7강 정리본 쿠버네티스

분류	
세부분류	
상위 항목	Cloud Native Spring In Action

목차

1. 도커와 쿠버네티스 비교

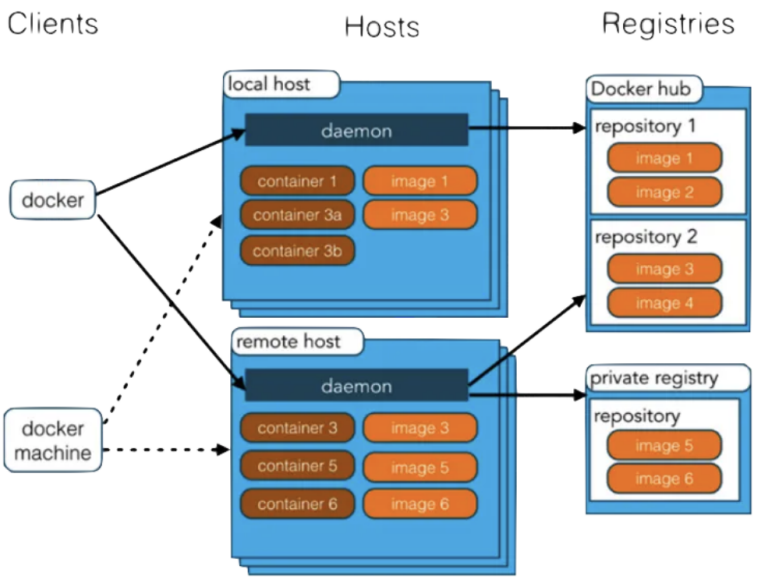
	도커	쿠버네티스
컨트롤러	도커 데몬	컨트롤 플레인
관리 서버 가능 대수	1대	여러대
리소스 관리	도커 호스트	쿠버네티스 클러스터

Docker CLI 및 Docker Compose 는 Docker Demon과 상호작용합니다.

또한 Docker Compose는 여러 도커 컨테이너를 쉽게 관리할 수 있습니다.

'docker compose up -d' 이런 명령어가 대표적입니다.

다만, 도커 호스트를 통해 도커 시스템 내에서 관리하기 때문에 컨테이너를 확장하는 것은 불가능합니다.

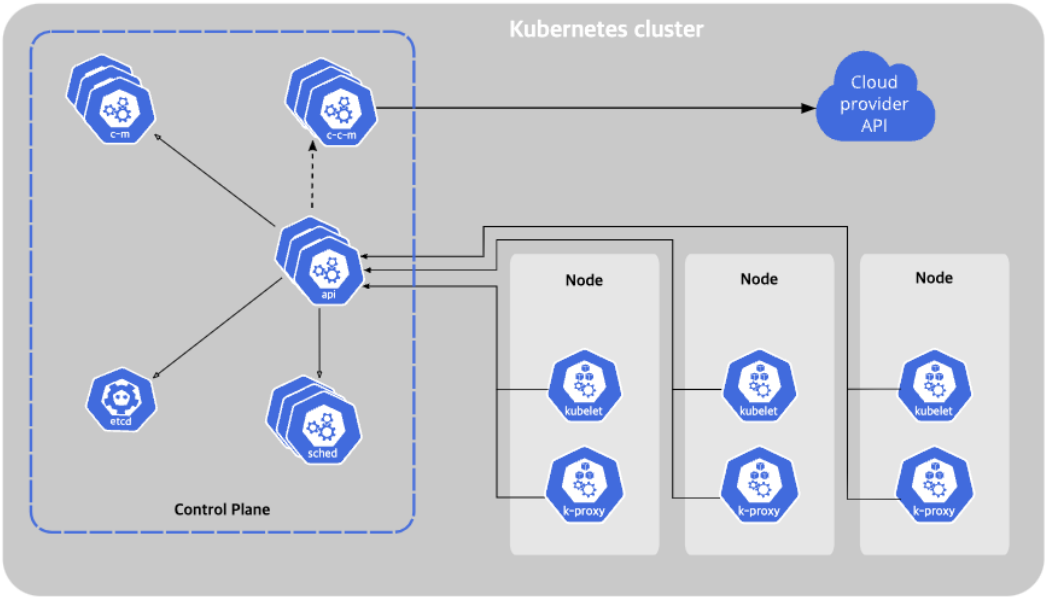


도커의 컨트롤 방법

더보기

반면에, 쿠버네티스는 쿠버네티스 클라이언트 API를 통해 쿠버네티스 컨트롤 플레인과 상호 작용을 합니다.

컨트롤 플레인의 경우 쿠버네티스 클러스터에 객체를 만들고 관리하기 때문에 한번에 여러 대와 연동하여 개발이 가능합니다.

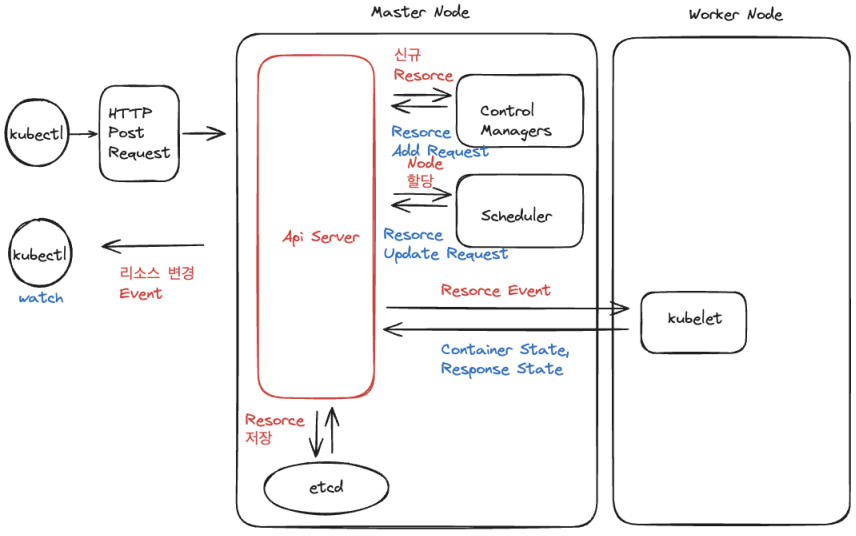


https://kubernetes.io/ko/docs/concepts/overview/components/

내부에서 동작하는 원리를 자세히 알아보면 kube-apiserver를 통해 주요 구현이 시작된다. 그리고 kube-apiserver는

수평적으로 확장되도록 디자인되어 있습니다. 워커 노드, 스케줄러, 컨트롤러 매니저 모두 Kube-apiserver를 바라보도록

설계되었습니다.



빨간 색의 요청들을 Api Server가 진행하고, 파란 색의 Response를 받아 상태를 관리하며 Worker Node에 구현, etcd에 Resorce를 저장해 관리함으로써, 지속가능한 개발이 가능하게 만들어줍니다.

더보기

2. 쿠버네티스 구성요소

쿠버네티스는 크게 클러스터, 컨트롤 플레인, 작업자 노드, 파드로 구성됩니다.

클러스터는 컨트롤 플레인과 한 개 이상의 작업자 노드로 구성된 것을 의미합니다.

컨트롤 플레인은 파드의 라이프 스타일을 정의하고 배포하기 위한 API 및 인터페이스를 제공합니다.

작업자 노드는 컨테이너가 실행하고 네트워크에 연결할 수 있게 기능을 제공하는 머신을 의미합니다.

파드는 애플리케이션 컨테이너를 감싸는 가장 작은 배포 단위를 의미합니다.

쉽게 설명하면

Pod는 하나의 애플리케이션,

Node는 그런 파드들이 네트워크에 연결하는 머신,

컨트롤 플레인은 정의되어 있는 상태로 유지하는 컨틀롤 타워,

클러스터는 하나의 작업 환경이라고 보면 됩니다.

3. 쿠버네티스 객체

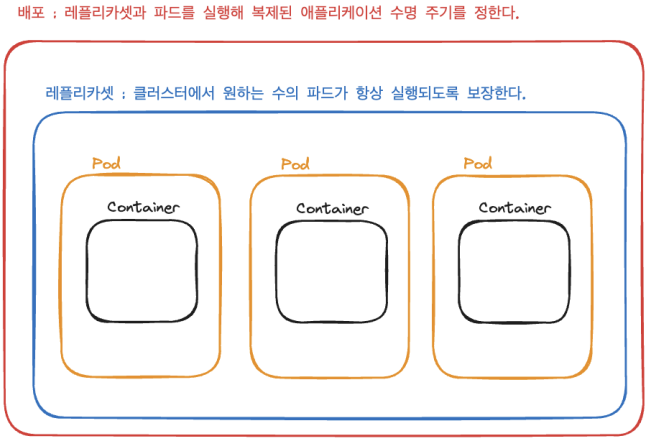
도커에서는 도커 CLI와 Compose를 통해 컨테이너 관리를 했다면, 쿠버네티스는 Pod를 관리합니다.

보통 Pod 하나에는 컨테이너 하나를 실행하지만, 로깅, 모니터링, 보안과 같은 추가 기능을

수행하는 헬퍼 컨테이너를 선택적으로 실행합니다.

많은 파일을 한번에 관리하고 확장, 수축하기 위해 추상화를 만드는데 이를 **배포 객체**라고 합니다.

코드를 수정해 업데이트를 실행하며, 롤백하거나 업그레이드를 진행합니다.



Ansible과 Puppet과 같은 명령형 도구와 달리 쿠버네티스는 그 상태로 유지하는데 의의가 있습니다.

4. 스프링부트 애플리케이션을 위한 배포 객체

코드로 manifest 파일을 만들어 객체(선언적 설정)를 기술하기 위해 종류에 대해 알아보겠습니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
    name: catalog-service
    labels:
```

```
        app: catalog-service
spec:
  replicas: 1
  selector: # 확장할 피드를 선택하기 위해 사용할 레이블을 정의
    matchLabels:
      app: catalog-service
  template: # 파드 생성을 위한 템플릿
    metadata:
      labels: # 파드 객체에 추가되는 레이블 레이블은 셀렉터로 사용하는 것과 일치
        app: catalog-service
    spec:
      containers:
        - name: catalog-service # 파드 이름
          image: catalog-service # 파드 생성에 사용할 이미지 이름(없으면 latest 사용)
          imagePullPolicy: IfNotPresent # 이미지가 로컬에 없을 때 컨테이너 저장소에서 받을것을 지시
          lifecycle:
            preStop:
              exec:
                command: [ "sh", "-c", "sleep 5" ]
          ports:
            - containerPort: 9001
          env:
            - name: BPL_JVM_THREAD_COUNT # 메모리 계산을 위한 스레드 수를 설정하기 위한 패키토 빌드팩 환경 변수?
              value: "50"
            - name: SPRING_CLOUD_CONFIG_URI
              value: http://config-service
            - name: SPRING_DATASOURCE_URL # postgre pod 가리키는 url spring.datasource.url 속성 값
              value: jdbc:postgresql://polar-postgres/polardb_catalog
            - name: SPRING_PROFILES_ACTIVE # testdata 스프링 프로파일 활성화
              value: testdata
```

apiVersion

특정 객체를 표현하기 위한 버전을 지정합니다.

더보기

kind

pod, replicaset, deployment, service를 작성할 수 있다.

metadata

작성할 객체의 세부 정보 제공이 가능하다.

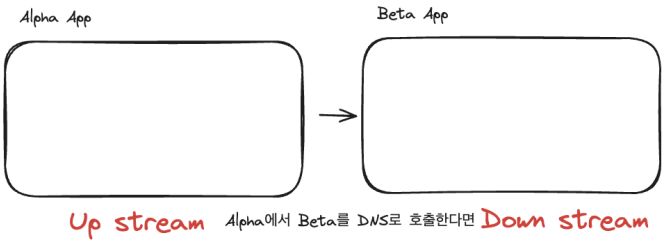
key, value 형태로 구성되어있다.

spec

객체 유형에 따라 원하는 설정을 선언하는데 사용한다.

5. 서비스 검색 및 부하 분산

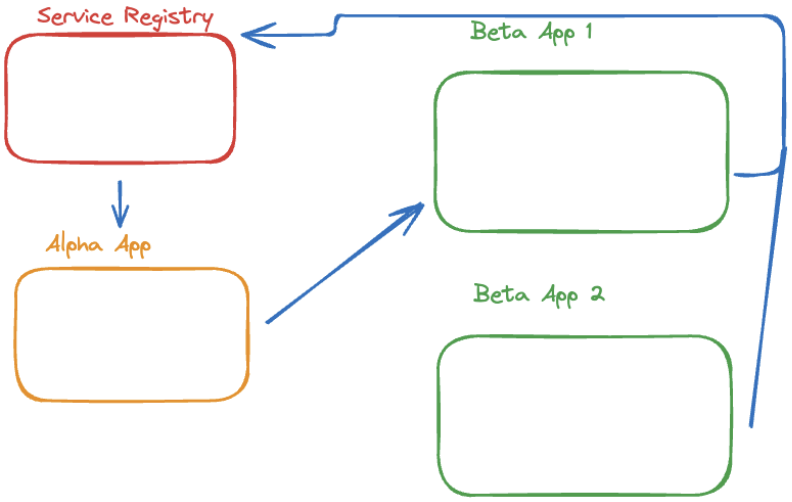
Pod 하나는 다른 Pod와 네트워크를 구성하려면 노출되어야 합니다.



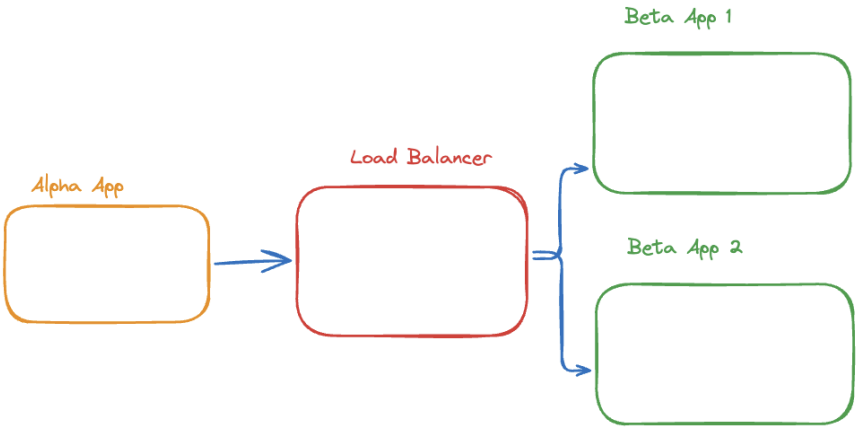
api를 보내는 것을 UpStream, 받는 것을 DownSream이라고 합니다.

우선, 통상 많은 인스턴스를 연결하는 방법을 알아보겠습니다.

1. 넷플릭스가 개발한 '유레카'
2. spring cloud consul
3. spring cloud zookeeper discovery
4. spring cloud alibaba nacos



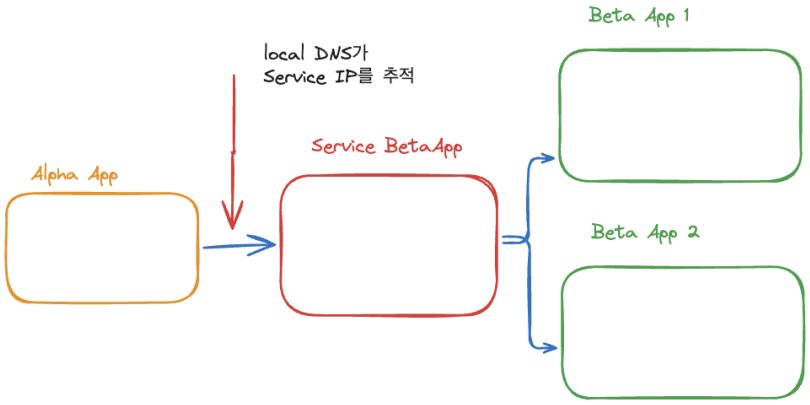
Service Registry에서 Alpha App으로 연결하고, Alpha App에서 Beta App으로 분산해 주는 형태입니다.



다른 방법으로는 플랫폼에 의존하는 방식이 있습니다.

들어오는 요청을 로드벨런서를 통해 부하 분산을 담당하고, 개발 책임의 많은 부분을 배포 플랫폼에서 담당함으로써

개발자는 비즈니스 로직에 집중하는 형태입니다.



쿠버네티스에서는 직접 파드를 호출하지 않고, 서비스 이름을 사용합니다.

이때, 서비스 이름의 IP를 찾으면 쿠버네티스는 큐브 프록시를 사용해 서비스 객체에 대한 연결을 가로채 서비스 대상 파드 중 하나로 전달합니다.

6. 노출 서비스 Cluster IP

노출 서비스로는 Load Balancer, Node Port, Cluster IP가 있습니다.

클러스터 IP는 파드의 집합을 클러스터에 노출하는 것으로, 파드가 서로 통신할 수 있게 만듭니다.

주로 내부 IP만을 이용하는 경우에 사용되는 기능입니다.

```
apiVersion: v1
kind: Service
metadata:
    name: catalog-service # 유효한 DNS 이름일것
    labels:
        app: catalog-service # 서비스에 추가될 레이블
spec:
    type: ClusterIP
selector:
    app: catalog-service # 대상으로 삼고 노출해야하는 파드를 찾는데 사용할 레이블
ports:
- protocol: TCP # 사용할 네트워크 프로토콜
    port: 80 # 서비스가 노출할 포트
    targetPort:  9001 # 서비스 대상이 되는 파드가 노출할 포트
```

selector : 서비스를 통해 노출할 대상이 되는 파드를 찾을 때 사용하는 레이블

protocol : 서비스가 사용하는 네트워크 프로토콜

port : 서비스가 듣는 포트

targetPort : 서비스가 파드로 요청을 할때 사용하는 포트

7. 우아한 종료

```
server:
    port: 9001
    shutdown: graceful
    tomcat:
        connection-timeout: 2s
        keep-alive-timeout: 15s
        threads:
            max: 50
            min-spare: 5

spring:
    application:
        name: catalog-service
    lifecycle:
        timeout-per-shutdown-phase: 15s # 15s로 변경
        ## 변경요소
        preStop:
            exec:
                command: ["sh", "-c", "sleep 5"]
```

쿠버네티스에서는 소스코드를 수정하면, 미니큐브에 이미지를 새로 만들어 로드합니다.

쿠버네티스의 sigterm 신호를 전송하면 이 신호를 가로채 우아한 종료를 시작합니다.

이때, sigKill 신호를 주면서, 신호를 보내는 곳에 요청 전달 중지 명령을 줄수 있는 시간을 확보해야 요청 실패가 발생하지 않는다.

이를 위해 lifecycle에 preStop을 5s 줘서 서버가 중지된 것을 유저들이 알지 못하게 만든다.

8. 틸트를 사용한 로컬 쿠버네티스 개발

수동으로 컨테이너를 재빌드하고, kubctl cli를 통한 관리는 파드를 업데이트하는 번거러움이 있습니다.

틸트를 사용하면 중앙 집중 방식으로 여러 애플리케이션 및 서비스의 배포를 조정하는 데 유용합니다.

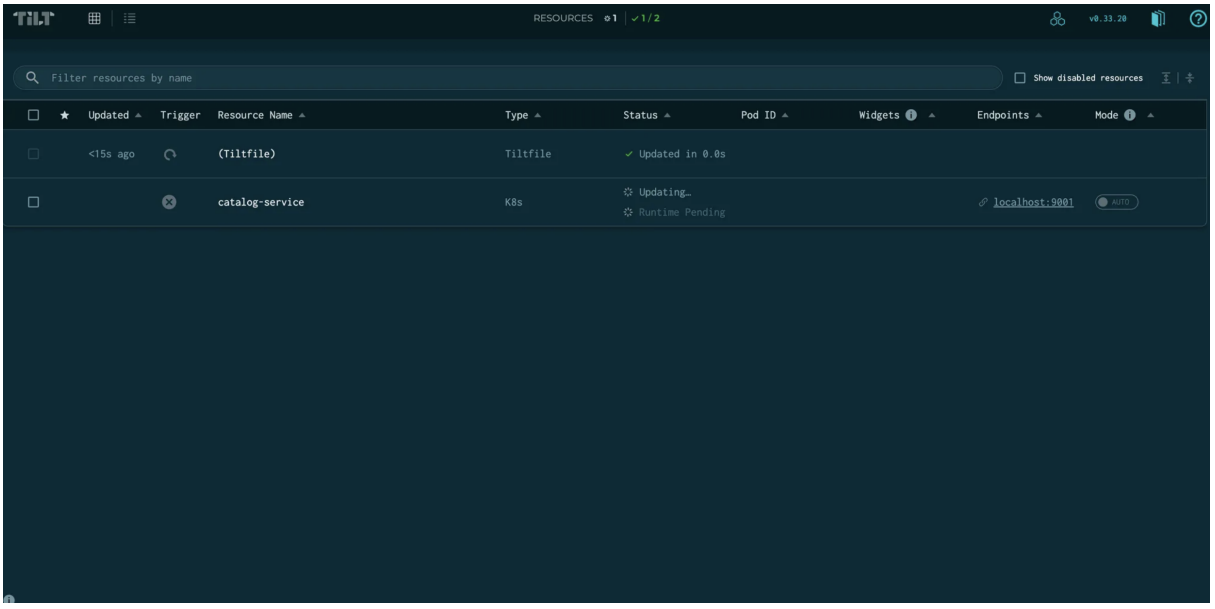
참고 : 틸트 파일이라는 설정파일로 설정하며, 스타라크 라는 언어로 개발됨

```
# Build
custom_build(
    # Name of the container image
    ref = 'catalog-service',
    # Command to build the container image
    # On Windows, replace $EXPECTED_REF with %EXPECTED_REF%
    command = './gradlew bootBuildImage --imageName $EXPECT
ED_REF',
    # Files to watch that trigger a new build
    deps = ['build.gradle', 'src']
)

# Deploy
k8s_yaml(['k8s/deployment.yml', 'k8s/service.yml'])

# Manage
k8s_resource('catalog-service', port_forwards=['9001'])
```

tilt up →모두 설치된다.



!!! 대박..

게다가 틸트를 사용하면 애플리케이션이 소스 코드와 동기화가 된다. 애플리케이션을 변경할 때마다 틸트는 업데이트 작업을 실행하려 새 컨테이너 이미지를 만들고 배포한다.

note : 코드를 변경할때 모든 이미지를 바꾸는 것은 효율적이지 못하다. 틸트는 변경된 파일만 동기화해 현재 이미지에 업로드하도록 만들 수 있다. spring boot devtools와 패키토 빌드팩이 사용하는 기능이다.

tilt down → 모두 내려간다.