

쿠버네티스

Container Orchestration

쿠버네티스는 컨테이너를 쉽고 빠르게 배포/확장하고 관리를 자동화해주는 오픈소스 플랫폼이다.

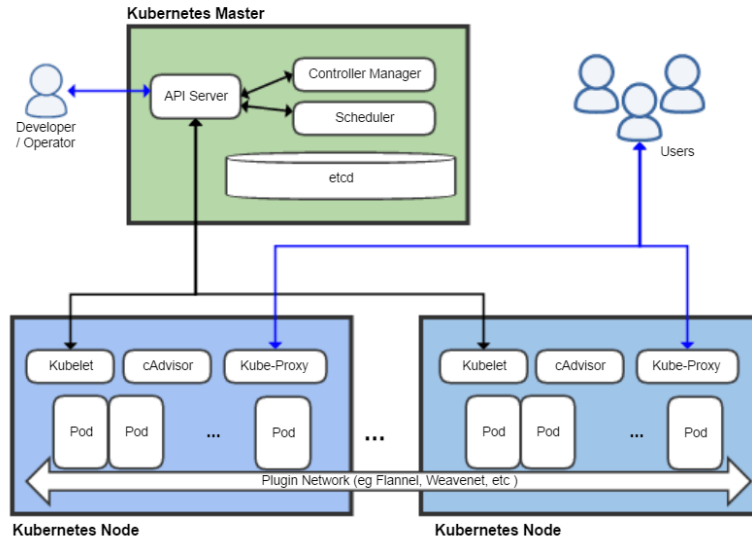
- 배경

도커 기반의 컨테이너 환경으로 서버 관리가 쉬워졌지만, 컨테이너가 많아짐에 따라 한번에 컨테이너를 조작하기가 쉽지 않았다.

- 아키텍처

[현재 상태를 원하는 상태로 유지하기 위해서 노력함]

1. Observe (상태체크) : current state == desired state
2. Diff (차이점 발견) : current state \neq desired state
3. Act (조치) : current state \Rightarrow desired state



• 매커니즘

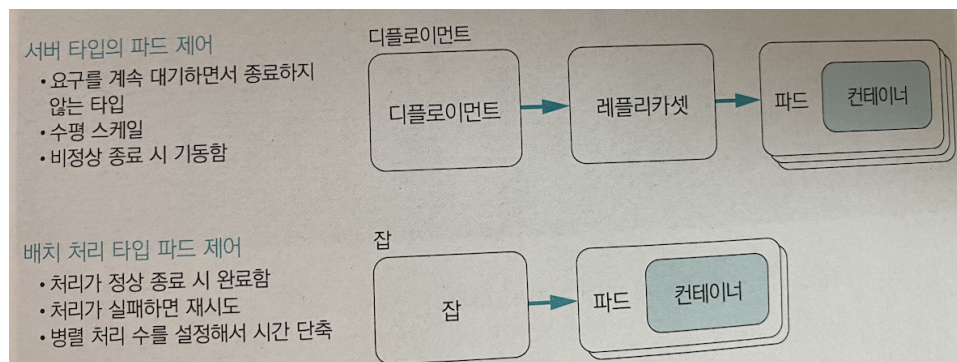
- API Server 에 Replica set 생성 요청
- controller 는 Replica set 생성 요청을 감시하다가 API server 에 pod 생성하여 API Server에 알리고 API Server는 etcd에 저장
- scheduler 는 pod 생성 요청을 감시하다가 worker 노드에 pod을 할당하고 이를 API server에 알림
- kubelet 자신의 node 에 할당되었으나 아직 생성되지 않은 pod 이 있는지 주기적으로 체크하다가 pod 을 생성하고 pod 상태를 API Server 에 주기적으로 알림

API Object

kubectl (클라이언트) — manifest → API Server (object 생성/변경/제거)

- pod
 - 컨테이너를 실행하기 위한 object
 - N개 ($N \geq 1$) 의 container 를 포함
 - 싱글 컨테이너 파드

- 멀티 컨테이너 파드
- 가장 작은 배포 단위
- controller
 - pod의 실행을 제어하고 관리하는 object
 - 끊임 없이 상태를 체크하고 원하는 상태 유지
 - 복잡도를 낮추기 위해서 하나의 프로세스로 실행
 - 논리적으로 다양한 컨트롤러가 존재하므로 워크로드에 맞게 선택
 - deployment : 지속적으로 서비스를 제공해야 하는 워크로드에 적합 (ex. API 서버)
 - job controller : 배치 처리와 같은 워크로드에 적합



기본 명령어

- 리소스 목록보기 : `kubectl get [TYPE]`
- 리소스 상세 상태보기 : `kubectl describe [TYPE]/[NAME]` 또는 `[TYPE] [NAME]`
- 리소스 제거 : `kubectl delete [TYPE]/[NAME]` 또는 `[TYPE] [NAME]`
- 컨테이너 로그 조회 : `kubectl logs [POD_NAME]`

- 컨테이너 명령어 전달 : `kubectl exec [-it] [POD_NAME] -- [COMMAND]`
- 설정 관리 : `kubectl config`

manifest 작성

- 쿠버네티스 object 를 생성하기 위한 메타 정보를 기술한 파일로 주로 YAML 로 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:latest
```

- 주요 항목
 - apiVersion
 - kind : pod, deployment, service, ingress ...
 - metadata
 - spec
 - status (read-only)
- 적용 방법
 - 생성 : `kubectl apply -f 파일명`
 - 삭제 : `kubectl delete -f 파일명`

Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: webap1
spec:
  containers:
  - name: webap1
    image: maho/webap1:0.1      # (1)핸드러를 구현한 애플리케이션
    livenessProbe:              # (2)애플리케이션이 살아있는지 확인
      httpGet:
        path: /healthz         # 확인 경로
        port: 3000
      initialDelaySeconds: 3    # 검사 개시 대기 시간
      periodSeconds: 5         # 검사 간격
    readinessProbe:            # (3) 애플리케이션이 준비되었는지 확인
      httpGet:
        path: /ready           # 확인 경로
        port: 3000
      initialDelaySeconds: 15
      periodSeconds: 6
```

- 파드 목록 : `kubectl get pod`
- 특정 파드 보기 : `kubectl get pod [이름] -o wide`
- 헬스 체크 기능 (kubelet이 수행)
 - liveness probe : 컨테이너의 애플리케이션이 정상적으로 실행 중인지 검사하고, 검사 실패하면 컨테이너를 강제 종료하고 재시작
 - readiness probe : 컨테이너의 애플리케이션이 요청을 받을 준비가 되었는지 검사하고, 검사 실패하면 요청 트래픽 전송을 중지
 - probe 대응 핸들러 : exec, tcpSocket, HttpGet
- 초기화 전용 컨테이너
 - `initContainers`
 - 요청을 처리하는 컨테이너와 별개로, 초기화만을 담당
 - ex) 스토리지 안에 새 디렉터리 만들고, 소유자를 변경하고 데이터를 저장하는 역할의 컨테이너 만들
- 사이드카 패턴

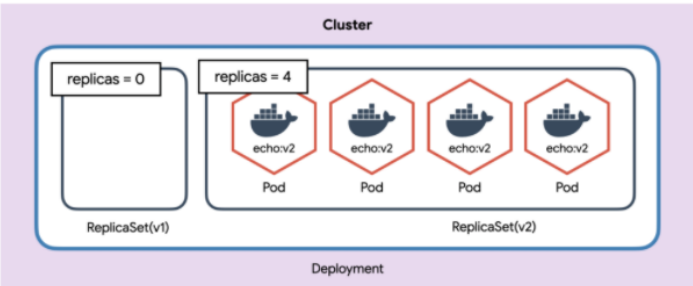
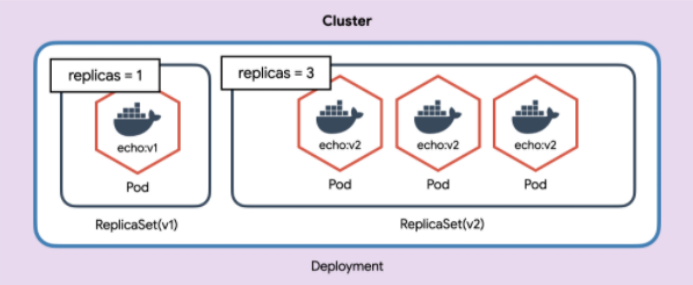
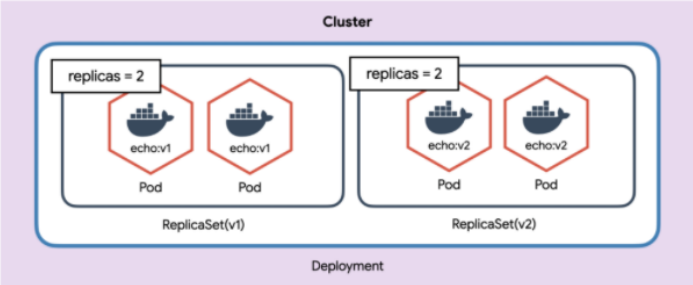
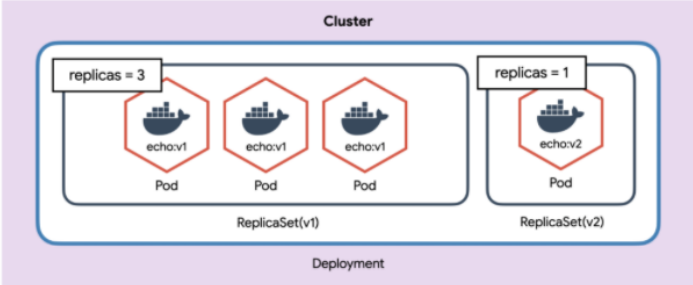
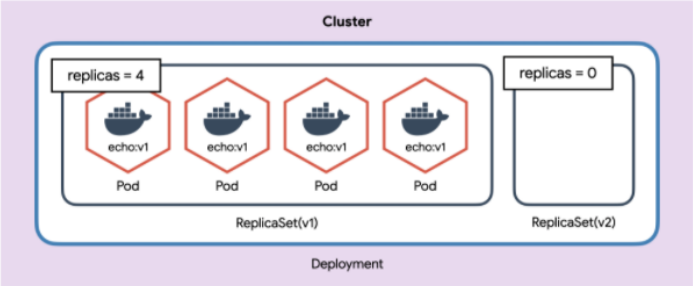
- 하나의 pod 안에 여러 개의 컨테이너를 담아서 동시에 실행시키는 패턴
- 재사용성, 생산성이 높아짐
- 컨테이너 종료 요청 시그널(SIGTERM)에 대한 종료 처리를 수행하도록 구현해야 함
⇒ 그렇지 않으면, 종료 처리 대기 시간만큼 기다린 후 SIGKILL 로 강제 종료됨

Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web #디플로이먼트와 파드를 묶는 레이블
  template:
    metadata:
      labels:
        app: web #디플로이먼트와 파드를 묶는 레이블
    spec:
      containers:
        - name: nginx
          image: nginx:1.16
```

- 서버 타입의 워크로드에 적합한 컨트롤러
- 요청한 개수만큼 파드를 기동하여 장애 등의 이유로 파드의 개수가 줄어들면 새롭게 파드를 만들어 기동
- 디플로이먼트와 파드 템플릿은 레이블에 의해 연관됨
- 기능
 1. 스케일 : 파드의 개수를 늘이거나 줄일 수 있음
 - `kubectl scale`
 2. 롤아웃 & 롤백 : 서비스를 유지하면서 파드를 교체
 - `kubectl rollout`

- `kubectl rollout undo`




- 단, 컨테이너는 stateless 하기 때문에 파드가 삭제되면서 데이터를 잃어버리므로 세션 정보 등은 외부 캐시에 보존해야 함
3. 자동복구 : 노드 수준의 장애가 발생했을 때 파드를 복구
- 파드는 컨테이너에 문제가 생긴 경우, 컨테이너를 재시작 (컨테이너 수준의 장애 대응)
- deployment + persistent volume + service controller ⇒ HA 구성 가능
 - 노드에 장애가 나면 일시적인 장애인지 기다렸다가 다른 노드로 옮겨가 파드를 기동

Reference

쿠버네티스 시작하기 - Kubernetes란 무엇인가?

쿠버네티스는 컨테이너를 쉽고 빠르게 배포/확장하고 관리를 자동화해주는 오픈소스 플랫폼입니다. 1주일째 수십억 개의 컨테이너를 생성하는 구글이 내부 배포시스템으로 사용하던 borg를 기반으로 2014년 프로젝트를

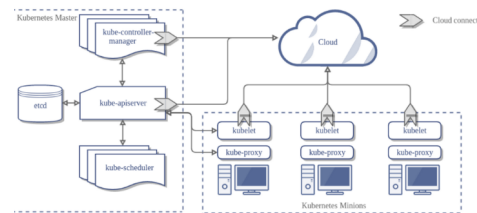
 <https://subicura.com/2019/05/19/kubernetes-basic-1.html>



쿠버네티스 #4 - 아키텍처

쿠버네티스에 대한 개념 이해가 끝났으면, 이제 쿠버네티스가 실제로 어떤 구조로 구현이 되어 있는지 아키텍처를 살펴보고자 하자. 아키텍처를 이용하면 동작 원리를 이해할 수 있기 때문에, 쿠버네티스의 사용법을 이해하는

 <https://bcho.tistory.com/1258>



Container, Docker, Kubernetes의 개념

사용자 jaehyunonline 2019. 5. 5. 19:23 Image Container Docker Hub / Registry Docker Engine Docker Container는 하나의 서비스를 모두 파편화하는 Microservice를 구성할 때 많은 이점이 있다. (하나의 마

 <https://iamnerd.tistory.com/4>

