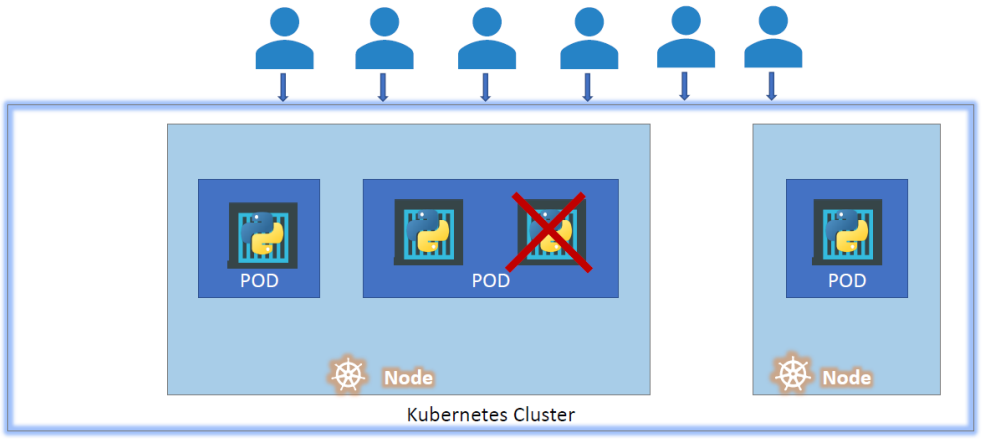
(4) Pods, ReplicaSets

▼ 목차

❤️Pods❤️

pod란?

- **앱의 단일 인스턴스**
- 쿠버네티스에서 만들 수 있는 **가장 작은 물체**
- 보통 pod은 **앱을 실행하는 컨테이너와 1:1 관계**를 가짐
  - 규모를 키우려면 새 pod을 생성하고, 줄이려면 기존 pod을 삭제함



- pod에 어떤 컨테이너로 구성됐는지 설정하면, 알아서 배포해줌

YAML로 Pods 생성

(1) Pods 정의 파일 생성

- **<u>쿠버네티스 정의 파일은 항상 4개의 상위 레벨 필드를 포함</u>**함

1. **apiVersion**
   - 문자열 형태
   - 개체를 생성할 때 사용하는 쿠버네티스 api 버전
2. **kind**
   - 문자열 형태
   - 만들려는 개체 유형
3. **metadata**
   - 딕셔너리 형태
   - 개체 상위의 데이터 ex) 이름, 라벨
   - metadata에 포함될만한 name/label만 허용함. 원하는대로 다른 속성 추가할 수 X

     but, metadata 하의 label 내에서는 원하는대로 key-value 쌍 작성 가능
4. **spec**
   - 딕셔너리 형태
   - 생성하려는 개체에 따라, 쿠버네티스에게 그 개체와 관련된 추가정보를 제공함
   - 각각에 맞는 형식을 갖추기 위해서는 문서 섹션을 이해/참조하는 것이 중요
   - 구성

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

1st Item in List

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

```
kubectl create -f pod-definition.yml
```

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

- **containers**: list/array 형태. (여러 개의 컨테이너가 하나의 pod 안에 들어갈 수 있기 때문)

(2) 정의 파일 실행

```
# 사용 가능한 포드 목록 확인
> kubectl get pods

# pod에 대한 자세한 설명 확인
> kubectl describe pod [pod 이름]
```

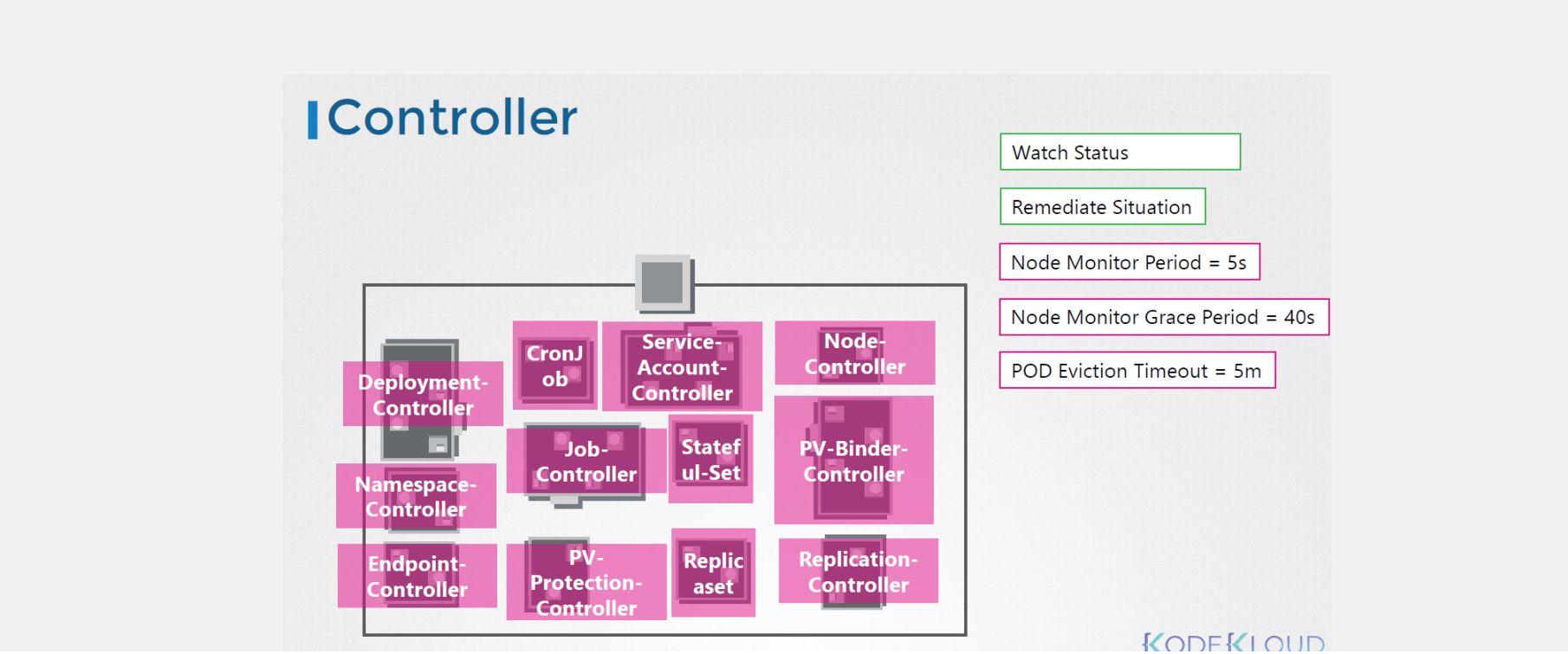
🧡ReplicaSets🧡

ReplicaSets 개요

- 쿠버네티스 컨트롤러 → 쿠버네티스의 두뇌!
  - 쿠버네티스를 모니터링, 그에 따라 반응함

▼ 쿠버네티스 내에는 **다양한 controller가 존재**함

⇒ 이들을 하나로 묶어 **Kube Controller Manager라는 프로세스로 패키지화**되어 있음



→ 이 중에 하나가 ReplicaSet!

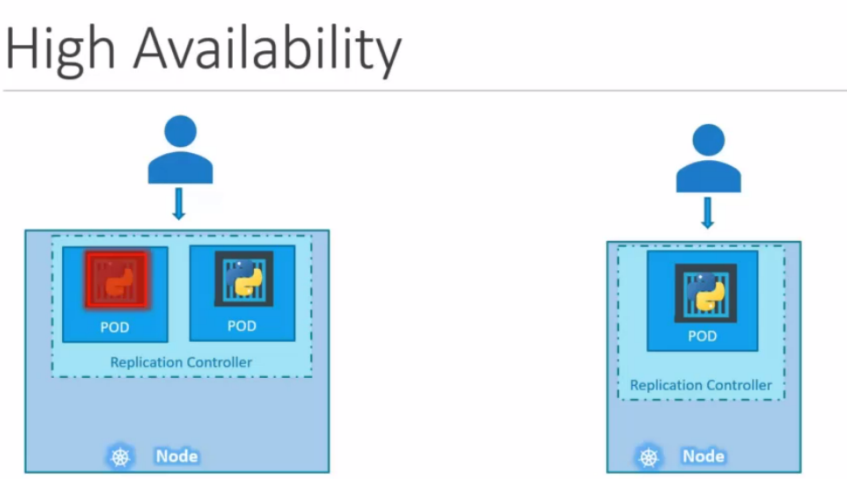
ReplicaSet의 필요성?

1. 사용자가 서비스에 계속해서 접근하려면, 한 개 이상의 인스턴스/pod이 실행되고 있어야 함

   ⇒ **ReplicaSet은 특정 pod이 항상 실행되도록 보장함!**

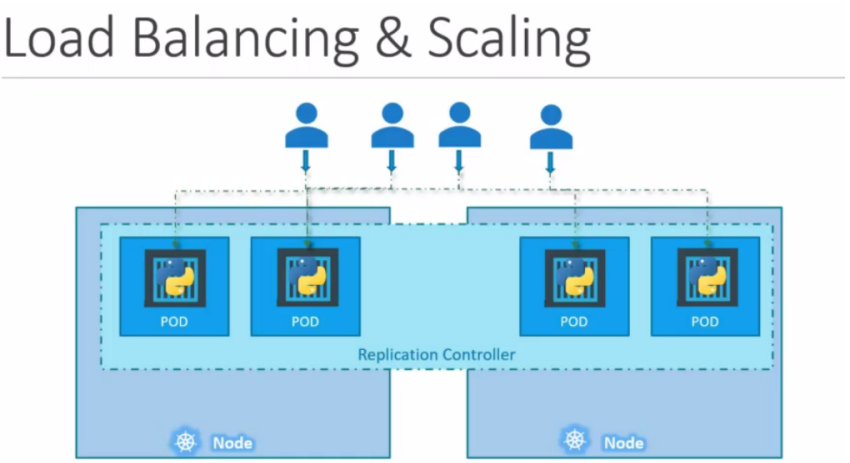
   - **Replication Controller**

- 쿠버네티스 클러스터에 있는 단일 pod의 다중 인스턴스를 실행하도록 도와줌 → **고가용성 (High Availability)** 제공
- pod이 하나라면, 기존 pod이 고장 났을 때 새로운 pod을 자동으로 불러오도록 함



2. **<u>여러 개의 pod을 만들어 로드를 공유</u>**하기 위함
   - **서로 다른 노드의 여러 Pod에 걸쳐 부하를 분산하는 데 도움**이 됨
   - 수요가 증가하면, **앱 scale도 조정** 가능

   ex) 서비스를 확장하기 위해 새로운 노드에 pod을 생성, 이들의 로드를 공유하기 위함



→ replication controller는 클러스터 내 여러 노드에 뻗어있음!

Replication Controller vs Replica Set

- 용도는 같지만, 다름!
- 앞선 내용을 둘 다 구현 가능하지만, 그 방법에 작은 차이가 있음

  → **Selector! (관리할 pod를 명시)**

Replication Controller

- 구식 기술
- Replica Set으로 대체되고 있음

Replica Set

- 복제를 설정하는 새로운 권장 방법

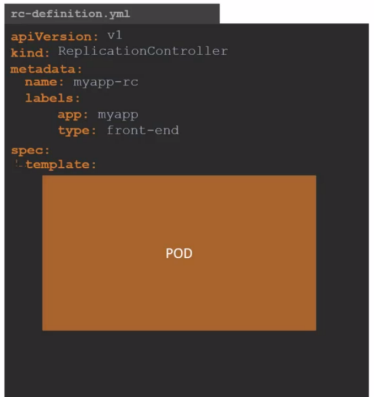
Replication Controller 생성 방법

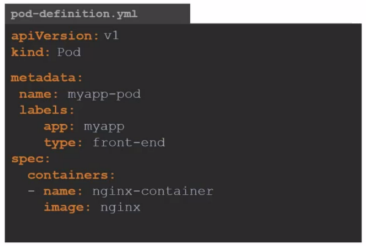
(1) 정의 파일 생성: YAML

→ Pod 생성과 마찬가지로 4개의 section 존재

1. **apiVersion**
   - 정의하려는 것에 따라 다름
   - Replication Controller는 api v1에서 지원함
2. **kind**
   - 만들려는 개체 유형

     : ReplicationController
3. **metadata**
   - name
   - labels
     - app
     - type
4. **spec**
   - 만드는 개체 안에 무엇이 들어 있는지 정의



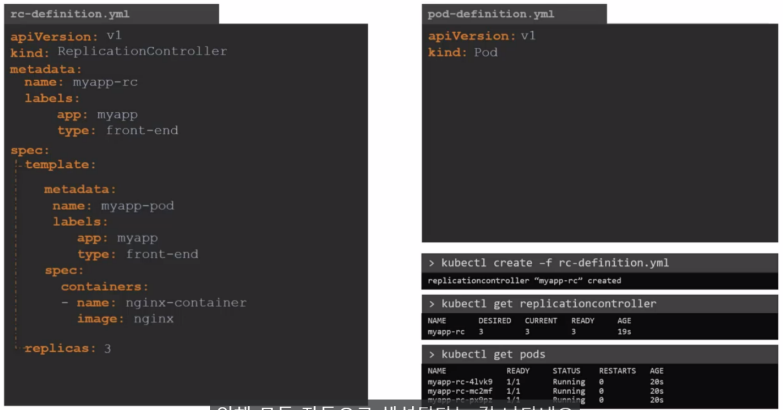


- Replication Controller는 pod에 여러 개의 인스턴스를 만듬

  → 하위에 pod 생성 템플릿이 필요함

  ⇒ 이를 template 섹션에 작성함

- **template 섹션**
  - replication controller가 복제본을 만들기 위해 사용할 pod template을 제공하기 위함
  - 기존 pod 정의 파일에서 apiVersion, kind를 제외한 전부를 가져오면 됨
- **replicas 섹션**
  - 필요한 복제본의 수 명시



(2) 정의 파일 실행

```
# 복제 컨트롤러 생성
# f: 파일 명시
> kubectl create -f [Replication Controller 정의 파일]

# 생성된 replication controler의 목록 확인
> kubectl get replicationcontroller

# replicatoin controller로 생성된 pod 목록 확인
> kubectl get pods
```

Replica Set 생성 방법

: Replication Controller와 매우 유사

(1) 정의 파일 생성: YAML

→ Pod 생성과 마찬가지로 4개의 section 존재

1. **apiVersion**
   - Replica Set은 apps/v1에서 지원함
2. **kind**
   - 만들려는 개체 유형

     : ReplicaSet
3. **metadata**
4. **spec**
   - **template, replicas**
   - **selector → Replication Controller와 다른 부분!**

> **어떤 pod을 관리 해야 할지** 알려줌!
>
> - **Replica Set 밑에 놓인 pod를 식별**할 수 있게 해줌
>
>   → label로 식별

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
      matchLabels:
        type: front-end
```

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
```

```
> kubectl create -f replicaset-definition.yml
replicaset "myapp-replicaset" created
> kubectl get replicaset
NAME            DESIRED   CURRENT   READY     AGE
myapp-replicaset   3         3         3         19s
> kubectl get pods
NAME                    READY     STATUS    RESTARTS   AGE
myapp-replicaset-9dd19   1/1       Running   0          45s
myapp-replicaset-9jtpx   1/1       Running   0          45s
myapp-replicaset-hq84m   1/1       Running   0          45s
```

> - 왜 정의 해야 할까?
>   - Replica Set은 Replica Set의 일부로 만들어지지 않은 pod도 관리할 수 있기 때문!

(2) 정의 파일 실행

```
# 정의파일에 따라 Replica Set 생성
> kubectl create -f [Replica Set 정의 파일]

# 생성된 Replica Set 목록 확인
> kubectl get replicaset

# 생성된 pod 목록 확인
> kubectl get pods
```

규모(Scale) 조정 방법

1. 정의 파일의 replicas 업데이트 → kubectl replace 명령

```
# replicas의 수를 수정하고, 다시 실행
> kubectl replace [수정한 Replica Set 정의 파일]
```

2. kubectl scale 명령

- replicas 파라미터를 업데이트

```
# 방법 1: 수정할 Replica Set 파일 명시
> kubectl scale --replicas=6 -f [수정할 Replica Set 정의 파일]

# 방법 2: 수정할 Replica Set의 name 명시
> kubectl scale --replicas=6 replicaset [Replica Set name]
```

> **scale 명령어 주의!**
>
> **정의 파일 자체가 변경되는 것은 아님!** 정의 파일은 변경되기 이전에 정의된 수 그대로임!

3. 부하에 따라 자동으로 replicas 수를 조절해주는 옵션도 있음

💡 **kubectl 명령어** 복습

```
# 생성: create
> kubectl create -f [정의 파일명]

# 목록 확인: get
> kubectl get [replicaset/pods]

# 삭제: delete
> kubectl delete
# name으로 replicaset 삭제
> kubectl delete replicaset [replicaset name]
# replicaset을 삭제할 때, 그 밑의 Pods도 제거됨

# 교체/업데이트: replace
> kubectl replace -f [정의 파일명]

# replicas 스케일 조정: scale
> kubectl scale -replicas=[조정 숫자] -f [정의 파일명]
```