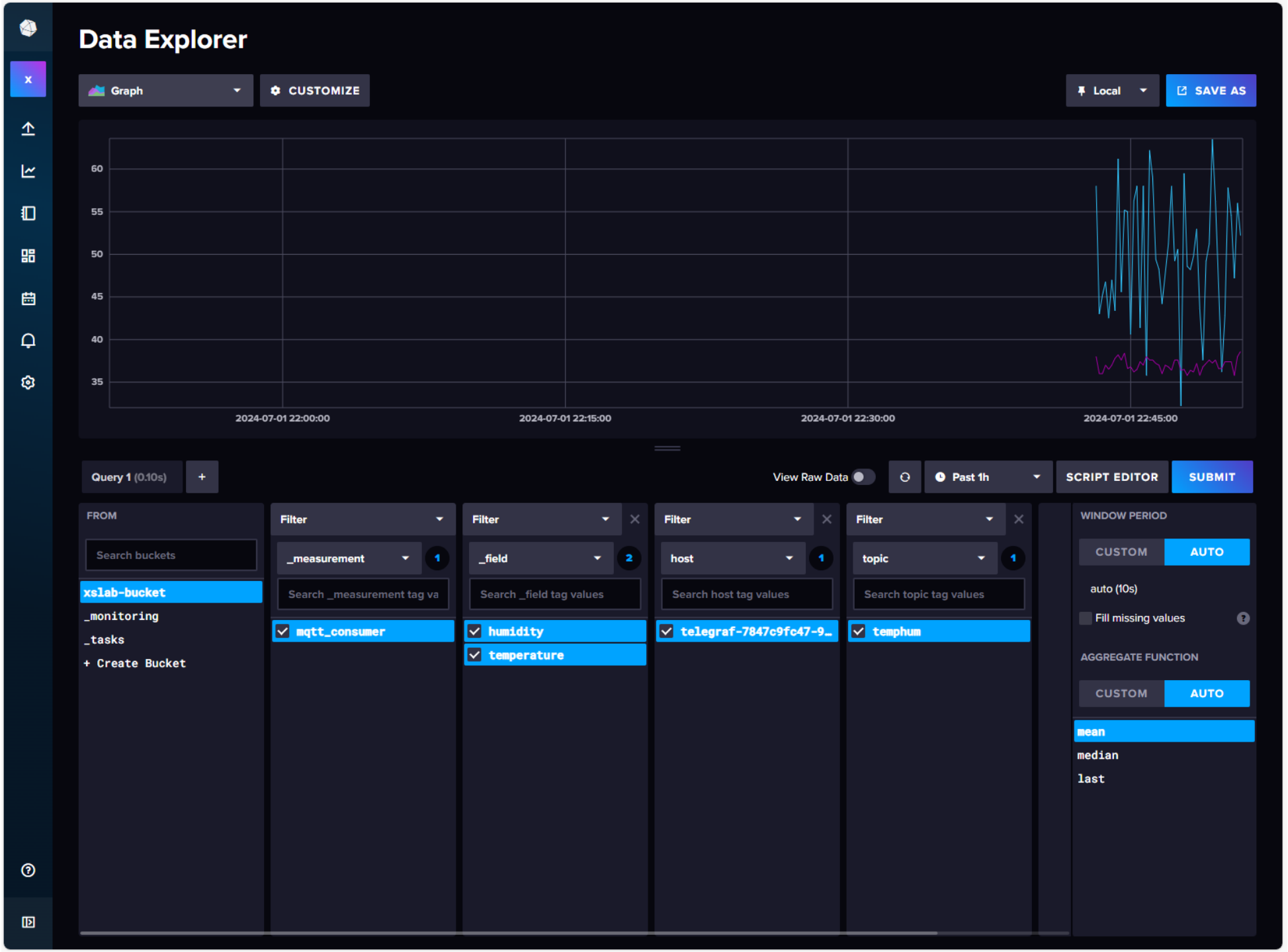
```
funfun@vm2:~$ kubectl get nodes
NAME   STATUS     ROLES           AGE   VERSION
vm2    NotReady   control-plane   2m    v1.30.2
vm3    NotReady   <none>          15s   v1.30.2
funfun@vm2:~$
```

```
funfun@vm4:~$ sudo kubeadm join 192.168.3.182:6443 --token ojhe3v.q1sxtu0eesenajon \
>         --discovery-token-ca-cert-hash sha256:f72c9f0f90737b2ed21bb9af723599a68f2f05b2a8216d0a32bf83e77975363d
[sudo] password for funfun:
[preflight] Running pre-flight checks
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet. This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 7.006241561s
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

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.

funfun@vm4:~$
```

```
funfun@vm2:~$ kubectl get nodes
NAME   STATUS     ROLES           AGE     VERSION
vm2    NotReady   control-plane   4m57s   v1.30.2
vm3    NotReady   <none>          3m12s   v1.30.2
vm4    NotReady   <none>          2m27s   v1.30.2
```

```
funfun@vm2:~/HOWTO-V-Raptor-SQ-nano/k8s$ kubectl get all --all-namespaces
NAMESPACE     NAME                                              READY   STATUS     RESTARTS   AGE
kube-system   pod/calico-kube-controllers-564985c589-6cmnz      0/1     Pending    0          21s
kube-system   pod/calico-node-55l7k                             0/1     Init:0/3   0          21s
kube-system   pod/calico-node-5pzzc                             0/1     Init:0/3   0          21s
kube-system   pod/calico-node-ldgqd                             0/1     Init:1/3   0          21s
kube-system   pod/coredns-7db6d8ff4d-5lxhh                      0/1     Pending    0          5m54s
kube-system   pod/coredns-7db6d8ff4d-7tqqb                      0/1     Pending    0          5m54s
kube-system   pod/etcd-vm2                                      1/1     Running    0          6m5s
kube-system   pod/kube-apiserver-vm2                            1/1     Running    0          6m5s
kube-system   pod/kube-controller-manager-vm2                   1/1     Running    0          6m5s
kube-system   pod/kube-proxy-4km96                              1/1     Running    0          4m24s
kube-system   pod/kube-proxy-5hj5d                              1/1     Running    0          3m39s
kube-system   pod/kube-proxy-p6www                              1/1     Running    0          5m54s
kube-system   pod/kube-scheduler-vm2                            1/1     Running    0          6m5s

NAMESPACE     NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
default       service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP                  6m11s
kube-system   service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   6m7s

NAMESPACE     NAME                         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
kube-system   daemonset.apps/calico-node   3         3         0       3            0           kubernetes.io/os=linux   22s
kube-system   daemonset.apps/kube-proxy    3         3         3       3            3           kubernetes.io/os=linux   6m7s

NAMESPACE     NAME                                   READY   UP-TO-DATE   AVAILABLE   AGE
kube-system   deployment.apps/calico-kube-controllers   0/1     1            0           22s
kube-system   deployment.apps/coredns                0/2     2            0           6m7s

NAMESPACE     NAME                                              DESIRED   CURRENT   READY   AGE
kube-system   replicaset.apps/calico-kube-controllers-564985c589   1         1         0       22s
kube-system   replicaset.apps/coredns-7db6d8ff4d                2         2         0       5m55s
funfun@vm2:~/HOWTO-V-Raptor-SQ-nano/k8s$
```

```
funfun@vm2:~/HOWTO-V-Raptor-SQ-nano/edgex$ kubectl get all
NAME                              READY   STATUS      RESTARTS   AGE
pod/influxdb-0                    1/1     Running     0          11m
pod/influxdb-setup-282fl          0/1     Completed   0          11m
pod/rabbitmq-5fd5946d89-4skbn     1/1     Running     0          13m
pod/telegraf-7847c9fc47-98s8f     1/1     Running     0          10m

NAME                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                                        AGE
service/influxdb     NodePort    10.101.127.111   <none>        8086:30201/TCP                                 11m
service/kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP                                        21m
service/rabbitmq     NodePort    10.111.169.85    <none>        1883:30101/TCP,5672:30102/TCP,15672:30103/TCP   13m
service/telegraf     NodePort    10.97.20.88      <none>        8125:30202/TCP                                 10m

NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/rabbitmq     1/1     1            1           13m
deployment.apps/telegraf     1/1     1            1           10m

NAME                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/rabbitmq-5fd5946d89     1         1         1       13m
replicaset.apps/telegraf-7847c9fc47     1         1         1       10m

NAME                          READY   AGE
statefulset.apps/influxdb     1/1     11m

NAME                          STATUS     COMPLETIONS   DURATION   AGE
job.batch/influxdb-setup      Complete   1/1           101s       11m
funfun@vm2:~/HOWTO-V-Raptor-SQ-nano/edgex$
```


Data Explorer
Graph
CUSTOMIZE
Local
SAVE AS
60
55
50
45
40
35
2024-07-01 22:00:00
2024-07-01 22:15:00
2024-07-01 22:30:00
2024-07-01 22:45:00
Query 1 (0.10s)
View Raw Data
Past 1h
SCRIPT EDITOR
SUBMIT
FROM
Search buckets
xslab-bucket
_monitoring
_tasks
+ Create Bucket
Filter
_measurement
1
Search _measurement tag va
mqtt_consumer
Filter
_field
2
Search _field tag values
humidity
temperature
Filter
host
1
Search host tag values
telegraf-7847c9fc47-9_
Filter
topic
1
Search topic tag values
temphum
WINDOW PERIOD
CUSTOM
AUTO
auto (10s)
Fill missing values
AGGREGATE FUNCTION
CUSTOM
AUTO
mean
median
last