

# Kubernetes\_Label & Controller

## - 실습가이드

최초 작성일 : 2024/02/29

최종 제출일 : 2024/03/04

김민경

### 목차

I. 개념	2
1. POD	2
1-1. 개념	2
1-2. 파드의 상태	2
1-3. restartPolicy (재시작 정책)	3
1-4. probe	3
2. Label & Selector	4
3. Controller	5
3-1. 개념	5
3-2. 종류	5
II. 실습	9
1. Label 활용하기	9
2. Selector 활용하기	13
3. 커맨드(Command) 및 Arguments(인자) 설정하기	14
4. Env 설정하기	15
5. Hostpath 볼륨 설정하기	16
6. Probe	18
7. Onfailure (exit 1) 설정하기	19
8. Onfailure (exit 0) 설정하기	21
9. Livenessprobe 설정하기	22
10. Replicas 설정하기	25
11. Replicaset의 라벨과 동일한 pod 생성하기	28
12. Deployment 설정하기	30
13. Rolling update 설정하기	33

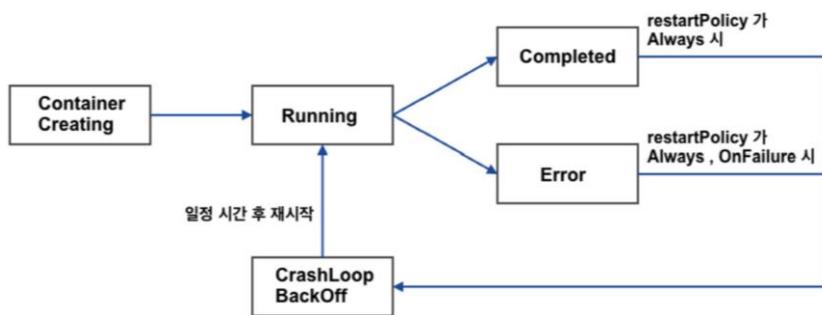
# I. 개념

## 1. POD

### 1-1. 개념

- 쿠버네티스가 생성하고 관리하는 가장 작은 컴퓨팅 단위

### 1-2. 파드의 상태



- ① **Pending** : 클러스터 내 파드 생성이 승인되었지만 하나 이상의 컨테이너가 설정되지 않았고 실행할 준비가 되지 않은 상태
- ② **Running** : 파드가 생성돼 정상적으로 실행된 상태
- ③ **Completed** : 정상적으로 실행되고 끝난 상태, 정상적으로 종료되면 '0' 표시를 보냄
- ④ **Error** : 파드 안의 컨테이너 중 하나의 컨테이너가 실패하여 종료된 상태
- ⑤ **CrashLoopBackOff** : 컨테이너가 반복해서 시작되고 죽는 오류 상태
- ⑥ **Unknown** : 어떤 이유로 파드와 통신할 수 없는 상태
- ⑦ **Terminating** : 파드가 삭제되기 위해 삭제 상태에 머물러 있는 경우

### 1-3. restartPolicy (재시작 정책)

- 파드에 포함된 모든 각각의 컨테이너에는 오류 발생 시 재시작을 하기 위한 정책을 가짐
- 파드의 yaml 중 spec.restartPolicy를 통해 직접 부여할 수 있음
- 이 정책은 크게 3종류로 구분됨
  - ① **Always** : 컨테이너가 정상/비정상 종료 시 항상 재시작 함(기본값)
  - ② **Onfailure** : 컨테이너가 비정상 종료되었을 때만 재시작 함
  - ③ **Never** : 재시작 하지 x

- 재시작 설정은 일정한 지연시간을 두고 이루어짐
  - 최대 300초 한도 내, 10s의 단위로 반영됨
  - ex) 첫번째 재시작 시도는 10s 후에, 그 다음번의 재시작 시도는 20s 후에, 그 다음은 40s 후
  - 만약 재시작에 성공한 컨테이너가 10분 동안 이상 x이 구동 시 해당 지연 시간은 초기화 됨

#### **1-4. probe**

##### 1) 개념

- probe를 활용하여 컨테이너의 상태를 체크해 재시작 정책을 수행하게 됨
- probe는 kubelet이 파드 안의 컨테이너 상태를 진단하여 컨테이너가 재시작/종료가 필요한지 그 여부를 확인할 수 있게 함

##### 2) 동작 방식

- ① **HTTP 방식** : HTTP GET 요청을 보내서 2XX 또는 3XX 응답이 오지 x을 경우 애플리케이션의 상태 검사가 실패한 것으로 간주함
- ② **TCP 방식** : TCP의 3-Way Handshake가 잘 맺어졌는지 체크함
- ③ **Exec 방식** : 컨테이너 내부에 명령어를 실행해 상태 검사 함. 명령어 종료 코드가 0이 x는 경우 애플리케이션의 상태 검사가 실패한 것으로 간주함  
⇒ 이러한 진단 결과는 success, failure, unknown으로 구분됨

##### 3) 종류

- ① **Liveness Probe**: 컨테이너가 멈추거나 비정상 종료되면 해당 컨테이너를 재시작함
- ② **Readiness Probe**: 컨테이너 요청을 처리할 수 x는 상황이 감지되면 해당 파드로의 로드밸런싱을 중단함 (파드 자체는 건드리지 X)
- ③ **Startup Probe**: 컨테이너가 시작되어 정상 구동되는 시점을 감지함. 그 전까지는 다른 프로브 (Liveness, Readiness)의 동작을 막고, 해당 시점을 기준으로 자기 자신의 역할을 종료한 뒤 프로브 기능을 다른 프로브들에게 넘김

## 2. Label & Selector

### 2-1. Label

- 리소스 식별 시 사용됨
- 쿠버네티스 오브젝트에 key-value 형식으로 Label(태그)을 붙임
- 파드, 서비스, 노드 등의 쿠버네티스 오브젝트에 라벨을 추가하여 특정 그룹으로 그들을 식별할 수 있음

```
apiVersion: v1
kind: Pod
metadata:
  name: myweb1
  labels:
    environment: production
    app: nginx
spec:
  containers:
    - image: nginx
      name: myweb1
    ports:
      - containerPort: 80
        protocol: TCP
```

### 2-2. Selector

- Label을 사용하여 특정 그룹의 리소스를 선택하는데 사용됨
- Selector 사용 시 특정 라벨이나 라벨의 조합을 기반으로 리소스를 쿼리할 수 있음

```
#labels-node
cat > labels-node.yaml
apiVersion: v1
kind: Pod
metadata:
  name: myweb
spec:
  containers:
    - image: nginx
      name: myweb
    ports:
      - containerPort: 80
        protocol: TCP
  nodeSelector:
    cloud: babo
```

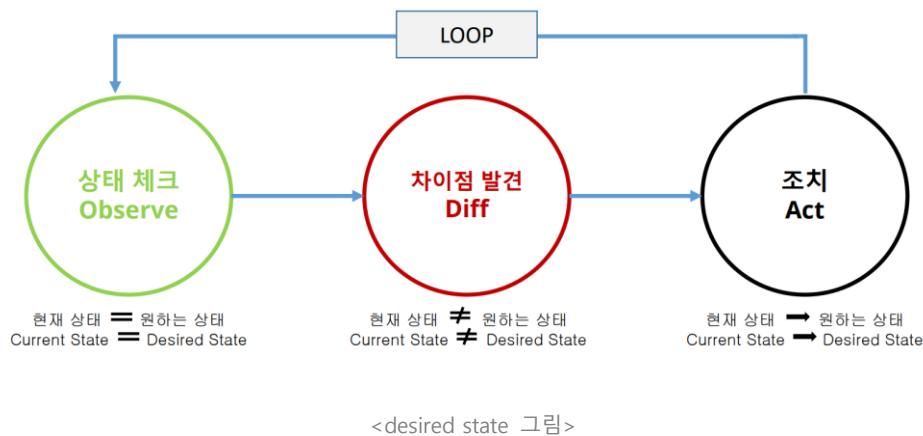
정리)

- **Label** : 리소스 식별
- **Selector** : 필요한 리소스 선택

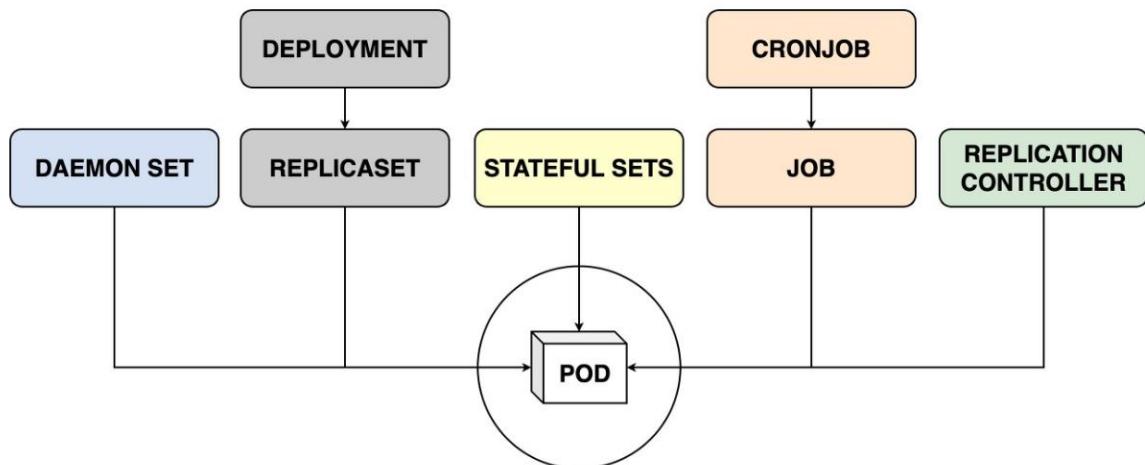
### 3. Controller

#### 3-1. 개념

- K8S는 여러 가지 컨트롤러를 제공하여 파드와 다른 리소스의 상태를 관리함
- Controller는 클러스터의 상태를 감시하고 current state와 desired state가 일치하도록 관리하는 역할



#### 3-2. 종류



##### ① Replication controller (RC)

- k8s의 초기버전에서 도입된 가장 basic한 컨트롤러
- 실행할 파드의 개수 지정 가능하고, 지정한 개수만큼 파드가 유지되도록 관리함
- but label selector에 유연성이 x음

- 등가성 (equality-based) 셀렉터만을 지원해 label의 key와 value가 정확히 일치해야 함

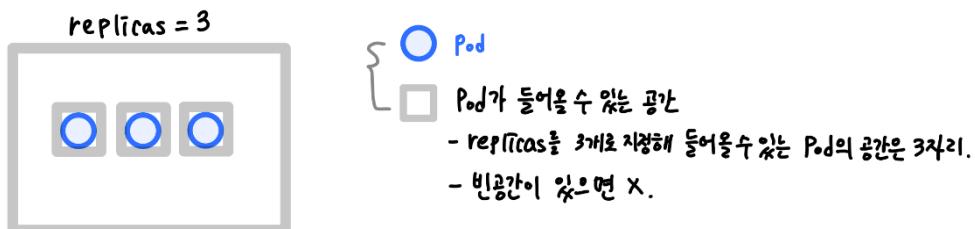
#### 참고) 등가성 셀렉터

- 예시) 만약 '색깔이 빨간색인 사과'만 팔고 싶다면
  - key: 색깔이고, value: 빨간색임
  - 정확히 빨간색인 사과만 선별됨

⇒ 이를 보완하기 위해 ReplicaSet이 등장

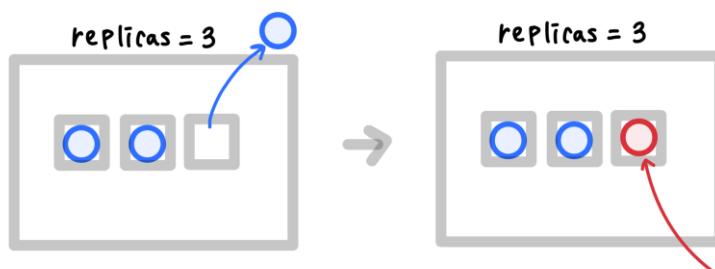
(현재는 ReplicaSet이 Replication controller를 완전히 대체함)

## ② ReplicaSet (RS)

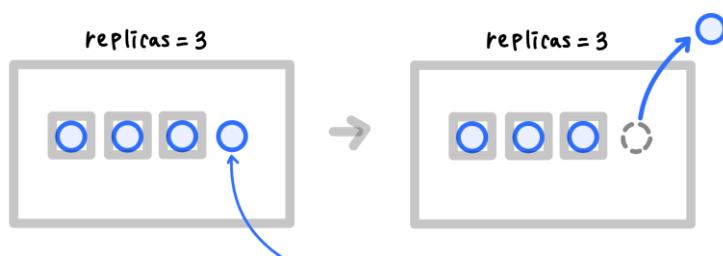


- RC의 차세대 버전
- RC처럼 실행할 파드의 개수 지정 가능하고, 지정한 개수만큼 파드가 유지되도록 관리함
  - 만약 replicas=3으로 지정해 pod가 3개여야하지만 하나가 사라졌다면

아래의 오른쪽 빨간색 pod처럼 그 자리가 다시 채워져야 함



- 만약 pod 한 개가 더 들어오면 더 이상 pod가 채워질 공간이 없기 때문에 다시 빼내야 함



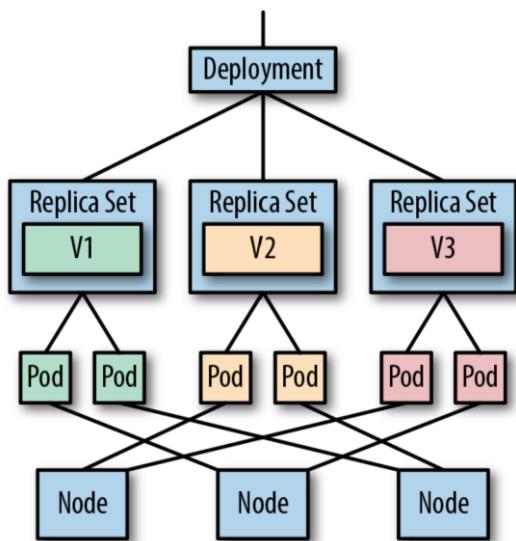
- label selector에 유연성이 있음
  - 집합성(set-based) 셀렉터를 지원하여 보다 복잡한 선택 기준을 사용할 수 있음  
(key가 여러 개라도 선택 가능)

**참고) 집합성 셀렉터**

- 예시) 만약 '색깔이 빨간색이거나 초록색인 사과'를 팔고 싶다면
  - key: 색깔인데, value: 빨간색, 초록색임
  - 여러 value 중에 하나를 선택할 수 있음

- Deployment와 함께 사용되어 파드의 룰 아웃과 룰 백을 관리하는데 사용됨

### ③ Deployment



- RS의 부모 역할 (Deployment를 만들면 RS가 만들어짐)
  - Deployment는 실행시킬 Pod의 버전과 replicas를 ReplicaSet에게 전달함.  
그 후에 ReplicaSet이 전달받은 정보를 기반으로 Pod를 생성하고 Pod 수를 보장하게 됨

⇒ Deployment가 ReplicaSet을 컨트롤하고, ReplicaSet이 Pod를 컨트롤하는 구조

- Rolling update(애플리케이션 업데이트)를 지원함

**참고) Rolling update**

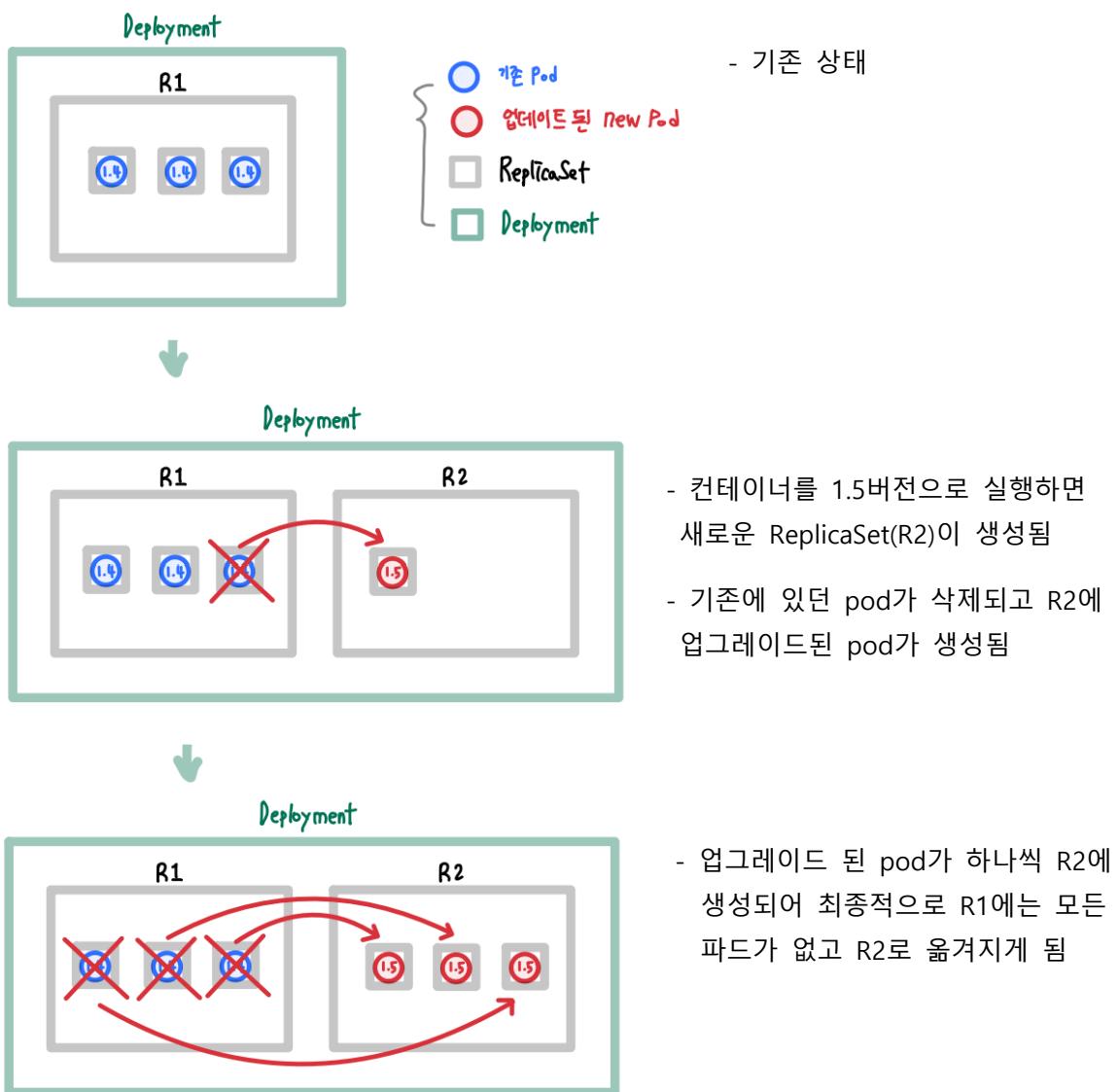
- pod를 하나씩 점진적으로 업데이트해 새로운 버전으로 만들어 Deployment 업데이트가 서비스 중단 없이 이루어질 수 있도록 해 줌

- Rolling back(배포)을 지원함

#### 참고) Rolling back

- 이전 버전으로 되돌아 가는 것

- 동작과정



## II. 실습

\* 전제) alias 설정으로 kubectl → k로 설정해놓은 상태

### 1. Label 활용하기

#### 1-1. Vagrantfile을 label.yaml 내용으로 변경하기

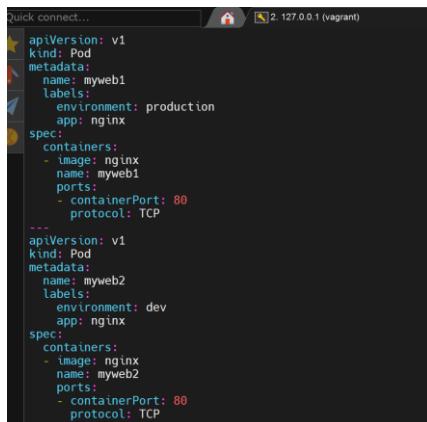
- label.yaml 내용을 모바일스팀에 전체 복붙하기

```
cat > label.yaml
apiVersion: v1
kind: Pod      // 항상 kind를 확인하기
metadata:
  name: myweb1
  labels:
    environment: production
    app: nginx
spec:
  containers:
    - image: nginx
      name: myweb1
      ports:
        - containerPort: 80
          protocol: TCP
---
apiVersion: v1
kind: Pod
metadata:
  name: myweb2
  labels:
    environment: dev
    app: nginx
spec:
  containers:
    - image: nginx
      name: myweb2
      ports:
        - containerPort: 80
          protocol: TCP
```

- but 아래처럼 잘 안되고 결과값도 이상하게 나와서 파일 자체를 하나 생성하고, 그 파일 내용을 수정하는 방식으로 진행함



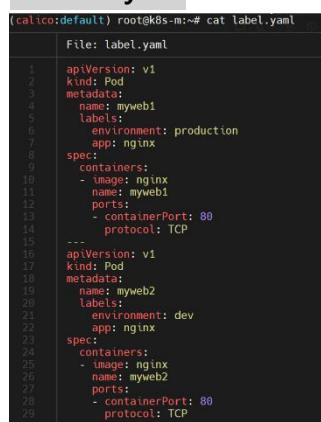
- vim label.yaml (첫째줄 'cat > ~'부분은 빼고 복붙하기)



```
apiVersion: v1
kind: Pod
metadata:
  name: myweb1
  labels:
    environment: production
    app: nginx
spec:
  containers:
    - image: nginx
      name: myweb1
      ports:
        - containerPort: 80
          protocol: TCP
...
apiVersion: v1
kind: Pod
metadata:
  name: myweb2
  labels:
    environment: dev
    app: nginx
spec:
  containers:
    - image: nginx
      name: myweb2
      ports:
        - containerPort: 80
          protocol: TCP
```

### 1-2. yaml 파일이 잘 만들어졌는지 확인하기

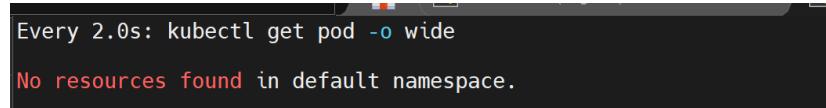
cat label.yaml



```
(calico:default) root@k8s-m:~# cat label.yaml
File: label.yaml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: myweb1
5    labels:
6      environment: production
7      app: nginx
8  spec:
9    containers:
10   - image: nginx
11     name: myweb1
12     ports:
13       - containerPort: 80
14         protocol: TCP
...
15  apiVersion: v1
16  kind: Pod
17  metadata:
18    name: myweb2
19    labels:
20      environment: dev
21      app: nginx
22  spec:
23    containers:
24      - image: nginx
25        name: myweb2
26        ports:
27          - containerPort: 80
28            protocol: TCP
```

### 1-3. 모니터링용 탭 추가 생성하기

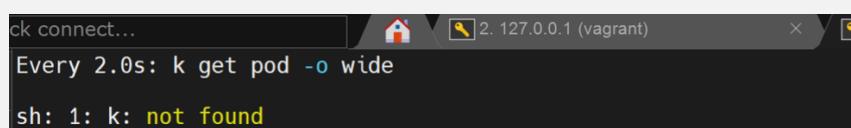
watch -d 'kubectl get pod -o wide'



```
Every 2.0s: kubectl get pod -o wide
No resources found in default namespace.
```

- 현재는 아무것도 만들지 않아 no resources found라고 뜸

참고) watch -d 'k get pod -o wide' 이렇게 alias 설정 해놓은걸로 명령어 치면 모니터링이 됨



```
Every 2.0s: k get pod -o wide
sh: 1: k: not found
```

#### 1-4. label.yaml 파일 실행하기

```
k apply -f label.yaml
```

- 결과값 확인하기

- **k get pod** //pod 확인하기

- 모니터링으로도 확인해보기

```
ck connect... | 2. 127.0.0.1 (vagrant) | 6. 127.0.0.1 (vagrant) | 4. 127.0.0.1 (vagrant)
Every 2.0s: kubectl get pod -o wide
NAME      READY   STATUS    RESTARTS   AGE     IP          NODE    NOMINATED NODE   READINESS GATES
myweb1    1/1     Running   0          2m18s   172.30.46.4   k8s-w2   <none>        <none>
myweb2    1/1     Running   0          2m18s   172.30.46.5   k8s-w2   <none>        <none>
```

#### 1-5. label 확인하기

- 모든 라벨 확인하기

```
k get pod --show-labels
```

```
(calico:default) root@k8s-m:~# k get pod --show-labels
NAME      READY   STATUS    RESTARTS   AGE     LABELS
myweb1    1/1     Running   0          4m46s   app=nginx,environment=production
myweb2    1/1     Running   0          4m46s   app=nginx,environment=dev
(calico:default) root@k8s-m:~#
```

- 특정 라벨만 확인하기

```
k get pod -l {확인하고자 하는 라벨}
```

```
(calico:N/A) root@k8s-m:~# k get pod -l environment=production
NAME      READY   STATUS    RESTARTS   AGE
myweb1    1/1     Running   0          11m
```

- environment=production 라벨에만 해당하는 pod를 확인할 수 있음

#### 1-6. 라벨로 특정 pod 삭제하기

```
k delete pod -l {지우려는 라벨}
```

- 지우려는 라벨을 가진 모든 파드가 삭제됨

```
(calico:N/A) root@k8s-m:~# k delete pod -l environment=production
pod "myweb1" deleted
(calico:N/A) root@k8s-m:~# k get pod --show-labels
NAME      READY   STATUS    RESTARTS   AGE     LABELS
myweb2    1/1     Running   0          15m     app=nginx,environment=dev
```

## 1-7. 특정 라벨 삭제하기

```
k label pod {파드이름} {지우려는 라벨}
```

- ex) k label pod myweb2 app

// "myweb2"라는 이름을 가진 모든 Pod의 라벨 중에서 "app" 라벨을 제거함

```
(calico:N/A) root@k8s-m:~# k get pod --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
myweb2   1/1     Running   0          15m   app=nginx,environment=dev
(calico:N/A) root@k8s-m:~#
(calico:N/A) root@k8s-m:~# ^C
(calico:N/A) root@k8s-m:~# k label pod myweb2 app-
pod/myweb2 unlabeled
(calico:N/A) root@k8s-m:~# k get pod --show-labels
NAME      READY   STATUS    RESTARTS   AGE   LABELS
myweb2   1/1     Running   0          18m   environment=dev
```

## 1-8. 모든 파드 지우기

```
k delete pod -all
```

## 1-9. 특정 노드 하나에 라벨 붙이기

- 노드의 라벨 확인하기

- master node의 라벨이 worker node의 라벨보다 더 길다는 것을 확인할 수 있음

```
(calico:default) root@k8s-m:~# k get nodes --show-labels
NAME      STATUS   ROLES      AGE   VERSION   LABELS
k8s-m    Ready    control-plane   24h   v1.27.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-m,kubernetes.io/os=linux,node-role:kubernetes.io/control-plane=node.kubernetes.io/exclude-from-external-load-balancers=
k8s-w1   Ready    <none>    23h   v1.27.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-w1,kubernetes.io/os=linux
k8s-w2   Ready    <none>    23h   v1.27.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-w2,kubernetes.io/os=linux
```

- 특정 워커노드에 라벨 추가하기

```
k label node {노드 이름} {라벨}
```

- w1에 'cloud=babo' 라벨 추가하기

```
(calico:default) root@k8s-m:~# k label node k8s-w1 cloud=babo
node/k8s-w1 labeled
```

- 'cloud=babo' 라벨 추가가 잘 되었는지 확인하기

```
(calico:default) root@k8s-m:~# k get nodes --show-labels
NAME      STATUS   ROLES      AGE   VERSION   LABELS
k8s-m    Ready    control-plane   24h   v1.27.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-m,kubernetes.io/os=linux,node-role:kubernetes.io/control-plane=node.kubernetes.io/exclude-from-external-load-balancers=
k8s-w1   Ready    <none>    23h   v1.27.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,cloud=babo,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-w1,kubernetes.io/os=linux
k8s-w2   Ready    <none>    23h   v1.27.11   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=k8s-w2,kubernetes.io/os=linux
```

## 2. Selector 활용하기

### 2-1. Vagrantfile을 labels-node.yaml 내용으로 변경하기

- 1-1과 동일하게 vim labels-node.yaml로 파일 생성 및 수정하기

```
apiVersion: v1
kind: Pod
metadata:
  name: myweb
spec:
  containers:
    - image: nginx
      name: myweb
      ports:
        - containerPort: 80
          protocol: TCP
  nodeSelector:
    cloud: babo
```

### 2-2. yaml 파일이 잘 만들어졌는지 확인하기

```
cat labels-node.yaml
```

### 2-3. label.yaml 파일 실행하기

```
k apply -f labels-node.yaml
```

- 모니터링으로 확인해보기

```
Every 2.0s: kubectl get pod -o wide
NAME     READY   STATUS    RESTARTS   AGE     IP           NODE   NOMINATED NODE   READINESS GATES
myweb   1/1     Running   0          3m50s  172.30.228.74   k8s-w1 <none>        <none>
```

### 2-4. 기존 yaml 파일 내용 수정하여 pod 추가 생성하기

```
cat labels-node.yaml | sed "s/name: myweb/name: myweb4/g" | k apply -f -
```

// cat labels-node.yaml : labels-node.yaml 파일의 내용을 읽어들여서

// sed "s/name: myweb/name: myweb4/g"

: sed 명령어를 사용하여 name:myweb → name:myweb4로 바꾸기

// k apply -f - : Kubernetes 클러스터에 변경사항 적용하기

```
(calico:default) root@k8s-m:~# cat labels-node.yaml | sed "s/name: myweb/name: myweb4/g" | k apply -f -
pod/myweb41 created
```

- 모니터링으로 myweb4 생성 확인하기

```
Every 2.0s: kubectl get pod -o wide
NAME      READY   STATUS    RESTARTS   AGE       IP          NODE     NOMINATED NODE   READINESS GATES
myweb     1/1     Running   0          6m38s   172.30.228.74   k8s-w1   <none>        <none>
myweb1    1/1     Running   0          4m44s   172.30.228.75   k8s-w1   <none>        <none>
myweb41   1/1     Running   0          3m31s   172.30.228.76   k8s-w1   <none>        <none>
```

### 3. 커맨드(Command) 및 Arguments(인자) 설정하기

- pod 생성 시 pod 내부에서 동작하는 컨테이너에게 전달할 커맨드와 인자 설정하기

#### 참고) command & arguments

- command

- 컨테이너가 시작될 때 실행할 실행 파일을 지정하는 것
- ex) 위의 예시 사진에서는 컨테이너 시작 시 '/bin/echo'가 실행 파일로 지정된 것임

```
command: ["/bin/echo"]
args: ["hello"]
```

- arguments

- 컨테이너가 시작될 때 실행 파일에 전달할 인수(argument)들
- ex) 위의 예시 사진에서는 "hello"가 인수로 지정된 것임

정리) 컨테이너 시작 시 '/bin/echo hello' 명령이 실행되는 것

#### 3-1. 새로운 파드 생성 및 확인하기

- **k run {파드 이름} --image {이미지}**

```
(calico:default) root@k8s-m:~# k run cmd-args --image nginx
pod/cmd-args created
```

- 현재 nginx 이미지를 사용하는 컨테이너가 포함된 cmd-args라는 파드를 실행함

- 생성한 파드 확인하기

**k logs {파드 이름}** //해당 파드의 로그 표시하기

```
(calico:default) root@k8s-m:~# k logs cmd-args
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/02/29 04:21:01 [notice] 1#1: using the "epoll" event method
2024/02/29 04:21:01 [notice] 1#1: nginx/1.25.4
2024/02/29 04:21:01 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/02/29 04:21:01 [notice] 1#1: OS: Linux 5.4.0-172-generic
2024/02/29 04:21:01 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/02/29 04:21:01 [notice] 1#1: start worker processes
2024/02/29 04:21:01 [notice] 1#1: start worker process 30
2024/02/29 04:21:01 [notice] 1#1: start worker process 31
```

### 3-2. 다른 방식으로 Vagrantfile을 cmd-args.yaml 내용으로 변경하기

- 1-1과 동일하게 vim cmd-args.yaml로 파일 생성 및 수정하기

```
apiVersion: v1
kind: Pod
metadata:
  name: cmd-args
spec:
  restartPolicy: OnFailure
  containers:
  - image: nginx
    name: cmd-args-nginx
    ports:
    - containerPort: 80
      protocol: TCP
    command: ["/bin/echo"]
    args: ["hello"]
```

- yaml 파일이 잘 만들어졌는지 확인하기

```
cat cmd-args.yaml
```

- cmd-args.yaml 파일 실행하기

```
k apply -f cmd-args.yaml
```

- 결과 확인하기

```
k logs cmd-args
```

```
(calico:default) root@k8s-m:~# k logs cmd-args
hello
```

- 3-1과 달리 'hello'라고 출력됨을 확인할 수 있음
- 이렇게 yaml 파일에서 command와 args를 지정할 수 있음

## 4. Env 설정하기

### 4-1. Vagrantfile을 env.yaml 내용으로 변경하기

- 1-1과 동일하게 vim env.yaml로 파일 생성 및 수정하기

```
apiVersion: v1
kind: Pod
metadata:
```

```

name: mynginx
spec:
  containers:
    - image: nginx:latest
      name: mynginx
      ports:
        - containerPort: 80
      env:
        - name: MYPASSWORD
          value: QWE123

```

#### 4-2. env.yaml 파일이 잘 만들어졌는지 확인하기

```
cat env.yaml
```

#### 4-3. env.yaml 파일 실행하기

```
k apply -f env.yaml
```

#### 4-4. 비밀번호 확인하기

- yaml 파일에 지정했던 비밀번호 확인하기

```
k exec {파드이름} -- env |grep MYPASSWORD
```

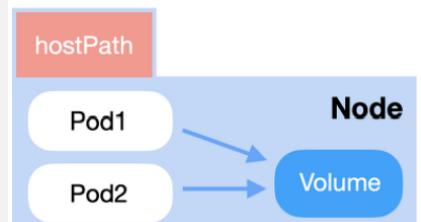
// 해당 파드 내에서 실행 중인 컨테이너에서 "MYPASSWORD"를 포함한 환경변수 확인하기

```
(calico:default) root@k8s-m:~# k exec mynginx -- env |grep MYPASSWORD
MYPASSWORD=QWE123
```

## 5. Hostpath 볼륨 설정하기

### 참고) Hostpath 볼륨

- 노드의 디스크에 볼륨을 생성하여 Pod가 삭제되더라도 볼륨에 있던 데이터는 유지됨
- hostPath 볼륨에는 많은 보안 위험이 있음  
⇒ so 가능하면 HostPath를 사용하지 않는 것이 좋음
- HostPath 볼륨을 사용해야 하는 경우,  
필요한 파일 또는 디렉터리로만 범위를 지정하고 ReadOnly로  
마운트해야 함



## 5-1. Vagrantfile을 hostpath.yaml 내용으로 변경하기

```
apiVersion: v1
kind: Pod
metadata:
  name: hostpath
spec:
  containers:
    - name: hostpath
      image: busybox
      args: [ "tail", "-f", "/dev/null" ]
      volumeMounts:
        - name: hostpath-volume
          mountPath: /etc/data
  volumes:
    - name: hostpath-volume
      hostPath:
        path: /cloud // 마스터 노드의 '/etc/data' 경로와 워커 노드의 '/cloud' 경로를 매칭해 볼륨을 생성한다는 뜻
```

## 5-2. hostpath.yaml 파일이 잘 만들어졌는지 확인하기

```
cat hostpath.yaml
```

## 5-3. hostpath.yaml 파일 실행하기

```
k apply -f hostpath.yaml
```

## 5-4. 볼륨 생성 확인하기

- scheduler에 의해 어떤 worker node에 생길지 결정됨
  - w1에서는 만들어지지 x음

```
root@k8s-w1:~# cd /cloud
-bash: cd: /cloud: No such file or directory
root@k8s-w1:~# ls /
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  lost+found  media  mnt  opt  proc  root  run  sbin  snap  srv  sys  tmp  usr  var
```

- w2에 생성됨 (volume을 공유 중)

```
root@k8s-w2:~# cd /cloud
root@k8s-w2:/cloud# dir
root@k8s-w2:/cloud# ls /
bin  boot  cloud  dev  etc  home  lib  lib32  lib64  libx32  lost+found  media  mnt  opt  proc  root  run  sbin  snap  srv  sys  tmp  usr  var
```

- master node의 /etc/data 와 w2의 /cloud의 연동 확인하기
  - master node의 /etc/data 경로에 babo.txt 파일 생성하기

```
(calico:N/A) root@k8s-m:~# k exec hostpath -- touch /etc/data/babo.txt
```

- w2의 /cloud 경로에 babo.txt가 생긴 것을 확인할 수 있음

```
root@k8s-w2:/cloud# dir  
babo.txt
```

⇒ 이를 통해 master node의 /etc/data 와 w2의 /cloud가 연동됨을 확인할 수 있음

## 6. Probe

참고) I. 개념'의 '1-4. probe' 참고

### 6-1. Vagrantfile을 completed.yaml 내용으로 변경하기

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: completed-pod  
spec:  
  containers:  
    - name: completed-pod  
      image: busybox  
      command: ["sh"]  
      args: ["-c", "sleep 5 && exit 0"] //쉘에서 5s 동안 대기한 후에 종료 코드 0 을 전송함
```

참고) exit 0 & exit 1

- **exit 0** : 스크립트/명령이 성공적으로 실행되었을 때 종료 코드를 0으로 설정해 프로세스를 정상적으로 종료했다고 알리는 것
- **exit 1** : 스크립트/명령이 실패했다고 알리는 것

### 6-2. yaml 파일이 잘 만들어졌는지 확인하기

```
cat completed.yaml
```

### 6-3. completed.yaml 파일 실행하기

```
kubectl apply -f completed.yaml
```

### 6-4. 결과 확인하기

```
kubectl apply -f completed.yaml && kubectl get pod -w
```

- 모니터링으로 확인 시 status가 creating → running → completed → CrashLoopBackOff → ... 가 반복됨

```
(calico:default) root@k8s-m:~# kubectl apply -f completed.yaml && kubectl get pod -w
pod/completed-pod created
NAME        READY   STATUS            RESTARTS   AGE
completed-pod 0/1    ContainerCreating 0          0s
completed-pod 0/1    ContainerCreating 0          1s
completed-pod 1/1    Running           0          5s
completed-pod 0/1    Completed         0          10s
completed-pod 1/1    Running           1 (<invalid> ago) 13s
completed-pod 0/1    Completed         1 (<invalid> ago) 18s
completed-pod 0/1    CrashLoopBackOff  1 (0s ago)   29s
completed-pod 1/1    Running           2 (1s ago)   32s
completed-pod 0/1    Completed         2 (6s ago)   37s
completed-pod 0/1    CrashLoopBackOff  2 (2s ago)   52s
```

- 여기서 CrashLoopBackOff가 되는 이유

#### 참고) CrashLoopBackOff

- 컨테이너가 반복해서 시작되고 죽는 오류 상태
- 컨테이너가 시작 후 바로 종료되거나, 시작 후 일정 시간이 지난 후에 비정상적으로 종료될 때 발생함
  - 원래 yaml의 argument가 `args: ["-c", "sleep 5 && exit 0"]`로 설정되어 있어서 5초 후에 exit 0 코드로 종료됨.

but yaml에 restartPolicy 설정을 아무것도 해주지 않았기 때문에 기본 값인 Always(컨테이너 정상/비정상 종료 시 항상 재시작)로 설정되어버려 계속적으로 파드를 재시작하는 시도를 함

⇒ 따라서 원래는 종료되어야 하는 파드가 계속 재시작되기 때문에 CrashLoopBackOff가 됨

- 또한 처음에 재시작되는 횟수 시간도 점점 늘어남

completed-pod	0/1	CrashLoopBackOff	2 (2s ago)	55s
completed-pod	1/1	Running	3 (17s ago)	70s
completed-pod	0/1	Completed	3 (22s ago)	75s

## 7. Onfailure (exit 1) 설정하기

- 그렇다면 restartPolicy를 Always가 아닌 Onfailure로 설정하면 어떻게 되는지 살펴보기

#### 참고) Onfailure

- 컨테이너가 비정상 종료가 되었을 때만 재시작하는 것

## 7-1. Vagrantfile을 onfailure.yaml 내용으로 변경하기

```
apiVersion: v1
kind: Pod
metadata:
  name: completed-pod
spec:
  restartPolicy: OnFailure
  containers:
    - name: completed-pod
      image: busybox
      command: ["sh"]
      args: ["-c", "sleep 5 && exit 1"]
```

## 7-2. yaml 파일이 잘 만들어졌는지 확인하기

```
cat onfailure.yaml
```

## 7-3. label.yaml 파일 실행하기

```
k apply -f onfailure.yaml && kubectl get pod -w
```

- 모니터링으로 확인 시 status가 Running → CrashLoopBackOff → Error → Running → ... 이 반복됨

```
(calico:default) root@k8s-m:~# kubectl get pod -w
NAME        READY   STATUS            RESTARTS   AGE
completed-pod  0/1    CrashLoopBackOff  2 (16s ago)  66s
completed-pod  1/1    Running          3 (17s ago)  67s
completed-pod  0/1    Error            3 (22s ago)  72s
completed-pod  0/1    CrashLoopBackOff  3 (0s ago)   83s
completed-pod  1/1    Running          4 (38s ago)  2m3s
completed-pod  0/1    Error            4 (46s ago)  2m11s
completed-pod  0/1    CrashLoopBackOff  4 (4s ago)   2m26s
```

- 현재 yaml은 args: ["-c", "sleep 5 && exit 1"]로 설정되어 있으므로 5초 대기 후 프로세스가 실패 했다고 알리게끔 설정되어 있음  
⇒ 따라서 Error라고 뜸
- but Onfailure 설정으로 인해 컨테이너가 비정상 종료 시 다시 재시작하게 되어 있음  
⇒ 따라서 다시 CrashLoopBackOff가 되고 Running이 됨
- 아래의 명령어를 통해 파드의 생성, 시작, 종료 등과 관련된 내용을 확인할 수도 있음

## kubectl describe pod completed-pod | grep Events -A 12

참고) 명령어 해석

- **kubectl describe pod completed-pod** //completed-pod라는 파드에 대한 자세한 정보 출력
- **grep Events -A 12** //Events라는 문자열을 찾고 그 다음 12개 줄을 함께 출력하기

```
root@k8s-w1:~# kubectl describe pod completed-pod | grep Events -A 12
Events:
  Type   Reason     Age           From            Message
  ----   ----      --           --             --
Normal  Scheduled  2m33s        default-scheduler  Successfully assigned default/completed-pod to k8s-w2
Normal  Pulled    2m16s        kubelet         Successfully pulled image "busybox" in 2.018588723s (2.018734708s including waiting)
Normal  Pulled    2m8s         kubelet         Successfully pulled image "busybox" in 1.816651285s (1.81693452s including waiting)
Normal  Pulled    108s        kubelet         Successfully pulled image "busybox" in 1.811247178s (1.811259568s including waiting)
Normal  Created   73s (x4 over 2m16s)  kubelet         Created container completed-pod
Normal  Started   73s (x4 over 2m16s)  kubelet         Started container completed-pod
Normal  Pulled    73s         kubelet         Successfully pulled image "busybox" in 1.756241081s (1.756253777s including waiting)
Warning BackOff   33s (x7 over 2m2s)   kubelet         Back-off restarting failed container completed-pod in pod completed-pod_default(97636b93e1549)
Normal  Pulling   18s (x5 over 2m18s)  kubelet         Pulling image "busybox"
Normal  Pulled    16s         kubelet         Successfully pulled image "busybox" in 1.667017738s (1.667029987s including waiting)
Windows 정품 인증
root@k8s-w1:~#
```

- Back-off restarting failed container 메시지

- 컨테이너가 계속해서 다시 시작되고 실패했다는 것을 나타내는 메시지
- 컨테이너가 시작한 후에 오류로 인해 실패하고 다시 시작되려고 시도했으나 여전히 실패했음을 의미

⇒ 이를 통해 파드가 계속 다시 시작되고 실패되어 CrashLoopBackOff가 나타났다고 파악할 수 있음

## 8. Onfailure (exit 0) 설정하기

- 그렇다면 exit 1 → exit 0으로 수정해보기

### 8-1. Vagrantfile을 다음 내용(exit 0)으로 변경하기

```
apiVersion: v1
kind: Pod
metadata:
  name: completed-pod
spec:
  restartPolicy: OnFailure
  containers:
    - name: completed-pod
      image: busybox
      command: ["sh"]
      args: ["-c", "sleep 5 && exit 0"]
```

### 8-2. yaml 파일이 잘 만들어졌는지 확인하기

```
cat onfailure.yaml
```

### 8-3. label.yaml 파일 실행하기

```
kubectl apply -f onfailure.yaml && kubectl get pod -w
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
completed-pod	0/1	Completed	0	2m31s	172.30.46.8	k8s-w2	<none>	<none>

- 모니터링으로 확인 시 pod가 잘 생성되고 종료되었음을 확인할 수 있음
  - 현재 yaml은 args: ["-c", "sleep 5 && exit 0"]로 설정되어 있으므로 5초 대기 후 프로세스가 성공적으로 실행되어 종료되었다고 알리게끔 설정되어 있음  
⇒ Onfailure로 설정되어 있지만 컨테이너가 비정상적인 종료가 된 것이 아니기 때문에 다시 재시작되지 않고 종료된 상태로 남아 있음

## 9. Livenessprobe 설정하기

### 참고) Liveness Probe

- 컨테이너가 멈추거나 비정상 종료되면 해당 컨테이너를 재시작 함
- 'I. 개념'의 '1-4. probe' 참고

### 9-1. Vagrantfile을 livenessprobe.yaml 내용으로 변경하기

```
apiVersion: v1
kind: Pod
metadata:
  name: livenessprobe
spec:
  containers:
    - name: livenessprobe
      image: nginx
      livenessProbe:
        httpGet:
          port: 80
          path: /index.html
```

- Http Get 요청을 통해 Linveness probe를 실행하기
- Http Get 요청을 보낼 컨테이너의 포트는 80번으로 설정하기
- Http Get 요청이 보내질 경로는 컨테이너의 80번 포트에 있는 /index.html으로 설정하기

### 9-2. yaml 파일이 잘 만들어졌는지 확인하기

```
cat livenessprobe.yaml
```

### 9-3. livenessprobe.yaml 파일 실행하기

```
k apply -f livenessprobe.yaml
```

### 9-4. livenessprobe 설정 정보 확인하기

```
k describe pod livenessprobe |grep Liveness
```

참고) 명령어 해석

- livenessprobe라는 파드의 상세 정보 중에서 "Liveness"와 관련된 정보만 필터링해서 보여주라는 명령어
- livenessprobe 설정에 관한 정보를 확인할 수 있음
- 주로 파드의 Liveness Probe 설정 상태나 현재 상태를 확인하는데 사용됨

```
(calico:default) root@k8s-m:~# k describe pod livenessprobe |grep Liveness
  Liveness:          http-get http://:80/index.html delay=0s timeout=1s period=10s #success=1 #failure=3
```

- 위의 빨간 줄 해석
  - **Liveness** : 파드의 Liveness Probe를 나타냄
  - **http-get http://:80/index.html** : Liveness Probe가 HTTP GET 요청을 함. 요청은 파드의 80 번 포트에 있는 '/index.html' 경로로 보내짐
  - **delay=0s** : Probe가 컨테이너 시작 후 즉시 시작함. (초기 지연 시간이 x음)
  - **timeout=1s** : Probe가 응답을 기다리는 시간이 1초임.  
1초 이내에 응답이 없으면 실패로 간주함
  - **period=10s** : Probe가 매 10초마다 실행됨
  - **#success=1** : Probe가 한 번 성공하면 컨테이너가 살아 있다고 판단함
  - **#failure=3** : 3번 연속으로 Probe 실패 시 컨테이너가 살아 있지 않다고 판단해 재시작됨

### 9-5. log 확인하기

- livenessprobe 파드의 로그 확인하기

```
k logs livenessprobe
```

- 아래의 사진처럼 10초마다 한 번씩 확인하는 것을 알 수 있음

```
(calico:default) root@k8s-m:~# k logs livenessprobe
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/02/29 05:46:52 [notice] 1#1: using the "epoll" event method
2024/02/29 05:46:52 [notice] 1#1: nginx/1.25.4
2024/02/29 05:46:52 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/02/29 05:46:52 [notice] 1#1: OS: Linux 5.4.0-172-generic
2024/02/29 05:46:52 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/02/29 05:46:52 [notice] 1#1: start worker processes
2024/02/29 05:46:52 [notice] 1#1: start worker process 29
2024/02/29 05:46:52 [notice] 1#1: start worker process 30
10.0.2.15 - - [29/Feb/2024:05:46:57 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
10.0.2.15 - - [29/Feb/2024:05:47:07 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
10.0.2.15 - - [29/Feb/2024:05:47:17 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
10.0.2.15 - - [29/Feb/2024:05:47:27 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
10.0.2.15 - - [29/Feb/2024:05:47:37 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
10.0.2.15 - - [29/Feb/2024:05:47:47 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
10.0.2.15 - - [29/Feb/2024:05:47:57 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
10.0.2.15 - - [29/Feb/2024:05:48:07 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
(calico:default) root@k8s-m:~#
```

## 9-6. index.html 삭제한 후 log 확인하기

- index.html 삭제하기

```
k exec livenessprobe -- rm /usr/share/nginx/html/index.html
```

- 다시 log 확인하기

```
k logs livenessprobe -f
```

- 아래 그림처럼 3번 error가 나오 재시작 함을 확인할 수 있음

```
(calico:default) root@k8s-m:~# k logs livenessprobe -f
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/02/29 05:58:39 [notice] 1#1: using the "epoll" event method
2024/02/29 05:58:39 [notice] 1#1: nginx/1.25.4
2024/02/29 05:58:39 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2024/02/29 05:58:39 [notice] 1#1: OS: Linux 5.4.0-172-generic
2024/02/29 05:58:39 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/02/29 05:58:39 [notice] 1#1: start worker processes
2024/02/29 05:58:39 [notice] 1#1: start worker process 29
2024/02/29 05:58:39 [notice] 1#1: start worker process 30
10.0.2.15 - - [29/Feb/2024:05:58:44 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
10.0.2.15 - - [29/Feb/2024:05:58:54 +0000] "GET /index.html HTTP/1.1" 200 615 "-" "kube-probe/1.27" "-"
2024/02/29 05:59:08 [error] 30#30: *3 open() "/usr/share/nginx/html/index.html" failed (2: No such file or directory), client: 10.0.2.15, server: localhost, request: "GET /index.html HTTP/1.1", host: 172.30.46.19:80
10.0.2.15 - - [29/Feb/2024:05:59:08 +0000] "GET /index.html HTTP/1.1" 404 153 "-" "kube-probe/1.27" "-"
2024/02/29 05:59:14 [error] 30#30: *4 open() "/usr/share/nginx/html/index.html" failed (2: No such file or directory), client: 10.0.2.15, server: localhost, request: "GET /index.html HTTP/1.1", host: 172.30.46.19:80
10.0.2.15 - - [29/Feb/2024:05:59:14 +0000] "GET /index.html HTTP/1.1" 404 153 "-" "kube-probe/1.27" "-"
2024/02/29 05:59:24 [error] 30#30: *5 open() "/usr/share/nginx/html/index.html" failed (2: No such file or directory), client: 10.0.2.15, server: localhost, request: "GET /index.html HTTP/1.1", host: 172.30.46.19:80
10.0.2.15 - - [29/Feb/2024:05:59:24 +0000] "GET /index.html HTTP/1.1" 404 153 "-" "kube-probe/1.27" "-"
2024/02/29 05:59:24 [notice] 1#1: signal 3 (SIGQUIT) received, shutting down
2024/02/29 05:59:24 [notice] 29#29: gracefully shutting down
2024/02/29 05:59:24 [notice] 29#29: exiting
2024/02/29 05:59:24 [notice] 29#29: exit
2024/02/29 05:59:24 [notice] 30#30: gracefully shutting down
2024/02/29 05:59:24 [notice] 30#30: exiting
2024/02/29 05:59:24 [notice] 30#30: exit
2024/02/29 05:59:24 [notice] 1#1: signal 17 (SIGCHLD) received from 29
2024/02/29 05:59:24 [notice] 1#1: worker process 29 exited with code 0
2024/02/29 05:59:24 [notice] 1#1: signal 29 (SIGTTOI) received
2024/02/29 05:59:24 [notice] 1#1: signal 17 (SIGCHLD) received from 30
2024/02/29 05:59:24 [notice] 1#1: worker process 30 exited with code 0
2024/02/29 05:59:24 [notice] 1#1: exit
```

Windows 정품 인증  
[설정]으로 이동하여 Windows를 정품 인증합니다.

# 10. Replicas 설정하기

참고) I. 개념'의 '3번' 참고

## 10-1. Vagrantfile을 replicaset.yaml 내용으로 변경하기

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: replicaset-cloud
spec:
  replicas: 3    //pod 3 개 유지하기
  selector:
    matchLabels:  //라벨 셀렉터가 매치해야 하는 라벨
      app: cloud-nginx-pods  //app' 라벨의 값이 'cloud-mgimx-pods'인 파드를 이 RelicaSet 이 관리하도록 지정
    template:  //Replicaset 이 생성할 파드의 템플릿을 정의
      metadata:
        name: cloud-nginx-pod  //생성되는 파드 이름
        labels:
          app: cloud-nginx-pods  //이 라벨은 RelicaSet 의 셀렉터 'matchLables'와 일치하여 이 템플릿으로 생성된
                                    //파드가 이 RelicaSet 에 의해 관리되도록 지정
      spec:
        containers:
          - name: nginx
            image: nginx:latest
```

## 10-2. yaml 파일이 잘 만들어졌는지 확인하기

```
cat replicaset.yaml
```

## 10-3. replicaset.yaml 파일 실행하기

```
kubectl apply -f replicaset.yaml
```

## 10-4. 모니터링으로 확인해보기

```
watch -d 'kubectl get pods,rs,deploy -o wide'
```

Every 2.0s: kubectl get pods,rs,deploy -o wide										
NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS	GATES	
pod/replicaset-cloud-lns9w	1/1	Running	0	65s	172.30.46.20	k8s-w2	<none>	<none>		
pod/replicaset-cloud-ts45d	1/1	Running	0	65s	172.30.46.21	k8s-w2	<none>	<none>		
pod/replicaset-cloud-v8jv5	1/1	Running	0	65s	172.30.228.77	k8s-w1	<none>	<none>		
NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR			
replicaset.apps/replicaset-cloud	3	3	3	65s	nginx	nginx:latest	app=cloud-nginx-pods			

- pod가 3개 생성되었고, Replicaset도 생성됨을 확인할 수 있음

```
k get replicaset.apps -o wide
```

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset-cloud	3	3	3	2m3s	nginx	nginx:latest	app=cloud-nginx-pods

## 10-5. 정보 조회하기

- replicaset-cloud라는 이름을 가진 ReplicaSet 리소스에 대한 정보 조회하기

```
k describe replicaset.apps replicaset-cloud
```

(calico:default) root@k8s-m:~# k describe replicaset.apps replicaset-cloud					
Name: replicaset-cloud					
Namespace: default					
Selector: app=cloud-nginx-pods					
Labels: <none>					
Annotations: <none>					
Replicas: 3 current / 3 desired					
Pods Status: 3 Running / 0 Waiting / 0 Succeeded / 0 Failed					
Pod Template:					
Labels: app=cloud-nginx-pods					
Containers:					
nginx:					
Image: nginx:latest					
Port: <none>					
Host Port: <none>					
Environment: <none>					
Mounts: <none>					
Volumes: <none>					
Events:					
Type	Reason	Age	From	Message	
Normal	SuccessfulCreate	3m22s	replicaset-controller	Created pod: replicaset-cloud-lns9w	
Normal	SuccessfulCreate	3m22s	replicaset-controller	Created pod: replicaset-cloud-v8jv5	
Normal	SuccessfulCreate	3m22s	replicaset-controller	Created pod: replicaset-cloud-ts45d	

- 각 파드의 label 살펴보기

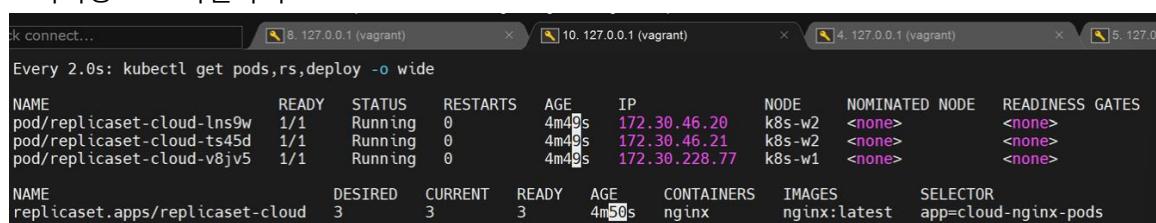
```
k get pod --show-labels
```

(calico:default) root@k8s-m:~# k get pod --show-labels					
NAME	READY	STATUS	RESTARTS	AGE	LABELS
replicaset-cloud-lns9w	1/1	Running	0	3m51s	app=cloud-nginx-pods
replicaset-cloud-ts45d	1/1	Running	0	3m51s	app=cloud-nginx-pods
replicaset-cloud-v8jv5	1/1	Running	0	3m51s	app=cloud-nginx-pods

- 3개 다 다른 pod가 동작 중임

- 서로 다른 라벨이 알아서 붙는다는 것을 확인할 수 있음

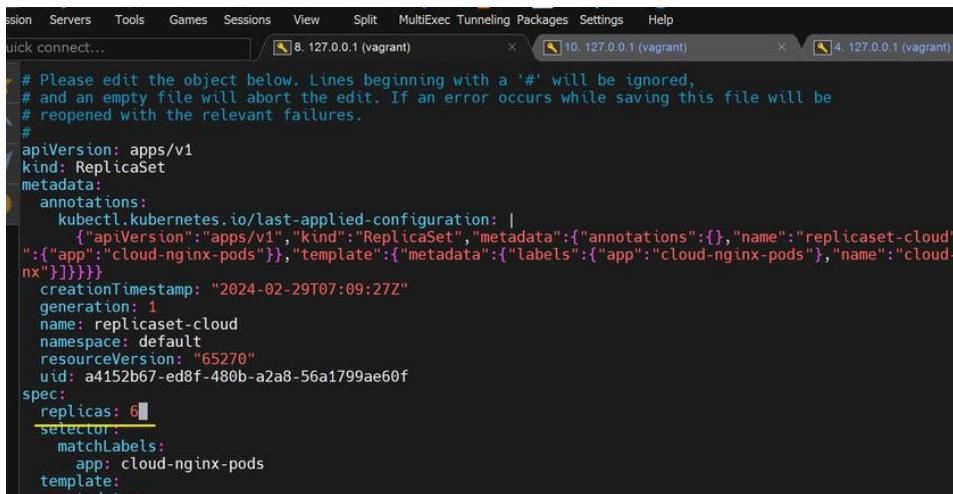
- 모니터링으로 확인하기



## 10-6. Replicaset 개수 수정한 후 확인하기

- Replicaset 개수 수정하기

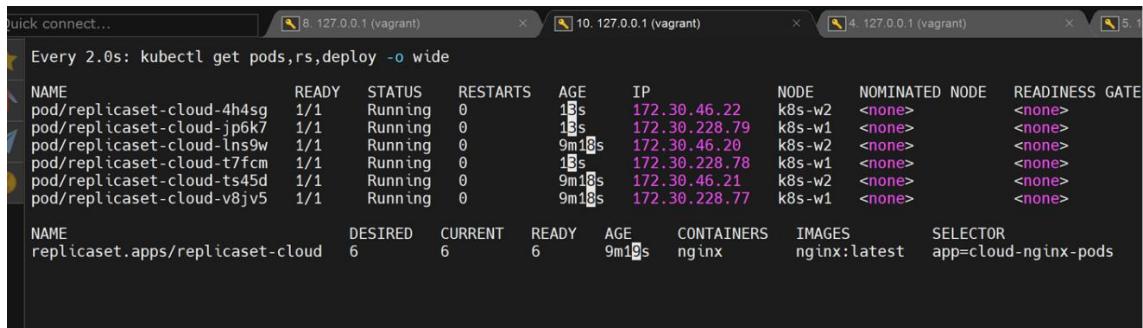
```
k edit replicsets.apps replicaset-cloud
```



```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"apps/v1","kind":"ReplicaSet","metadata":{"annotations":{},"name":"replicaset-cloud"},"spec":{"app":"cloud-nginx-pods"},"template":{"metadata":{"labels":{"app":"cloud-nginx-pods"},"name":"cloud-nginx-pods}}}}
  creationTimestamp: "2024-02-29T07:09:27Z"
  generation: 1
  name: replicaset-cloud
  namespace: default
  resourceVersion: "65270"
  uid: a4152b67-ed8f-480b-a2a8-56a1799ae60f
spec:
  replicas: 6
  selector:
    matchLabels:
      app: cloud-nginx-pods
  template:
    metadata:
```

- 개수를 6개로 늘려줌

- 모니터링으로 확인하기



NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATE
pod/replicaset-cloud-4h4sg	1/1	Running	0	13s	172.30.46.22	k8s-w2	<none>	<none>
pod/replicaset-cloud-jp6k7	1/1	Running	0	13s	172.30.228.79	k8s-w1	<none>	<none>
pod/replicaset-cloud-lns9w	1/1	Running	0	9m18s	172.30.46.20	k8s-w2	<none>	<none>
pod/replicaset-cloud-t7fcn	1/1	Running	0	13s	172.30.228.78	k8s-w1	<none>	<none>
pod/replicaset-cloud-ts45d	1/1	Running	0	9m18s	172.30.46.21	k8s-w2	<none>	<none>
pod/replicaset-cloud-v8jv5	1/1	Running	0	9m18s	172.30.228.77	k8s-w1	<none>	<none>

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset.apps/replicaset-cloud	6	6	6	9m18s	nginx	nginx:latest	app=cloud-nginx-pods

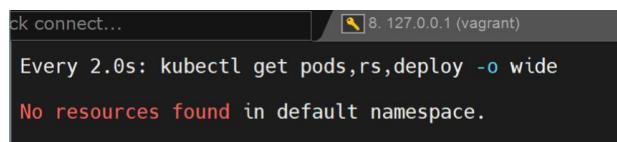
- pod 개수가 6개로 늘어남을 확인할 수 있음

## 10-7. Replicaset 지우기

```
k delete replicsets.apps replicaset-cloud
```

- 모니터링으로 확인해보면 pod가 다 지워짐을 확인할 수 있음

⇒ Replicaset은 pod를 관리하는 큰 개념이기 때문에 Replicaset 삭제 시 pod 또한 삭제됨



```
Every 2.0s: kubectl get pods,rs,deploy -o wide
No resources found in default namespace.
```

# 11. Replicaset의 라벨과 동일한 pod 생성하기

## 11-1. Vagrantfile을 pod-rs.yaml 내용으로 변경하기

- pod 2개를 생성하는데, Replicaset의 라벨과 동일한 라벨 붙이기

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    app: cloud-nginx-pods
spec:
  containers:
    - name: hello1
      image: gcr.io/google-samples/hello-app:1.0
---
apiVersion: v1
kind: Pod
metadata:
  name: pod2
  labels:
    app: cloud-nginx-pods
spec:
  containers:
    - name: hello2
      image: gcr.io/google-samples/hello-app:2.0
```

## 11-2. yaml 파일이 잘 만들어졌는지 확인하기

```
cat pod-rs.yaml
```

## 11-3. pod-rs.yaml 파일 실행하고 모니터링하기

```
k apply -f pod-rs.yaml && k get pod -w
```

## 11-4. 모니터링으로 확인해보기

```
k apply -f pod-rs.yaml && k get pod -w
```

Every 2.0s: kubectl get pods,rs,deploy -o wide										
NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS	GATES	
DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR				
1	1	0/1	Terminating	0s	<none>	k8s-w2	<none>	<none>	<none>	
1	1	0/1	Terminating	0s	<none>	k8s-w1	<none>	<none>	<none>	
3	3	3/3	Running	0s	5m55s	172.30.228.80	k8s-w1	<none>	<none>	
3	3	3/3	Running	0s	5m55s	172.30.46.23	k8s-w2	<none>	<none>	
3	3	3/3	Running	0s	5m55s	172.30.46.24	k8s-w2	<none>	<none>	
NAME										
replicaset.apps/replicaset-cloud										

- 초반에는 pod1과 pod2가 생성되었지만 Terminating 상태임을 확인할 수 있음

```

Every 2.0s: kubectl get pods,rs,deploy -o wide
k8s-m: Thu Feb 29

NAME           READY   STATUS    RESTARTS   AGE   IP          NODE   NOMINATED NODE   READINESS GATES
pod/replicaset-cloud-69rl   1/1     Running   0          6m12s  172.30.228.80   k8s-w1   <none>        <none>
pod/replicaset-cloud-c4n5c   1/1     Running   0          6m12s  172.30.46.23   k8s-w2   <none>        <none>
pod/replicaset-cloud-c98s2   1/1     Running   0          6m12s  172.30.46.24   k8s-w2   <none>        <none>

NAME            DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES   SELECTOR
replicaset.apps/replicaset-cloud   3         3         3      6m12s  nginx       nginx:latest   app=cloud-nginx-pods

```

- 일정 시간이 지난 후에는 위의 사진처럼 pod1과 pod2가 사라짐

⇒ Replicaset의 개수를 3개로 지정해놓았기 때문에 3개 이상의 pod를 가질 수 없음

따라서 pod1과 pod2가 생성되어도 사라지는 것임

## 11-5. 라벨을 통해 pod 삭제하기

- 해당 라벨을 가진 pod들을 삭제해보기

**k delete pod -l app=cloud-nginx-pods**

```

Every 2.0s: kubectl get pods,rs,deploy -o wide
k8s-m: Thu Feb 29 16:33

NAME           READY   STATUS    RESTARTS   AGE   IP          NODE   NOMINATED NODE   READINESS GATES
pod/replicaset-cloud-lx6tb   1/1     Running   0          35s   172.30.228.82   k8s-w1   <none>        <none>
pod/replicaset-cloud-mbzfr   1/1     Running   0          35s   172.30.46.27   k8s-w2   <none>        <none>
pod/replicaset-cloud-w8ps9   1/1     Running   0          35s   172.30.46.26   k8s-w2   <none>        <none>

NAME            DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES   SELECTOR
replicaset.apps/replicaset-cloud   3         3         3      8m46s  nginx       nginx:latest   app=cloud-nginx-pods

```

- 언뜻 보기에는 pod가 삭제되지 않은 것처럼 보임

- 하지만 app=cloud-nginx-pods 라벨을 가진 pod 3개가 삭제되고, 새로운 pod 3개가 생성된 것임

- 이는 위 사진의 노란색 표시와 11-4의 사진을 비교해보면 알 수 있음

- 11-4의 사진과 위의 사진(노란색 표시)의 pod 이름이 다 다름

이는 기존의 pod 3개가 삭제되고, 새로운 pod 3개가 생성됨을 알 수 있음

## 11-6. 동일 라벨을 가진 pod들 중 특정 pod의 라벨 삭제하기

- 현재 pod 3개의 라벨은 모두 app=cloud-nginx-pods임

```
(calico:default) root@k8s-m:~# k get pod --show-labels
NAME           READY   STATUS    RESTARTS   AGE   LABELS
replicaset-cloud-lx6tb   1/1     Running   0          108s  app=cloud-nginx-pods
replicaset-cloud-mbzfr   1/1     Running   0          108s  app=cloud-nginx-pods
replicaset-cloud-w8ps9   1/1     Running   0          108s  app=cloud-nginx-pods
```

- app=cloud-nginx-pods라는 라벨을 가진 pod 중 특정 pod 1개(replicaset-cloud-lx6tb)의 라벨을 삭제해보기

**k label pod replicaset-cloud-lx6tb app-**

```
ck connect... 8. 127.0.0.1 (vagrant) 10. 127.0.0.1 (vagrant) 4. 127.0.0.1 (vagrant) 5. 127.0.0.1 (vagrant)
Every 2.0s: kubectl get pods,rs,deploy -o wide
k8s-m: Thu Feb 2
NAME          READY   STATUS    RESTARTS   AGE      IP           NODE     NOMINATED NODE   READINESS GATES
pod/replicaset-cloud-lx6tb   1/1     Running   0          7m22s   172.30.228.82   k8s-w1   <none>        <none>
pod/replicaset-cloud-mblw9   1/1     Running   0          21s     172.30.228.83   k8s-w1   <none>        <none>
pod/replicaset-cloud-mbzfr   1/1     Running   0          7m22s   172.30.46.27    k8s-w2   <none>        <none>
pod/replicaset-cloud-w8ps9   1/1     Running   0          7m22s   172.30.46.26    k8s-w2   <none>        <none>
NAME            DESIRED   CURRENT   READY   AGE      CONTAINERS   IMAGES   SELECTOR
replicaset.apps/replicaset-cloud   3         3         3       15m     nginx        nginx:latest   app=cloud-nginx-pods
```

- 오히려 pod 1개가 생성되어 총 pod는 4개가 됨

- 이는 replicaset-cloud-lx6tb라는 파드는 해당 라벨을 지워 더 이상 replicaset에 속하지 x는 pod가 됨
- 따라서 replicaset에 해당하는 pod는 2개가 되므로, 한 자리를 채워줘야 해서 replicaset에 속하는 pod 한 개가 더 생성된 것임

## 11-7. replicaset에 통제를 받지 않게된 일반 pod 삭제하기

- 11-6.에서 더 이상 replicaset에 통제를 받지 않게된 replicaset-cloud-lx6tb 파드를 삭제해보기

**k delete pod replicaset-cloud-lx6tb**

⇒ 삭제가 됨! (replicaset에 통제를 받지 x아서)

# 12. Deployment 설정하기

참고) 'I. 개념'의 '3번' 참고

## 12-1. Vagrantfile을 deployment.yaml 내용으로 변경하기

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-cloud
spec:
  replicas: 3
  selector:
    matchLabels:
      app: cloud-nginx-pods
  template:
    metadata:
      name: cloud-nginx-pod
      labels:
```

```

    app: cloud-nginx-pods
  spec:
    containers:
      - name: nginx
        image: nginx:latest      args: ["hello"]

```

## 12-2. yaml 파일이 잘 만들어졌는지 확인하기

**cat deployment.yaml**

## 12-3. label.yaml 파일 실행하기

**k apply -f deployment.yaml**

## 12-4. 모니터링으로 확인해보기

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
pod/deployment-cloud-5fc5956697-7ntvp	1/1	Running	0	51s	172.30.46.16	k8s-w2	<none>	<none>
pod/deployment-cloud-5fc5956697-vx2ht	1/1	Running	0	51s	172.30.228.85	k8s-w1	<none>	<none>
pod/deployment-cloud-5fc5956697-zth2g	1/1	Running	0	51s	172.30.46.17	k8s-w2	<none>	<none>

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset.apps/deployment-cloud-5fc5956697	3	3	3	51s	nginx	nginx:latest	app=cloud-nginx-pods,pod-template-hash=5fc5956697

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
deployment.apps/deployment-cloud	3/3	3	3	51s	nginx	nginx:latest	app=cloud-nginx-pods

- deployment에 의해 replicas가 생성되고, 파드까지 생성된 것을 확인할 수 있음

**k get deployments.apps -o wide**

```
(calico:N/A) root@k8s-m:~# k get deployments.apps -o wide
NAME          READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES   SELECTOR
deployment-cloud  3/3     3           3           108s  nginx       nginx:latest  app=cloud-nginx-pods
```

**k describe deployments.apps deployment-cloud**

```
(calico:N/A) root@k8s-m:~# k get deployments.apps -o wide
NAME          READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES   SELECTOR
deployment-cloud  3/3     3           3           108s  nginx       nginx:latest  app=cloud-nginx-pods
(calico:N/A) root@k8s-m:~# k describe deployments.apps deployment-cloud
Name:           deployment-cloud
Namespace:      default
CreationTimestamp: Sun, 03 Mar 2024 20:48:02 +0900
Labels:         <none>
Annotations:   deployment.kubernetes.io/revision: 1
Selector:       app=cloud-nginx-pods
Replicas:      3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:  RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=cloud-nginx-pods
  Containers:
    nginx:
      Image:      nginx:latest
      Port:       <none>
      Host Port: <none>
      Environment: <none>
      Mounts:
      Volumes:
  Conditions:
    Type     Status  Reason
    ----  -----
    Available  True   MinimumReplicasAvailable
    Progressing  True   NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  deployment-cloud-5fc5956697 (3/3 replicas created)
Events:
  Type     Reason            Age   From               Message
  ----  -----            --  --    --
  Normal  ScalingReplicaSet  2m37s  deployment-controller  Scaled up replica set deployment-cloud-5fc5956697 to 3
(calico:N/A) root@k8s-m:~#
```

## 12-5. Deployment 가 관리하는 pod 개수 조정하기

k scale deployment --replicas 6 deployment-cloud

명령어 해석)

- **scale** : 리소스의 크기 조정하기
  - **deployment** : deployment 리소스에 적용하기
  - **--replicas 6** : deployment가 관리하는 pod의 개수를 6개로 설정하기
  - **deployment-cloud** : 조정할 deployment 이름
- 아래 그림처럼 6개로 pod가 늘어남

```
Every 2.0s: kubectl get pods,rs,deploy -o wide
k8s-m: Sun Mar  3
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE   NOMINATED NODE   READINESS GATES
pod/deployment-cloud-5fc5956697-284zv  1/1    Running   0          3m45s  172.30.228.86  k8s-w1  <none>        <none>
pod/deployment-cloud-5fc5956697-29nwp  1/1    Running   0          3m45s  172.30.228.87  k8s-w1  <none>        <none>
pod/deployment-cloud-5fc5956697-7ntvp  1/1    Running   0          3m45s  172.30.46.16   k8s-w2  <none>        <none>
pod/deployment-cloud-5fc5956697-1kbnq  1/1    Running   0          3m45s  172.30.46.18   k8s-w2  <none>        <none>
pod/deployment-cloud-5fc5956697-vx2ht  1/1    Running   0          3m45s  172.30.228.85  k8s-w1  <none>        <none>
pod/deployment-cloud-5fc5956697-zth2g  1/1    Running   0          3m45s  172.30.46.17   k8s-w2  <none>        <none>
NAME           DESIRED  CURRENT  READY   AGE     CONTAINERS   IMAGES          SELECTOR
replicaset.apps/deployment-cloud-5fc5956697  6       6        6      3m45s  nginx        nginx:latest  app=cloud-nginx-pods,pod-template-hash=5fc5956697
NAME           READY   UP-TO-DATE  AVAILABLE  AGE     CONTAINERS   IMAGES          SELECTOR
deployment.apps/deployment-cloud  6/6     6         6      3m45s  nginx        nginx:latest  app=cloud-nginx-pods
```

## 12-6. pod, replicaset, deployment 단위로 pod 삭제하기

### 1) pod 단위로 삭제하기

k delete pod -all

```
Every 2.0s: kubectl get pods,rs,deploy -o wide
k8s-m: Sun Mar  3
NAME           DESIRED  CURRENT  READY   AGE     CONTAINERS   IMAGES          SELECTOR
replicaset.apps/deployment-cloud-5fc5956697  6       6        6      5m9s   nginx        nginx:latest  app=cloud-nginx-pods,pod-template-hash=5fc5956697
NAME           READY   UP-TO-DATE  AVAILABLE  AGE     CONTAINERS   IMAGES          SELECTOR
deployment.apps/deployment-cloud  6/6     6         6      5m9s   nginx        nginx:latest  app=cloud-nginx-pods

Every 2.0s: kubectl get pods,rs,deploy -o wide
k8s-m: Sun Mar  3
NAME           READY   STATUS    RESTARTS   AGE     IP          NODE   NOMINATED NODE   READINESS GATES
pod/deployment-cloud-5fc5956697-24997  1/1    Running   0          51s    172.30.46.19  k8s-w2  <none>        <none>
pod/deployment-cloud-5fc5956697-2rbrs  1/1    Running   0          51s    172.30.228.90  k8s-w1  <none>        <none>
pod/deployment-cloud-5fc5956697-8vvs9  1/1    Running   0          51s    172.30.46.21  k8s-w2  <none>        <none>
pod/deployment-cloud-5fc5956697-dcvtc  1/1    Running   0          51s    172.30.46.20  k8s-w2  <none>        <none>
pod/deployment-cloud-5fc5956697-hj66m  1/1    Running   0          51s    172.30.228.89  k8s-w1  <none>        <none>
pod/deployment-cloud-5fc5956697-zhcxk  1/1    Running   0          51s    172.30.228.88  k8s-w1  <none>        <none>
NAME           DESIRED  CURRENT  READY   AGE     CONTAINERS   IMAGES          SELECTOR
replicaset.apps/deployment-cloud-5fc5956697  6       6        6      6m9s   nginx        nginx:latest  app=cloud-nginx-pods,pod-template-hash=5fc5956697
NAME           READY   UP-TO-DATE  AVAILABLE  AGE     CONTAINERS   IMAGES          SELECTOR
deployment.apps/deployment-cloud  6/6     6         6      6m9s   nginx        nginx:latest  app=cloud-nginx-pods
```

- 위처럼 모니터링으로 보면 모든 pod가 지워졌다가 다시 새로운 pod 6개가 생성됨을 알 수 있음

⇒ pod 단위로는 pod가 완전히 삭제되지 않음

### 2) replicaset 단위로 삭제하기

k delete replicaset.apps -all

Every 2.0s: kubectl get pods,rs,deploy -o wide										
NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES		
pod/deployment-cloud-5fc5956697-c8hpr	1/1	Running	0	15s	172.30.228.96	k8s-w1	<none>	<none>		
pod/deployment-cloud-5fc5956697-gp77l	1/1	Running	0	15s	172.30.228.95	k8s-w1	<none>	<none>		
pod/deployment-cloud-5fc5956697-ngqjv	1/1	Running	0	15s	172.30.228.94	k8s-w1	<none>	<none>		
pod/deployment-cloud-5fc5956697-q59r7	1/1	Running	0	15s	172.30.46.27	k8s-w2	<none>	<none>		
pod/deployment-cloud-5fc5956697-smx2x	1/1	Running	0	15s	172.30.46.26	k8s-w2	<none>	<none>		
pod/deployment-cloud-5fc5956697-vnrh5	1/1	Running	0	15s	172.30.46.25	k8s-w2	<none>	<none>		
replicaset.apps/deployment-cloud-5fc5956697	6	DESIRED	CURRENT	READY	15s	nginx	nginx:latest	SELECTOR		
deployment.apps/deployment-cloud	6/6	UP-TO-DATE	AVAILABLE	AGE	7m50s	nginx	nginx:latest	SELECTOR		

- 위처럼 모니터링으로 보면 지웠다가 다시 새로운 pod 6개가 생성됨을 알 수 있음

⇒ replica 단위로는 pod가 완전히 삭제되지 않음

### 3) deployment 단위로 삭제하기

**k delete deployments.apps deployment-cloud**

- 아래처럼 완전히 pod가 삭제됨을 알 수 있음

```
Every 2.0s: kubectl get pods,rs,deploy -o wide
No resources found in default namespace.
```

⇒ pod < node < replicas < deployment 순이기 때문에 가장 큰 단위인 deployment를 삭제해야지만 pod가 삭제됨

## 13. Rolling update 설정하기

참고) 'I. 개념'의 '3번' 참고

### 13-1. Vagrantfile을 deployment-1.yaml 내용으로 변경하기

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: cndk-nginx-pods
  template:
    metadata:
      name: cndk-nginx-pod
      labels:
        app: cndk-nginx-pods
    spec:
```

```
containers:  
- name: nginx  
  image: nginx:1.11
```

### 13-2. yaml 파일이 잘 만들어졌는지 확인하기

```
cat deployment-1.yaml
```

### 13-3. deployment-1.yaml 파일 실행하기

```
k apply -f deployment-1.yaml --record
```

#### 명령어 해석)

- **--record** : replicaset에 대한 정보를 기록해서 나중에 문제발생 시 history로 변경 이력 확인 가능함

### 13-4. 필터링하여 출력하기, rollout으로 업데이트 이력 확인하기

```
k get pod -o yaml |grep "image: nginx"
```

#### 명령어 해석)

- 클러스터 내에서 nginx 이미지를 사용하는 모든 파드를 식별하고, 해당 파드들의 컨테이너 이미지 정보 부분만을 출력

```
(calico:default) root@k8s-m:~# k get pod -o yaml |grep "image: nginx"  
- image: nginx:1.11  
- image: nginx:1.11
```

```
k rollout history deployment deployment-nginx
```

#### 참고) rollout

- 파드 관리하는 Deployment 등의 리소스에 적용되는 업데이트 or 변경사항을 새로운 버전으로 점진적으로 배포하는 과정을 말함
- 새로운 이미지로의 업데이트, 구성 변경, 스케일링 등과 같은 변경사항을 클러스터에 적용할 때 사용됨

#### 명령어 해석)

- deployment-nginx Deployment의 업데이트 이력을 볼 수 있음
- Deployment 설정을 변경한 후 예상대로 롤아웃이 진행되었는지 확인하거나, 문제가 발생했을 때 이전 버전으로 롤백하기 위해 어떤 버전을 선택해야 할지 결정하는 데 유용

```
(calico:default) root@k8s-m:~# k rollout history deployment deployment-nginx
deployment.apps/deployment-nginx
REVISION  CHANGE-CAUSE
1          kubectl apply --filename=deployment-1.yaml --record=true
```

### 위 사진 분석)

#### • REVISION

- 리비전 번호를 나타냄
- 이 경우 '1'이며, 이는 첫 번째 롤아웃을 의미함

#### • CHANGE-CAUSE

- 변경 원인을 나타냄.
- 여기서는 처음에 deployment-1.yaml을 apply할 때 --record 옵션을 사용했다는 것을 나타내고 있음

```
while true; do kubectl rollout status deployment deployment-nginx && echo "-----"
&& date; sleep 1; done
```

### 명령어 해석)

- deployment-nginx라는 이름의 Deployment에 대한 롤아웃 상태를 지속적으로 모니터링하고, 현재 시간과 함께 상태를 출력하는 무한 루프 스크립트

#### • **kubectl rollout status deployment deployment-nginx**

- deployment-nginx Deployment의 현재 롤아웃 상태를 확인하라는 뜻.
- 이 명령어는 롤아웃이 성공적으로 완료되었는지, 진행 중인지, 또는 실패했는지에 대한 정보를 제공함

#### • **&& echo "-----"**

- 이전 명령어가 성공적으로 실행된 후에만 구분선을 출력하기
- **&&** : 이전 명령어가 성공적으로 실행되었을 때 다음 명령어를 실행하라는 의미

#### • **&& date**

- 현재 날짜와 시간을 출력

#### • **; sleep 1**

- 이전 명령어들의 실행이 끝난 후 1초간 대기하기
- ;: 이전 명령어의 실행 성공 여부와 관계없이 다음 명령어를 실행하라는 의미

#### • **done**

- while true 루프의 끝을 나타내며, 조건이 참(true)인 동안 무한히 반복 실행됨

```
Last login: Thu Feb 29 14:55:54 2024 from 10.0.2.2
(calico:default) root@k8s-m:~# while true; do kubectl rollout status deployment deployment-nginx && echo "-----" && date; sleep 1;
done
deployment "deployment-nginx" successfully rolled out
-----
Thu Feb 29 17:24:01 KST 2024
deployment "deployment-nginx" successfully rolled out
-----
Thu Feb 29 17:24:03 KST 2024
deployment "deployment-nginx" successfully rolled out
-----
Thu Feb 29 17:24:04 KST 2024
deployment "deployment-nginx" successfully rolled out
-----
Thu Feb 29 17:24:05 KST 2024
deployment "deployment-nginx" successfully rolled out
```

### 13-5. 버전 업데이트 하기

- 기존 버전인 1.11버전에서 1.12 버전으로 업데이트하기

```
kubectl set image deployment deployment-nginx nginx=nginx:1.12 --record && kubectl get pod -w
```

명령어 해석)

- kubectl set image** : Deployment 내의 컨테이너 이미지를 업데이트하는 명령어
- deployment deployment-nginx** : deployment-nginx라는 이름의 Deployment를 대상으로 함
- nginx=nginx:1.12** : 컨테이너(nginx)의 이미지를 nginx:1.12 버전으로 설정

```
kubectl get pod -o yaml | grep "image: nginx"
```

- 모니터링으로 확인하기

```
while true; do kubectl rollout status deployment deployment-nginx && echo "-----" && date; sleep 1; done
```

```
[1]+ Stopped                  kubectl rollout status deployment deployment-nginx
(calico:default) root@k8s-m:~# while true; do kubectl rollout status deployment deployment-nginx && echo "-----" && date; sleep 1;
done
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "deployment-nginx" rollout to finish: 1 old replicas are pending termination...
-----
```

- Deployment의 업데이트가 진행 중임을 나타내며, 3개 중 2개의 새로운 파드(replicas)가 업데이트되었다는 것을 의미
- 이전 버전의 1개 파드(replica)가 종료를 기다리고 있음을 나타냄
- 참고) 지금부터 1.16으로 업데이트한 사진을 캡쳐함

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset.apps/deployment-nginx-5c569bd9c5e-hash=5c569bd9c5	3	3	3	112s	nginx	nginx:1.15	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-6566c8bdb8e-hash=6566c8bdb8	1	1	0	4s	nginx	nginx:1.16	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-675576d94be-hash=675576d94b	0	0	0	4m59s	nginx	nginx:1.14	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-68594d9d7ce-hash=68594d9d7c	0	0	0	7m59s	nginx	nginx:1.13	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-6b8c48676ce-hash=6b8c48676c	0	0	0	15m	nginx	nginx:1.11	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-7d87ffc656e-hash=7d87ffc656	0	0	0	10m	nginx	nginx:1.12	app=cndk-nginx-pods,pod-template
NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
deployment.apps/deployment-nginx	3/3	1	3	15m	nginx	nginx:1.16	app=cndk-nginx-pods

```
| Waiting for deployment "deployment-nginx" rollout to finish: 1 out of 3 new replicas have been updated.
```

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset.apps/deployment-nginx-5c569bd9c5 e-hash=5c569bd9c5	2	2	2	2m9s	nginx	nginx:1.15	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-6566c8bdb8 e-hash=6566c8bdb8	2	2	1	21s	nginx	nginx:1.16	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-675576d94b e-hash=675576d94b	0	0	0	5m16s	nginx	nginx:1.14	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-68594d97c e-hash=68594d97c	0	0	0	8m16s	nginx	nginx:1.13	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-6b8c48676c e-hash=6b8c48676c	0	0	0	15m	nginx	nginx:1.11	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-7d87ffc656 e-hash=7d87ffc656	0	0	0	10m	nginx	nginx:1.12	app=cndk-nginx-pods,pod-template
READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR	
deployment.apps/deployment-nginx	3/3	2	3	15m	nginx	nginx:1.16	app=cndk-nginx-pods

```
Thu Feb 29 17:36:14 KST 2024
Waiting for deployment "deployment-nginx" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
```

NAME	DESIRED	CURRENT	READY	AGE	CONTAINERS	IMAGES	SELECTOR
replicaset.apps/deployment-nginx-5c569bd9c5 e-hash=5c569bd9c5	0	0	0	2m31s	nginx	nginx:1.15	app=cndk-nginx-pods,pod-template
replicaset.apps/deployment-nginx-6566c8bdb8 e-hash=6566c8bdb8	3	3	3	48s	nginx	nginx:1.16	app=cndk-nginx-pods,pod-template

```
Thu Feb 29 17:36:14 KST 2024
Waiting for deployment "deployment-nginx" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "deployment-nginx" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "deployment-nginx" rollout to finish: 1 old replicas are pending termination...
deployment "deployment-nginx" successfully rolled out
```

- **k rollout history deployment deployment-nginx**로도 1.16으로 업데이트 되었다는 사실을 확인 할 수 있음

```
(calico:default) root@k8s-m:~# k rollout history deployment deployment-nginx
deployment.apps/deployment-nginx
REVISION  CHANGE-CAUSE
1          kubectl apply --filename=deployment-1.yaml --record=true
2          kubectl set image deployment deployment-nginx nginx=nginx:1.12 --record=true
3          kubectl set image deployment deployment-nginx nginx=nginx:1.13 --record=true
4          kubectl set image deployment deployment-nginx nginx=nginx:1.14 --record=true
5          kubectl set image deployment deployment-nginx nginx=nginx:1.15 --record=true
6          kubectl set image deployment deployment-nginx nginx=nginx:1.16 --record=true
```

## 13-6. 롤백하기

- 기존 버전으로 되돌리기

**k rollout undo deployment deployment-nginx --to-revision 2**

- 13-5의 가장 마지막 사진에서 보면 Revision 2에 해당하는 것은 1.12 버전임

so 위의 명령어는 1.12로 롤백하라는 의미임

```
(calico:default) root@k8s-m:~# k rollout undo deployment deployment-nginx --to-revision 2
deployment.apps/deployment-nginx rolled back
```

- 확인하기

### k rollout history deployment deployment-nginx

```
(calico:default) root@k8s-m:~# k rollout history deployment deployment-nginx
deployment.apps/deployment-nginx
REVISION  CHANGE-CAUSE
1          kubectl apply --filename=deployment-1.yaml --record=true
3          kubectl set image deployment deployment-nginx nginx=nginx:1.13 --record=true
4          kubectl set image deployment deployment-nginx nginx=nginx:1.14 --record=true
5          kubectl set image deployment deployment-nginx nginx=nginx:1.15 --record=true
6          kubectl set image deployment deployment-nginx nginx=nginx:1.16 --record=true
7          kubectl set image deployment deployment-nginx nginx=nginx:1.12 --record=true
(calico:default) root@k8s-m:~#
```

- 가장 최신으로 설정한 버전이 가장 아래에 뜸  
⇒ 1.12이 버전으로 뒤로 한 것이 성공함을 알 수 있음

## 출처

[https://velog.io/@\\_zero\\_/%EC%BF%A0%EB%B2%84%EB%84%A4%ED%8B%B0%EC%8A%A4-%EC%B%BA4%EB%A7%A8%EB%93%9CCommand-%EB%B0%8F-%EC%9D%B8%EC%9E%90Arguments-%EC%84%A4%EC%A0%95](https://velog.io/@_zero_/%EC%BF%A0%EB%B2%84%EB%84%A4%ED%8B%B0%EC%8A%A4-%EC%B%BA4%EB%A7%A8%EB%93%9CCommand-%EB%B0%8F-%EC%9D%B8%EC%9E%90Arguments-%EC%84%A4%EC%A0%95)

<https://nirsa.tistory.com/156>

<https://kimjingo.tistory.com/137>

<https://seongjin.me/kubernetes-pods/>

<https://jh-labs.tistory.com/503>

<https://jh-labs.tistory.com/506>

지원님, 동진님, 대수님 감사합니다.