An Introduction to Kubernetes

쿠버네티스 살펴보기

Kim Chungsub

@subicura

subicura@subicura.com

purpleworks

what is kubernetes?

쿠버네티스란 무엇일까요?

쿠버네티스의 인기 > 특징 > 아키텍처 > 구성 오브젝트 > 단점 > 개발하기

일단 얼마나 인기있는 플랫폼인지 알아보고 학습하는 방법까지 알아봅시다

2013년 3월 **도커**가 등장했습니다

컨테이너 기반의 도커는 서버 관리 방식을 **완전히** 바꾸고 있습니다

Docker - Build, Ship, and Run Any App, Anywhere

docker.com의 **타이틀** 입니다

Build

도커 이미지를 만들고

Ship

레지스트리에 저장하고

Run

도커 컨테이너를 실행합니다

Deployment?

그럼, 배포는??

```
$ DOCKER_HOST=production.host.ip docker run -d -p 80:80 awesome-app
```

<production.host.ip>에 도커 명령어를 실행합니다

매번 `docker` 명령어를 입력하는 것을 자동화 해볼까?

newrelic/centurion

A deployment tool for Docker

NewRelic에서 만든 ruby기반의 도커 컨테이너 배포툴입니다

```ruby
namespace :environment do
  task :common do
    set :image, 'example.com/newrelic/radio-radio'
    host 'docker-server-1.example.com'
    host 'docker-server-2.example.com'
  end

  desc 'Staging environment'
  task :staging => :common do
    set_current_environment(:staging)
    env_vars YOUR_ENV: 'staging'
    env_vars MY_DB: 'radio-db.example.com'
    host_port 10234, container_port: 9292
    host_volume '/mnt/volume1', container_volume: '/mnt/volume2'
  end

  desc 'Production environment'
  task :production => :common do
    set_current_environment(:production)
    env_vars YOUR_ENV: 'production'
    env_vars MY_DB: 'radio-db-prod.example.com'
    host_port 22234, container_port: 9292
    command ['/bin/bash', '-c', '/path/to/server -e production']
  end
end
```

대략 설정은 이렇게 생겼습니다. 참..쉽죠?

서버 2-3대까지는 이러한 방식 ~~한땀한땀~~ 도 괜찮습니다

orchestration

복잡한 서버를 효과적으로 관리하기 위해 오케스트레이션 툴을 도입해 봅니다

Deis v1 the CoreOS and Docker PaaS

Rancher container management platform

Mesos Deployment Made Easy

HashiCorp

Nomad

Nomad a tool for managing a cluster of machines and running applications on them

Swarm a Docker-native clustering system

컨테이너 관리도구 춘추전국시대

저마다의 장점을 가진 다양한 도구들이 등장했습니다

But, De facto

하지만, 사실상의 표준

Kubernetes Production-Grade Container Orchestration

why kubernetes?

Google

갓-구글

1주일에 20억개의 컨테이너를 생성하는 google이
컨테이너 배포 시스템으로 사용하던 borg를 기반으로 만든 오픈소스

이미 대세는 kubernetes로

Kubernetes Native Platform

Deis Workflow

Rancher 2.0

OpenShift (by redhat)

Tectonic (by CoreOS)

Docker EnterpriseEdition

kubernetes를 기반으로한 플랫폼의 등장

deis는 기존 아키텍처를 버리고 kubernetes위에 Workflow라는 플랫폼 개발

HELM
Get Helm
Blog
Docs
Charts
The package manager for Kubernetes
Helm is the best way to find, share, and use software built for Kubernetes.

deis => helm 개발

Rancher도 힘들게 만든 cattle을 버리고(?) kubernetes 기반으로 =_=/

RANCHER LABS

See what else Rancher is up to

K3S

Docs Github

Star 7,077 Fork 399

Lightweight Kubernetes

Easy to install. A binary of less than 40 MB. Only 512 MB of RAM required to run.

Watch our recorded k3s demo on-demand, and get a copy of our slides here

Watch Demo

This shouldn't take long...

```
curl -sfL https://get.k3s.io | sh -
# Check for Ready node, takes maybe 30 seconds
k3s kubectl get node
```

For detailed installation, refer to the docs

k3s from rancher

docker

Why Docker? Products Solutions Customers Resources Sign In **Get Started**

Docker Kubernetes Service

The easiest way to securely run and manage Kubernetes.

도커도 ~~어쩔수없이~~ 쿠버네티스를 지원합니다..! (ㅂㄷㅂㄷ)

Cloud Support

AWS EKS / Amazon Elastic Container Service for Kubernetes

Google Cloud GKE / Google Kubernetes Engine

Azure Container Service

Container Platform

Knative Serverless

Istio Service Mesh

kubeflow Machine Learning

...

쏘리! 리눅스, 이제 주인공은 '쿠버네티스'다

Kubernetes is..

쿠버네티스란 뭘까요

Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

쿠버네티스는 컨테이너 응용 프로그램의 배포, 확장 및 관리를 자동화하는 오픈 소스 시스템입니다

The name Kubernetes originates from Greek, meaning helmsman or pilot.

쿠버네티스는 그리스어로 조타수 또는 조종사에서 따왔습니다

K8s is an abbreviation derived by replacing the 8 letters “ubernete” with “8”.

쿠버네티스는 줄여서 K8s라고도 합니다

Google is donating it to a newly formed foundation — the Cloud Native Computing Foundation (CNCF) that will be run by the Linux Foundation

Google에서 만들어서 Cloud Native Computing Foundation(CNCF) 재단에 기부했습니다

특징

기본 기능

쿠버네티스는 프로덕션 수준의 컨테이너 오케스트레이션으로 기본기능을 충실하게 제공합니다.

- **상태관리** 상태를 선언하고 선언한 상태를 유지 / 노드가 죽거나 컨테이너 응답이 없을 경우 자동으로 복구
- **스케줄링** 클러스터의 여러 노드 중 조건에 맞는 노드를 찾아 컨테이너를 배치
- **클러스터** 가상 네트워크를 통해 하나의 서버에 있는 것처럼 통신
- **서비스 디스커버리** 서로 다른 서비스를 쉽게 찾고 통신할 수 있음
- **리소스 모니터링** cAdvisor를 통한 리소스 모니터링
- **스케일링** 리소스에 따라 자동으로 서비스를 조정함
- **RollOut/RollBack** 배포/롤백 및 버전 관리

빠른 업데이트

쿠버네티스는 현재 가장 빠르게 발전하고 있는 오픈소스입니다.

- **v1.14** 2019/3/26
- **v1.13** 2018/12/4
- **v1.12** 2018/9/28
- **v1.11** 2018/6/28
- **v1.10** 2018/3/27
- **v1.9** 2017/11/9
- **v1.8** 2017/8/25
- **v1.7** 2017/6/14
- **v1.6** 2017/3/23

1.10 이후 많은 부분이 안정화 되고 정리됨

Eco system

CNCF Cloud Native Landscape
2019-05-03T19:55:38Z a3c7df8
Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io
Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
App Definition and Development
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Orchestration & Management
Cloud-Native Storage
Container Runtime
Cloud-Native Network
Runtime
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Provisioning
Public
Kubernetes Certified Service Provider
Kubernetes Training Partner
Cloud
Special
Platform
Observability and Analysis
Serverless
l.cncf.io
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path
CLOUD NATIVE COMPUTING FOUNDATION
CLOUD NATIVE Landscape

다양한 배포방식

DEPLOYMENT
CRONJOB
DAEMON SET
REPLICASET
STATEFUL SETS
JOB
REPLICATION CONTROLLER
POD

Ingress 설정

Ingress
example.com
subicura.com/blog
subicura.com/help
...
example.com
subicura.com/blog
subicura.com/help
Service
Service
Service
POD
POD
POD
POD
POD

Namespace & Label

RBAC

RoleBinding
Role
User subicura
User admin
User jenkins
User dev01
Pod
Service
Deployment
Ingress
PV
Secrets
...
list, get, create
watch
delete
patch
Subjects
API Resources
Operations
(Verbs)

다양한 지원

쿠버네티스는 다양한 환경을 지원합니다.

- **Cluster Networking** ACI, Cilium, Contiv, Contrail, Flannel, Google Compute Engine (GCE), Kube-router, L2 networks and linux bridging, Multus (a Multi Network plugin), NSX-T, Nuage Networks VCS (Virtualized Cloud Services), OpenVSwitch, OVN (Open Virtual Networking), Project Calico, Romana, Weave Net from Weaveworks, CNI-Genie from Huawei
- **Volume Type** awsElasticBlockStore, azureDisk, azureFile, cephfs, configMap, csi, downwardAPI, emptyDir, fc (fibre channel), flocker, gcePersistentDisk, gitRepo, glusterfs, hostPath, iscsi, local, nfs, persistentVolumeClaim, projected, portworxVolume, quobyte, rbd, scaleIO, secret, storageos, vsphereVolume
- **Ingress Service** ingress-nginx, ingress-gce

뭔가.. 많이 지원하며, 다양한 기능을 쉽게 커스터마이징 할 수 있어 여러 클라우드 서비스에 적절하게 적용할 수 있고 쿠버네티스 기반의 플랫폼이 나오기 쉬운 구조

기본 개념

Desired State

Kubernetes Object

[The Illustrated Children's Guide to Kubernetes]

8분동안 쿠버네티스 개념을 **쉽게** 소개합니다

Kubernetes Object - Pod

Kubernetes Object - ReplicaSet

Kubernetes Object - etc

- Service
- Volume
- Namespace
- ...

Object Spec - YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: example
spec:
    containers:
    - name: busybox
        image: busybox:1.25
```

Kubernetes 사용하기

원하는 상태(desired state)를 다양한 오브젝트(object)에 라벨Label을 붙여 정의(yaml)하고 API 서버에 전달

Architecture

Server - Agent

Master - Node

Master - Node

- Master - 마스터 서버는 다양한 모듈이 확장성을 고려하여 기능별로 쪼개져 있는 것이 특징. 관리자만 접속할 수 있도록 보안 설정을 해야 하고 마스터 서버가 죽으면 클러스터를 관리할 수 없기 때문에 보통 3대를 구성하여 안정성을 높임
- Node - 노드 서버는 마스터 서버와 통신하면서 필요한 Pod을 생성하고 네트워크와 볼륨을 설정
- kubectl - 명령 도구

Master

master
scheduler
kube-controller
cloud-controller
kubectl
helm
web ui
api
API server
etcd
node
node
node

Master

- **API Server** 운영자 및 내부 노드와 통신하기 위한 인터페이스 / HTTP(S) RestAPI로 노출되어 있고 모든 명령은 여기를 통함
- **Controller Manager** 다양한 컨트롤러 (복제/배포/상태/크론/..)를 관리하고 API Server와 통신하여 작업을 수행
- **Scheduler** 서비스를 리소스 상황에 맞게 적절한 노드에 배치하는 역할 / predicates와 priorities (LeastRequestedPriority, BalancedResourceAllocation, ServiceSpreadingPriority, EqualPriority) 설정
- **etcd** 가볍고 빠른 분산형 key-value 저장소 / 설정 및 상태를 저장

Node

Node

실제 서비스(컨테이너)가 실행되는 서버입니다. 마스터의 API Server와 통신하며 서비스를 생성하고 상태를 관리합니다.

- **kubelet** 서비스(컨테이너)를 실행/중지하고 상태를 체크하여 계속해서 살아있는 상태로 관리
- **Proxy** 네트워크 프록시와 로드 발란서 역할 (creates a iptable rule/...)
- **Docker(containerd 또는 rkt 또는 Hyper)** 도커 뿐만 아니라 다양한 컨테이너 엔진을 지원

ReplicaSet 오브젝트 생성 흐름

모든 명령은 API Server를 통해서..

단점

복잡한 개념

공부할게 너무 많습니다. 쿠버네티스를 구성하고 관리하려면 알아야 할 내용이 많기 때문에 복잡한 클러스터를 구성하는게 아니라면 배보다 배꼽이 더 큰 상황은 아닌지 고민해봐야 합니다.

복잡한 설치

설치가 너무 어렵습니다. 클라우드를 이용합시다.

한땀한땀 설치해보면서 제대로 쿠버네티스를 익히고 싶은분은 여기를 참고하세요!

무거운 환경

기본적으로 실행되는 프로그램이 많고 그에 따라 기본 리소스 사용이 많은 편입니다.

복잡한 설정파일

모든 설정이 마이크로서비스 스럽게 나뉘어져 있기 때문에 길고 복잡합니다. swarm 설정과 비교해봅시다.

docker swarm

```yaml
version: "3"
services:
  api:
    image: vfarcic/go-demo-2
    environment:
      - DB=db
    ports:
      - 8080
    deploy:
      replicas: 3
  db:
    image: mongo:3.3
```

Kubernetes Disadvantages

== kubernetes config file(뒤에 짤림..) 😱

Deployments do provide a few potentially important functional differences. But, before we dive into the functional comparison, we'll take a moment to explore differences in how we define objects.

An example Kubernetes Deployment and Service definition is as follows.

```yaml
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: go-demo-2-db
spec:
  selector:
    matchLabels:
      type: db
      service: go-demo-2
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        type: db
        service: go-demo-2
        vendor: MongoLabs
    spec:
      containers:
      - name: db
        image: mongo:3.3
        ports:
        - containerPort: 28017

---

apiVersion: v1
kind: Service
metadata:
  name: go-demo-2-db
spec:
  ports:
  - port: 27017
  selector:
    type: db
    service: go-demo-2

---

apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: go-demo-2-api
spec:
  replicas: 3
  selector:
    matchLabels:
      type: api
      service: go-demo-2
  template:
    metadata:
      labels:
        type: api
        service: go-demo-2
        language: go
    spec:
      containers:
      - name: api
        image: vfarcic/go-demo-2
        env:
        - name: DB
          value: go-demo-2-db
        readinessProbe:
          httpGet:
            path: /demo/hello
            port: 8080
          periodSeconds: 1
        livenessProbe:
          httpGet:
            path: /demo/hello
            port: 8080

---

apiVersion: v1
kind: Service
metadata:
  name: go-demo-2-api
spec:
  type: NodePort
  ports:
  - port: 8080
  selector:
```

Thanks 🙏