쿠버네티스 세미나

https://www.youtube.com/watch?v=ZuIQurh_kDk

세미나 개요

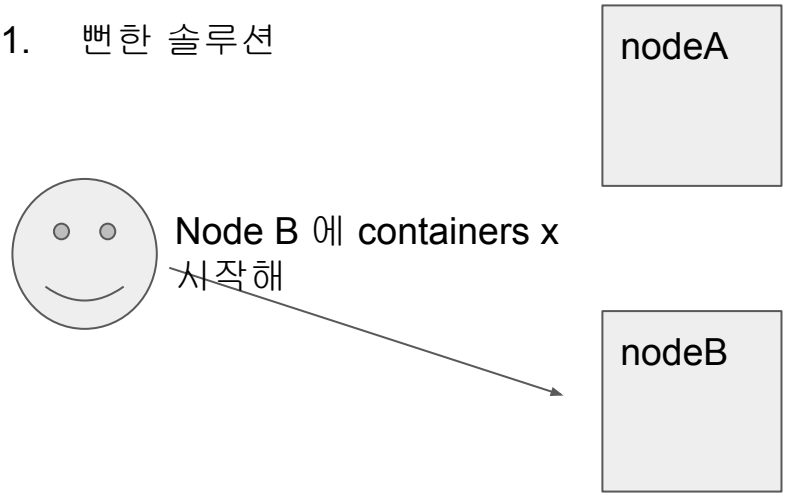
- 목적: 쿠버네티스에 대한 더 깊은 이해
- 배움을 위한 중요한 도구
  - 문제를 이해하는 것
  - 무엇에 그치지 않고 왜라고 묻는 것

쿠버네티스

- 컨테이너화 하는 것이 key
  - 아주 다양한 시스템에서의 지속가능한, 반복가능한, 신뢰성있는 배포
- 누가 관리할까
  - 당신? 스크립트? 당신이 작성한 시스템?
- 쿠버네티스가 당신의 시스템을 관리한다
  - 컨테이터화된 workload를 배포, 모니터링

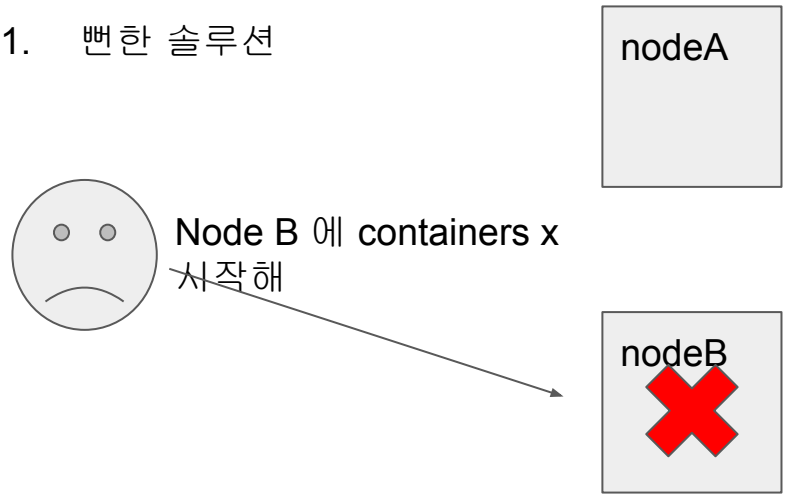
어떻게 workload를 배포

1. 뻔한 솔루션



어떻게 workload를 배포할까

1. 뻔한 솔루션



2. 만약:
   - 컨테이너가 죽는다면
   - 노드가 죽는다면
   - 노드B가 일시적으로 장애가 발생한다면

3. 사용자는:
   - 모든 컨테이너와 노드의 상태를 모니터링하고 저장해야 함
   - 확인되지 않은 모든 실패한 노드를 처리해야 함

=> 복잡한 커스텀 로직

쿠버네티스 디자인 원칙 #1

쿠버네티스API 는 명령형이 아니라 선언적임

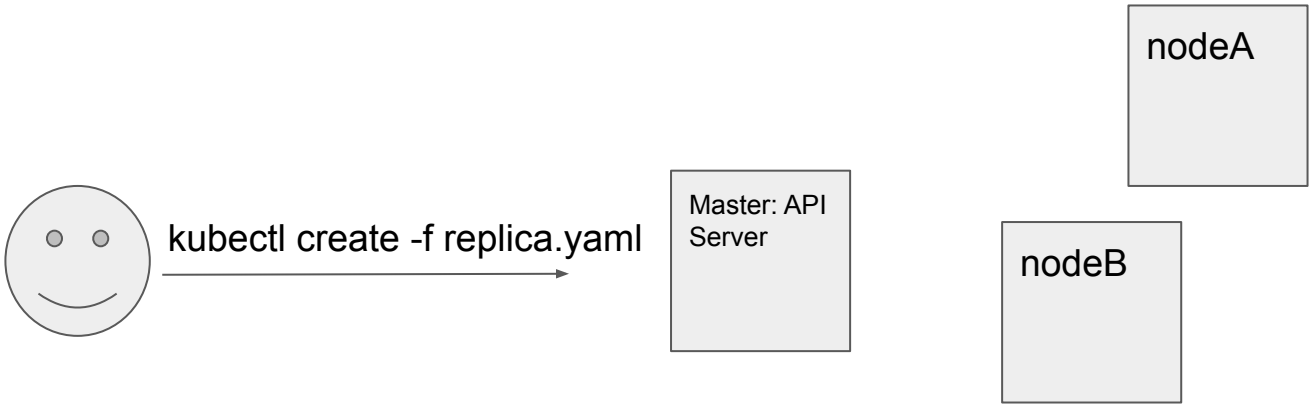
선언적인 APIs

- 이전
  - 당신: 원하는 상태로 만들기 위해 정확한 명령어의 집합을 제공
  - 시스템: 명령어를 실행
  - 당신: 시스템을 모니터링하고 필요하다면 다른 명령어를 제공
- 이후
  - 당신: 원하는 상태를 정의함
  - 시스템: 그 상태로 만들기 위해 일함

어떻게 workload를 배포할까

쿠버네티스 방식

- 당신: 삭제 전까지 kube API 서버에 존재하는 API 객체를 생성
- 시스템: 모든 요소가 그 상태로 만들기 위해 병렬적으로 일함



어떻게 workload를 배포할까

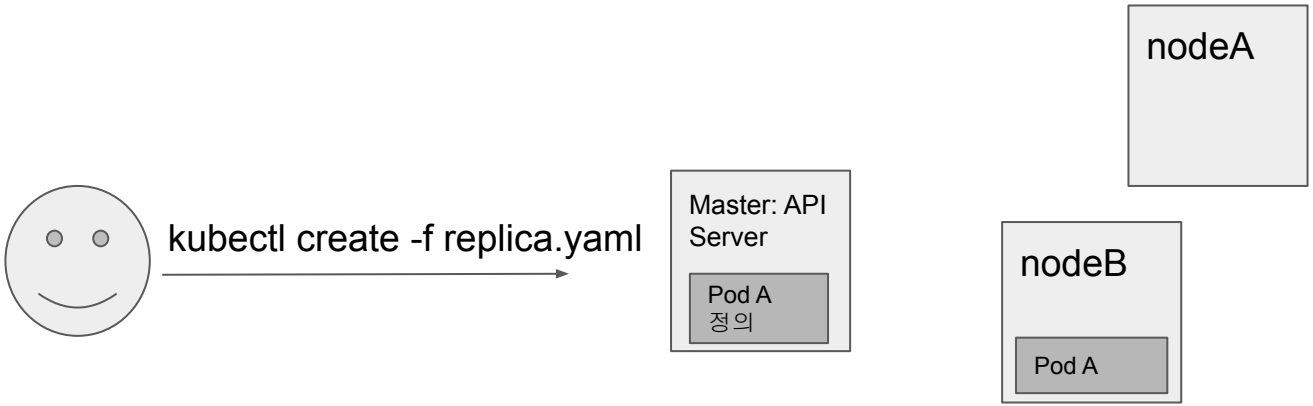
쿠버네티스 방식
- 당
- 사

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
spec:
  replicas: 1
  template:
    metadata:
…
    spec:
…
      containers:
      - name: nginx
        image: internal.mycorp.com:5000/mycontainer:1.7.9
```

어떻게 workload를 배포할까

쿠버네티스 방식

- 당신: 삭제 전까지 kube API 서버에 존재하는 API 객체를 생성
- 시스템: 모든 요소가 그 상태로 만들기 위해 병렬적으로 일함

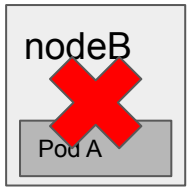


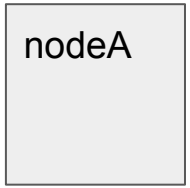
왜 명령형보다 선언형이 좋을까

- 자동적인 회복

왜 명령형보다 선언형이 좋을까

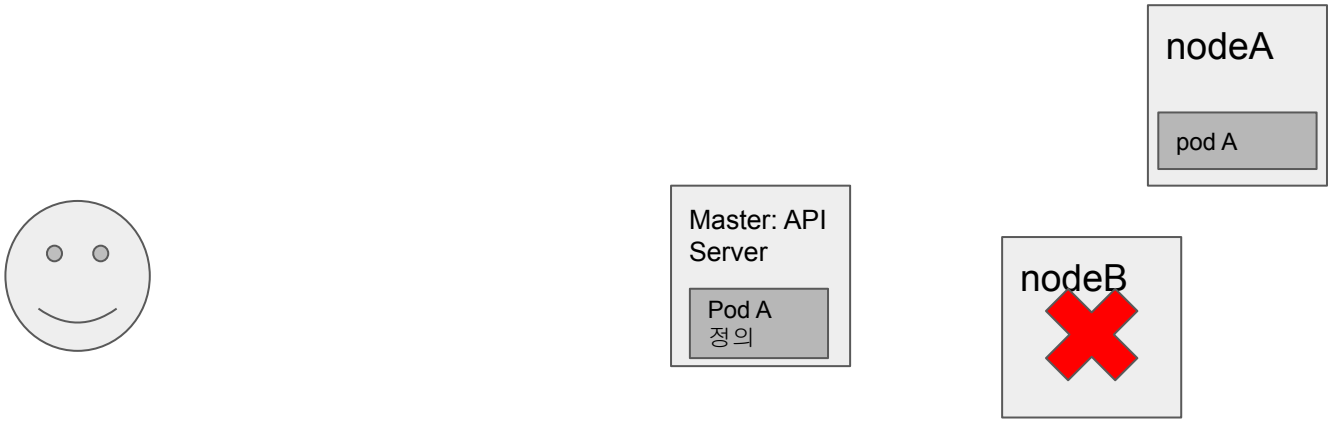
- 자동적인 회복 예시
  - 1. 노드 장애





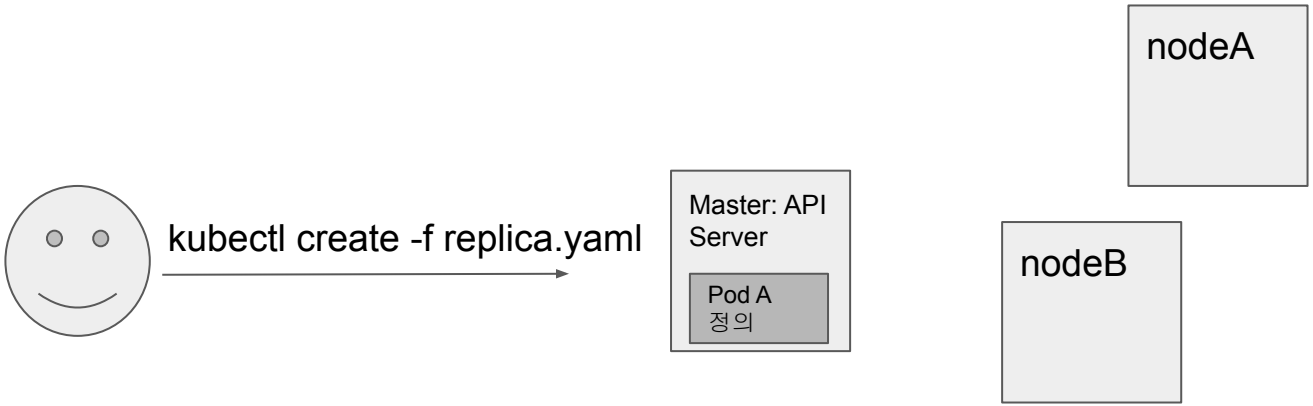
왜 명령형보다 선언형이 좋을까

- 자동적인 회복 예시
  - 1. 노드 장애
  - 2. 시스템이 자동적으로 파드를 헬시 노드로 옮김



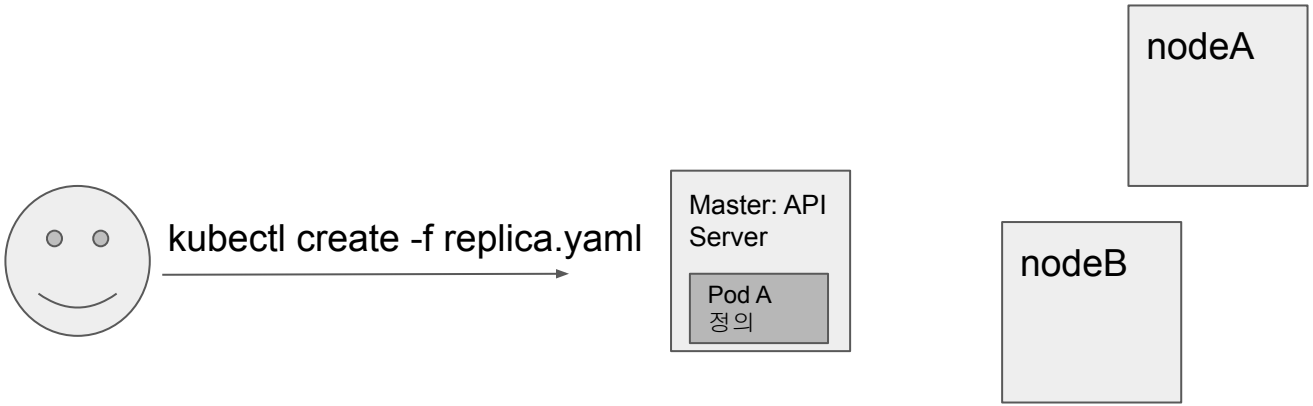
어떻게 workload를 배포할까

쿠버네티스 방식을 더 깊게 살펴보자



어떻게 workload를 배포할까

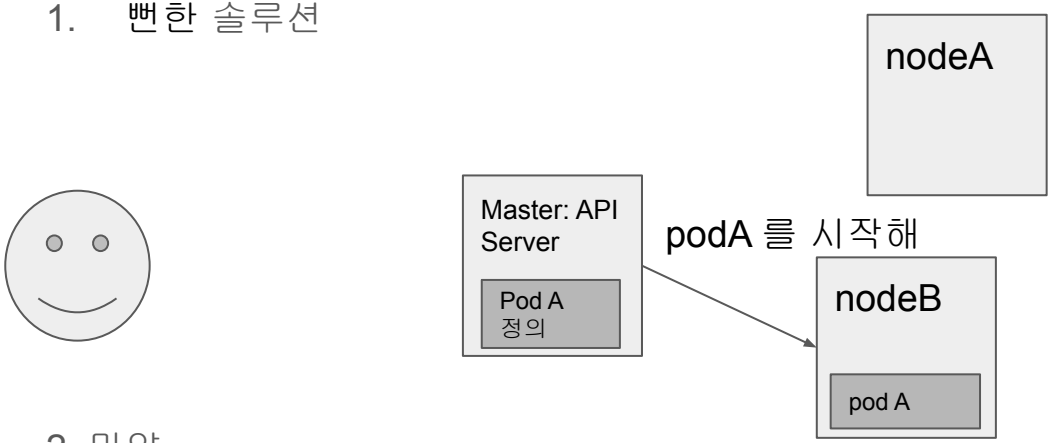
노드는 무슨일을 해야 할 지 어떻게 알아낼까?



어떻게 workload를 배포할까

노드는 무슨일을 할지 어떻게 알아낼까?

1. 뻔한 솔루션



2. 만약

- 컨테이너가 죽으면
- 노드가 죽으면
- 노드 B가 일시적인 장애가 있으면

3. 마스터는

- 모든 컨테이너와 노드의 상태를 모니터링하고 저장해야 함
- 확인되지 않는 모든 실패한 노드를 처리해야 함

=> 마스터는 복잡하고
취약하고 확장성이 떨어짐

쿠버네티스 디자인 원칙 #2

쿠버네티스 컨트롤플레인은 투명하다

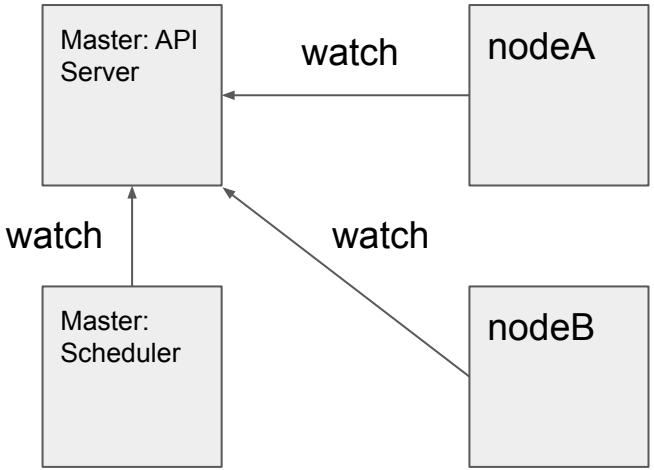
숨겨진 내부 API는 없다

투명한 API

- 이전
  - 마스터: 원하는 상태로 만들기 위해 정확한 명령어의 집합을 제공
  - 노드: 명령어를 실행
  - 마스터: 시스템을 모니터링하고 필요하다면 다른 명령어를 제공
- 이후
  - 마스터: 원하는 상태를 정의함
  - 노드: 자신을 그 상태로 만들기 위해 독립적으로 일함

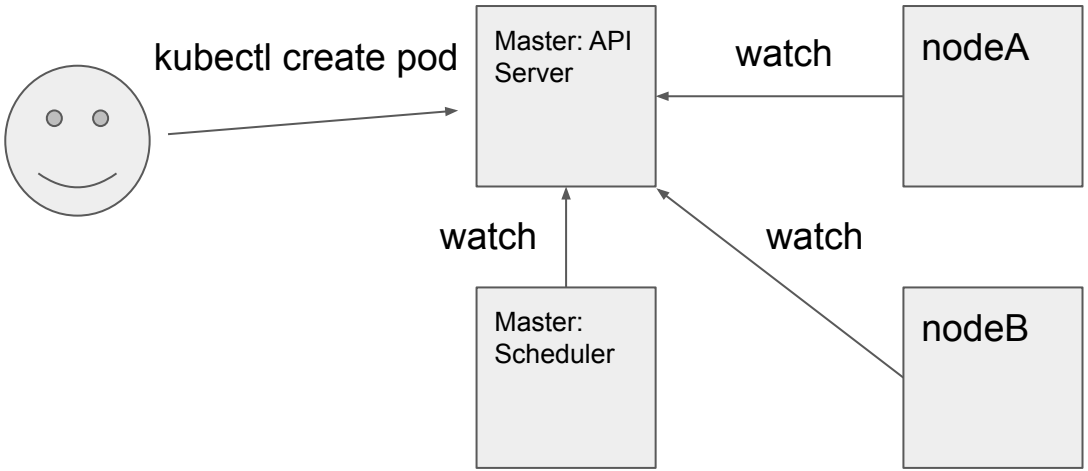
투명한 API

모든 요소는 쿠버네티스 API를 보고 그들이 해야할 일을 알아낸다



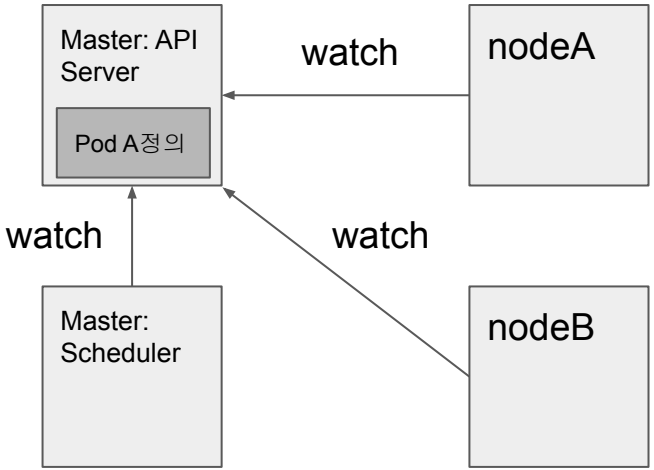
투명한 API

모든 컴포넌트는 쿠버네티스 API를 보고 그들이 해야할 일을 알아낸다



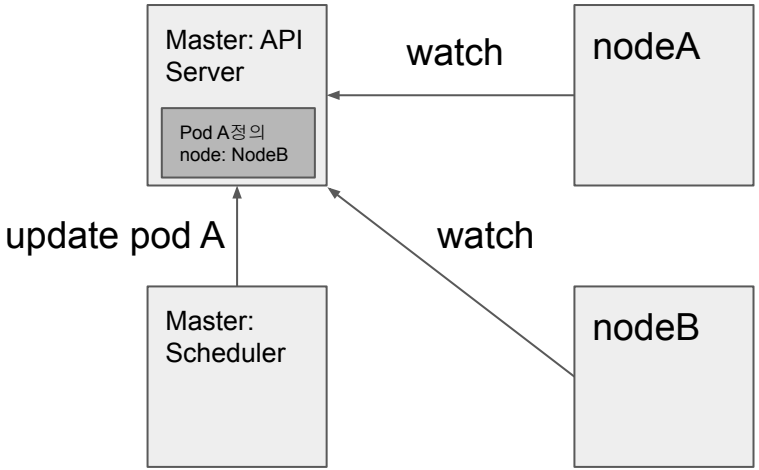
투명한 API

모든 요소는 쿠버네티스 API를 보고 그들이 해야할 일을 알아낸다



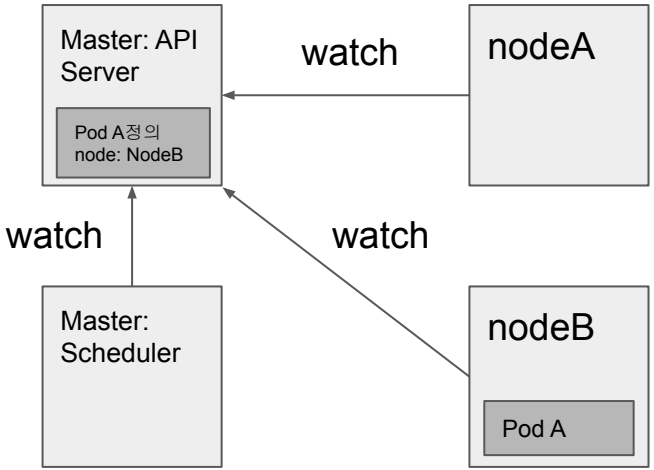
투명한 API

모든 요소는 쿠버네티스 API를 보고 그들이 해야할 일을 알아낸다



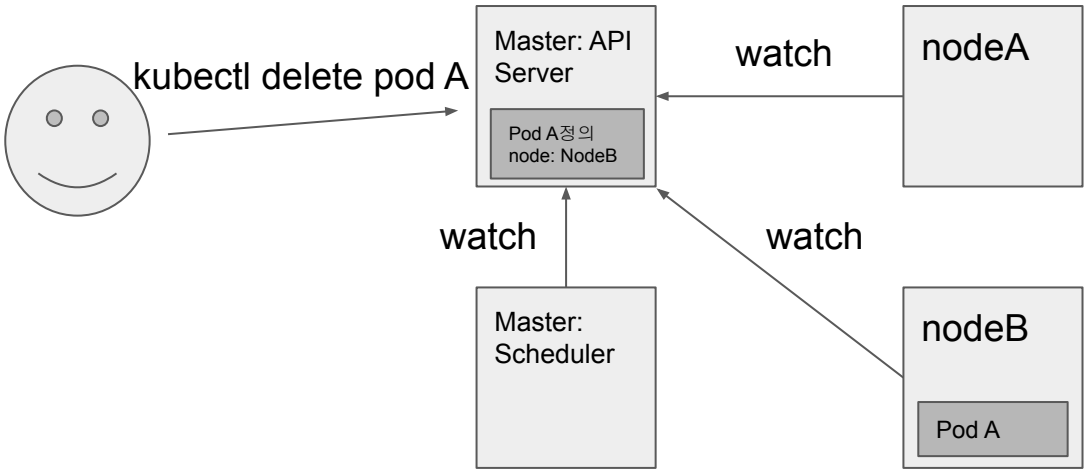
투명한 API

모든 요소는 쿠버네티스 API를 보고 그들이
해야할 일을 알아낸다



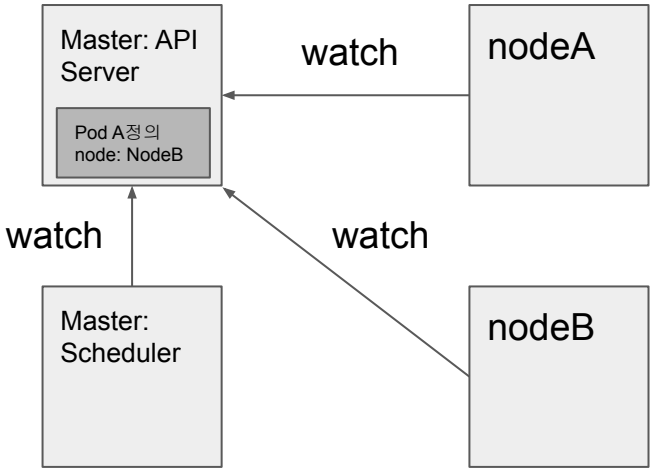
투명한 API

모든 요소는 쿠버네티스 API를 보고 그들이 해야할 일을 알아낸다



투명한 API

모든 요소는 쿠버네티스 API를 보고 그들이 해야할 일을 알아낸다



투명한 API 의 장점

- 선언형 API 는 내부 요소와 동일한 이점을 제공한다
  - 컴포넌트는 엣지 트리거가 아닌 레벨 트리거됨 -> 이벤트를 놓치는 문제가 없음
- 컴포넌트 장애를 쉽게 회복할 수 있는 더 단순하고 강력한 시스템을 만듦
  - 단일 장애점(동작하지 않으면 전체 시스템이 중단되는 요소) 이 없음
  - 단순한 마스터 컴포넌트
- 쿠버네티스를 조립가능하고 확장가능하게 함
  - 기본 컴포넌트가 당신에게 동작하지 않는다면
    - 당신만의 컴포넌트로 대체하라
  - 추가 기능이 아직 사용가능하지 않다면
    - 직접 당신만의 기능을 작성하고 추가하라

Kube API 데이터

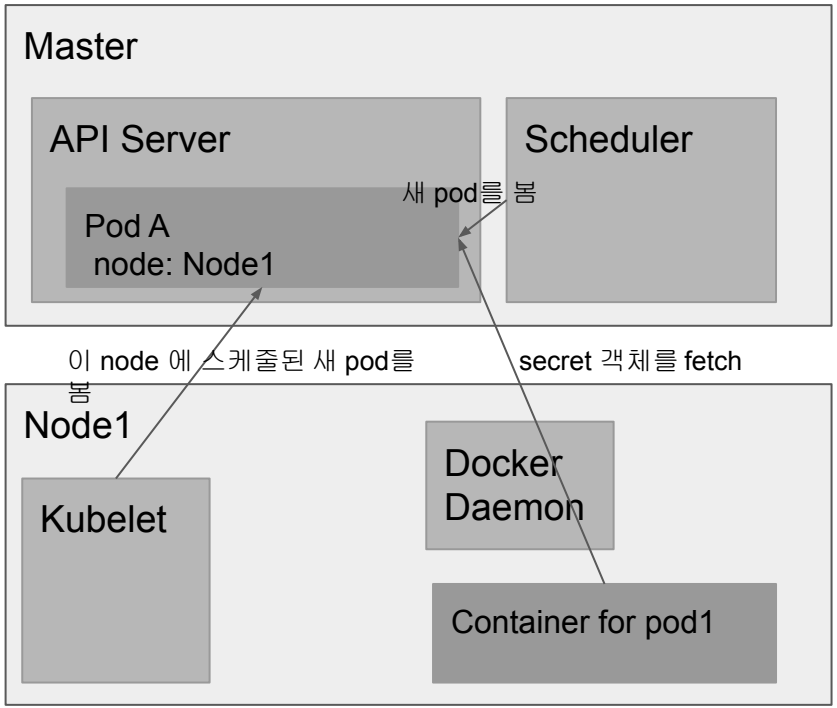
쿠버네티스 API 서버는 워크로드에 관련된 수많은 데이터를 가진다

- secrets - KubeAPI 에 저장된 민감한 정보
  - e.g. 비밀번호, 인증서 등
- config map - Kube API 에 저장된 설정 정보
  - e.g. 애플리케이션 startup 파라미터 등
- DownwardAPI - KubeAPI 에 저장된 Pod 정보
  - e.g. 현재 pod의 이름/namespace/uid

Kube API 데이터를 fetch 하기

애플리케이션은 어떻게 Secrets, config map 등의 정보를 fetch 할까

- 원칙: 숨겨진 내부 API 없음
- 뻔한 솔루션: API서버에서 데이터를 직접 읽어오도록 앱을 수정함

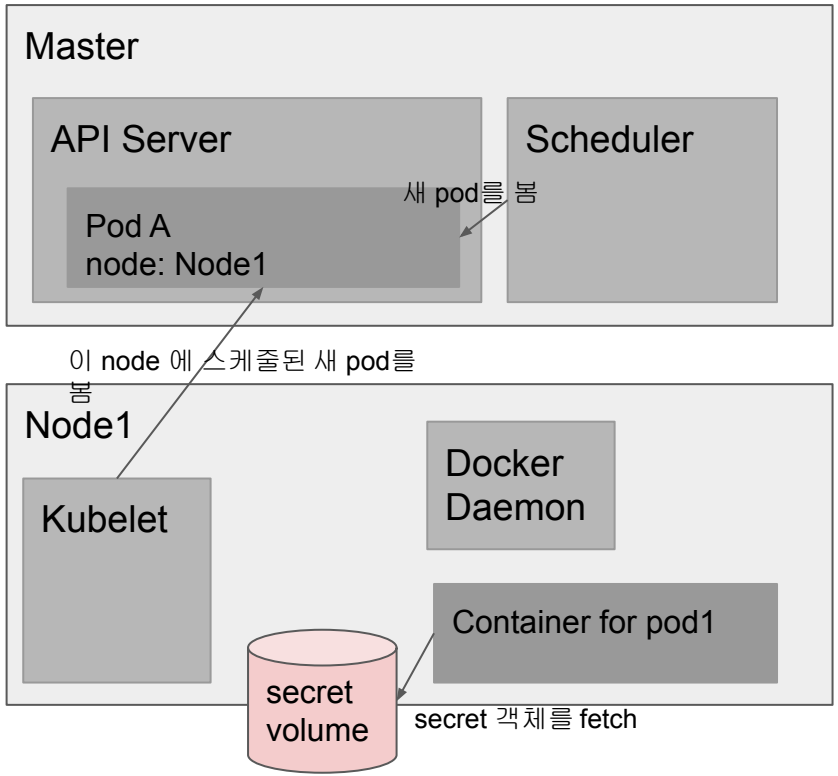


쿠버네티스 디자인 원칙 #3

- 사용자들을 그들이 있는 곳에서 만나라

사용자들을 그들이 있는 곳에서 만나기

- 이전:
  - 앱은 쿠버네티스를 알아차리기 위해 수정되어야만 함
- 이후:
  - 앱이 파일이나 환경변수로부터 설정이나 secret 데이터를 가져올 수 있다면 수정되지 않아도 됨



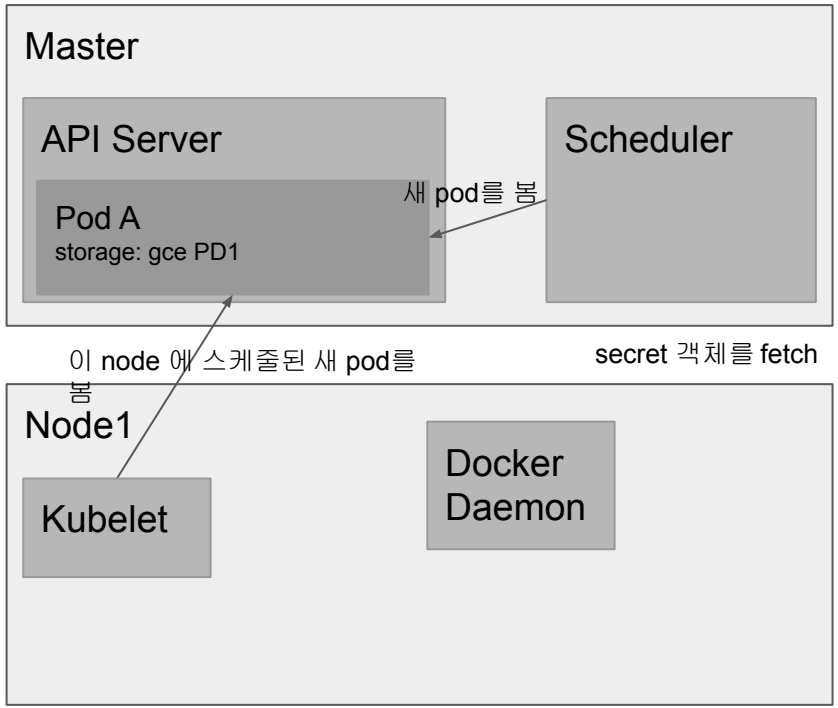
사용자들을 그들이 있는곳에서 만나는 것의 장점

쿠버네티스에 워크로드를 배포하기 위한 허들을 최소화

쿠버네티스 채택(adoption)을 증가시킴

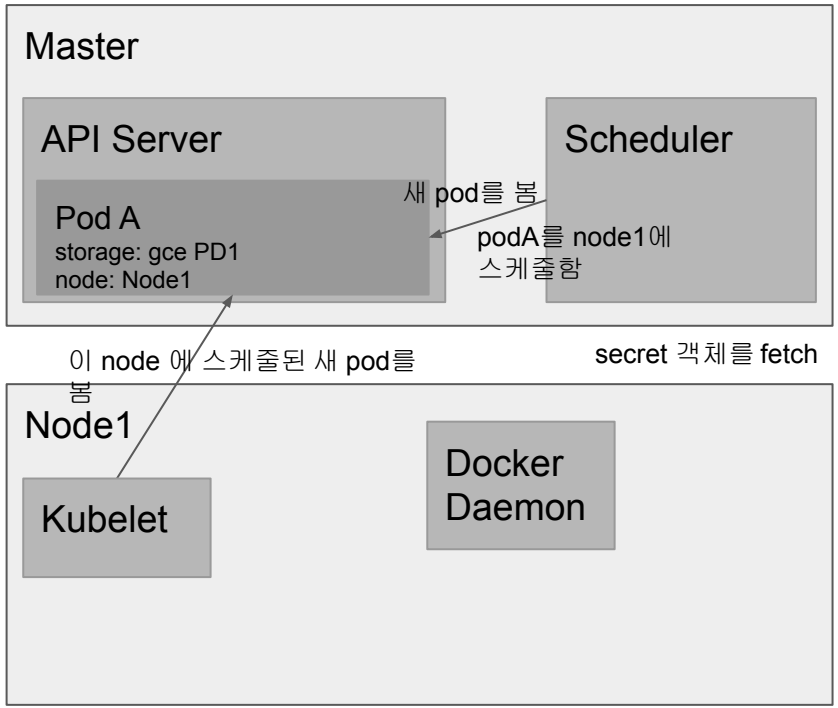
원격 저장소

- pod 정의에서 직접 원격 저장소(GCE PD, AWS EBS, NFS etc)를 참조할 수 있음



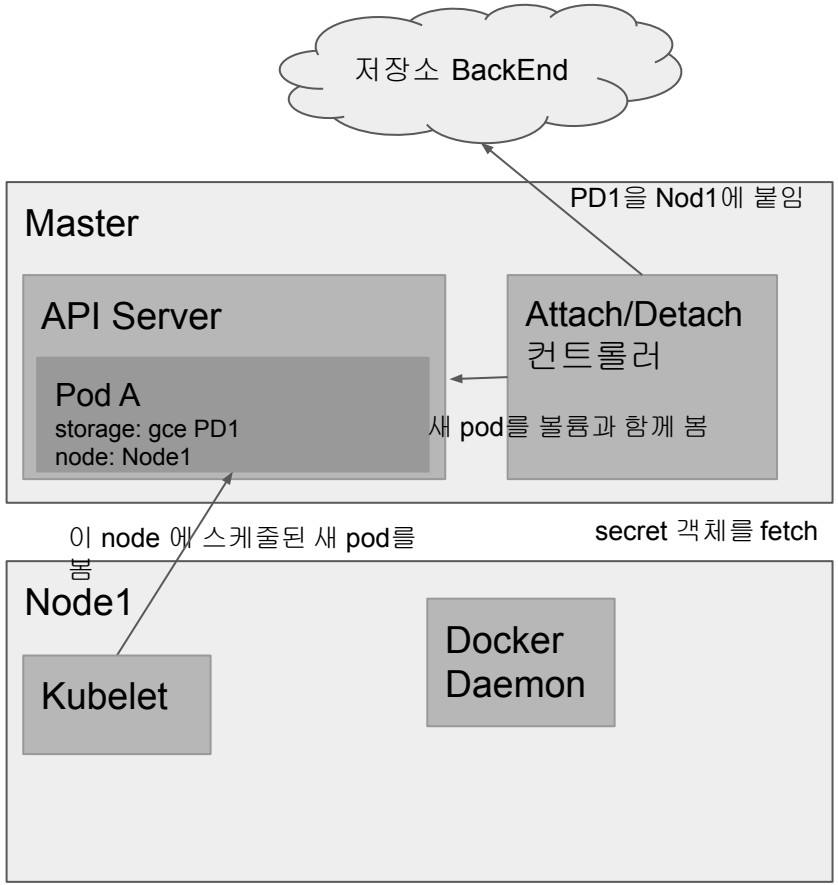
원격 저장소

- pod 정의에서 직접 원격 저장소(GCE PD, AWS EBS, NFS etc)를 참조할 수 있음
- 쿠버네티스는 자동적으로 원격 저장소가 워크로드에 사용가능하도록 만든다



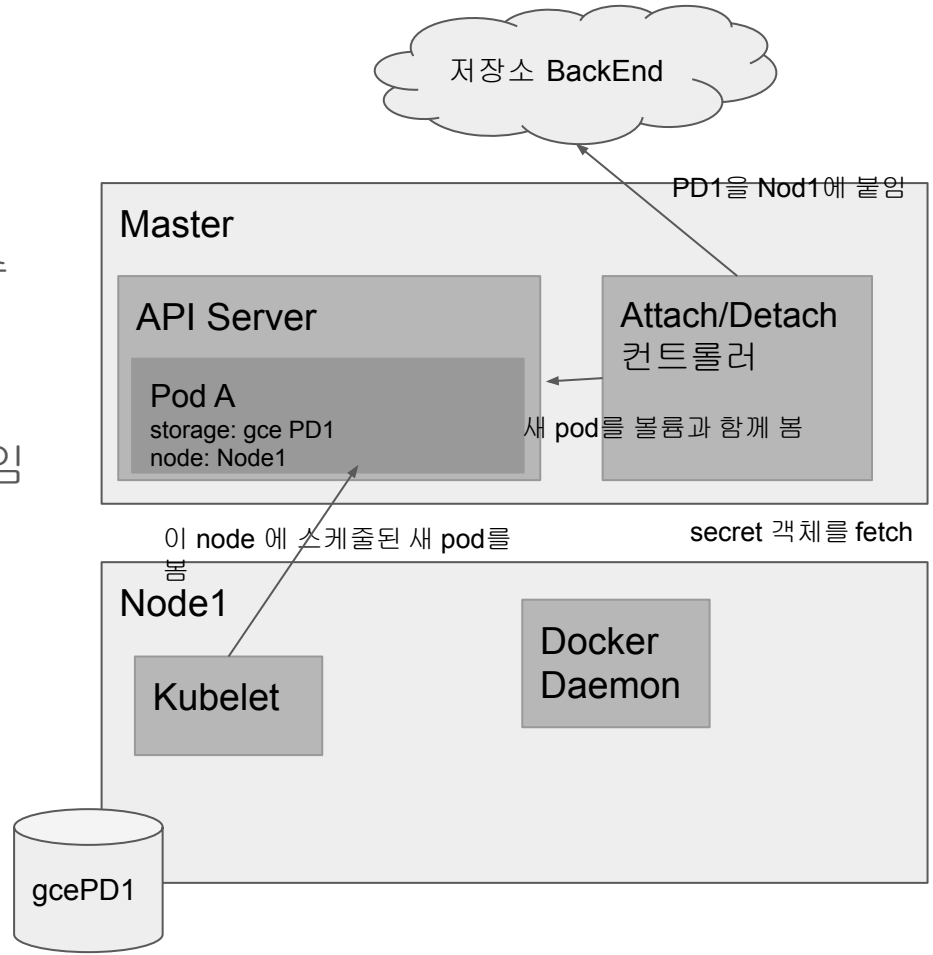
원격 저장소

- pod 정의에서 직접 원격 저장소(GCE PD, AWS EBS, NFS etc)를 참조할 수 있음
- 쿠버네티스는 자동적으로 원격 저장소가 워크로드에 사용가능하도록 만든다



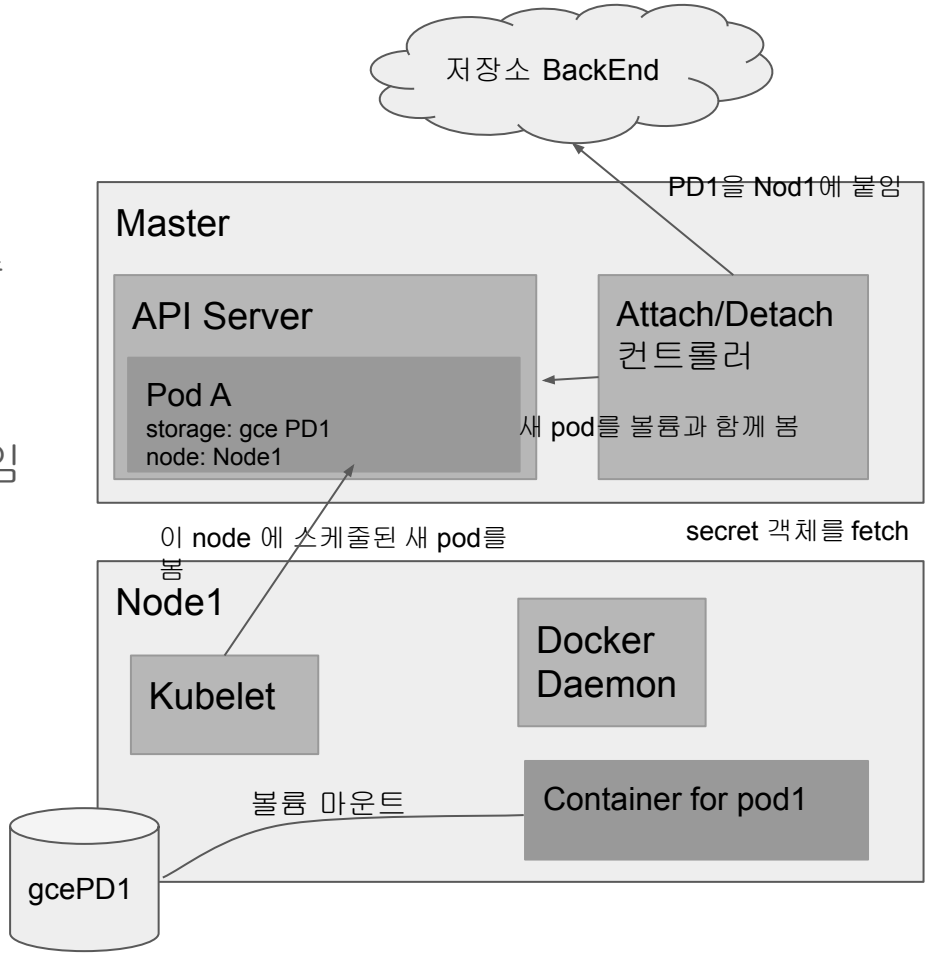
원격 저장소

- pod 정의에서 직접 원격 저장소(GCE PD, AWS EBS, NFS etc)를 참조할 수 있음
- 쿠버네티스는 자동으로 그것을 워크로드에 사용가능하도록 만들 것임



원격 저장소

- pod 정의에서 직접 원격 저장소(GCE PD, AWS EBS, NFS etc)를 참조할 수 있음
- 쿠버네티스는 자동으로 그것을 워크로드에 사용가능하도록 만들 것임



문제

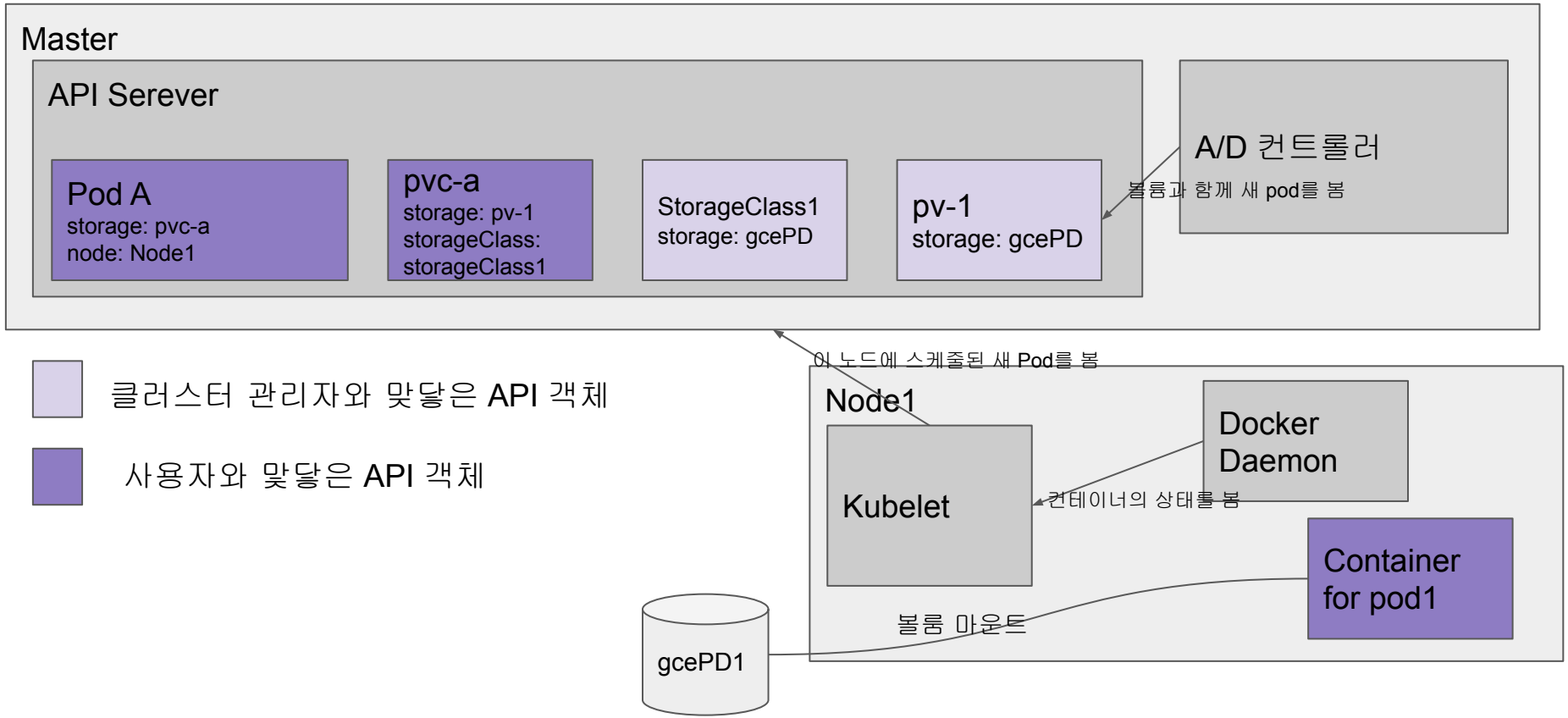
만약 당신이 (Pod에서) 볼륨을 직접 참조하고 있다면 좋지 않은 시간을 보내게 될 것이다

PVC/PC

Persistent Volume 과 Persistent Volume Claim 추상화

저장소 구현과 저장소 소비(consumption)를 분리시키는 것

PVC/PC


Master
API Serever
Pod A
storage: pvc-a
node: Node1
pvc-a
storage: pv-1
storageClass:
storageClass1
StorageClass1
storage: gcePD
pv-1
storage: gcePD
A/D 컨트롤러
볼륨과 함께 새 pod를 봄
이 노드에 스케줄된 새 Pod를 봄
Node1
Kubelet
Docker
Daemon
컨테이너의 상태를 봄
Container
for pod1
볼륨 마운트
gcePD1
클러스터 관리자와 맞닿은 API 객체
사용자와 맞닿은 API 객체

왜 워크로드 이식성이 중요한가

분산 시스템 애플리케이션 개발을 클러스터 구현과 분리시키는 것

쿠버네티스를 OS 와 같은 참된 추상화 계층으로 만듦

Q. 왜 선언형 API

- Self healing, able to rollback, extensible

Q. 왜 투명한 API

- immutable(불변함)
- 설정을 코드화한 yaml 파일(템플릿)만 있으면 반복적으로 동일하게 앱을 실행시킬 수 있음

Q. 왜 사용자가 있는 곳에서 만나야 하는가

- 배포 시스템으로써 쿠버네티스를 채택하는 것을 증가시키기 위해

Q. 왜 워크로드의 이식성이 중요

CLOUD AGNOSTIC(특정 클라우드 플랫폼에 종속되지 않음)

관심사의 분리 => 테스트, 업그레이드, 유지하기 좋음