Lambda, Cloudwatch를 활용하여 ec2 stop/start 스케줄 설정하기

Lambda 란 ?

서버리스 컴퓨팅 서비스로, 애플리케이션을 실행하기 위해 별도의 서버 셋업 없이 곧바로 코드를 실행해주는 서비스 이다.
초 단위로 비용을 계산하는 ec2와는 별개로 Lambda는 1ms 당 요금을 계산해 정확히 사용한 만큼만 비용이 발생한다.

1. Lambda 에 붙일 IAM 정책 및 역할 생성

**정책 및 역할을 연결하지 않고 Lambda를 바로 생성했을 경우에는 정책 및 역할이 자동 생성된다.

[Policy]

정책 이름 : Lambda_policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ec2:Describe*",
                "ec2:Start*",
                "ec2:Stop*"
            ],
            "Resource": "*"
        }
    ]
}
```

[Role]

역할 이름 : Lambda_role

생성한 policy(Lambda_policy)추가

2. Lambda 생성

```
Auther from scratch 선택
Lambda Function 이름 : StartEC2Instance / StopEC2Instance
Runtime : Python3.8
Permissions - Use an existing role - 생성해두었던 role 선택
```

[Lambda Code]

```
import boto3
region = 'ap-northeast-2'
instances = []
ec2_r = boto3.resource('ec2')
ec2 = boto3.client('ec2', region_name=region)
//
for instance in ec2_r.instances.all():
    instances.append(instance.id)

def lambda_handler(event, context):
    ec2.start_instances(InstanceIds=instances)
    print('started your instances: ' + str(instances))
```

태그를 설정하여 특정 ec2에만 스크립트 적용될 수 있도록 할수도 있습니다.

```
import boto3
region = 'ap-northeast-2'
instances = []
ec2_r = boto3.resource('ec2')
ec2 = boto3.client('ec2', region_name=region)

for instance in ec2_r.instances.all():
    for tag in instance.tags:
        if tag['Key'] == 'auto-schedule':
            if tag['Value'] == 'auto':
                instances.append(instance.id)

def lambda_handler(event, context):
    ec2.start_instances(InstanceIds=instances)
    print('start your instances: ' + str(instances))
```

정상적으로 스크립트가 구동되는지 테스트를 해봅시다.

```
Lambda - Deploy - Test
```

구동이 제대로 안된다면,
1) IAM Policy, Lambda code 확인
2) Lambda-General configuration - Timeout 을 3(default) -> 15로 변경.
3) 지정한 region에 있는 ec2의 Instance state가 1EA라도 terminate 되어있는 상태라면 스크립트가 구동되지 않음.

3. CloudWatch 스케줄링 등록

Event - Rules - Go to Amazon EventBridge

step1)
규칙 이름 : StartEC2Instance
Event bus : default
Rule type : Schedule

step2)
GMT 기준 cron 표현식으로 원하는 시간 지정

step3)
target types : AWS service
Select a target : Lambda function
Function : Lambda Function 선택 (StartEC2Instance)
++ 타겟 추가 가능

step4)
태그 설정

(선택) 4. Lmabda code 성공/실패 메일 알람

[SNS]

topic 생성
subscription 생성
    Protocol : Email
    Endpoint : 수신받을 이메일 지정

[Lambda]

Configuration - Destinations - Add destination
source : Asynchronous invocation
Condition : On Failure / On Success
Destination Type : SNS topic
Destination : 생성한 topic 선택

CentOS 7 설치 후 할 일 – 필수 패키지 및 권장 패키지 설치 (2)

CentOS 7 패키지 설치

CentOS 7 기본 설정 후 서버 관리 및 운영을 도와주는 몇 가지 패키지들을 설치해주어야 한다.

해당 포스트에서는 필수 패키지 / 권장 패키지로 나누어 설명하도록 하겠다.

1. 패키지 업데이트

커널 및 각종 패키지들이 업데이트 되며 서버의 구성환경을 고려하여 진행한다.

```
## 패키지명을 지정하지 않을경우 설치되어있는 모든 패키지를 업데이트 한다.
yum update [패키지명]
## 설치된 패키지 중 업데이트된 패키지 목록 확인
yum list updates
```

2. 필수 패키지

일반적으로 설치해야하는 패키지들과 서버 모니터링을 위한 패키지로 구분하여 설명한다.

일반 패키지

epel-release : 각종 패키지의 최신 버전을 사용하기 위해 설치
해당 패키지 설치후 다른 패키지 설치시에 최신 버전으로 설치되므로 disable 설정 후 enablerepo 옵션 사용

vi /etc/yum.repos.d/epel.repo
enabled=1 을 0으로

[epel]
...
enabled=0
...

yum repolist
yum --enablerepo=epel install [패키지명]

net-tools : 각종 네트워크 명령어들을 관리하는 패키지로 IP 확인을 위한 ifconfig, 네트워크 연결 상태 확인을 위한 netstat 둘을 포함한 명령어를 사용하기 위해 설치한다.

<ifconfig>
ifconfig -a
ifconfig [interface] up
ifconfig [interface] down

<netstat>
netstat -nap
netstat -an | grep [Port]
netstat -nlpt

unzip : zip으로 압축된 파일을 풀기 위해 사용한다.

unzip [file_name].zip
unzip -l [file_name].zip
unzip -t [file_name].zip

wget : 웹 서버로부터 콘텐츠를 가져오는 명령어로 HTTP, HTTPS, FTP 프로토콜클 지원한다.

wget -O [파일 이름] [URL 주소]

wget --no-check-certificate [URL 주소]

curl : 커맨드 라인용 데이터 전송 도구로 HTTP, FTP등 다양한 프로토콜을 지원한다.
Web 파일 다운받을 경우 wget 권장

curl -o [파일 이름] [URL 주소]
curl -T [파일 이름] [서버 IP]
curl -L [URL 주소]

chrony : NTP를 구현한 Server/Client로 ntpd를 대체하며 서버의 시간을 동기화한다.

동기화 시킬 NTP 서버의 IP 추가
vi /etc/chrony.conf

server [NTP server IP] iburst <iburst = 동기화 시간을 단축 시켜주는 옵션>

systemctl restart chronyd
chronyc sources

gcc, gcc-c++ : 각각 C언어, C++언어 컴파일러로 서버에 개발환경을 구축하기 위해 설치한다.

openssl-devel : openssl을 설치하기 위한 의존성 패키지로 openssl을 사용할 애플리케이션 개발용 파일이다.

서버 모니터링 관련 패키지

htop : 리눅스에서 사용중인 리소스등을 확인 (윈도우의 작업 관리자와 비슷한 역할을 함)

프로세스

PID

USER
PR
NI
VIRT
RES
SHR
S
%CPU
%MEM
TIMR+
COMMAND

iftop : 네트워크 인터페이스의 트래픽 모니터링

```
iftop -i eno1
iftop -f "dst port 22"
```

dstat : 실시간으로 디스크별 I/O 사용량 체크 및 부하 유발을 주는 프로세스를 쉽게 확인 가능하다.

```
dstat -tcdml
```

sysstat : 리눅스 성능 측정 도구 패키지로 sar, iostat 등을 사용하기 위해 설치

<iostat> : CPU와 디스크의 처리량, 대기열 길이등을 실시간으로 모니터링 한다.

```
iostat -d 3
iostat -c 3
```

<sar> : 과거 모니터링 정보를 확인 /var/log/sa 폴더에 기록된 sa파일들을 조회한다.

```
sar -u
sar -r
sar -dp
sar -n DEV
```

lsof : 시스템에서 열려있는 파일에 대한 정보를 출력

lsof -u [사용자 이름]
lsof -i
lsof -c [서비스 이름]

psmisc : proc 파일 시스템을 사용하는 유틸리티 패키지로 fuser, killall, pstree등을 사용

<fuser> : 특정 파일시스템을 umount 할 때 어떠한 프로세스 때문에 진행이 안될 경우, 그 프로세스를 kill 하기 위해 사용
fuser -v [디렉터리]
fuser -ck [디렉터리]
<killall> : 특정 프로세스를 한 번에 죽이기 위해 사용
killall -i [프로세스]
killall -v [프로세스]
killall -w [프로세스]

<pstree> : 프로세스의 상태를 Tree 구조로 출력하여 확인
pstree -anp

3. 권장 패키지

mlocate : 파일을 검색하기 위해 find 명령어 대신 데이터베이스 기반으로 파일을 검색하기 때문에 빠른 검색을 위해 사용한다.

updatedb
locate [파일 이름]
locate -n [숫자] [파일 이름]

ncat : 서버와 클라이언트가 소켓 통신을 하기위해 사용한다.

<서버>
ncat -l [Port]
ncat -lk [Port]

<클라이언트>
ncat [Server IP] [Port]

whois : 도메인 소유자나 IP 주소 위치 등 이외 다른 정보를 확인할 때 사용한다.

whois [도메인 주소]
whois [IP 주소]

cloud-utils-growpart : LVM 구성이 아닌 상태에서 root 파티션 용량을 추가하기 위해 사용한다.

growpart [디스크 명] [파티션 번호]
resize2fs [파티션 경로]

tcping : 클라이언트에서 서버의 TCP 포트를 ping 유틸리티로 체크하기 위해 사용한다.

tcping [Server IP] [Port]

About OOM Killer ?

본 글은 Kernel 5.4.0-104-generic 을 기반으로 작성하였습니다.

OOM(Out Of Memory) 이란 ?

Linux의 swap 메모리와 물리 메모리를 모두 사용 중 일 때, 프로세스가 추가적인 물리 메모리 할당을 요청 하여도 할당이 불가능 한 상태를 의미한다.
주로 Over Commit 상태일 때 해당 현상이 동반된다.

OOM(Out Of Memory) Killer 란?

Linux 의 메모리 부족으로 Out Of Memory가 발생 하였을 때, OOM Score 를 기준으로 점수가 높은 프로세스를 Kill 하여 메모리 확보를 하는 Linux Kernel 기능이다.
OOM Killer 로 인한 프로세스 강제 종료 시 /var/log/ 내부의 시스템 로그에서 확인 할 수 있다.

oom_killer 로 인한 프로세스 종료 Log

```
$ cat /var/log/syslog | grep oom
 Mar  7 19:14:00 zabbix-node01 kernel: [1132818.054201] ib_log_writer invoked oom-killer: gfp_mask=0x100cca(GFP_HIGHUSER_MOVABLE), order=0, oom_score_adj=0
```

OOM Score 점수 산정 및 OOM Killer 구동의 직접적인 영향을 주는 요인을 알아보자.

1. 프로세스에 할당 된 메모리 양 + 프로세스로 부터 fork() 된 자식 프로세스의 메모리 양
2. 프로세스의 동작 중 인 시간
3. 프로세스가 하드웨어에 직접 접근하거나, root계정 (super user) 에 의해 실행 된 프로세스
4. 프로세스의 nice 값이 1 이상일 경우 Score 점수 2배 증가
5. /proc/[PID]/oom_score_adj 값 (유저가 직접 변경 가능)
6. /proc/[PID]/oom_adj 값 (유저가 직접 변경 가능)

프로세스의 OOM Score를 확인하는 방법은 다음과 같다.

```
# oom_score 확인 방법
$ cat /proc/890081/oom_score
1048
```

890081은 PID 이며, 현재 OOM Score는 1048이다.

oom_adj / oom_score_adj 에 대한 부연 설명

종료되면 안되는 주요 프로세스가 OOM Score 가 높아 OOM Killer에 의한 종료가 예상 될 때에는, oom_adj /oom_score_adj 값의 조절을 통하여, OOM Score 조절이 가능하다.

oom_adj 의 범위는 -17 ~ 15 이며, -17로 지정 시 OOM Killer Disable과 같은 의미를 가진다.

oom_score_adj의 범위는 -1000 ~ 1000 이며, 현재 OOM Score 에서 oom_score_adj 값을 가감한다.(oom_scoer_adj 의 -1000은 oom_adj 의 -17과 같다.)

oom_adj / oom_score_adj 의 값을 변경 시 변경 한 값을 참조하여 oom_score의 재 산정이 이루어진다.

프로세스의 OOM Score 확인 및 oom_score_adj 를 통한 OOM Score 조절은 아래 예를 참고하자.

```
# 현재 oom_score 확인
$ cat /proc/890081/oom_score
1048
```

890081은 PID 이며, 현재 OOM Score는 1048이다.

```
# oom_score_adj 를 통한 OOM Score 조절
$ echo -1000 > /proc/890081/oom_score_adj

$ cat /proc/890081/oom_score_adj
-1000
# oom_score_adj 값에 가장 낮은 값인 -1000 으로 설정

# 설정 후 oom_score / oom_adj 확인
$ cat /proc/890081/oom_score
0
$ cat /proc/890081/oom_adj
-17
# oom_score_adj 설정 후 확인 결과, oom_adj 는 -17 (OOM Killer Disable)
# oom_score는 1048 -> 0 으로 감소한 것을 확인할 수 있다.
```

추가적으로, overcommit에 대한 설정을 할 수 있다. over commit의 설정 방법은 다음과 같다.

over commit 설정

```
$ cat /etc/sysctl.conf | grep overcommit_memory
vm.overcommit_memory = 1 # 0~2 의 설정값을 가진다.
# 0 = Heuristic overcommit. Default 설정 값이며, 휴리스틱하게(유동적이게) 커널 자체적으로 over commit 비율을 조정한다.
# 1 = 항상 over commit을 허용한다. OOM Killer가 동작하지 않으나, 프로세스의 메모리할당에 실패 할 수 있다.
# 2 = vm.overcommit_ratio 의 값을 참조하여 over commit 비율을 조정한다.

# 사용 가능한 물리 메모리의 백분율
$ cat /etc/sysctl.conf | grep overcommit_ratio
vm.overcommit_ratio = 90
# 해당 설정은 vm.overcommit_memory가 2일때 의미를 가진다.
# 90%의 물리메모리 사용 + swap 메모리 사용을 초과하게 되면 OOM Killer가 동작한다.

# 설정 적용
$ systemctl -w
```

그렇다면, 현재 commit 되고있는 메모리와 허용가능한 over commit 메모리양은 어떻게 확인할까?
답은 /proc/meminfo 과 sysstat 패키지로 조회할 수 있다. 하기 예를 참고하자.

```
# commit 정보 확인
$ cat /proc/meminfo | grep Commit
CommitLimit:    30229156 kB
# vm.overcommit_memory 의 설정이 2 일 때 vm.overcommit_ratio 의 설정을 기반으로 commit 가능 한 메모리 양
Committed_AS:   64267776 kB
# 현재 commit 되고 있는 메모리 양
```

sysstat 패키지를 이용한 현재 commit 양 확인

```
$ apt install -y sysstat
# sysstat 패키지 설치

# sar 명령어를 이용한 메모리 상태 조회
$ sar -r 1
Linux 5.4.0-104-generic (zabbix-node02)         04/14/2022      _x86_64_        (16 CPU)

09:25:17 AM kbmemfree   kbavail kbmemused  %memused kbbuffers kbcached  kbcommit   %commit  kbactive   kbinact   kbdirty
09:25:18 AM    208884   2446760  20804112     85.73    990436   1441416  64262056    196.79  19538260   3304352      1508
# %commit 부분이 현재 commit된 메모리 양을 백분율로 표기해주는 부분이다.
# 현재 100% 이상 commit 되고 있으므로, over commit 상태이다.
# sar 명령어는 시스템 자원(CPU, Memory, I/O) 사용량을 수집, 레포트 하는 명령어다.
# -r 옵션은 Memory 사용량을 조회하는 옵션이며, 뒤의 1은 1초 마다 수집한다는 의미이다.
Linux Memory Commit / Memory Over Commit 이란?
Memory Commit 이란
프로세스가 커널에게 메모리 공간을 요청 하여 가상 메모리 영역 주소를 할당 받은 상태

이해를 위해 아래 예를 참고 하자.

A 프로세스가 커널에게 메모리 할당을 요청 하였다고 가정 했을 때,
커널은 이에 해당하는 가상 메모리 주소를 A 프로세스에게 할당한다.
결과적으로 A프로세스는 커널에게 메모리 영역을 할당 받았다고 생각하지만, 실제론 물리 메모리에 할당 받지 않은 상태다.
이를 Memory Commit (메모리 커밋) 이라고 한다.
그렇다면, 왜 바로 물리 메모리 영역에 프로세스를 할당하지 않고, 가상 영역에 할당을 할까?
```

물리 메모리 영역에 프로세스를 바로 할당하게 되면 몇가지 문제점이 있기 때문이다.
프로세스의 메모리 주소 할당을 바로 물리 메모리 주소로 할당 하면 생길 수 있는 문제점은 다음과 같다.

1. A 프로세스가 메모리를 할당받고 사용하지 않을 수 있다.
2. 이미 사용중인 메모리 주소값으로 접근 할 수 있는 위험성이 있다.
 - 예로, 이미 커널이 사용 중 인 물리 메모리 주소가 반환될 경우 매우 위험
3. 메모리 단편화(Fragmentation)가 발생한다.
 - 메모리 단편화 : RAM 에서 메모리 공간이 작은 공간으로 나뉘어 사용가능한 메모리가 존재하지만 할당이 불가능한 상태
 - 메모리 단편화 발생 사유 및 해결방법에 대한 자세한 내용은 다음에 다루도록 한다.

Memory Over Commit이란
말 그대로, 리눅스의 실제 물리 Memory보다 Over(초과) 하여 Commit 된 상태를 의미한다.
결론적으로 over commit이 발생한다고 하더라도, 시스템에 문제가 생기진 않는다.
하지만, over commit의 양이 지속적으로 증가한다면 oom killer로 중요 프로세스가 갑자기 종료될 수 있다.

그러니 시스템의 설정과 프로세스가 요구하는 메모리를 잘 파악하고 메모리 누수가 없는지 주기적인 점검이 필요하다.

[CKA] #2. Pod - 1

[CKA] #2. Pod (1)

※ 본 글은 작성 후에도 기재 순서의 변경 및 내용 업데이트 예정

CKA 시험은 오픈북 형식으로 크롬에서 몇 가지 URL 를 허용해준다.
대부분의 커맨드 및 설명은 공식문서에 있으나, 시험 중에 바로바로 찾으려면 평소에 숙달이 되어야 한다
주요 명령어의 북마크 등을 미리 설정해 놓는 것도 좋을 것 같다.

Pod ?

Kubernetes 에서 최소 배포 단위
최소 옵션으로 생성 시, Pod 라는 하나의 공간 안에 서비스 Container 와

Pod의 Network 정보를 가지고 있는 /pause 컨테이너로 기본적인 생성이 된다.
즉 Pod 라는 최소 단위 안에는, 컨테이너가 한 개 이상 및 볼륨 등의 다른 서비스도 존재할 수 있다.
외부에서는 Pod 의 Network 만 인지하고 있겠지만, /pause 가 내부 컨테이너들을 포트로 구분하는 역할을 해준다.

Pod Create

```
# 실행 전에 yaml 형식의 사전 테스트 모드인 dry run 을 항상 실행해보는 것이 좋다
$ kubectl run hello --image=nginx --dry-run=client -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: hello
  name: hello
spec:
  containers:
  - image: nginx
    name: hello
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}

# 실제 실행은 위 명령어를 | (파이프)를 사용하여 실제 적용(apply) 의 - 표준 입력으로 넘겨 사용 하는 형식도 권장한다

kubectl run hello2 --image=nginx --dry-run=client -o yaml |
kubectl apply -f -
kubectl get pods -o wide

# nodeName 을 추가하여 원하는 노드에 pod 생성이 가능하다.
이 방식은 Taint 의 스케쥴을 무시한다.
yaml 파일에 nodeName 을 추가 지정해 주면 된다.
$ kubectl run hi --image=nginx --dry-run=client -o yaml  >
hi.yaml
apiVersion: v1
kind: Pod
metadata:
```

```
  creationTimestamp: null
  labels:
    run: hi
  name: hi
spec:
  containers:
  - image: nginx
    name: hi
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
  nodeName: user-controller      ## Node 기입
status: {}

# 정의된 YAML 을 이용한 Pod 생성
$ kubectl create -f hi.yaml
$ kubectl get pods -o wide
NAME                     READY   STATUS    RESTARTS   AGE
IP               NODE              NOMINATED NODE   READINESS
GATES
hello-776c774f98-894tt   1/1     Running   0          18h
192.168.153.193   user-worker       <none>           <none>
hi                       1/1     Running   0          3m10s
192.168.136.5     user-controller   <none>           <none>
```

Pod status 확인하기

```
$ kubectl describe pod hi

# Pod 에 직접 접근해보기. -- kubectl 의 arg 와의 옵션 구분을 위해 -- 를 사용이 권장되는 추세이다
$ kubectl exec -it hi -- /bin/bash
root@hi:/#
```

어플리케이션 Pod 배포하기

```
# 다중 수량 옵션인 --replicas 를 지정할 수 있다. deployment 옵션이 커맨드에 들어간다.
$ kubectl create deployment web --image=nginx --replicas=3
deployment.apps/web created
```

```
$ kubectl get pods -o wide | grep -i web
web-76b56fd968-c2pk9     1/1     Running   0          11s
192.168.153.217   user-worker       <none>          <none>
web-76b56fd968-chr4w     1/1     Running   0          11s
192.168.136.6     user-controller   <none>          <none>
web-76b56fd968-mmdfn     1/1     Running   0          11s
192.168.153.218   user-worker       <none>          <none>
```

Pod log 확인하기

```
# Pod info 확인
kubectl describe pod hi

# Pod log 확인
kubectl logs hi

# journal Log(kubelet) 확인
sudo journalctl -u kubelet

# 하위 레벨 Log 확인.
hi POD 내의 Container 탐색.
/pause 컨테이너가 Pod의 포워딩 네트워크 정보. nignx 컨테이너가 서비스 컨테이너이다
$ sudo docker ps -a | grep -i hi
d507d8b298c3   nginx                             "/docker-
entrypoint.…"       26  hours  ago       Up  26  hours
k8s_hi_hi_default_6a1464a1-0fea-4ff8-a5c6-426afe281173_0
d8fb1a992247    k8s.gcr.io/pause:3.6           "/pause"
26     hours     ago               Up     26     hours
k8s_POD_hi_default_6a1464a1-0fea-4ff8-a5c6-426afe281173_0

# nginx Service Container info
$ sudo docker inspect d507d8b298c3
$ sudo docker logs d507d8b298c3

# Pod Network info
$ sudo docker inspect d8fb1a992247
$ sudo docker logs d8fb1a992247

# Container 레벨에서 접근해보기. 서비스 컨테이너만 접근 가능하다
$ sudo docker exec -it d507d8b298c3 ls
```

```
bin   docker-entrypoint.d   home   media  proc  sbin  tmp
boot  docker-entrypoint.sh  lib    mnt    root  srv   usr
dev   etc                   lib64  opt    run   sys   var

$ sudo docker exec -it d507d8b298c3 /bin/bash
root@hi:/#

# 컨테이너 내에 Net 관련 명령어가 없어 불편하다면 nsenter 를 사용해본다.
ip 명령이 없어서 확인이 어렵다
$ $ sudo docker exec -it d507d8b298c3 ip addr
OCI runtime exec failed: exec failed: container_linux.go:380:
starting container process caused: exec: "ip": executable file
not found in $PATH: unknown

컨테이너 PID 추출한다
$ sudo docker inspect --format '{{ .State.Pid }}' d507d8b298c3
1244489

nsenter 를 사용하여 해당 PID 를 대상으로 클라이언트(pod namespace가 위치
한) 레벨의 명령어 실행이 가능하다.
$ sudo nsenter -t 1244489 -n ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group
default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
4: eth0@if12: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc
noqueue state UP group default
     link/ether c6:3d:04:5d:80:82 brd ff:ff:ff:ff:ff:ff link-
netnsid 0
    inet 192.168.136.5/32 scope global eth0
       valid_lft forever preferred_lft forever
```

Pod delete

```
# 일반 Pod 삭제
$ kubectl delete hi
```

```
# 어플리케이션 Pod Delete. replicas 로 다중 배포한 모두가 삭제된다.
$ kubectl delete deployment web
deployment.apps "web" deleted
$ kubectl get pods -o wide | grep -i web
-
```

참고 자료

Pod : https://kubernetes.io/docs/concepts/workloads/pods/
Pod Networking : https://www.digitalocean.com/community/tutorials/how-to-inspect-kubernetes-networking