WORKLOAD API

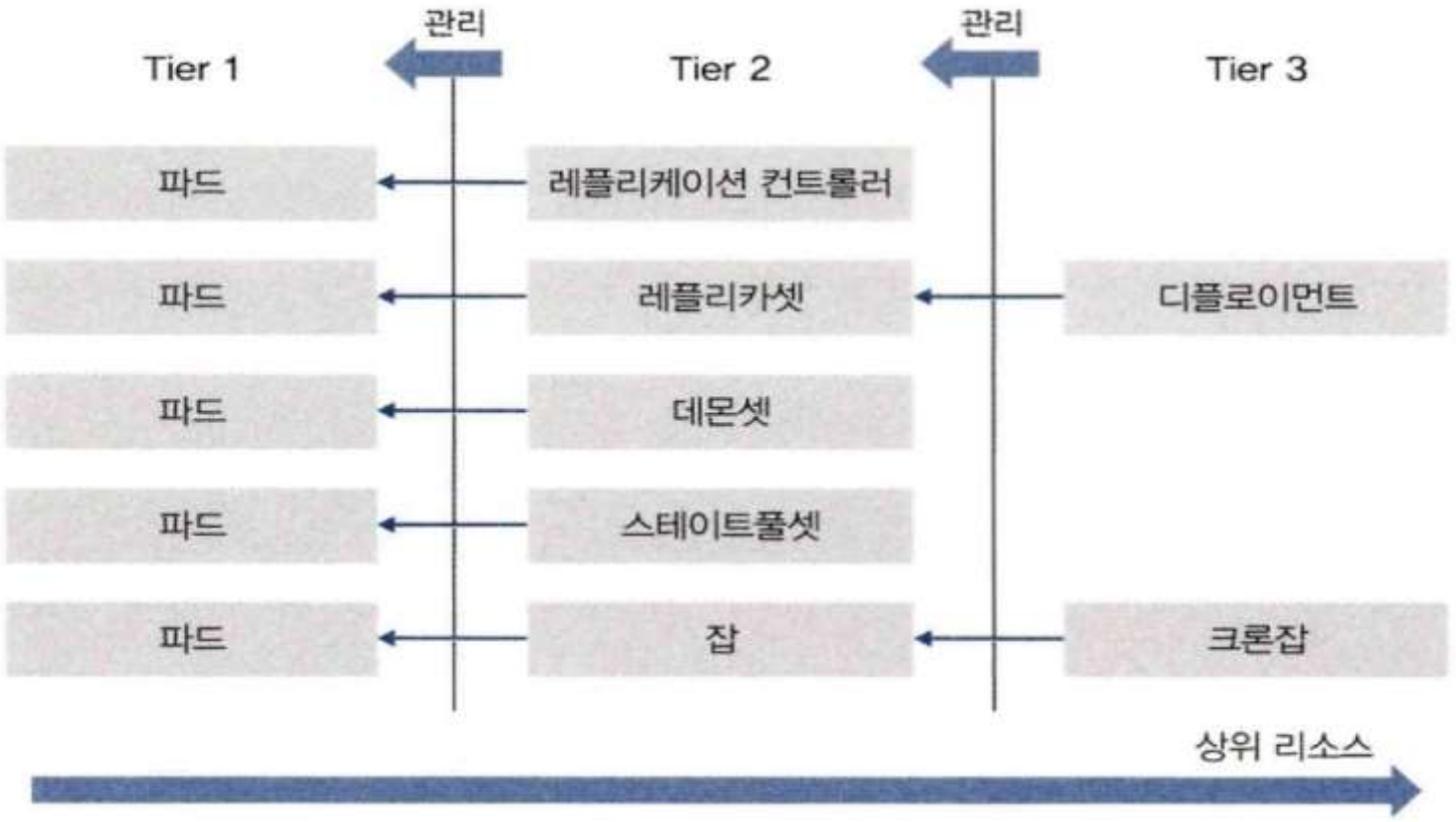
WORKLOAD API

- WORKLOAD API
  - ✓ 클러스터에 컨테이너를 기동시키기 위해 사용되는 리소스
  - ✓ 종류
    - Pod
    - ReplicaController
    - ReplicaSet
    - Deployment
    - DaemonSet
    - StatefulSet
    - Job
    - CronJob

WORKLOAD API

❖ WORKLOAD API

✓ 리소스 간의 관계

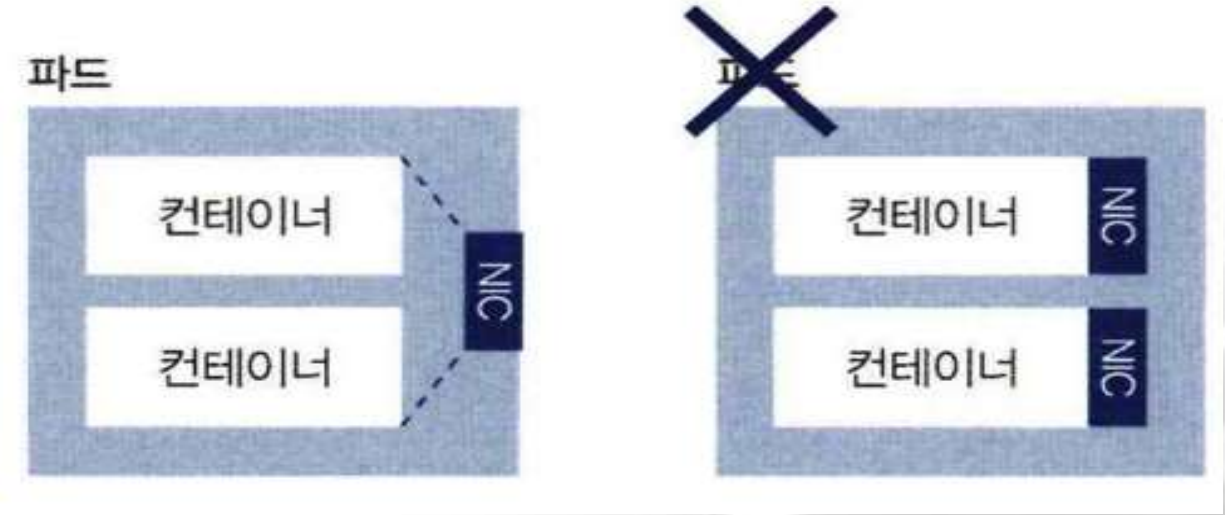


Pod

- ❖ Pod
  - ✓ 개요
    - Pod는 Kubernetes의 기본 배포(최소) 단위로 다수의 컨테이너가 모인 집합체
    - 하나의 Pod는 한 개 이상의 컨테이너로 구성되며 하나의 파드에 여러 개의 컨테이너를 포함시킬 수 있는데 이러한 구조를 사이드카(sidecar) 패턴이라고 함
    - 하나 이상의 컨테이너로 애플리케이션을 구축하다 보면 Nginx 컨테이너와 Go 애플리케이션 컨테이너처럼 서로 강한 결합을 유지하는 쪽이 더 나은 경우가 있는데 쿠버네티스에서는 이렇게 결합이 강한 컨테이너를 파드로 묶어 일괄 배포

Pod

- Pod
  - ✓ 개요
    - 같은 파드에 포함된 컨테이너끼리는 네트워크적으로 격리되어 있지 않고 IP 주소를 공유
    - 컨테이너가 두 개 들어 있는 파드를 생성한 경우 이 두 컨테이너는 같은 IP 주소를 가지며 이로 인해 파드 내부의 컨테이너는 서로 localhost로 통신할 수 있음

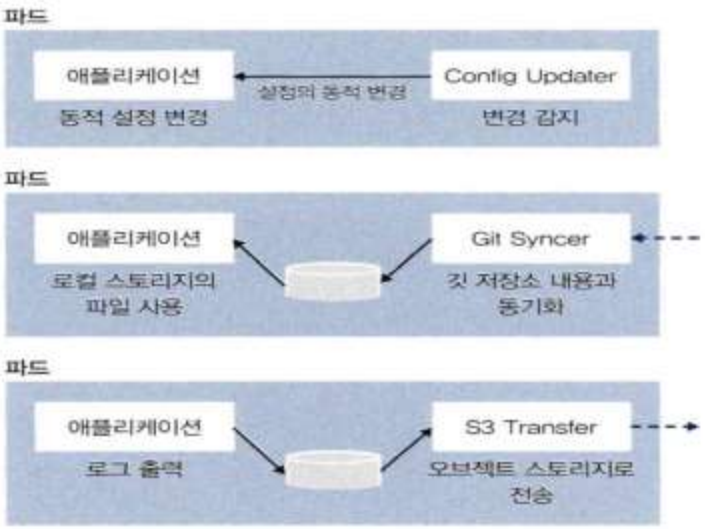


Pod

- Pod
  - ✓ 파드 디자인 패턴
    - 대부분의 경우 하나의 파드에 하나의 컨테이너를 가지지만 메인 컨테이너 이외에 메인 컨테이너를 지원하는 서브 컨테이너도 포함하듯이 하나의 파드에 여러 컨테이너를 가질 수도 있음
    - 서브 컨테이너라고 하면 프록시 역할을 하는 컨테이너, 설정 값을 동적으로 변경시키는 컨테이너, 로컬 캐시용 컨테이너, SSL용 컨테이너 등을 예로 들 수 있음
    - 하나의 파드에 여러 컨테이너가 존재하는 구성에 대해 각 특징 별로 파드의 디자인 패턴들이 제시되어 있는데 nginx 컨테이너와 redis 컨테이너와 같은 메인 컨테이너를 하나의 파드 안에 두는 구성은 개별 파드의 이동이 어려워지고 복잡해지기 때문에 추천하지 않음
    - 종류
      - Sidecar pattern
      - Ambassador pattern
      - Adapter pattern

Pod

- ❖ Pod
  - ✓ 파드 디자인 패턴
    - Sidecar pattern
      - ◆ 사이드카 패턴은 메인 컨테이너 외에 보조적인 기능을 추가하는 서브 컨테이너를 포함하는 패턴
      - ◆ 예를 들어 특정 변경 사항을 감지하여 동적으로 설정을 변경하는 컨테이너 그리고 깃 저장소와 로컬 스토리지를 동기화하는 컨테이너 와 애플리케이션의 로그 파일을 오브젝트 스토리지로 전송하는 컨테이너 라는 구성이 자주 사용됨
      - ◆ 파드는 데이터 영역을 공유하고 가지고 있을 수 있기 때문에 대부분 데이터와 설정에 관련된 패턴이라 할 수 있음

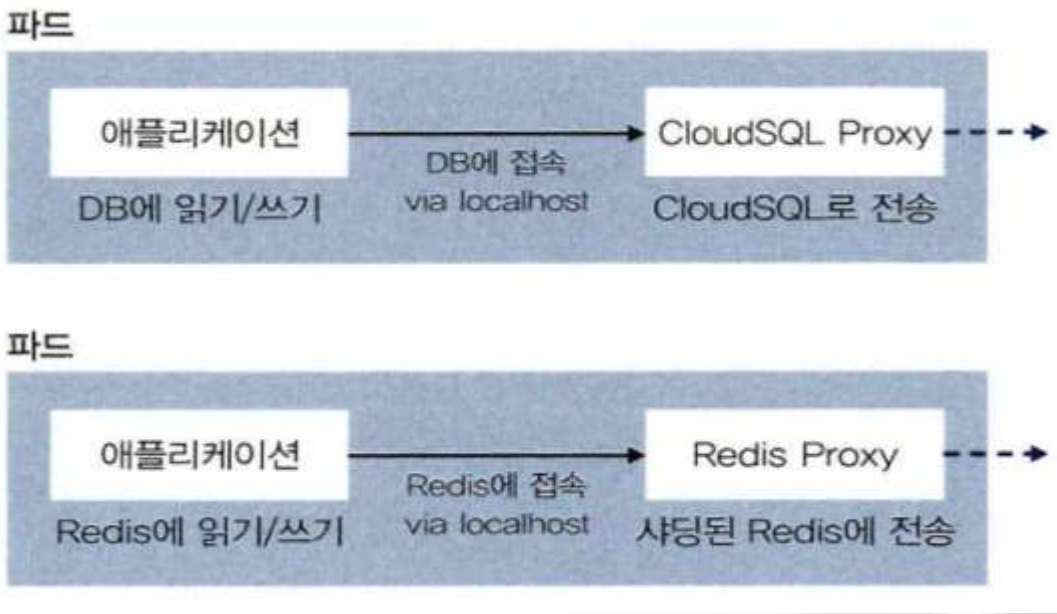


Pod

- Pod
  - ✓ 파드 디자인 패턴
    - Ambassador pattern
      - ◆ 앰배서더 패턴은 메인 컨테이너가 외부 시스템과 접속할 때 대리로 중계해주는 서브 컨테이너(앰배서더 컨테이너)를 포함한 패턴
      - ◆ 파드에 두 개의 컨테이너가 있어 메인 컨테이너에서 목적지에 localhost를 지정하여 앰배서더 컨테이너로 접속할 수 있음
      - ◆ 앰배서더 컨테이너를 사용하지 않고 메인 컨테이너에서 동작 중인 애플리케이션이 sharding 된 데이터베이스 하나를 선택하여 접속하게 된다면 메인 컨테이너는 데이터베이스와의 결합도가 강해짐
      - ◆ 앰배서더 컨테이너를 사용함으로써 메인 컨테이너에서는 항상 localhost를 지정하여 앰배서더 컨테이너로만 접속하고 앰배서더 컨테이너가 여러 목적지에 중계하여 연결하도록 구성하면 느슨한 결합을 유지할 수 있게 하는데 앰배서더 컨테이너를 경유하면 단일 데이터베이스를 사용하는 개발 환경이나 분산된 데이터베이스를 사용하는 서비스 환경 모두에서 특별한 변경 사항 없이 애플리케이션을 사용할 수 있음

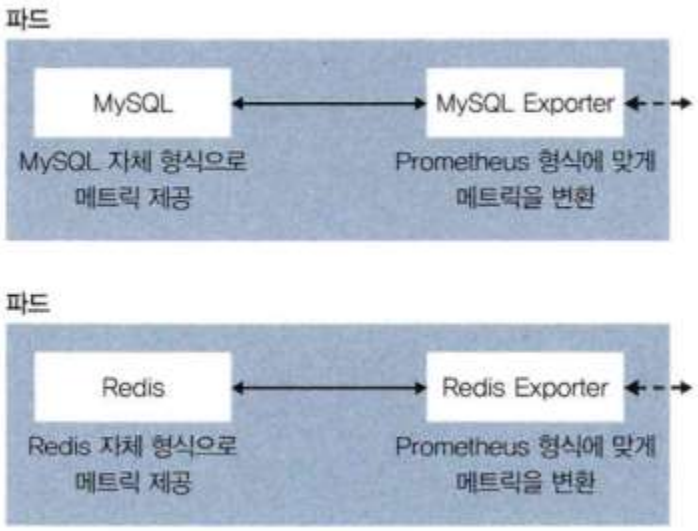
Pod

- Pod
  - ✓ 파드 디자인 패턴
    - Ambassador pattern



Pod

- ❖ Pod
  - ✓ 파드 디자인 패턴
    - Adapter pattern
      - ◆ 어댑터 패턴은 서로 다른 데이터 형식을 변환해주는 컨테이너(어댑터 컨테이너)를 포함하는 패턴
      - ◆ 프로메테우스(Prometheus) 등의 모니터링 소프트웨어에서는 정의된 형식으로 메트릭을 수집해야 하는데 대부분의 미들웨어가 제공하는 메트릭 출력 형식은 프로메테우스 메트릭 형식을 지원하지 않기 때문에 어댑터 컨테이너를 사용해서 외부 요청에 맞게 데이터 형식으로 변환하고 데이터를 반환해서 사용
      - ◆ 어댑터 패턴의 경우도 메인 컨테이너와 어댑터 컨테이너 간에는 localhost를 통해 접속



Pod

- ❖ Pod
  - ✓ 파드 생성
    - sample-pod.yaml 파일 생성 및 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
spec:
  containers:
  - name: nginx-container
    image: nginx
```

    - 리소스 파일 적용

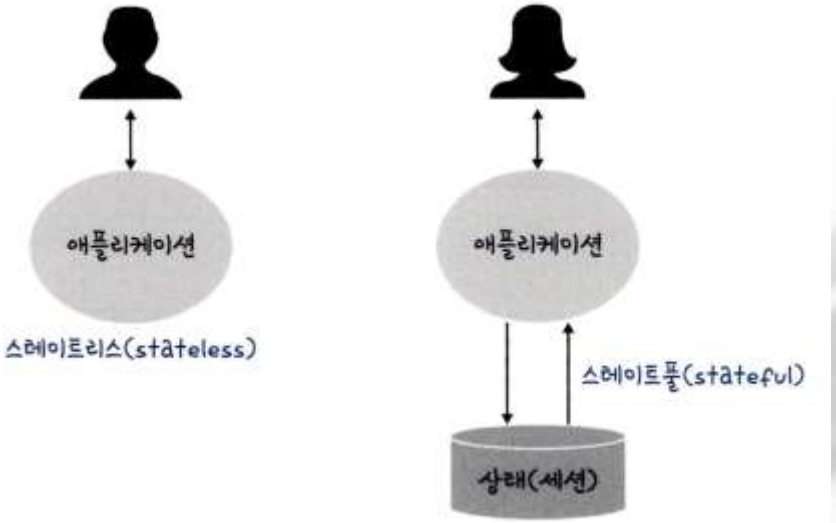
```
kubectl apply -f sample-pod.yaml


pod/sample-pod created
```

Pod

- Pod
  - ✓ 컨테이너 실행 과 관리
    - 쿠버네티스가 직접 컨테이너를 실행하지는 않음
    - 컨테이너를 생성할 책임을 해당 노드에 설치된 컨테이너 런타임에 맡기는 형태
    - 컨테이너 런타임은 containerd 나 docker 가 될 수 있음
    - 파드는 쿠버네티스가 관리하는 리소스고 컨테이너는 쿠버네티스 외부에서 관리

Pod

- Pod
  - ✓ Stateless & Stateful
    - 스테이트리스(stateless, 상태가 없는)는 사용자가 애플리케이션을 사용할 때 상태나 세션을 저장해 둘 필요가 없는 애플리케이션에서 사용
    - 스테이트풀(stateful)은 사용자가 애플리케이션을 사용할 때 상태나 세션을 별도의 데이터베이스에 저장해야 하는 애플리케이션에서 사용
    - 상태나 세션을 저장해야 하는 애플리케이션으로는 상품을 장바구니에 담고 구매하는 쇼핑몰, 금융 거래 사이트 등이 있음



Pod

- Pod
  - ✓ 파드 정보 확인
    - 정상적으로 생성되었는지 확인

      kubectl get pods

      ```
      NAME         READY   STATUS    RESTARTS   AGE
      sample-pod   1/1     Running   0          7m50s
      ```

Pod

- Pod
  - ✓ 파드 정보 확인
    - 추가적인 정보를 얻으려면 -o wide 옵션을 사용

      kubectl get deployment -o wide

      NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

      sample-pod 1/1 Running 0 10m 10.244.2.20 minikubecluster-m03 <none> <none>

Pod

- Pod
  - ✓ 두 개의 컨테이너를 포함한 파드 생성
    - sample-2pod.yaml 파일을 생성하고 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-2pod
spec:
  containers:
  - name: nginx-container
    image: nginx:1.16
  - name: redis-container
    image: redis:3.2
```

Pod

- Pod
  - ✓ 두 개의 컨테이너를 포함한 파드 생성
    - 파드 생성

      kubectl apply -f sample-2pod.yaml
    - 파드 확인

      kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
sample-2pod	2/2	Running	0	2m6s
sample-pod	1/1	Running	0	16m

Pod

- Pod
  - ✓ 두 개의 컨테이너를 포함한 파드 생성
    - sample-2pod-fail.yaml 파일을 생성하고 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-2pod-fail
spec:
  containers:
  - name: nginx-container-112
    image: nginx:1.16
  - name: nginx-container-113
    image: nginx:1.17
```

Pod

- Pod
  - ✓ 두 개의 컨테이너를 포함한 파드 생성
    - 파드 생성

      kubectl apply -f sample-2pod-fail.yaml
    - 파드 확인 – 동일한 포트를 사용하므로 첫번째만 컨테이너로 기동되고 두번째는 컨테이너 기동 안됨

      kubectl get pods

NAME	READY	STATUS	RESTARTS	AGE
sample-2pod	2/2	Running	0	5m15s
sample-2pod-fail	1/2	Error	0	5s
sample-pod	1/1	Running	0	19m

Pod

- ❖ Pod
  - ✓ 두 개의 컨테이너를 포함한 파드 생성
    - ● 로그 확인

      kubectl logs sample-2pod-fail -c nginx-container-113

      2024/12/18 02:53:28 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address already in use)

      nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)

      2024/12/18 02:53:28 [emerg] 1#1: bind() to 0.0.0.0:80 failed (98: Address already in use)

Pod

- Pod
  - ✓ 컨테이너 접속 과 명령어 실행
    - 생성된 컨테이너에 접속할 때는 kubectl exec 명령어를 사용하는데 가상 터미널을 생성(-t)하고 표준 입력을 패스 스루(-i)하면서 /bin/sh를 실행하면 컨테이너에 SSH로 로그인 한 상태가 됨

      kubectl exec -it sample-pod -- /bin/bash
    - 정상 수행 중 인지 명령어 실행

      root@sample-pod:/# apt update && apt -y install iproute2 procps

      root@sample-pod:/# ip a | grep "inet"

      inet 127.0.0.1/8 scope host lo

      inet 10.244.2.20/24 brd 10.244.2.255 scope global eth0

      root@sample-pod:/# ss -napt | grep LISTEN

      LISTEN 0 511 0.0.0.0:80 0.0.0.0:* users:(("nginx",pid=1,fd=6))

      LISTEN 0 511 [::]:80 [::]:* users:(("nginx",pid=1,fd=7))

Pod

- ❖ Pod
  - ✓ 컨테이너 접속 과 명령어 실행
    - ● 외부에서 명령어 실행
      - ◆ 컨테이너가 1개 인 파드에 명령어 수행

        kubectl exec -it sample-pod -- /bin/ls

        bin docker-entrypoint.d home mnt root srv usr
        boot docker-entrypoint.sh lib opt run sys var
        dev etc media proc sbin tmp
      - ◆ 특정 파드를 지정해서 명령어 수행

        kubectl exec -it sample-2pod -c nginx-container -- /bin/ls

        bin dev home media opt root sbin sys usr
        boot etc lib mnt proc run srv tmp var

Pod

- ❖ Pod
  - ✓ 컨테이너 접속 과 명령어 실행
    - ● 외부에서 명령어 실행
      - ◆ 옵션을 활용한 명령어 실행

        kubectl exec -it sample-pod -- /bin/ls --all --classify

        ./ bin@ docker-entrypoint.d/ home/ mnt/ root/ srv/ usr/

        ../ boot/ docker-entrypoint.sh* lib@ opt/ run/ sys/ var/

        .dockerenv* dev/ etc/ media/ proc/ sbin@ tmp/

      - ◆ 특수문자가 포함된 명령어 실행

        kubectl exec -it sample-pod -- /bin/bash -c "ls --all --classify | grep lib"

        lib@

Pod

- Pod
  - ✓ ENTRYPOINT 명령/CMD 명령과 command/args
    - 도커 파일로 이미지를 생성할 때는 ENTRYPOINT 명령과 CMD 명령을 사용하여 컨테이너 실행 시 명령어를 정의하는데 쿠버네티스에서는 도커용 용어와 조금 다르게 ENTRYPOINT를 command로 CMD를 args라고 부름
    - CMD 명령이 command에 해당하지 않는다는 점을 주의
    - 컨테이너를 실행할 때 도커 이미지의 ENTRYPOINT와 CMD를 덮어 쓰기하려면 파드 정의 내용 중 spec.containers[ ].command 와 spec.containers[ ].args를 지정

Pod

- Pod
  - ✓ ENTRYPOINT 명령/CMD 명령과 command/args
    - sample-entrypoint.yaml 파일 생성 및 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-entrypoint
spec:
  containers:
  - name: nginx-container-112
    image: nginx
    command: ["/bin/sleep"] # ENTRYPOINT
    args: ["3600"] # CMD
```

Pod

- Pod
  - ✓ ENTRYPOINT 명령/CMD 명령과 command/args
    - sample-entrypoint.yaml 파일 실행

      kubectl apply –f sample-entrypoint.yaml
    - sample-entrypoint 파드에 접속

      kubectl exec -it sample-entrypoint -- /bin/sh

    - 프로세스 실행 확인

      # apt update && apt -y install procps

      # ps aux

```
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.0   2216  1072 ?        Ss   05:08   0:00 /bin/sleep 3600
root           7  0.0  0.0   2316  1348 pts/0    Ss   05:09   0:00 /bin/sh
root         196  0.0  0.0   8160  3824 pts/0    R+   05:11   0:00 ps aux
```

Pod

- Pod
  - ✓ 파드명 제한
    - 이용 가능한 문자는 영문 소문자와 숫자
    - 이용 가능한 기호는 - 와 .
    - 시작과 끝은 영문 소문자

Pod

- Pod
  - ✓ 호스트의 네트워크 구성을 사용한 파드 기동
    - 쿠버네티스에 기동하는 파드에 할당된 ip 주소는 쿠버네티스 노드의 호스트 ip 주소와 범위가 달라 외부에서 볼 수 없는 IP 주소가 할당됨
    - 호스트의 네트워크를 사용하는 설정(spec.hostNetwork)을 활성화하면 호스트상에서 프로세스를 기동하는 것과 같은 네트워크 구성(IP 주소, DNS 설정, host 설정 등)으로 파드를 기동시킬 수 있음
    - hostNetwork를 사용한 파드는 쿠버네티스 노드의 IP 주소를 사용하는 관계로 포트 번호 충돌을 방지하기 위해 기본적으로 사용하지 않고 사용할 때는 에지(Edge) 환경에서의 사용이나 호스트 측의 네트워크를 감시 또는 제어와 같은 특수한 애플리케이션 등에서만 사용

Pod

- ❖ Pod
  - ✓ 호스트의 네트워크 구성을 사용한 파드 기동
    - Pod 확인

      kubectl get pod sample-pod -o wide

      NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

      sample-pod 1/1 Running 0 3h42m 10.244.2.20 minikubecluster-m03 <none> <none>

      kubectl get node minikubecluster-m03 -o wide

      NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME

      minikubecluster-m03 Ready <none> 5d6h v1.28.3 192.168.58.4 <none> Ubuntu 22.04.3 LTS 6.10.11-linuxkit docker://24.0.7

Pod

- Pod
  - ✓ 호스트의 네트워크 구성을 사용한 파드 기동
    - sample-hostnetwork .yaml 파일 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-hostnetwork
spec:
  hostNetwork: true
  containers:
  - name: nginx-container
    image: nginx
```

Pod

- ❖ Pod
  - ✓ 호스트의 네트워크 구성을 사용한 파드 기동
    - Pod 확인

      kubectl get pod sample-hostnetwork -o wide

      NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

      sample-hostnetwork 1/1 Running 0 10s 192.168.58.4 minikubecluster-m03 <none> <none>

      kubectl get node minikubecluster-m03 -o wide

      NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME

      minikubecluster-m03 Ready <none> 5d6h v1.28.3 192.168.58.4 <none> Ubuntu 22.04.3 LTS 6.10.11-linuxkit docker://24.0.7

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - DNS 서버에 관한 설정(dnsPolicy)은 파드 정의 spec.dnsPolicy에 설정
    - dnsPolicy 설정값
      - ClusterFirst(기본값)
      - None
      - Default
      - ClusterFirstWithHostNet

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - ClusterFirst(기본값)
      - 일반적으로 파드는 클러스터 내부 DNS를 사용하여 이름을 해석
      - 서비스 디스커버리나 클러스터 내부 로드 밸런싱에서 사용
      - dnsPolicy가 ClusterFirst인 경우 클러스터 내부의 DNS 서버에 질의를 하고 클러스터 내부 DNS에서 해석이 안 되는 도메인에 대해서는 업스트림 DNS 서버에 질의

Pod

- ❖ Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - ● ClusterFirst(기본값)
      - ◆ sample-dnspolicy-clusterfirst.yaml 파일 생성 및 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-dnspolicy-clusterfirst
spec:
  dnsPolicy: ClusterFirst
  containers:
  - name: nginx-container
    image: nginx
```

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - ClusterFirst(기본값)
      - ◆ 파드 배포

      kubectl apply -f sample-dnspolicy-clusterfirst.yaml
      - ◆ 컨테이너 내부의 DNS 설정 파일 /etc/resolv.conf 확인

      kubectl exec -it sample-dnspolicy-clusterfirst -- cat /etc/resolv.conf

      nameserver 10.96.0.10
      search default.svc.cluster.local svc.cluster.local cluster.local
      options ndots:5
      - ◆ 클러스터 내부의 DNS Service에 할당된 IP 주소 확인

      kubectl get services -n kube-system

      NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
      kube-dns ClusterIP 10.96.0.10 <none> 53/UDP,53/TCP,9153/TCP 6d7h

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - None
      - 특별한 요건에 따라서는 클러스터 외부 DNS 서버를 참조하는 경우도 있음
      - DNS 서버를 수동으로 설정하려면 spec.dnsPolicy: None이라고 설정한 후 dnsConfig에 설정하고 싶은 값을 작성
      - 정적으로 외부 DNS 서버만 설정하면 클러스터 내부 DNS를 사용한 서비스 디스커버리는 사용할 수 없게 되므로 주의

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - None
      - sample-dnspolicy-none.yaml 파일 생성 및 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-dnspolicy-none
spec:
  dnsPolicy: None
  dnsConfig:
    nameservers:
    - 8.8.8.8
    - 8.8.4.4
    searches:
    - example.com
    options:
    - name: ndots
      value: "5"
  containers:
  - name: nginx-container
    image: nginx
```

Pod

- ❖ Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - ● None
      - ◆ 파드 배포

      kubectl apply -f sample-dnspolicy-none.yaml
      - ◆ 컨테이너 내부의 DNS 설정 파일 /etc/resolv.conf 확인

      kubectl exec -it sample-dnspolicy-none -- cat /etc/resolv.conf

      nameserver 8.8.8.8

      nameserver 8.8.4.4

      search example.com

      options ndots:5

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - Default
      - 쿠버네티스 노드의 DNS 설정을 그대로 상속받는 경우에는 spec.dnsPolicy: Default로 설정
      - dnsPolicy의 기본값은 Default가 아니므로 주의
      - 쿠버네티스 노드의 DNS 설정을 상속받게 설정하면 클러스터 내부의 DNS를 사용한 서비스 디스커버리를 할 수 없음

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - Default
      - sample-dnspolicy-default.yaml 파일 생성 및 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-dnspolicy-default
spec:
  dnsPolicy: Default
  containers:
  - name: nginx-container
    image: nginx
```

Pod

- ❖ Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - ● Default
      - ◆ 파드 배포

      kubectl apply -f sample-dnspolicy-default.yaml
      - ◆ 컨테이너 내부의 DNS 설정 파일 /etc/resolv.conf 확인

      kubectl exec -it sample-dnspolicy-default -- cat /etc/resolv.conf

      nameserver 192.168.65.254

      options ndots:0

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - ClusterFirstWithHostNet
      - hostNetwork를 사용한 파드에 클러스터 내부의 DNS를 참조하고 싶은 경우에는 spec.dnsPolicy: ClusterFirstWithHostNet을 설정
      - hostNetwork를 사용하는 경우 기본값 ClusterFirst의 설정 값은 무시되고 쿠버네티스 노드의 네트워크 설정이 사용되기 때문에 명시적으로 ClusterFirstWithHostNet을 지정해야 함

Pod

- Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - ClusterFirstWithHostNet
      - sample-dnspolicy-clusterfirstwithhostnet.yaml 파일 생성 및 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-dnspolicy-clusterfirstwithhostnet
spec:
  dnsPolicy: ClusterFirstWithHostNet
  hostNetwork: true
  containers:
  - name: nginx-container
    image: nginx
```

Pod

- ❖ Pod
  - ✓ 파드 DNS 설정과 서비스 디스커버리
    - ● ClusterFirstWithHostNet
      - ◆ 파드 배포

      kubectl apply -f sample-dnspolicy-clusterfirstwithhostnet.yaml
      - ◆ 컨테이너 내부의 DNS 설정 파일 /etc/resolv.conf 확인

      kubectl exec -it sample-dnspolicy-clusterfirstwithhostnet -- cat /etc/resolv.conf

      nameserver 10.96.0.10

      search default.svc.cluster.local svc.cluster.local cluster.local

      options ndots:5

Pod

- Pod
  - ✓ 정적 호스트명 설정
    - 리눅스 운영체제에서는 DNS로 호스트명을 해석하기 전에 /etc/hosts 파일로 정적 호스트명을 해석
    - 쿠버네티스에서는 파드 내부 모든 컨테이너의 /etc/hosts를 변경하는 기능이 준비되어 있으며 spec.hostAliases로 지정하여 사용
    - sample-hostaliases.yaml 파일 생성 및 설정

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-hostaliases
spec:
  containers:
  - name: nginx-container
    image: nginx
  hostAliases:
  - ip: 8.8.8.8
    hostnames:
    - google-dns
    - google-public-dns
```

Pod

- ❖ Pod
  - ✓ 정적 호스트명 설정
    - 파드 배포

      kubectl apply -f sample-hostaliases.yaml
    - /etc/hosts 조회

      kubectl exec -it sample-hostaliases -- cat /etc/hosts

```
# Kubernetes-managed hosts file.
127.0.0.1               localhost
::1         localhost ip6-localhost ip6-loopback
fe00::0  ip6-localnet
fe00::0  ip6-mcastprefix
fe00::1  ip6-allnodes
fe00::2  ip6-allrouters
10.244.1.20             sample-hostaliases

# Entries added by HostAliases.
8.8.8.8  google-dns             google-public-dns
```

Pod

- Pod
  - ✓ 작업 디렉터리 설정
    - 컨테이너에서 동작하는 애플리케이션의 작업 디렉터리(Working Directory)는 도커 파일의 W0RKDIR 명령 설정을 따르지만 spec.containers[].workingDir로 덮어 쓸 수 있음
    - 특정 스크립트 등이 배치된 볼륨을 파드에 마운트할 때 그 스크립트가 배치된 디렉터리로 이동한 후 실행하고 싶은 경우가 있는데 이는 작업 디렉터리를 변경하는 기능을 사용하여 해결할 수 있음
    - 컨테이너에 workingDir을 설정한 경우 프로세스가 실행되는 디렉터리가 변경된 것을 확인할 수 있음

Pod

- Pod
  - ✓ 작업 디렉터리 설정
    - sample-workingdir.yaml 파일 생성 및 설정

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-workingdir
spec:
  containers:
  - name: nginx-container
    image: nginx:1.16
    workingDir: /tmp
```

    - 파드 배포

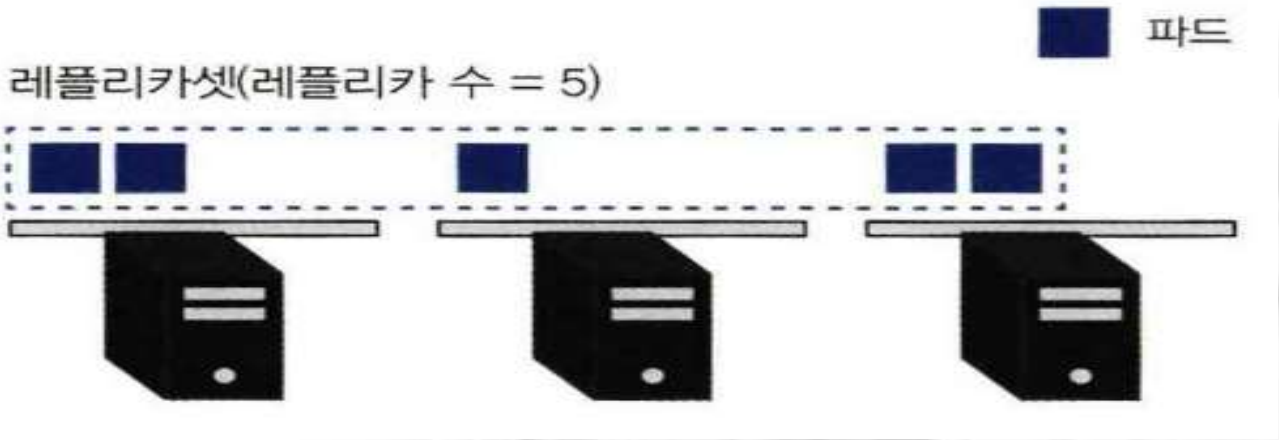
```
kubectl apply -f sample-workingdir.yaml
```

    - 현재 디렉토리 확인

```
kubectl exec -it sample-workingdir -- pwd

/tmp
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 개요
    - Replicaset은 일정한 개수의 동일한 Pod가 항상 실행되도록 관리하는데 이러한 기능이 필요한 이유는 서비스의 지속성 때문
    - 노드의 하드웨어에서 발생하는 장애 등의 이유로 Pod를 사용할 수 없을 때 다른 노드에서 Pod를 다시 생성해서 사용자에게 중단 없는 서비스를 제공할 수 있음
    - 원래 파드의 레플리카를 생성하는 리소스의 이름은 레플리케이션 컨트롤러였는데 레플리카셋을 추가하고 일부 기능을 추가
    - 레플리케이션 컨트롤러는 앞으로 사용하지 않는 추세이기 때문에 기본적으로 레플리카셋을 사용하는 것을 권장



Pod

- ReplicaSet/ReplicationController
  - ✓ 생성
    - ReplicaSet을 위한 yaml 파일 작성 – sample-rs.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: sample-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 생성
    - ReplicaSet 생성

      kubectl apply -f sample-rs.yaml

      replicaset.apps/sample-rs created
    - ReplicaSet 확인

      kubectl get replicasets -o wide

      NAME DESIRED CURRENT READY AGE CONTAINERS IMAGES SELECTOR

      sample-rs 3 3 3 17s nginx-container nginx app=sample-app

Pod

- ReplicaSet/ReplicationController
  - ✓ 생성
    - label(app=sample-app)을 지정하여 파드 목록 표시

      kubectl get pods -l app=sample-app -o wide

      NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

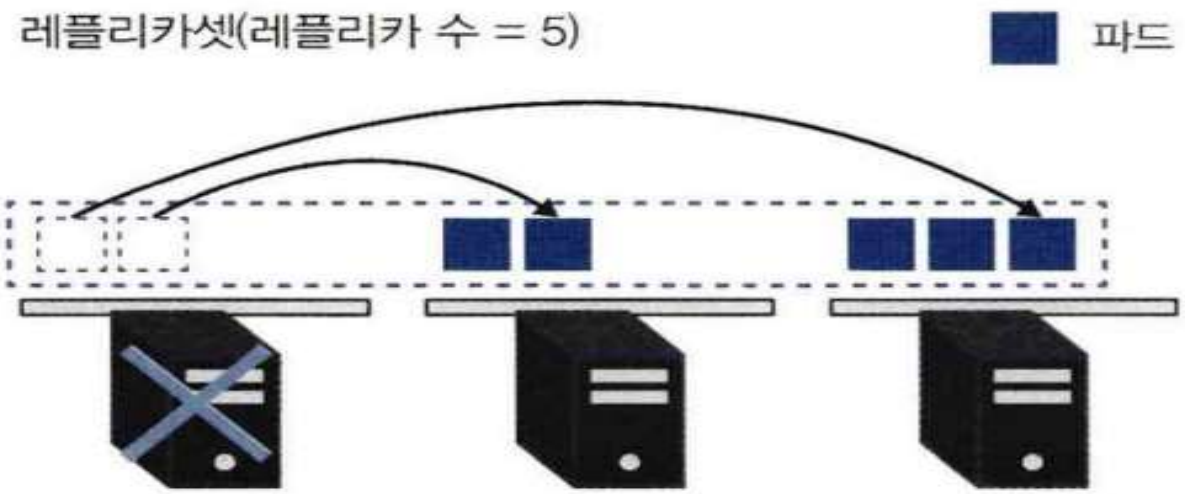
      sample-rs-45hp4 1/1 Running 0 2m53s 10.244.1.23 minikubecluster-m02 <none> <none>

      sample-rs-97749 1/1 Running 0 2m53s 10.244.0.5 minikubecluster <none> <none>

      sample-rs-x8n5m 1/1 Running 0 2m53s 10.244.2.29 minikubecluster-m03 <none> <none>
    - 레플리카셋이 생성하는 파드는 레플리카셋이름-임의의문자열 로 설정

Pod

- ReplicaSet/ReplicationController
  - ✓ 파드 정지 와 자동화된 복구
    - 레플리카셋에서는 노드나 파드에 장애가 발생했을 때 지정한 파드 수를 유지하기 위해 다른 노드에서 파드를 기동시켜 주기 때문에 장애 시에도 많은 영향을 받지 않는데 이것이 쿠버네티스의 중요한 콘셉트 중 하나로 자동화된 복구 라는 기능



노드 장애 시에도 다른 노드에서 파드를 기동하고
레플리카 수를 유지(자동화된 복구)

Pod

- ReplicaSet/ReplicationController
  - ✓ 파드 정지 와 자동화된 복구
    - 자동화된 복구 기능을 확인하기 위해 하나의 파드 정지

      kubectl delete pods sample-rs-45hp4

      pod "sample-rs-45hp4" deleted
    - 파드 확인

      kubectl get pods -o wide

      NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

      sample-rs-97749 1/1 Running 0 6m23s 10.244.0.5 minikubecluster <none> <none>

      sample-rs-lpmsn 1/1 Running 0 9s 10.244.1.24 minikubecluster-m02 <none> <none>

      sample-rs-x8n5m 1/1 Running 0 6m23s 10.244.2.29 minikubecluster-m03 <none> <none>

Pod

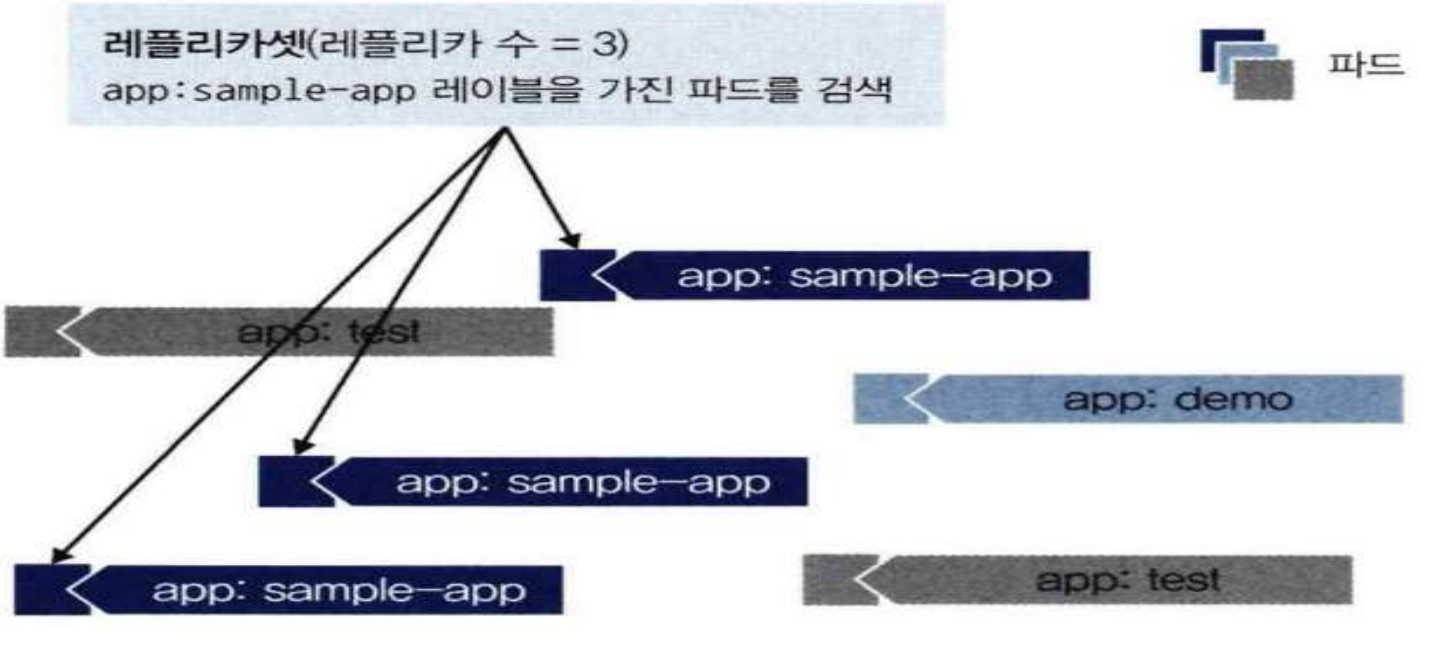
- ReplicaSet/ReplicationController
  - ✓ 파드 정지 와 자동화된 복구
    - 증감 이력 확인

      kubectl describe replicaset sample-rs

```
Name:         sample-rs
Namespace:    default
Selector:     app=sample-app
Labels:       <none>
Annotations:  <none>
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  8m28s  replicaset-controller  Created pod: sample-rs-x8n5m
  Normal  SuccessfulCreate  8m28s  replicaset-controller  Created pod: sample-rs-45hp4
  Normal  SuccessfulCreate  8m28s  replicaset-controller  Created pod: sample-rs-97749
  Normal  SuccessfulCreate  2m14s  replicaset-controller  Created pod: sample-rs-lpmsn
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 레플리카셋 과 레이블
    - 레플리카셋은 쿠버네티스가 파드를 모니터링하여 파드 수를 조정
    - 모니터링은 특정 레이블을 가진 파드 수를 계산하는 형태로 이루어 짐
    - 레플리카 수가 부족한 경우 매니페스트에 기술된 spec.template로 파드를 생성하고 레플리카 수가 많을 경우 레이블이 일치하는 파드 중 하나를 삭제



Pod

- ReplicaSet/ReplicationController
  - ✓ 레플리카셋 과 레이블
    - 레이블은 spec.selector 와 spec.template.metadata.labels 부분에 해당

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: sample-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 레플리카셋 과 레이블
    - 레이블은 spec.selector 와 spec.template.metadata.labels 부분에 해당

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: sample-rs
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 레플리카셋 과 레이블
    - 두 개의 레이블이 다른 yaml 파일 생성: sample-rs-fail.yaml

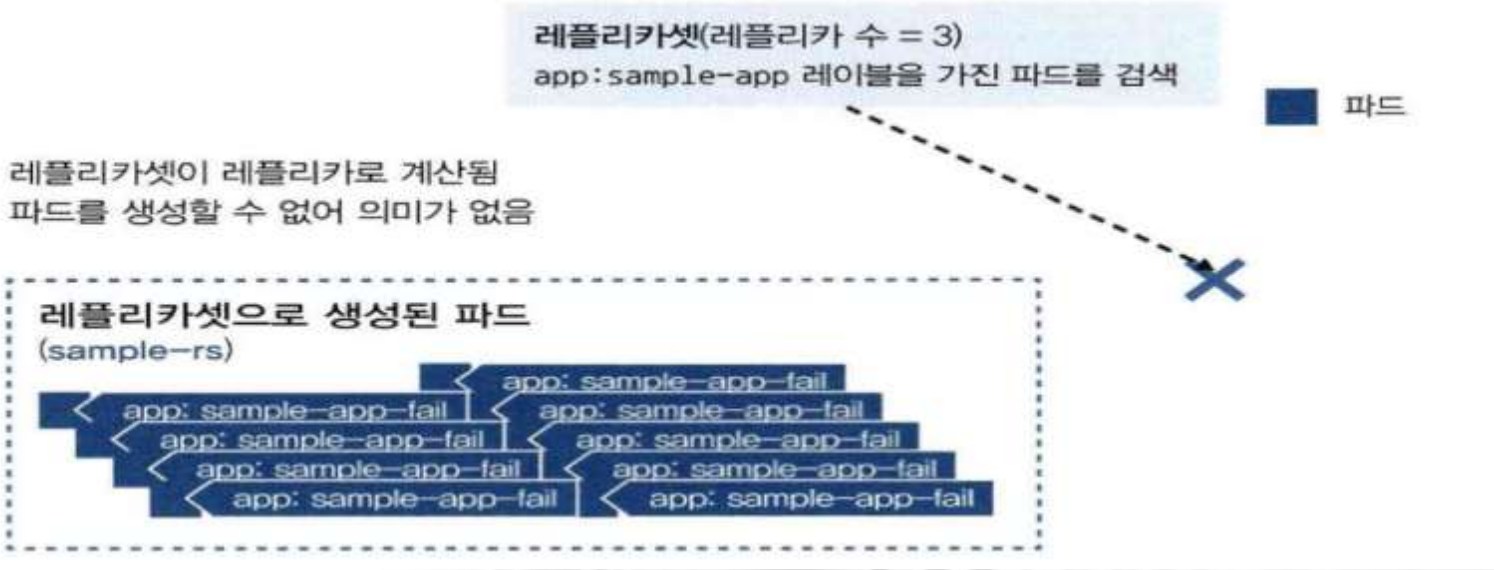
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: sample-rs-fail
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app-fail
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 레플리카셋 과 레이블
    - 두 개의 레이블이 다른 yaml 파일 생성: sample-rs-fail.yaml

      kubectl apply -f sample-rs-fail.yaml

      The ReplicaSet "sample-rs-fail" is invalid: spec.template.metadata.labels: Invalid value: map[string]string{"app":"sample-app-fail"}: `selector` does not match template `labels`



Pod

- ReplicaSet/ReplicationController
  - ✓ 레플리카셋 과 레이블
    - 동일한 레이블을 가진 파드를 레플리카셋 외부에서 생성: sample-rs-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-rs-pod
  labels:
    app: sample-app
spec:
  containers:
  - name: nginx-container
    image: nginx
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 레플리카셋 과 레이블
    - 동일한 레이블을 가진 파드를 레플리카셋 외부에서 생성: sample-rs-pod.yaml

      kubectl apply -f sample-rs-pod.yaml

      pod/sample-rs-pod created

      kubectl get pods -L app

```
NAME              READY   STATUS        RESTARTS   AGE    APP
sample-rs-97749   1/1     Running       0          26m    sample-app
sample-rs-lpmsn   1/1     Running       0          19m    sample-app
sample-rs-pod     0/1     Terminating   0          2s     sample-app
sample-rs-x8n5m   1/1     Running       0          26m    sample-app
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 레플리카셋 과 스케일링
    - 방법
      - 매니페스트를 수정하여 kubectl apply –f 명령어를 실행
      - kubectl scale 명령어를 사용하여 스케일 처리
    - kubectl scale 명령어를 이용해서 replica의 개수를 5로 수정

      kubectl scale replicaset sample-rs --replicas 5

      replicaset.apps/sample-rs scaled

      kubectl get pods

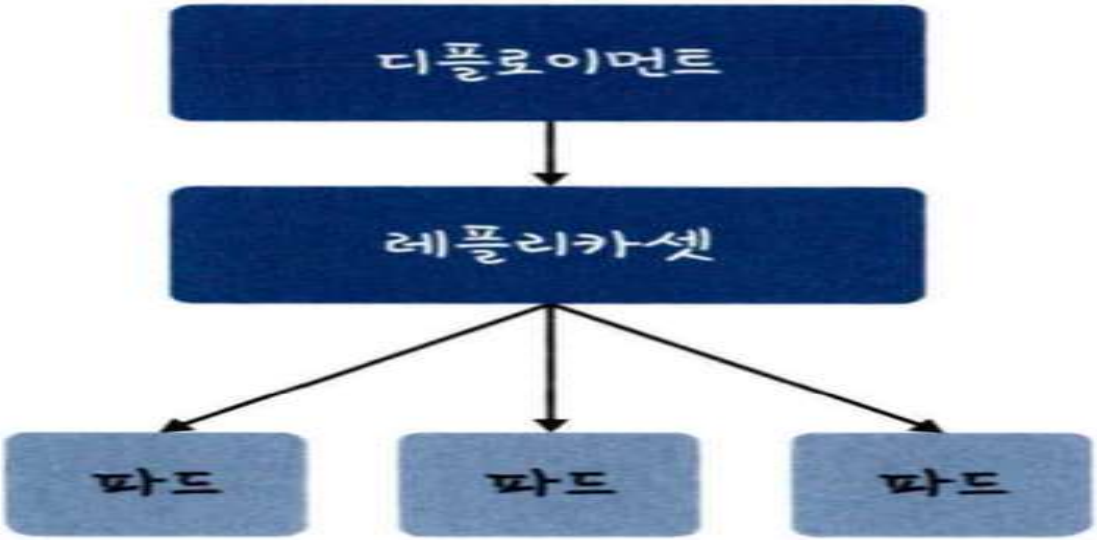
```
NAME              READY   STATUS    RESTARTS   AGE
sample-rs-97749   1/1     Running   0          31m
sample-rs-bzwcj   1/1     Running   0          5s
sample-rs-d2tv2   1/1     Running   0          5s
sample-rs-lpmsn   1/1     Running   0          25m
sample-rs-x8n5m   1/1     Running   0          31m
```

Pod

- ReplicaSet/ReplicationController
  - ✓ 일치성 기준 조건과 집합성 기준 조건
    - 레플리카 제어 조건은 서비스 중단 예정인 레플리케이션 컨트롤러의 일치성 기준(equality-based) 셀렉터였지만 레플리카셋에서는 좀 더 강화된 집합성 기준(set-based) 셀렉터를 사용하여 유연한 제어도 가능
      - 일치성 기준: 조건부에 일치 불일치(=, !=) 조건 지정
      - 집합성 기준: 조건부에 일치 불일치(=, !=) 조건 지정과 집합(in, notin, exists) 조건 지정 가능
    - 쿠버네티스에서 어떤 조건을 지정할 때는 이 두 가지 방법이 있고 레플리카셋 외에 스케줄링할 때도 이 조건 지정이 사용됨
    - 일치성 기준 조건에서는 조건부에 같은지 혹은 같지 않은지에 대한 조건을 사용할 수 있는데 app=sample-app과 같이 지정하는 것
    - 집합성 기준 조건에서는 일치성 기준 조건과 함께 집합 조건을 지정할 수 있는데 env In [development,staging]과 같이 지정할 수 있으며 In에 해당하는 연산자는 스케줄링 조건에서 사용할 때 수치 비교도 가능

Pod

- Deployment
  - ✓ 개요



Pod

- Deployment
  - ✓ 개요
    - 여러 레플리카셋을 관리하여 롤링 업데이트나 롤백 등을 구현하는 리소스
    - Deployment가 레플리카셋을 관리하고 레플리카셋이 파드를 관리하는 관계
    - Deployment는 Replicaset의 상위 개념으로서 Pod의 개수를 유지할 뿐만 아니라 배포 작업을 좀 더 세분화해 관리할 수 있음
    - Deployment를 사용하면 신규 레플리카셋에 컨테이너가 기동되었는지 와 헬스 체크는 통과했는지를 확인하면서 전환 작업이 진행되며 레플리카셋의 이행 과정에서 파드 수에 대한 상세한 지정도 가능
    - 쿠버네티스에서 가장 권장하는 컨테이너 기동 방법으로 알려져 있음
    - 하나의 파드를 기동만 한다 하더라도 Deployment 사용을 권장하는데 파드로만 배포한 경우에는 파드에 장애가 발생하면 자동으로 파드가 다시 생성되지 않으며 레플리카셋으로만 배포한 경우에는 롤링 업데이트 등의 기능을 사용할 수 없음

Pod

- ❖ Deployment
  - ✓ 생성
    - sample-deployment.yaml 파일 생성 및 작성

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

Pod

- ❖ Deployment
  - ✓ 생성
    - 업데이트 이력을 저장하는 옵션을 사용하여 리소스를 생성

      kubectl apply -f sample-deployment.yaml --record=true
    - 이력은 각 레플리카셋의 metadata.annotations[kubernetes.io/change-cause]에 저장되고 현재 레플리카셋의 수정 버전 번호도 마찬가지로 metadata.annotations[deployment.kubernetes.io/revision]에 저장
    - 레플리카셋 목록 확인

      kubectl get replicasets -o yaml | head

```
apiVersion: v1
items:
- apiVersion: apps/v1
  kind: ReplicaSet
  metadata:
    annotations:
      deployment.kubernetes.io/desired-replicas: "3"
      deployment.kubernetes.io/max-replicas: "4"
      deployment.kubernetes.io/revision: "1"
      kubernetes.io/change-cause: kubectl apply --filename=sample-
deployment.yaml
```

Pod

- ❖ Deployment
  - ✓ 생성
    - ● 계층 확인

kubectl get deployments

```
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
sample-deployment   3/3     3            3           7m
```

kubectl get replicasets

```
NAME                           DESIRED   CURRENT   READY   AGE
sample-deployment-b6cfdc59f    3         3         3       7m6s
```

kubectl get pods

```
NAME                                 READY   STATUS    RESTARTS   AGE
sample-deployment-b6cfdc59f-hz65m    1/1     Running   0          7m10s
sample-deployment-b6cfdc59f-qpphh    1/1     Running   0          7m10s
sample-deployment-b6cfdc59f-wp8sq    1/1     Running   0          7m10s
```

Pod

- ❖ Deployment
  - ✓ 생성
    - ● 이미지 업데이트

      kubectl set image deployment sample-deployment nginx-container=nginx:1.17 --record

      deployment.apps/sample-deployment image updated

      kubectl rollout status deployment sample-deployment

      deployment "sample-deployment" successfully rolled out

      kubectl get deployments

      NAME READY UP-TO-DATE AVAILABLE AGE
      sample-deployment 3/3 3 3 28m

Pod

- ❖ Deployment
  - ✓ 생성
    - ● 이미지 업데이트

      kubectl get replicasets

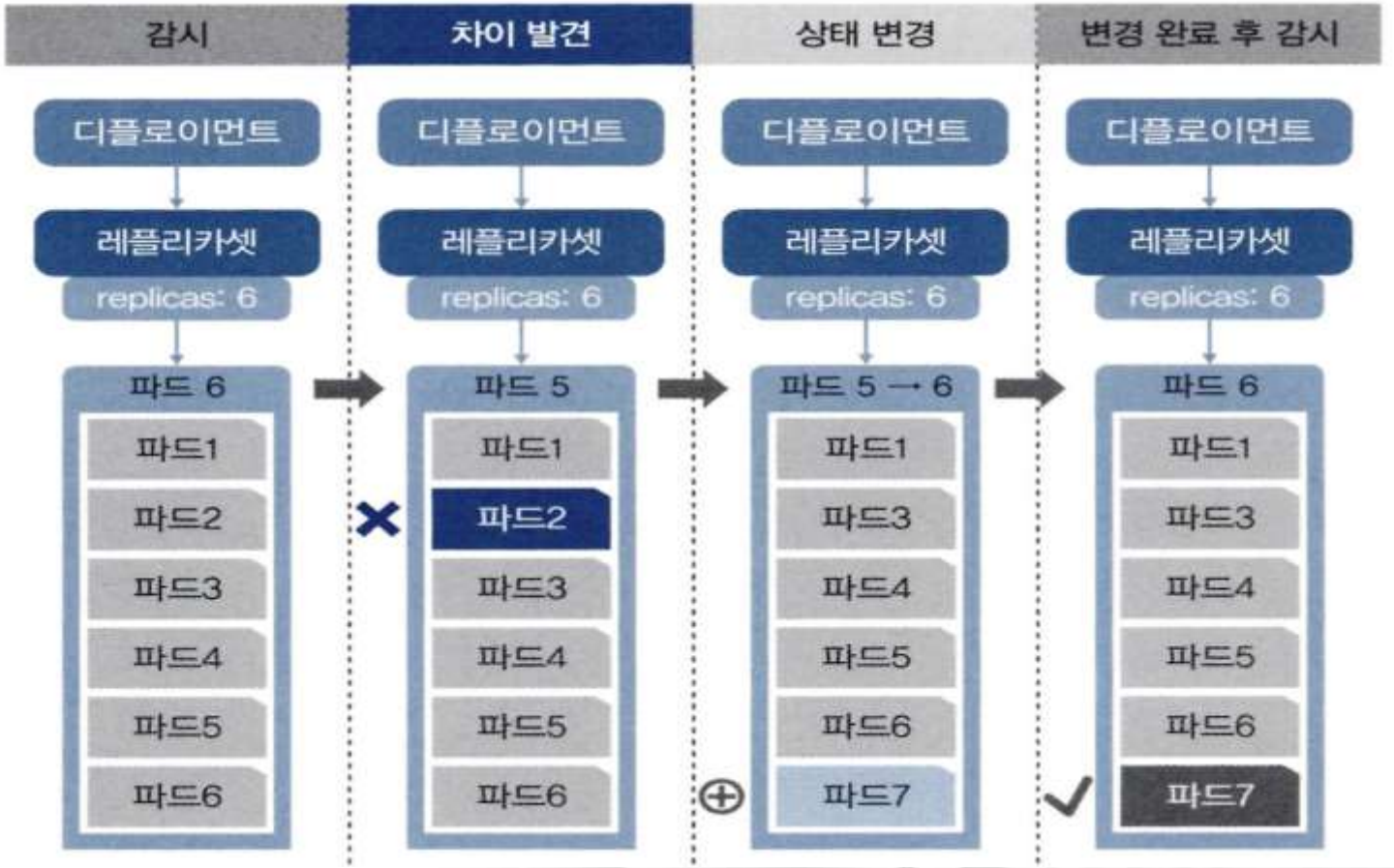
```
NAME                          DESIRED   CURRENT   READY   AGE
sample-deployment-79598877b   3         3         3       44s
sample-deployment-b6cfdc59f   0         0         0       29m
```

Pod

- Deployment
  - ✓ 레플리카셋이 생성되는 조건
    - Deployment에서는 변경이 발생하면 레플리카셋이 생성되는데 변경에는 레플리카 수의 변경 등은 포함되어 있지 않으며 생성된 파드의 내용 변경이 필요
    - spec.template에 변경이 있으면 생성된 파드의 설정이 변경되기 때문에 레플리카셋을 신규로 생성하고 롤링 업데이트를 수행
    - 실제로 매니페스트를 쿠버네티스에 등록한 후 레플리카셋의 정의를 보면, spec.template 아래의 해시값(파드 템플릿 해시 - Pod Template Hash)을 계산하고 이 값을 사용한 레이블로 관리
    - 수작업으로 이미지 등을 이전 버전으로 재변경하여 해시값이 동일해진 경우에는 레플리카셋을 신규로 생성하지 않고 기존 레플리카셋을 사용

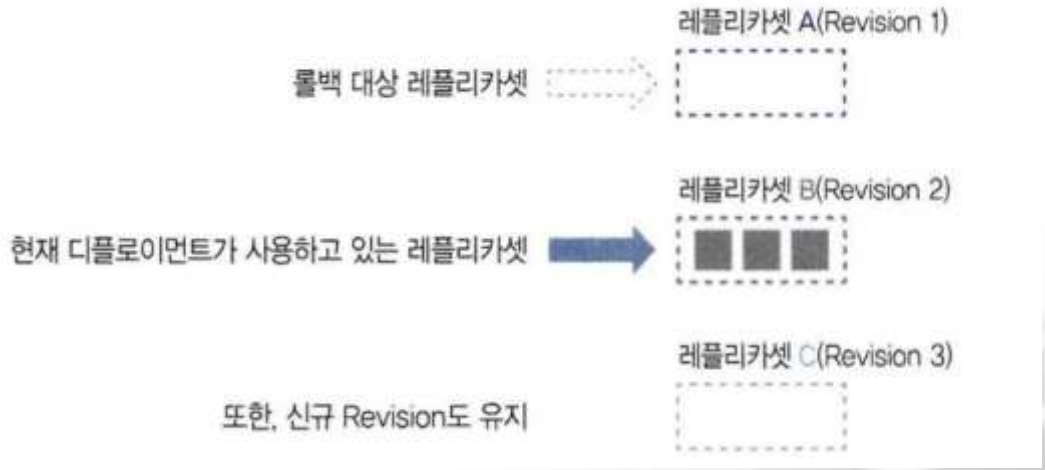
Pod

- Deployment
  - ✓ Deployment에 속한 파드의 삭제 및 복구 과정



Pod

- ❖ Deployment
  - ✓ 변경 롤백
    - Deployment에는 롤백 기능이 있음
    - 롤백 기능의 실체는 현재 사용 중인 레플리카셋의 전환과 같은 것
    - Deployment가 생성한 기존 레플리카셋은 레플리카 수가 0인 상태로 남아 있기 때문에 레플리카 수를 변경시켜 다시 사용할 수 있는 상태가 됨



    - 변경 이력을 확인할 때는 kubectl rollout history 명령어를 사용하는데 CHANGE-CAUSE 부분은 Deployment를 생성할 때 --record 옵션을 사용하여 이력 내용이 있지만 --record 옵션을 사용하지 않았다면 <none>

Pod

- ❖ Deployment
  - ✓ 변경 롤백
    - ● 변경 이력 확인

      kubectl rollout history deployment sample-deployment

      deployment.apps/sample-deployment

      REVISION CHANGE-CAUSE

      1 kubectl apply --filename=sample-deployment.yaml --record=true

      2 kubectl set image deployment sample-deployment nginx-container=nginx:1.17 --record=true

Pod

- ❖ Deployment
  - ✓ 변경 롤백
    - ● 버전에 대한 상세 이력

      kubectl rollout history deployment sample-deployment --revision 1

```
deployment.apps/sample-deployment with revision #1
Pod Template:
  Labels:       app=sample-app
          pod-template-hash=b6cfdc59f
  Annotations:  kubernetes.io/change-cause: kubectl apply --filename=sample-
deployment.yaml --record=true
  Containers:
   nginx-container:
    Image:      nginx
    Port:<none>
    Host Port:  <none>
    Environment:  <none>
    Mounts:     <none>
  Volumes:      <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
```

Pod

- ❖ Deployment
  - ✓ 변경 롤백
    - ● 버전에 대한 상세 이력

      kubectl rollout history deployment sample-deployment --revision 2

```
deployment.apps/sample-deployment with revision #2
Pod Template:
  Labels:       app=sample-app
        pod-template-hash=79598877b
  Annotations:  kubernetes.io/change-cause: kubectl set image deployment
sample-deployment nginx-container=nginx:1.17 --record=true
  Containers:
   nginx-container:
    Image:      nginx:1.17
    Port:<none>
    Host Port:  <none>
    Environment:  <none>
    Mounts:     <none>
  Volumes:      <none>
  Node-Selectors:  <none>
  Tolerations:  <none>
```

Pod

- Deployment
  - ✓ 변경 롤백
    - 버전 번호를 이용한 롤백

      kubectl rollout undo deployment sample-deployment --to-revision 1
    - 바로 이전 리비전으로 롤백

      kubectl rollout undo deployment sample-deployment
    - 확인

      kubectl get replicasets

      NAME DESIRED CURRENT READY AGE

      sample-deployment-79598877b 3 3 3 13m

      sample-deployment-b6cfdc59f 0 0 0 41m
    - 실제 환경에서는 롤백 기능을 사용하는 경우가 많지 않은데 CI/CD 파이프라인에서 롤백을 하는 경우 kubectl rollout 명령어보다 이전 매니페스트를 다시 kubectl apply 명령어로 실행하여 적용하는 것이 호환성 면에서 더 좋기 때문이며 spec.template을 같은 내용으로 되돌렸을 경우에는 pod-template-hash 값이 같기 때문에 kubectl rollout 처럼 기존에 있었던 레플리카셋의 파드가 기동됨

Pod

- ❖ Deployment
  - ✓ 변경 일시 중지
    - 일반적으로 Deployment를 업데이트하면 바로 적용되어 업데이트 처리가 실행되지만 안전을 위해 Deployment에 대한 업데이트를 하더라도 바로 적용되지 않는 것을 원하는 경우도 있음
    - 업데이트 일시 정지

      kubectl rollout pause deployment sample-deployment

      deployment.apps/sample-deployment paused
    - 이미지 업데이트

      kubectl set image deployment sample-deployment nginx-container=nginx:1.16

      deployment.apps/sample-deployment image updated

      kubectl rollout status deployment sample-deployment

      Waiting for deployment "sample-deployment" rollout to finish: 0 out of 3 new replicas have been updated...

Pod

- ❖ Deployment
  - ✓ 변경 일시 중지
    - ● 롤백

      kubectl rollout undo deployment sample-deployment

      error: you cannot rollback a paused deployment; resume it first with 'kubectl rollout resume' and try again
    - ● 업데이트 일시 정지 해제

      kubectl rollout resume deployment sample-deployment

      deployment.apps/sample-deployment resumed

Pod

- ❖ Deployment
  - ✓ Deployment 스케일링
    - ● Deployment가 관리하는 레플리카셋의 레플리카 수는 레플리카셋과 같은 방법으로 kubectl apply -f 또는 kubectl scale을 사용하여 스케일링

      kubectl scale deployment sample-deployment --replicas=5

Pod

- Deployment
  - ✓ 매니페스트를 사용하지 않고 생성

kubectl create deployment sample-deployment-by-cli --image nginx

deployment.apps/sample-deployment-by-cli created

kubectl get deployments

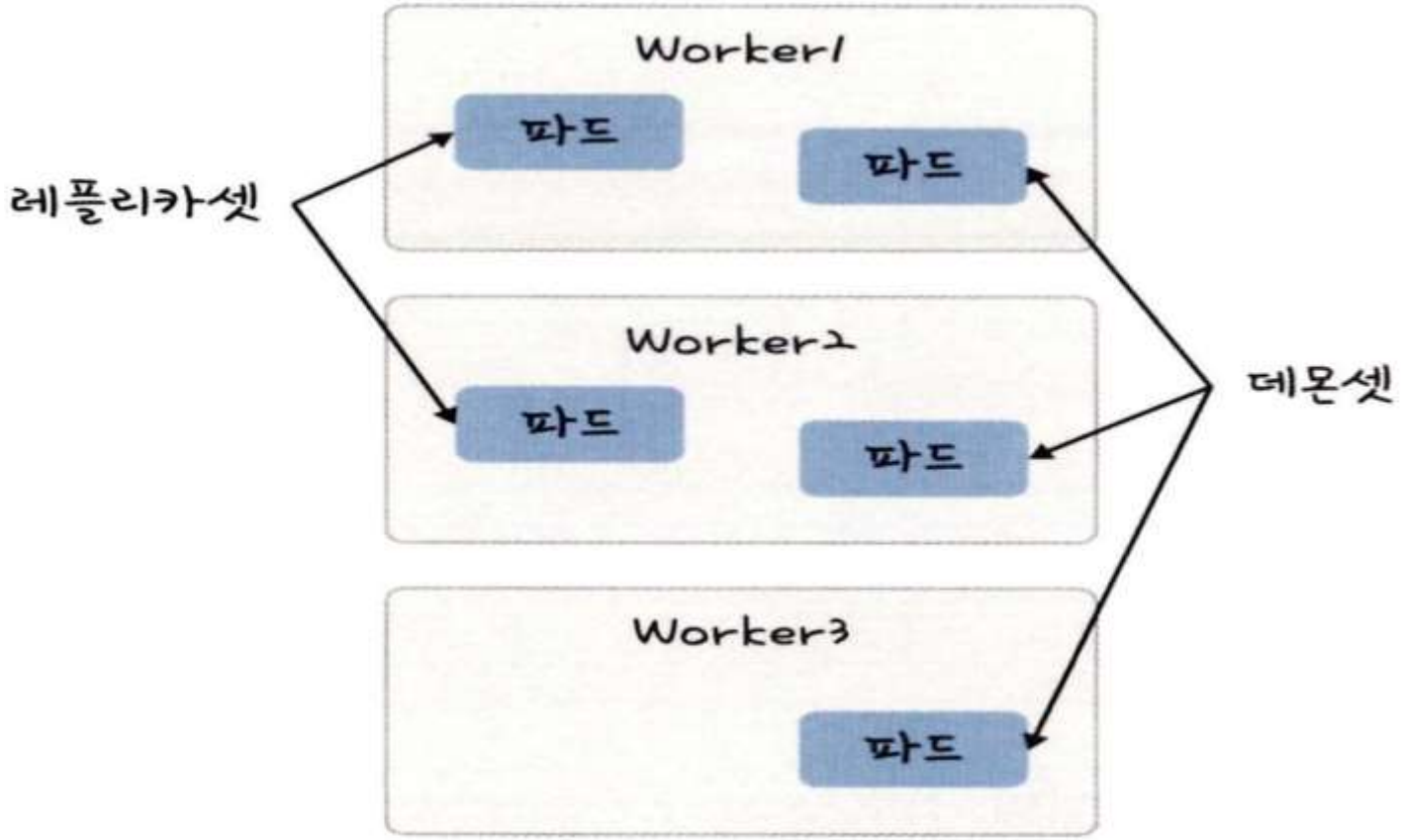
```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
sample-deployment-by-cli            1/1     1            1           14s
sample-deployment-rollingupdate     3/3     3            3           27m
```

Pod

- DaemonSet
  - ✓ 개요
    - ReplicaSet의 특수한 형태
    - DaemonSet(DaemonSet)은 모든 노드에 Pod를 생성
    - ReplicaSet이 특정 개수의 Pod를 유지할 때 사용하는 것이라면 DaemonSet은 모든 노드에 Pod를 배포할 때 사용
    - DaemonSet은 레플리카 수를 지정할 수 없고 하나의 노드에 두 개의 파드를 배치할 수도 없음
    - 파드를 배치하고 싶지 않은 노드가 있을 경우 nodeselector나 노드 안티어피니티를 사용한 스케줄링으로 예외 처리
    - 쿠버네티스 노드를 늘렸을 때도 DaemonSet의 파드는 자동으로 늘어난 노드에서 기동하기 때문에 DaemonSet은 각 파드가 출력하는 로그를 호스트 단위로 수집하는 플루언트디 (Fluentd)나 각 파드 리소스 사용 현황 및 노드 상태를 모니터링하는 데이터독 (Datadog) 등 모든 노드에서 반드시 동작해야 하는 프로세스를 위해 사용하는 것이 유용

Pod

- DaemonSet
  - ✓ 개요



Pod

- DaemonSet
  - ✓ 생성
    - DaemonSet 생성을 위한 yaml 파일 작성 – daemonsets.yaml

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: prometheus-daemonset
spec:
  selector:
    matchLabels:
      tier: monitoring    # 사용자 정의 레이블로 모니터링 용도로 사용할 것임을 지정
      name: prometheus-exporter
  template:
    metadata:
      labels:
        tier: monitoring    #모니터링 용도
        name: prometheus-exporter
    spec:
      containers:
      - name: prometheus
        image: prom/node-exporter
        ports:
        - containerPort: 80
```

Pod

- ❖ DaemonSet
  - ✓ 생성
    - 리소스 배포

      kubectl apply -f daemonsets.yaml

      daemonset.apps/prometheus-daemonset created

    - 상세 정보 확인

      kubectl describe daemonset/prometheus-daemonset

      Name: prometheus-daemonset
      Selector: name=prometheus-exporter,tier=monitoring
      Node-Selector: <none>
      Labels: <none>
      Annotations: deprecated.daemonset.template.generation: 1
      Desired Number of Nodes Scheduled: 1
      Current Number of Nodes Scheduled: 1
      Number of Nodes Scheduled with Up-to-date Pods: 1
      Number of Nodes Scheduled with Available Pods: 1
      Number of Nodes Misscheduled: 0
      Pods Status: 1 Running / 0 Waiting / 0 Succeeded / 0 Failed

Pod

- ❖ DaemonSet
  - ✓ 생성
    - Pod 리스트 확인

      kubectl get pods -o wide

      NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES

      prometheus-daemonset-c9qd2 1/1 Running 0 3m39s 10.244.0.36 minikube <none> <none>
    - DaemonSet 삭제

      kubectl delete -f daemonsets.yaml

      daemonset.apps "prometheus-daemonset" deleted

Pod

- StatefulSet
  - ✓ 개요
    - 스테이트풀셋(StatefulSet)도 DaemonSet과 마찬가지로 레플리카셋의 특수한 형태라고 할 수 있는 리소스
    - 데이터베이스 등과 같은 스테이트풀(stateful)한 워크로드에 사용하기 위한 리소스
    - ReplicaSet 과 차이점
      - 생성되는 파드명의 접미사는 숫자 인덱스가 부여된 것
      - 파드명이 바뀌지 변경되지 않음
      - 데이터를 영구적으로 저장하기 위한 구조
      - 스테이트풀셋에서는 spec.volumeClaimTemplates를 지정할 수 있는데 이 설정으로 스테이트풀셋으로 생성되는 각 파드에 영구 볼륨 클레임(영구 볼륨 요청)을 설정할 수 있으며 영구 볼륨 클레임을 사용하면 클러스터 외부의 네트워크를 통해 제공되는 영구 볼륨을 파드에 연결할 수 있으므로 파드를 재기동할 때나 다른 노드로 이동할 때 같은 데이터를 보유한 상태로 컨테이너가 다시 생성되고 영구 볼륨은 하나의 파드가 소유할 수도 있고 영구 볼륨 종류에 따라 여러 파드에서 공유할 수도 있음

Pod

- StatefulSet
  - ✓ 생성
    - 리소스 파일 생성: sample-statefulset.yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sample-statefulset
spec:
  serviceName: sample-statefulset
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
```

Pod

- StatefulSet
  - ✓ 생성
    - 리소스 파일 생성: sample-statefulset.yaml

```
template:
  metadata:
    labels:
      app: sample-app
  spec:
    containers:
    - name: nginx-container
      image: nginx
      volumeMounts:
      - name: www
        mountPath: /usr/share/nginx/html
volumeClaimTemplates:
- metadata:
    name: www
  spec:
    accessModes:
    - ReadWriteOnce
    resources:
      requests:
        storage: 1G
```

Pod

- StatefulSet
  - ✓ 생성
    - 리소스 생성:

      kubectl apply -f sample-statefulset.yaml

      statefulset.apps/sample-statefulset created
    - 확인

      kubectl get statefulsets

      NAME READY AGE

      sample-statefulset 3/3 40s

Pod

- StatefulSet
  - ✓ 생성
    - 확인

      kubectl get pods -o wide

```
NAME                 READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
sample-statefulset-0   1/1     Running   0          88s   10.244.2.45
minikubecluster-m03   <none>           <none>
sample-statefulset-1   1/1     Running   0          84s   10.244.1.39
minikubecluster-m02   <none>           <none>
sample-statefulset-2   1/1     Running   0          79s   10.244.0.19
minikubecluster       <none>           <none>
```

Pod

- StatefulSet
  - ✓ 생성
    - 확인

      kubectl get persistentvolumeclaims

```
NAME                        STATUS   VOLUME                                     CAPACITY
ACCESS MODES   STORAGECLASS   AGE
www-sample-statefulset-0    Bound    pvc-e617621f-3529-4903-b662-2b25dda8c8fe
1G         RWO            standard       2m15s
www-sample-statefulset-1    Bound    pvc-930ab568-8a2a-45d0-ac94-7ffd04ce4961
1G         RWO            standard       2m11s
www-sample-statefulset-2    Bound    pvc-eefcfe6c-a49a-4a73-9aed-76d6a0a02672
1G         RWO            standard       2m6s
```

Pod

- ❖ StatefulSet
  - ✓ 생성
    - ● 확인

      kubectl get persistentvolumes

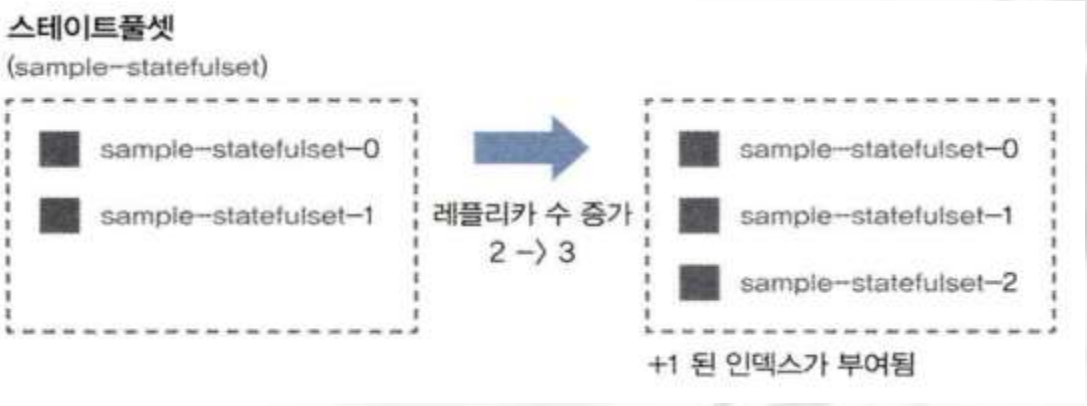
```
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                                  STORAGECLASS   REASON   AGE
pvc-930ab568-8a2a-45d0-ac94-7ffd04ce4961   1G         RWO            Delete
Bound    default/www-sample-statefulset-1      standard                2m29s
pvc-e617621f-3529-4903-b662-2b25dda8c8fe   1G         RWO            Delete
Bound    default/www-sample-statefulset-0      standard                2m33s
pvc-eefcfe6c-a49a-4a73-9aed-76d6a0a02672   1G         RWO            Delete
Bound    default/www-sample-statef
```

Pod

- ❖ StatefulSet
  - ✓ 스케일링
    - 스케일링 수행

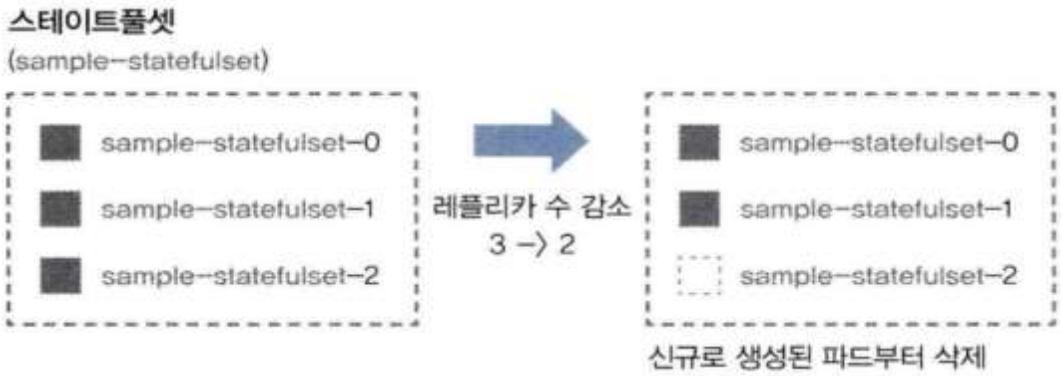
      kubectl scale statefulset sample-statefulset --replicas=5

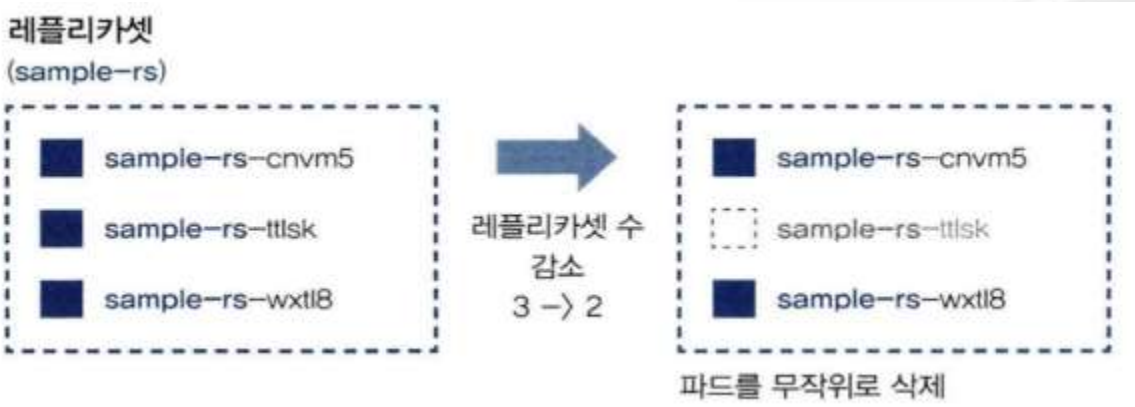
      statefulset.apps/sample-statefulset scaled
    - 스테이트풀셋에서 레플리카 수를 변경하여 파드를 생성하고 삭제하면 레플리카셋이나 DaemonSet 등과 달리 기본적으로 파드를 동시에 하나씩만 생성하고 삭제하기 때문에 조금 시간이 걸림
    - 스케일 아웃일 때는 인덱스가 가장 작은 것부터 파드를 하나씩 생성하고 이전에 생성된 파드가 Ready 상태가 되고 나서 다음 파드를 생성하기 시작하는데 위 예제에서는 sample-statefulset-0, sample-statefulset-1, sample-statefulset-2 순서로 생성



Pod

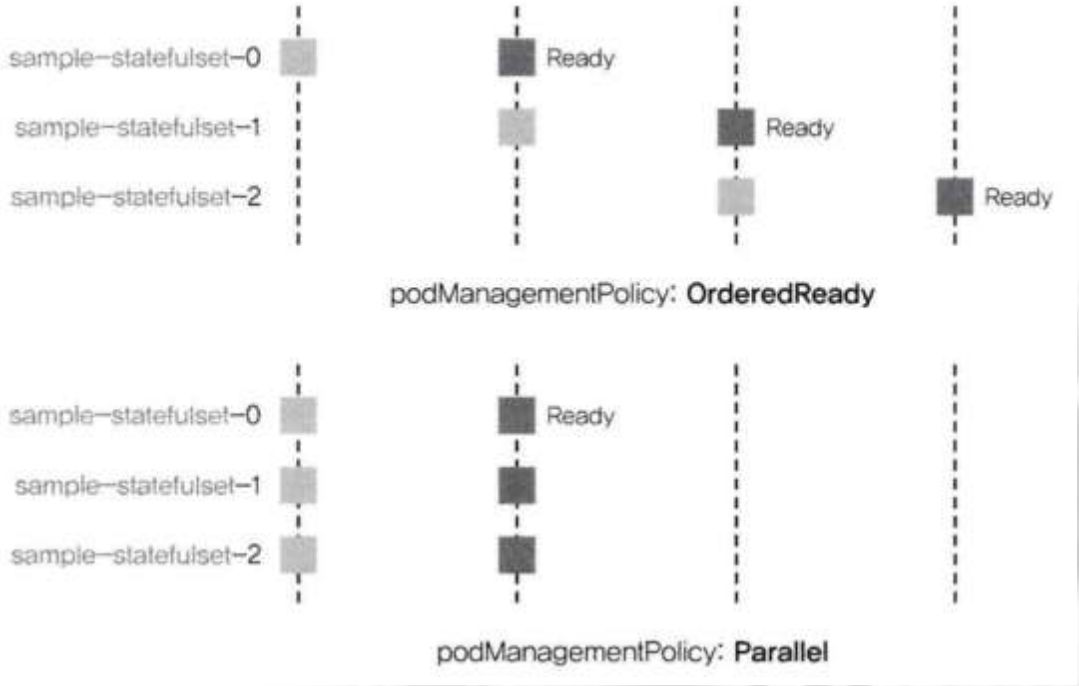
- ❖ StatefulSet
  - ✓ 스케일링
    - ● 스케일 인일 때는 인덱스가 가장 큰 파드(가장 최근에 생성된 컨테이너)부터 삭제되므로 sample-statefulset-2, sample-statefulset-1, sample-statefulset-0 순서로 삭제





Pod

- StatefulSet
  - ✓ 라이프사이클
    - 스테이트풀셋에서도 spec.podManagementPolicy를 Parallel로 설정하여 레플리카셋 등과 마찬가지로 병렬로 동시에 파드를 기동시킬 수 있음
    - spec.podManagementPolicy 기본값은 OrderedReady로 설정되어 있음



Pod

- StatefulSet
  - ✓ 라이프사이클
    - sample-statefulset-parallel.yaml

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sample-statefulset-parallel
spec:
  podManagementPolicy: Parallel
  serviceName: sample-statefulset-parallel
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

Pod

- StatefulSet
  - ✓ 영구 볼륨 데이터 저장 확인
    - 배포

      kubectl apply -f sample-statefulset.yaml

      statefulset.apps/sample-statefulset created
    - 마운트 확인

      kubectl exec -it sample-statefulset-0 -- df -h

```
Filesystem      Size  Used Avail Use% Mounted on
overlay          59G   30G   26G  54% /
tmpfs            64M     0   64M   0% /dev
/dev/vda1        59G   30G   26G  54% /etc/hosts
shm              64M     0   64M   0% /dev/shm
tmpfs           7.7G   12K  7.7G   1% /run/secrets/kubernetes.io/serviceaccount
tmpfs           3.9G     0  3.9G   0% /proc/scsi
tmpfs           3.9G     0  3.9G   0% /sys/firmware
```

Pod

- StatefulSet
  - ✓ 영구 볼륨 데이터 저장 확인
    - 파일 생성 및 존재 여부 확인

      kubectl exec -it sample-statefulset-0 -- ls /usr/share/nginx/html/sample.html

      ls: cannot access '/usr/share/nginx/html/sample.html': No such file or directory
      command terminated with exit code 2

      kubectl exec -it sample-statefulset-0 -- touch /usr/share/nginx/html/sample.html

      kubectl exec -it sample-statefulset-0 -- ls /usr/share/nginx/html/sample.html

      /usr/share/nginx/html/sample.html

Pod

- StatefulSet
  - ✓ 영구 볼륨 데이터 저장 확인
    - 파드를 삭제하거나 컨테이너 내부 에러로 컨테이너가 정된 경우 등의 상황에서도 복구 후에 파일이 그대로 존재

      kubectl delete pod sample-statefulset-0

      pod "sample-statefulset-0" deleted

      kubectl exec -it sample-statefulset-0 -- /bin/bash -c 'kill 1'

      kubectl exec -it sample-statefulset-0 -- ls /usr/share/nginx/html/sample.html

      /usr/share/nginx/html/sample.html

Pod

- StatefulSet
  - ✓ 영구 볼륨 데이터 저장 확인
    - 파드가 복구되면 IP는 변경되지만 이름은 동일

      kubectl get pods -o wide

```
NAME                 READY   STATUS    RESTARTS        AGE     IP           NODE
NOMINATED NODE   READINESS GATES
sample-statefulset-0   1/1     Running   1 (3m49s ago)   4m11s   10.244.2.54
minikubecluster-m03   <none>           <none>
sample-statefulset-1   1/1     Running   0               10m     10.244.1.46
minikubecluster-m02   <none>           <none>
sample-statefulset-2   1/1     Running   0               10m     10.244.0.24
minikubecluster
```

Pod

- StatefulSet
  - ✓ 삭제
    - 스테이트풀셋을 생성하면 파드에 영구 볼륨 클레임(영구 볼륨 요청)을 설정할 수 있어 영구 볼륨도 동시에 확보할 수 있음
    - 영구 볼륨은 스테이트풀셋이 삭제되어도 동시에 해제되지 않음
    - 스테이트풀셋이 영구 볼륨을 해제하기 전에 볼륨에서 데이터를 백업할 수 있도록 시간을 주기 때문
    - 스테이트풀셋을 삭제하고 스테이트풀셋이 영구 볼륨 클레임으로 확보한 영구 볼륨을 해제하지 않고 다시 스테이트풀셋을 생성한 경우 영구 볼륨 데이터는 그대로 파드가 기동되는데 스케일 인하여 레플리카 수를 줄인 경우도 마찬가지

Pod

- StatefulSet
  - ✓ 삭제
    - 스테이트풀셋 삭제

      kubectl delete statefulset sample-statefulset

      statefulset.apps "sample-statefulset" deleted

    - 스테이트풀셋 연결되는 영구 볼륨 클레임 확인

      kubectl get persistentvolumeclaims

      NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

      www-sample-statefulset-0 Bound pvc-e617621f-3529-4903-b662-2b25dda8c8fe 1G RWO standard 89m

      www-sample-statefulset-1 Bound pvc-930ab568-8a2a-45d0-ac94-7ffd04ce4961 1G RWO standard 89m

      www-sample-statefulset-2 Bound pvc-eefcfe6c-a49a-4a73-9aed-76d6a0a02672 1G RWO standard 89m

Pod

- ❖ StatefulSet
  - ✓ 삭제
    - ● 스테이트풀셋 연결되는 영구 볼륨 확인

      kubectl get persistentvolumes

```
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY
STATUS   CLAIM                                 STORAGECLASS   REASON   AGE
pvc-08ab034a-e371-489b-bd63-033994f23cab   1G         RWO            Delete
Bound    default/www-sample-statefulset-3   standard                84m
pvc-6d2d78fc-d140-4105-b1c3-e4d4dab2f292   1G         RWO            Delete
Bound    default/www-sample-statefulset-4   standard                84m
pvc-930ab568-8a2a-45d0-ac94-7ffd04ce4961   1G         RWO            Delete
Bound    default/www-sample-statefulset-1   standard                89m
```

Pod

- StatefulSet
  - ✓ 삭제
    - 스테이트풀셋 다시 생성

      kubectl apply -f sample-statefulset.yaml

      statefulset.apps/sample-statefulset created
    - 파일 존재 여부 확인

      kubectl exec -it sample-statefulset-0 -- ls /usr/share/nginx/html/sample.html

      /usr/share/nginx/html/sample.html

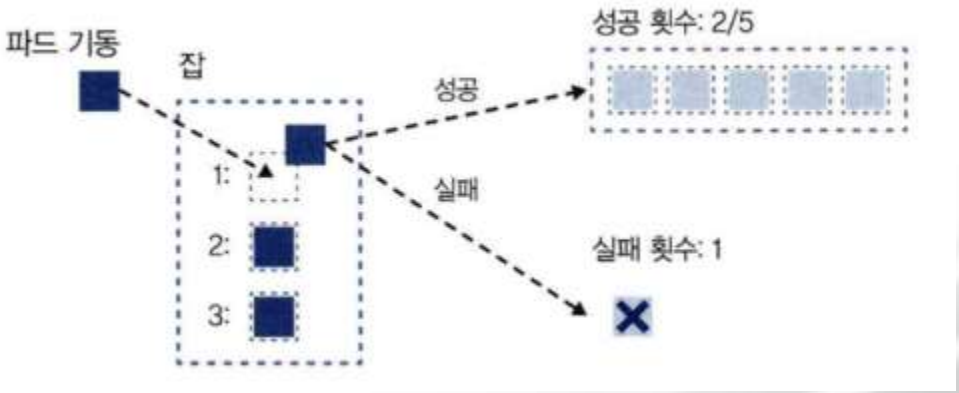
Pod

- StatefulSet
  - ✓ 삭제
    - 스테이트풀셋을 삭제해도 영구 볼륨이 확보된 상태일 경우 비용이 발생
    - 관리형 서비스의 종량 과금 방식인 경우 볼륨을 그대로 유지하면 비용이 발생하고 온프레미스 환경인 경우에도 디스크 공간이 확보된 상태로 남아 있기 때문에 스테이트풀셋을 삭제한 후에는 확보된 영구 볼륨도 해제해야 함
    - 스테이트풀셋이 확보한 영구 볼륨 해제

      kubectl delete persistentvolumeclaims www-sample-statefulset-{0..4}

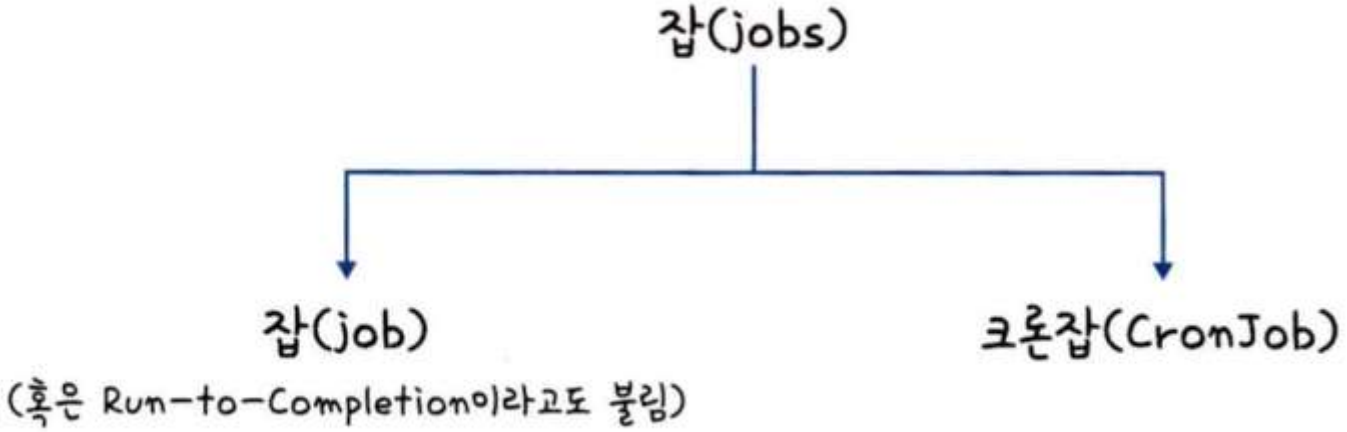
Job

- Job
  - ✓ 개요
    - 잡(Job)은 컨테이너를 사용하여 한 번만 실행되는 리소스
    - N개의 병렬로 실행하면서 지정한 횟수의 컨테이너 실행(정상 종료)을 보장하는 리소스
    - 3개의 병렬 잡으로 5회 실행이 성공할 때까지 파드를 기동



Job

- Job
  - ✓ 개요



- 크론잡은 주기적으로 어떤 액션(action)을 발생시키는 잡으로 어떤 액션을 얼마나 자주 발생시킬지는 매니페스트에서 지정
- 특정 시기에 주기적으로 발생시켜야 하는 크론잡은 애플리케이션이나 데이터를 백업해야 할 때 주로 사용
- 잡과 레플리카셋 간의 큰 차이점은 기동 중인 파드가 정지되는 것을 전제로 만들어졌는지에 있는데 레플리카셋 등에서 파드의 정지는 예상치 못한 에러지만 잡의 경우는 파드의 정지가 정상 종료되는 작업에 적합
- 레플리카셋 등에서는 정상 종료 횟수 등을 셀 수 없기 때문에 배치 처리인 경우에는 잡을 사용

Job

- Job
  - ✓ 생성
    - 파이를 2천자리까지 계산해서 출력하는 잡을 위한 매니페스트 작성: sample-job.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34.0
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Job

- Job
  - ✓ 생성
    - 잡 실행

      kubectl apply -f sample-job.yaml

      job.batch/pi created
    - 잡 확인

      kubectl get jobs

      NAME COMPLETIONS DURATION AGE

      pi 0/1 5s 5s
    - 파드 확인

      kubectl get pods --watch

      NAME READY STATUS RESTARTS AGE

      pi-kdscg 0/1 Completed 0 12s

Job

- Job
  - ✓ 생성
    - 잡 확인

      kubectl get jobs

      NAME COMPLETIONS DURATION AGE
      pi 1/1 6s 26s
    - 결과 확인

      kubectl logs pi-q2f85

      3.14159265358979323846264338327950288419716939937510582097494459230781
      64062862089986280348253421170679821480865132823066470938446095505822 3
      17253594081284811174502841027019385211055596446229489549303819644288 1
      09756659334461284756482337 8

Job

- Job
  - ✓ restartPolicy
    - 잡의 매니페스트에는 spec.template.spec.restartPolicy에 OnFailure 또는 Never 중 하나를 지정해야 하는데 Never를 지정한 경우 파드에 장애가 발생하면 신규 파드가 생성되고 OnFailure의 경우 다시 동일한 파드를 사용하여 잡을 다시 시작
    - Naver 적용
      - yaml 파일 작성: sample-job-never-restart.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: errorjob-unvalidcommand
spec:
  template:
    spec:
      containers:
      - name: errorjob-unvalidcommand
        image: busybox
        command: ["ls", "unvalid path"]
      restartPolicy: Never
```

Job

- ❖ Job
  - ✓ restartPolicy
    - ● Naver 적용
      - ◆ 리소스 생성

        kubectl apply -f sample-job-never-restart.yaml

        job.batch/errorjob-unvalidcommand created

      - ● 파드 확인

        kubectl get pods

        NAME READY STATUS RESTARTS AGE
        errorjob-unvalidcommand-bbc97 0/1 ContainerCreating 0 2s
        errorjob-unvalidcommand-hnflr 0/1 Error 0 16s

Job

- Job
  - ✓ restartPolicy
    - OnFailure 적용
      - ◆ yaml 파일 작성: sample-job-onFailure-restart.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: errorjob-unvalidcommand
spec:
  template:
    spec:
      containers:
      - name: errorjob-unvalidcommand
        image: busybox
        command: ["ls", "unvalid path"]
      restartPolicy: OnFailure
```

Job

- Job
  - ✓ restartPolicy
    - OnFailure 적용
      - ◆ 리소스 생성

        kubectl apply -f sample-job-onfailure-restart.yaml

        job.batch/errorjob-unvalidcommand created

      - 파드 확인

        kubectl get pods

        NAME READY STATUS RESTARTS AGE
        errorjob-unvalidcommand-48wgt 0/1 CrashLoopBackOff 1 (4s ago) 9s

Job

- Job
  - ✓ 태스크 와 작업 큐 병렬 실행
    - 성공 횟수를 지정하는 completions 와 병렬성을 지정하는 parallelism 설정들의 기본 값은 1이기 때문에 명시적으로 지정하지 않아도 동작은 변하지 않음
    - 이 두 파라미터는 잡을 병렬화하여 실행할 때 사용하는 옵션
    - backoffLimit는 실패를 허용하는 횟수
    - 병렬 잡 과 작업 큐 파라미터 설정(M, N, P는 임의의 수)

워크로드	completions	parallelism	backoffLimit
1회만 실행하는 태스크	1	1	0
N개 병렬로 실행하는 태스크	M	N	P
한 개씩 실행하는 작업 큐	미지정	1	P
N개 병렬로 실행하는 작업 큐	미지정	N	P

    - 파라미터들 중에서 성공 횟수(completions)는 나중에 변경할 수 없으며 병렬성(parallelism) 과 실패를 허용하는 횟수(backoffLimit)는 도중에 변경할 수 있는데 변경하려면 매니페스트를 수정하고 kubectl apply 명령어를 실행

Job

- Job
  - ✓ 일정 시간 후 잡 삭제
    - 잡 리소스는 한 번만 실행하는 태스크로 생성하는 경우가 있는데, 잡은 종료 후에 삭제되지 않고 계속 남는데 spec.ttlSecondsAfterFinished를 설정하여 잡이 종료한 후에 일정 기간(초) 경과 후 삭제하도록 설정할 수 있음
    - 작업 종료 후 30초 후 삭제
      - yaml 파일 작성: sample-job-ttl.yaml

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  ttlSecondsAfterFinished: 30
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34.0
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Job

- Job
  - ✓ 일정 시간 후 잡 삭제
    - ● 작업 종료 후 30초 후 삭제
      - ◆ 리소스 생성

        kubectl apply -f sample-job-ttl.yaml

        job.batch/pi created
      - ◆ 파드 확인

        kubectl get job pi --watch --output-watch-events

        EVENT NAME COMPLETIONS DURATION AGE
        ADDED pi 1/1 6s 32s
        MODIFIED pi 1/1 6s 36s
        DELETED pi 1/1 6s 36s

Job

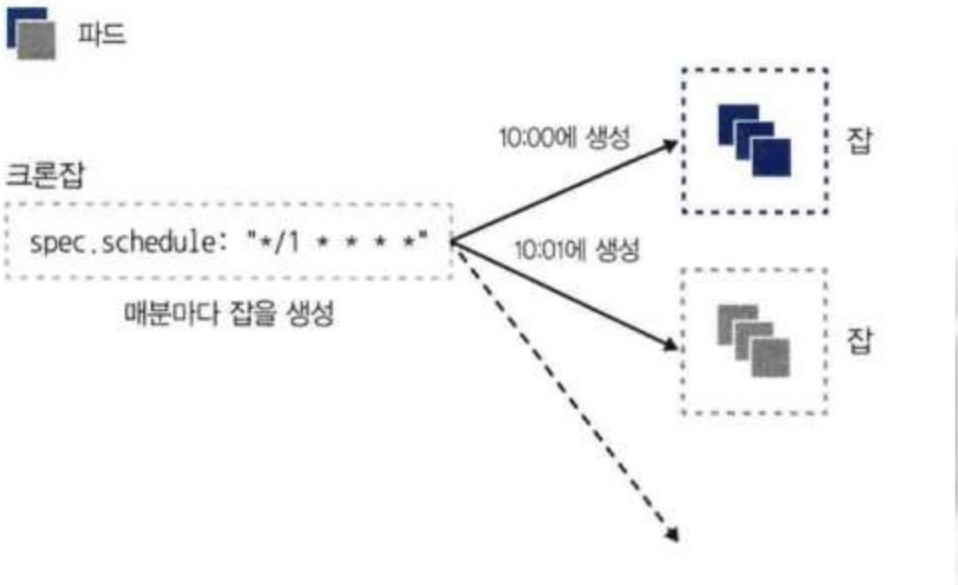
- Job
  - ✓ 매니페스트를 사용하지 않고 잡을 생성

    kubectl create job my-job --image=busybox -- date

    job.batch/my-job created

Job

- ❖ CronJob
  - ✓ 개요
    - 다른 매니페스트와 다르게 schedule(얼마나 자주), command(어떤 행동을 수행) 부분이 추가 됨



Job

- CronJob
  - ✓ 이 매니페스트는 busybox 이미지를 통해 Pod를 생성하고 생성된 Pod는 1분 단위(schedule)로 date; echo Hello from the Kubernetes cluster 메시지를 출력(command)하는 크론잡
    - yaml 파일 작성 – cronjob.yaml

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"     #스케쥴 지정
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            imagePullPolicy: IfNotPresent     #이미지가 없을 때에만 내려받음
            command:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure     #컨테이너에서 실행된 프로세스가 실패할 때에만 다시 시작
```

Job

- CronJob
  - ✓ 이 매니페스트는 busybox 이미지를 통해 Pod를 생성하고 생성된 Pod는 1분 단위(schedule)로 date; echo Hello from the Kubernetes cluster 메시지를 출력(command)하는 크론잡
    - 크론잡 생성

      kubectl apply -f cronjob.yaml

      job.batch/my-job created

    - 작업 확인

      kubectl get pods

      NAME READY STATUS RESTARTS AGE
      hello-28909935-66pzd 0/1 Completed 0 17s

      kubectl logs hello-28909935-66pzd

      Thu Dec 19 08:15:00 UTC 2024
      Hello from the Kubernetes cluster

Job

- CronJob
  - ✓ 이 매니페스트는 busybox 이미지를 통해 Pod를 생성하고 생성된 Pod는 1분 단위(schedule)로 date; echo Hello from the Kubernetes cluster 메시지를 출력(command)하는 크론잡
    - 크론 잡 상태 확인

      kubectl get cronjob hello

      NAME SCHEDULE SUSPEND ACTIVE LAST SCHEDULE AGE

      hello */1 * * * * False 0 <none> 34

      - LAST SCHEDULE 상태가 34로 출력
      - 스케줄을 34초 전에 실행했다는 의미
      - 경우에 따라<none>으로 출력될 수도 있는데 시간이 좀 지나면 숫자로 바뀔 것

Job

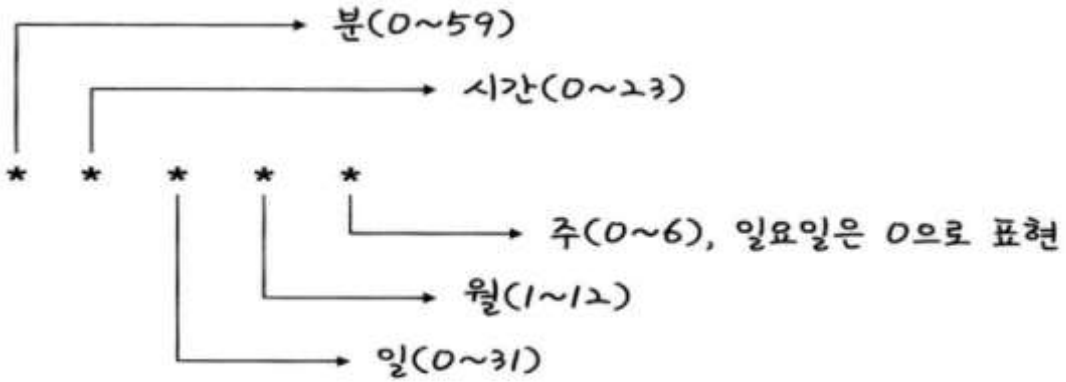
- CronJob
  - ✓ 이 매니페스트는 busybox 이미지를 통해 Pod를 생성하고 생성된 Pod는 1분 단위(schedule)로 date; echo Hello from the Kubernetes cluster 메시지를 출력(command)하는 크론잡
    - 크론 잡에 대해서 자세히 알아보기

      kubectl get cronjob -w

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE
hello	*/1 * * * *	False	1	1s	59s
hello	*/1 * * * *	False	0	8s	66s
hello	*/1 * * * *	False	0	8s	66s

Job

- CronJob
  - ✓ 스케쥴의 의미



Job

- CronJob
  - ✓ 크론 잡 삭제

    kubectl delete cronjob hello

    cronjob.batch "hello" deleted

Job

- CronJob
  - ✓ 일시 정지

```
kubectl patch cronjob hello -p '{"spec":{"suspend":true}}'
```

```
cronjob.batch/hello patched
```

```
kubectl patch cronjob hello -p '{"spec":{"suspend":false}}'
```

```
cronjob.batch/hello patched
```

Job

- CronJob
  - ✓ 정기적 실행 이외의 시점에 실행

    kubectl create job hello-job-from-cronjob --from cronjob/hello

    job.batch/hello-job-from-cronjob created

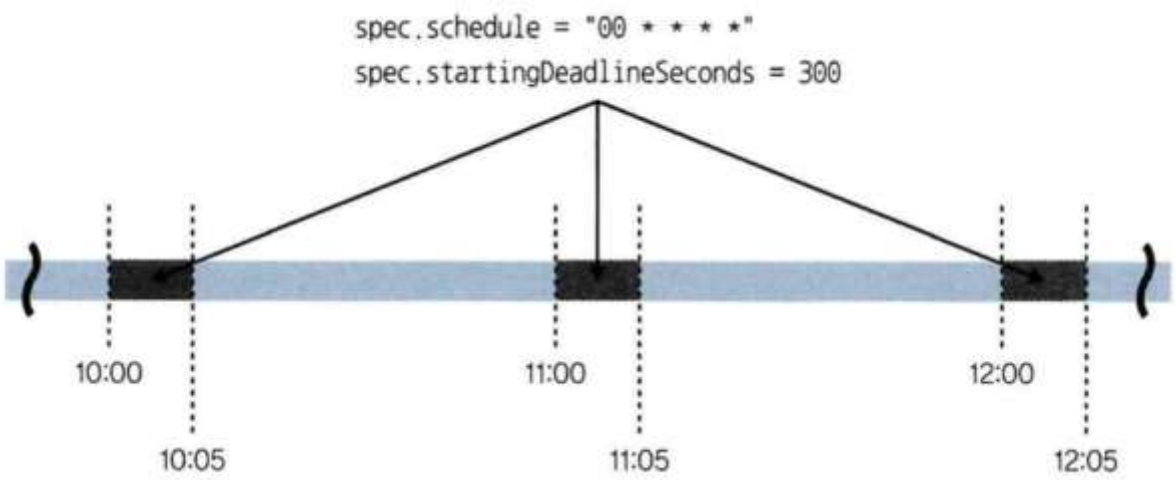
Job

- CronJob
  - ✓ 동시 실행 제어
    - 크론잡에서는 잡을 생성하는 특성상 동시 실행에 대한 정책을 설정할 수 있음
    - 동시 실행에 대한 정책은 spec.concurrencypolicy에 지정

정책	개요
Allow	기본값으로 동시 실행에 대한 제한을 하지 않음
Forbid	이전 잡이 종료되지 않았을 경우 다음 잡은 실행하지 않음
Replace	이전 잡을 취소하고 잡을 시작

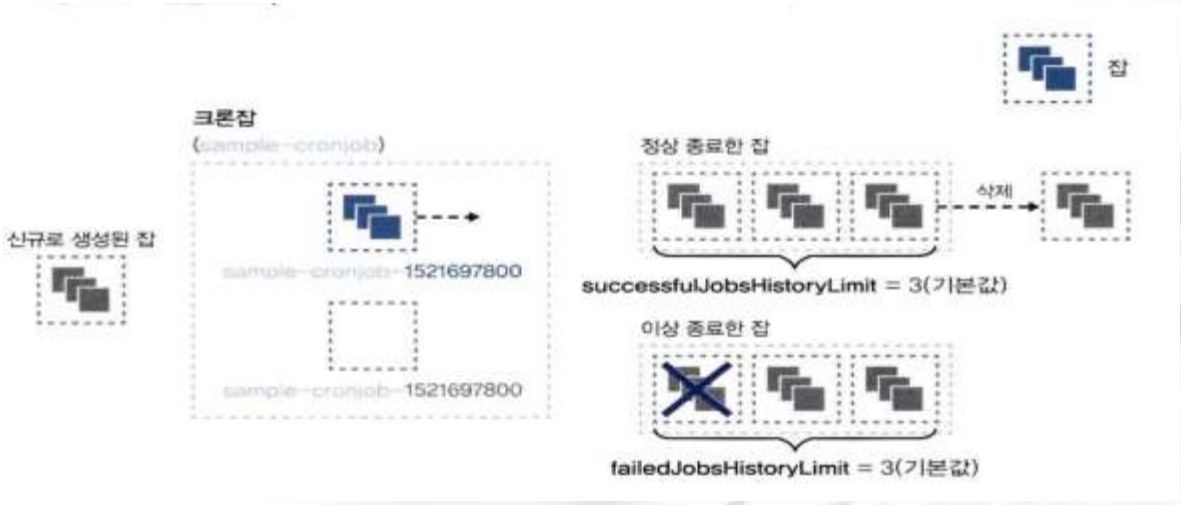
Job

- CronJob
  - ✓ 실행 시작 기한 제어
    - 크론잡은 지정한 시간이 되면 쿠버네티스 마스터가 잡을 생성하기 때문에 쿠버네티스 마스터가 일시적으로 정지되는 경우 등과 같이 시작 시간이 지연되면 그 지연 시간을 허용하는 시간(초 - spec.startingDeadlineSeconds) 지정할 수 있음
    - 매시 00분에 시작하는 잡을 매시 00~05분에만 실행 가능으로 설정할 경우 300초로 설정해야 할 수 있는데 기본값에서는 시작 시간이 아무리 늦어져도 잡을 생성하게 되어 있음



Job

- CronJob
  - ✓ 크론잡 이력
    - 크론잡 설정 항목에는 이외에도 저장할 잡 개수를 지정하는 spec.successfulJobsHistoryLimit(성공한 잡을 저장하는 개수) 와 spec.failedOobsHistoryLimit(실패한 잡을 저장하는 개수)가 있음
    - 잡이 남아 있다는 것은 잡에 연결되어 있는 파드도 Completed(정상 종료), Error(이상 종료)의 상태로 남아 있고 kubectl log 명령어로 로그를 확인할 수 있다는 의미
    - 실제 운영 환경에서는 컨테이너 로그를 외부 로그 시스템을 통해 운영하는 것을 추천하는데 로그를 별도 로그 시스템으로 수집하면 kubectl 명령어로 로그를 확인할 필요도 없고 가용성이 높은 환경에 로그를 저장할 수 있음



node

- ❖ 노드 자원 보호
  - ✓ 개요
    - 노드는 쿠버네티스 스케줄러에서 파드를 할당받고 처리하는 역할을 수행
    - 최근에 몇 차례 문제가 생긴 노드에 파드를 할당하면 문제가 생길 가능성이 높은데 노드를 제거하면 좋지만 어쩔 수 없이 해당 노드를 사용해야 한다면 파드의 문제를 최소화해야 하는데 쿠버네티스는 모든 노드에 균등하게 파드를 할당하려고 함
    - 쿠버네티스에서는 이런 경우에 cordon 기능을 이용해서 노드에 더 이상 파드를 배포하지 않도록 함

node

- ❖ 노드 자원 보호
  - ✓ 실습
    - sample-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - name: nginx-container
        image: nginx
```

    - 배포: kubectl apply -f sample-deployment.yaml

node

- ❖ 노드 자원 보호
  - ✓ 실습
    - ● 파드에서 원하는 정보를 추출하기 위한 조회:
      - ◆ kubectl get pod sample-deployment-556b965888-ckgxq -o yaml > pod.yaml
      - ◆ nano pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2025-10-14T06:50:34Z"
  generateName: sample-deployment-556b965888-
  labels:
    app: sample-app
    pod-template-hash: 556b965888
  name: sample-deployment-556b965888-ckgxq
  namespace: default
  ownerReferences:
  - apiVersion: apps/v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicaSet
    name: sample-deployment-556b965888
    uid: 240d7111-a9da-454a-9921-66fc517c183e
```

node

- ❖ 노드 자원 보호
  - ✓ 실습
    - 파드 확인: kubectl get pods -o custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName

```
NAME                                IP           STATUS    NODE
sample-deployment-556b965888-ckgxq  10.244.2.2   Running   worker1
sample-deployment-556b965888-wq92s  10.244.1.2   Running   worker2
```

    - 파드 개수 증가: kubectl scale deployment sample-deployment --replicas=4
    - 파드 확인: kubectl get pods -o custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName

```
NAME                                IP           STATUS    NODE
sample-deployment-556b965888-7k9xg  10.244.1.3   Running   worker2
sample-deployment-556b965888-ckgxq  10.244.2.2   Running   worker1
sample-deployment-556b965888-vd97z  10.244.2.3   Running   worker1
sample-deployment-556b965888-wq92s  10.244.1.2   Running   worker2
```

node

- ❖ 노드 자원 보호
  - ✓ 실습
    - 파드 개수 복원: kubectl scale deployment sample-deployment --replicas=2
    - 파드 확인: kubectl get pods -o custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName

```
NAME                                IP           STATUS    NODE
sample-deployment-556b965888-ckgxq  10.244.2.2   Running   worker1
sample-deployment-556b965888-wq92s  10.244.1.2   Running   worker2
```

node

- ❖ 노드 자원 보호
  - ✓ 실습
    - 노드의 현재 상태 보존: kubectl cordon worker1

      node/worker1 cordoned
    - 노드의 현재 상태 확인: kubectl get nodes

      NAME STATUS ROLES AGE VERSION

      master Ready control-plane 29m v1.30.14

      worker1 Ready,SchedulingDisabled <none> 28m v1.30.14

      worker2 Ready <none> 28m v1.30.14
    - 파드 개수 증가: kubectl scale deployment sample-deployment --replicas=4
    - 파드 확인: kubectl get pods -o custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName

      NAME IP STATUS NODE

      sample-deployment-556b965888-ckgxq 10.244.2.2 Running worker1

      sample-deployment-556b965888-hpg2c 10.244.1.4 Running worker2

      sample-deployment-556b965888-pfpfw 10.244.1.5 Running worker2

      sample-deployment-556b965888-wq92s 10.244.1.2 Running worker2

node

- ❖ 노드 자원 보호
  - ✓ 실습
    - 파드 개수 복원: kubectl scale deployment sample-deployment --replicas=2
    - 파드 확인: kubectl get pods -o custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName

      ```
      NAME                                IP           STATUS    NODE
      sample-deployment-556b965888-ckgxq  10.244.2.2   Running   worker1
      sample-deployment-556b965888-hpg2c  10.244.1.4   Running   worker2
      ```

    - 노드의 현재 상태 보존 취소: kubectl uncordon worker1

      ```
      node/worker1 cordoned
      ```

    - 노드의 현재 상태 확인: kubectl get nodes

      ```
      NAME      STATUS     ROLES           AGE   VERSION
      master    Ready      control-plane   29m   v1.30.14
      worker1   Ready      <none>          28m   v1.30.14
      worker2   Ready      <none>          28m   v1.30.14
      ```

node

- ❖ 노드 유지 보수
  - ✓ 개요
    - 쿠버네티스를 사용하다 보면 정기 또는 비정기적인 유지보수를 위해 노드를 꺼야 하는 상황이 발생하는데 이런 경우를 대비해 drain 기능을 제공
    - drain은 지정된 노드의 파드를 전부 다른 곳으로 이동시켜 해당 노드를 유지보수 할 수 있게 함
    - DaemonSet은 지울 수 없어서 DaemonSet이 있는 경우는 명령을 수행할 수 없음
    - drain은 실제로 파드를 옮기는 것이 아니라 노드에서 파드를 삭제하고 다른 곳에 다시 생성함
    - DaemonSet은 각 노드에 1개만 존재하는 파드라서 drain으로는 삭제 할 수 없음
    - Drain 기능을 수행할 때 --ignore-daemonsets 옵션을 추가하면 경고를 표시하지만 DaemonSet을 무시하고 진행

node

- ❖ 노드 유지 보수
  - ✓ 실습
    - worker1을 drain: kubectl drain worker1 --ignore-daemonsets
    - 파드 확인: kubectl get pods -o custom-columns=NAME:.metadata.name,IP:.status.podIP,STATUS:.status.phase,NODE:.spec.nodeName

      NAME IP STATUS NODE
      sample-deployment-556b965888-6st6h 10.244.1.6 Running worker2
      sample-deployment-556b965888-wq92s 10.244.1.2 Running worker2
    - 노드의 현재 상태 확인: kubectl get nodes

      NAME STATUS ROLES AGE VERSION
      master Ready control-plane 29m v1.30.14
      worker1 Ready,SchedulingDisabled <none> 28m v1.30.14
      worker2 Ready <none> 28m v1.30.14

node

- ❖ 노드 유지 보수
  - ✓ 실습
    - worker1을 복구: kubectl uncordon worker1
    - 노드의 현재 상태 확인: kubectl get nodes

```
NAME      STATUS     ROLES           AGE   VERSION
master    Ready      control-plane   29m   v1.30.14
worker1   Ready      <none>          28m   v1.30.14
worker2   Ready      <none>          28m   v1.30.14
```