KOREAN STUDENT VERSION

Beginning Kubernetes

Byungwook Hyeon
&
Dongwan Kang

Begining Kubernetes

“Begining Kubernetes” is for kubernetes beginers who want to leran using korean, not a expert. We hope that this book will be a ESSENTIAL for kubernetes beginners.

Byungwook Hyeon & Dongwan Kang

About the Authors.

현병욱 Byungwook Hyeon

- 소속 : 한양대학교 ERICA 소프트웨어융합대학 소프트웨어학부
- 이력 :

2007. 03 ~ 2007. 11 : 홍콩 어학연수
2009. 03 ~ 2012. 02 : 창동고등학교 졸업
2014. 03 ~ 2019. 02 : 청주대학교 컴퓨터정보공학과 학사(중퇴)
2016. 02 ~ 2017. 10 : 705 특공연대 특공병, 병장 만기 전역
2019. 03 ~ 2021. 02 : 한양대학교 ERICA 소프트웨어학부 학사
2019. 03 ~ 2019. 06 : (주)대양엔바이오 컨설팅
2019. 09 ~ 2020. 06 : 클라우드 네이티브 컴퓨팅 플랫폼 개발 With KTds
2020. 08 ~ 2020. 12 : 최강학원 수학 강사
2020. 09 ~ 2020. 12 : AWS기반 호텔정보시스템(PMS) 개발, Agora
2020. 10~ 2020. 12 : 대한민국 증권시장에서 거래되는 주식에 대한 R기반 리스크 측도 및 평균 수익률 분석
2020. 11 ~ 2020. 12 : 콜옵션 소유자와 발행자 각각의 거래일에 대한 R기반 현금 흐름과 만기 수익 분석

- 자격 :

Microsoft - Azure Fundamental : AZ-900
Linux Foundation - Certified Kubernetes Administrator : CKA

About the Authors.

강동완 Dongwan Kang

- 소속 : 한양대학교 ERICA 소프트웨어융합대학 소프트웨어학부
- 이력 :

2007. 03 ~ 2010. 02 : 광성고등학교 졸업
2010. 03 ~ 2014. 02 : 유한대학교 컴퓨터소프트웨어공학과 전문학사
2011. 02 ~ 2012. 10 : 3사단 직할 전차대대, 병장 만기 전역
2013. 03 ~ 2017. 01 : 웹플러스 근무, 대리
2019. 03 ~ 2021. 02 : 한양대학교 ERICA 소프트웨어학부 학사
2019. 09 ~ 2020. 06 : 클라우드 네이티브 컴퓨팅 플랫폼 개발 With KTds
2019. 10 ~ 2020. 07 : GS25 안산신안점 근무
2020. 09 ~ 2020. 12 : AWS기반 호텔정보시스템(PMS) 개발, Agora

- 자격 :

Microsoft - Azure Fundamental : AZ-900
대한민국 한국산업인력공단 - 정보처리 산업기사

Table of Contents

시작하면서..

Chapter 1. 쿠버네티스 소개

Kubernetes란?

Why Kubernetes?

Kubernetes 구성 요소

Chapter 2. 쿠버네티스 컨셉

Node

POD

Cluster

ETCD

Chapter 3. 쿠버네티스 APIs

Chapter 4. 쿠버네티스 구성요소

Chapter 5. 보조 프로젝트 소개

시작하면서..

저자인 현병욱과 강동완은 클라우드 엔지니어로의 진출을 준비중인 한양대학교 ERICA 소프트웨어학부 학생입니다. 현병욱과 강동완은 KTds Architecture CoE팀과 클라우드 네이티브 컴퓨팅 플랫폼 개발을 주제로 프로젝트를 수행하여 종합 플랫폼인 Agora를 개발했습니다. 또한, 현병욱은 LInux Foundation과 CNCF사의 kubernetes 운용 능력 검증 시험인 'Certified Kubernetes Administrator : CKA'를 취득했습니다.

'Beginning Kubernetes'는 저자가 프로젝트 수행과 자격 취득 준비 과정에서 경험한 어려웠던 점들을 쉽게 설명하여 입문자들이 쉽게 kubernetes에 접근할 수 있도록 하기위해 집필하게 되었습니다. 이 과정을 통해 kubernetes 오픈 소스 프로젝트가 활성화되길 바랍니다.

저자 현병욱

저자 강동완

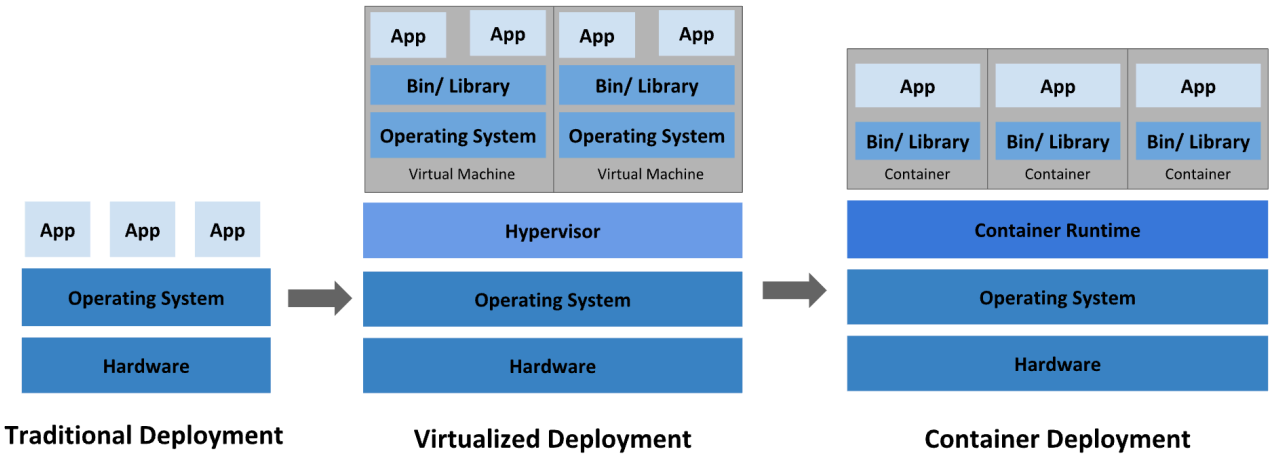
Chapter 1.

Kubernetes 소개

Kubernetes란?

Kubernetes는 컨테이너화된 애플리케이션의 자동 디플로이, 스케일링 등을 제공하는 오픈 소스 기반의 관리 시스템이다. 시스템의 초기 버전은 Google에 의해서 개발되어 2014년 중순에 공개되었다. 프로젝트 당시 kubernetes는 “프로젝트 세븐”으로 시작하였으나, 그리스어로 키잡이(helmsman)를 뜻하는 kubernetes로 2015년 7월 21일에 v1.0이 정식 출시되었다. 현재는 Linux Foundation에서 서비스를 주관하고 있다.

Why Kubernetes?



초창기 대부분의 애플리케이션은 기본적인 시스템의 물리 서버에서 서비스되었다. 그러나 여러개의 애플리케이션을 구동하고자 한다면 그에 상응하는 수의 물리 서버가 필요하다는 한계가 존재했는데, 이는 서비스 측면에 있어서 어플리케이션의 성능을 저하시킬 수 있다는 치명적인 단점으로 작용했다.

이를 해결하기 위해 VirtualBox, VMware와 같은 가상화 환경이 등장하게 된다. 가상환경은 기존의 단일 물리 서버의 CPU에서 다중으로 실행이 가능하도록하여 애플리케이션의 각 리소스들을 격리하여 성능 저하를 방지할 수 있다. 또한, 격리된 리소스는 서로간의 접근에 있어서 자율성을 제한하여 보안 측면까지 해결이 가능했다. 가상화의 등장은 기존에 요구된 많은 수의 물리 서버의 필요성을 감소시킴으로써 하드웨어 비용의 절감을 효과를 기대할 수 있다. 이는 동적 확장에 자율성을 보장해줄 수 있음은 물론이거니와, 기존 각 물리 서버에서 요구하는 독립적 운영체제의 도입에서 발생하는 비용을 감축킬 수 있기에 다수의 기업들이 가상환경을 도입하기 시작했다.

가상 환경의 도입으로 기존 물리 서버의 시대보다 효율적인 애플리케이션 운용이 가능해졌지만, 가상 환경에도 단점이 존재했다. 바로 리소스간의 접근 제한으로인한 개발 복잡성의 증가이다. 개발 단계에서 가상 환경내의 독립적인 리소스간는 정보를 주고받는 통신에 제약사항을 유발하여 개발 단계를 지연시키는 경우가 발생한 것이다. 이를 해결하기위해 등장하게 된 것이 바로 '컨테이너'기술이다. 컨테이너는 VM과 유사하지만 리소스간의 통신 제약사항을 완화하여 애플리케이션 간의 리소스 공유가 가능하도록 구상되었다. 또한, 컨테이너의 가장 큰 장점은 기존의 로우 레벨의 인프라와의 연결 고리가 끊어짐에따라 클라우드나 기타 다른 운영체제로의 이식이 가능해졌다는 점이다.

입문서의 메인 주제인 kubernetes는 컨테이너를 쉽게 관리할 수 있도록 고안된 플랫폼이다. Kubernetes는 분산 시스템에 대한 접근성 강화와 운영 단계를 단축시킬 수 있는 프레임 워크를 제공한다. 이를 통해 Kubernetes는 컨테이너에 대해 애플리케이션의 배포, 확장, 장애 조치, 모니터링, 스케쥴링 등을 신속하고 편리하게 처리할 수 있다. 결론적으로 Kubernetes는 컨테이너 시대를 총괄하는 플랫폼인 것이다.

Kubernetes 운용 능력 확보는 4차 산업혁명 흐름에 맞춰 변화중인 애플리케이션 개발 환경에 필수 역량으로 자리잡게 될 것이다.

Kubernetes 구성 요소

Kubernetes 운용 능력을 향상시키기 이전에 기본적으로 알아야 할 Kubernetes의 구성 요소는 다음과 같다.

1. 노드 (Node)

도커와 같은 패키징 프로그램을 통해 컨테이너화된 애플리케이션을 실행하는 단위이다.

2. 파드 (POD)

파드는 kubernetes에서 생성할 수 있는 가장 작은 단위의 오브젝트로, 애플리케이션의 작은 Instance 이다.

3. 클러스터 (Cluster)

클러스터는 컨테이너화된 애플리케이션을 실행하는 노드의 집합니다.

4. Kube-apiserver

Kube-apiserver는 kubernetes를 전반적으로 관리하는 메인 server이다. kuber-apiserver를 통해 클러스터 안의 모든 동작의 오케스트레이션을 관리할 수 있다.

5. Kubelet

Kubelet은 kube-apiserver로부터 커맨드를 받아 현재 컨테이너, 노드, 파드 등에 대한 정보를 받거나 노드에 컨테이너를 배포 혹은 삭제하는 기능을 수행한다.

6. Kube-proxy

Kube-proxy는 서비스와 클러스터 사이의 통신을 보조한다.

7. Kube-scheduler

Kube-scheduler는 현재 필요한 컨테이너를 가져오는 작업을 수행한다.

8. ETCD

ETCD는 reliable한 key-value를 간단하게 저장하여 보관할 수 있는 기능을 수행한다. 또한, 현재 클러스터에 존재하는 컨테이너, 파드, 노드 등에 대한 정보를 일괄적으로 백업하여 보관할 수 있다.

9. Replication-controller

Replication-controller는 노드를 새로운 클러스터에 가져오거나, 사용 불가 상태의 노드를 삭제할 수 있는 작업을 수행한다.

이 이외에도 kubernetes를 구성하고있는 수많은 요소들이 존재하지만, 다음 9가지는 kubernetes를 운용하는데 있어서 가장 기본적으로 숙지하고 있어야 할 구성 요소이다. 자세한 설명은 다음 장에서 진행하도록 한다.

Chapter 2.

Kubernetes 컨셉

Node

Kubernetes는 도커와 같은 패키징 프로그램을 통해 컨테이너화된 애플리케이션을 파드 내에 배치하여 애플리케이션을 실행하는 워크로드를 지원한다. 이처럼 노드는 클러스터 내의 파드에 배치되어 애플리케이션을 실행하게 되는데 이때의 노드 환경은 클러스터에 따라 가상 혹은 물리적 환경일 수 있다. 이는 사용자의 판단에 따라 kubernetes에서 설정할 수 있으며, 노드는 실행하는 단계에 필요한 라이프사이클, API, 인터페이스 등에 대한 정의가 포함된 컨트롤 플레인을 보유함으로써 파드 단계의 실행을 지원해준다. 또한, 각 노드별 OS를 업그레이드 시켜주는데 있어서 사용하는 방법인 ‘Rolling Update’를 진행하기 위해서도 노드의 역할은 중요하다. 왜냐하면 rolling update를 통해 어떠한 조치도 없이 단일 노드 단위의 OS를 업그레이드 시키게되면 컨테이너에 손상이 발생할 수 있다. 컨테이너의 손상은 애플리케이션 단위의 치명적 오류를 발생시킬 수 있다. 따라서, 컨테이너에 손상이 발생하지 않도록 노드에 ‘drain’을 적용하게되면 marked되어 scheduling과 deploy가 제한되고 자동 분산되어 컨테이너의 손상을 막고 애플리케이션의 downtime을 방지하여 서비스 공급에 제한이 발생하지 않도록 한다. 자세한 내용은 추후 실습 과정을 통해서 다시한번 설명하도록 한다.

POD

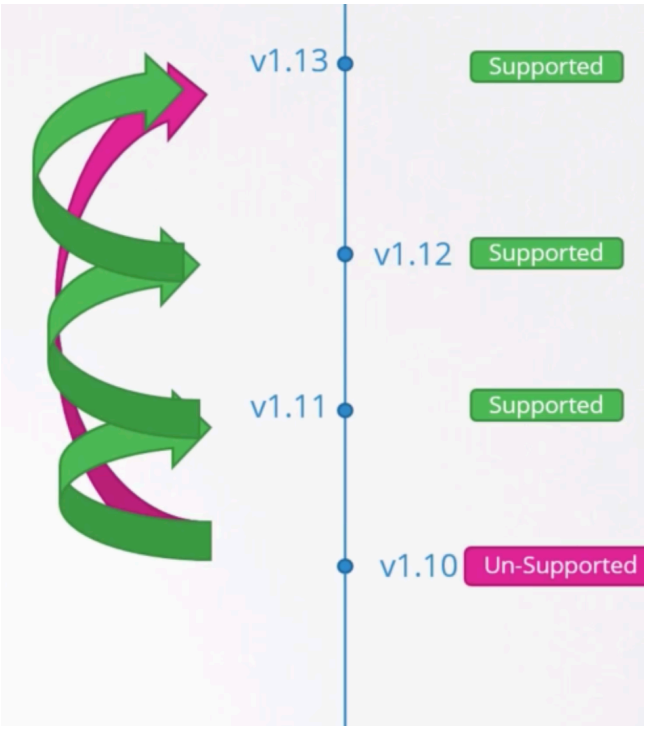
파드는 kubernetes에서 생성할 수 있는 가장 작은 단위의 오브젝트로, 애플리케이션의 작은 Instance이다. 즉, 노드 내에 존재하는 컨테이너로 이루어진 가장 작은 집합 단위가 파드이다. 파드 내에 존재하는 컨테이너는 패키징 과정을 통해 바로 노드에 디플로이되는 것이 아니라 kubernetes를 통해 파드로 압축되어서 노드에 디플로이되게 된다. 파드가 디플로이되면 해당 파드 내에 helper 컨테이너도 동시에 생성되게 되는데, helper 컨테이너는 파드 내의 컨테이너의 동작 수행을 지원하거나 동일 노드 내의 파드 간의 통신을 지원하기도 한다. 이때 생성된 helper 컨테이너는 파드의 메인 컨테이너가 삭제되면 helper 컨테이너도 함께 삭제된다. 추가적으로 일반적으로 싱글 파드는 컨테이너와 1:1 관계가 기본이기는 하나, 사용자의 요구에 따라 싱글 파드가 멀티플 컨테이너를 보유할 수도 있다. 또한, 동일

namespace에 위치한 파드의 컨테이너는 helper나 proxy의 도움 없이도 localhost를 통해 직접적으로 통신을 하거나 스토리지를 공유할 수도 있다.

※ POD의 생성은 kubernetes의 메인 API인 kube-apiserver가 노드에 접근하여 생성하게 되는데, 이와 관련해서는 kube-apiserver를 설명하면서 자세히 다루도록 한다.

Cluster

클러스터는 컨테이너화된 애플리케이션을 실행하는 노드의 집합으로, 단일 클러스터에 multiple server를 가질 수 있다. 클러스터에는 노드의 수, CPU, Memory, Network, Disk Utilization, POD에 대한 CPU의 사용량을 확인할 수 있는 정보들이 모두 들어가 있다. 이를 일반적으로 'metrics server'라 부르는데, 이 서버는 업무 수행 결과를 보여주는 계량적 분석 기능을 지원한다고 볼 수 있다. 이를 통해 현재 클러스터 내에서 동작중인 애플리케이션에 대한 모니터링을 수행할 수 있다. 뿐만 아니라 클러스터는 kubernetes 내의 API에 대한 업그레이드를 수행하는데 있어서도 큰 영향을 준다. 예를 들어 kubeadm을 통해서 deploy된 scheduler는 kubeadm이 클러스터를 upgrade하고 plan할 수 있도록 한다. 이와 반대로, GCP와 같은 cloud service providers에 deploy된 클러스터가 kubernetes 클러스터에 의해서 managed 되고 있다면 GCP kubernetes는 몇번의 클릭만으로도 업그레이드가 가능해진다. 이처럼 클러스터는 kubernetes 내에서 동작중인 주요 기능들과 직간접적으로 연결되어 있음으로 상황에 적합한 클러스터를 구성하는 것이 애플리케이션의 동작 환경을 최적화 시키는데 도움을 줄 수 있다.



✔ kube-apiserver

kubectl은 kube-apiserver에 접근한다.

api server가 노드에 접근하지 않고 POD를 생성하는 방법
1. ETCD server에 이미 생성되어진 POD를 업데이트한다.
2. 스케쥴러는 지속적으로 API server를 모니터링해서 새로운 POD가 노드에 생김을 인지한다.
3. API server는 2에서 받은 정보를 토대로 ETCD 클러스터를 업데이트한다.
4. 3에서의 업데이트 정보를 토대로 Node의 Kubelet에 정보를 전달한다.
5. kubelet은 API server로 정보를 업데이트해준다.
6. API Server는 ETCD 클러스터에 정보를 업데이트한다.
7. 클러스터의 모든 정보는 API server가 중심이 된다.

즉, Kube-api server의 역할
1. Authenticate User
2. Validate Request
3. Retrieve data
4. Update ETCD
5. Scheduler
6. Kubelet

※ Kube-api server는 ETCD Datastore와 직접적으로 상호작용하는 유일한 요소이다.
※ Kubelet, Kube-controller-manager, Scheduler는 kube-api server를 통해 클러스터 정보를 업데이트할 수 있다.

· 존재하는 클러스터의 kube-api server options 보는 방법
kubectl get pods -n kube-system

· kubeadm tool로 설정하면 kubeadm이 kube-api server를 pod로써 master node의 kubesystem name에 deploy한다.

· Pod definition file과 관련된 options를 보는 방법
cat /etc/kubernetes/manifests/kube-apiserver.yaml

· runnig process와 effective options의 list 보는 방법
ps -aux | grep kube-apiserver

✔Kube Controller manager

Node-controller는 kube-api server를 통해 정보를 받는다.

replication-controller는 레플리카 셋의 상태를 모니터링하고 필요한 수의 PODS를 셋에서 필요한 만큼 사용 가능하게 해준다. -> pods가 죽으면 다른 하나를 생성한다.

Kube Controller Manager에 Node-Controller, Replication-Controller, Deployments-Controller 등의 k8s에 존재하는 모든 Controller가 포함되어 있으므로, 따로 설치 할 필요 없이 Kube Controller Manager만 설치하면 일괄적으로 설치된다.

· Kube-Controller-Manager 설치 방법
wget https://storage.googleleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-controller-manager

※ 설치 후 중요한 값 :
—node-monitor-periods=5s
-node-monitor-grace-periods=40s
-pod-eviction-timeout=5m0s

· Kube-Controller-Manager server Options 보는 법
7. Kubeadm tool 쓰는 경우
kubeadm이 kube-controller-manager를 Master Node의 kube-system namespace에 Pod로 Deployments한다.

cat /etc/kubernetes/manifests/kube-controller-manager.yaml

✔ Kube Scheduler
Kube Scheduler는 오직 어떤 Pod가 어떤 Node로 갈지를 정한다.
단, Node의 Pod로 이동시키지는 않는다. → Kubelet 역할

※ Nodes가 Pods에 필요한 충분한 CPU와 Memory를 갖고 있지 않다면 처음 두개의 작은 Nodes가 filtered out된다.

· Kube-Scheduler 설치 방법
wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-scheduler

· Kube-scheduler server options 보는 방법
8. Kubeadm tool로 설정한 경우

kubeadm은 kube-scheduler를 Master Node의 Kube-system namespace에 Pod로 deploy한다.

cat /etc/kubernetes/manifests/kube-scheduler.yaml

✔ Kubelet
Kubelet은 배의 선장과 같다.
Kubelet은 컨테이너를 올릴지 말지를 결정하고 컨테이너의 상태를 체크한다.

· k8s의 Worker Node의 Kubelet은 k8s cluster에 Node를 등록한다.

· Container나 Node의 Pod를 불르라는 명령을 받게되면 Container run time engine인 Docker에 명령받은 image를 불러오라고 요청하고 불러와서 그 instance를 실행시킨다. 이후, Kubelet은 pods와 containers에 대한 지속적인 모니터링을 하고 그 결과를 kube-api server에 시간에 따라 정보를 전달한다.

· Kubelet 설치 방법
wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kubelet

cluster에 deploy하기위해 kubeadm tool을 통해서 kubelet을 설치하면 자동적으로 kubelet을 deploy하는 것이 아니다. 이것이 다른 설치 요소들과의 차이점이다.
worker nodes에 수동으로 설치해줘야한다.

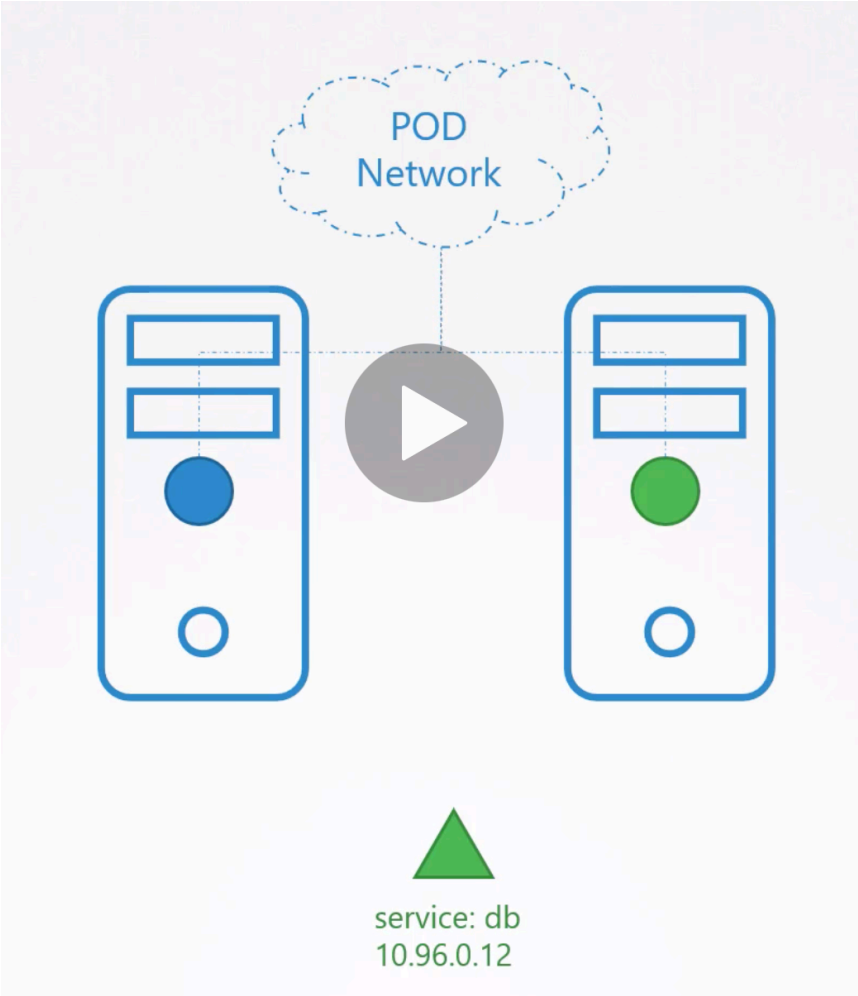
✔ Kube-proxy
k8s의 cluster는 모든 pods가 서로 다른 pods에 접근할 수 있도록 해준다.
이는 POD networking solution을 cluster에 deploy해줌으로써 가능하다.

※ 예제
상황 : Node1에 Web Application이 deploy되있고, Node2에 database application이 deploy 되어있다.

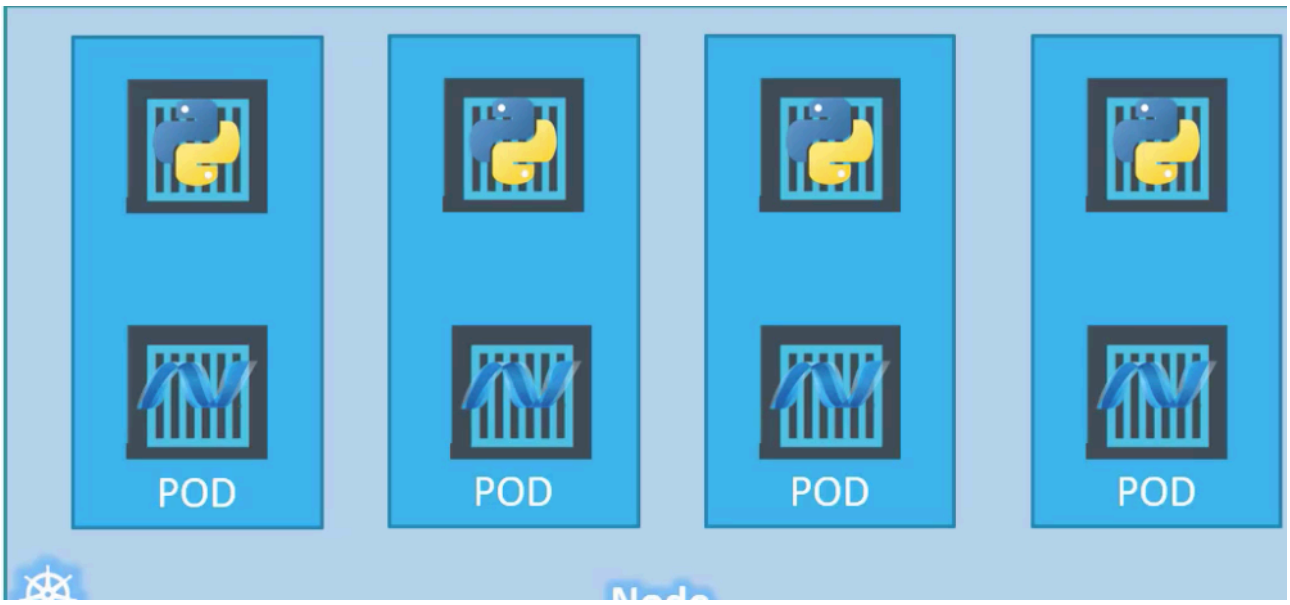
Web app은 database Pod의 IP를 통해 database에 접근할 수 있다.
하지만, database Pod의 IP가 항상 같은 상태로 남아있다고는 보장이 되지 않는다.

따라서, Web app이 database에 접근하고자 할 때의 최고의 방법은 service를 사용하는 것이다.

database application을 노출시키기 위해 cluster 밖에 Service: db를 생성한다.
이렇게 되면 web application은 service db를 통해 database에 접근할 수 있게 된다.



Service: db는 Pod가 service에 접근해서 IP나 name을 요청할때 마다 database에 할당된 IP 주소를 받아온다.



여기서 Service는 actual thing이 아닌 memory의 cabinet에 위치한 가상의 요소로 Pod networking에 접속할 수 없다.
따라서 service는 pod같은 container가 아니므로 interface가 없고, 능동적으로 process를 알 수 없다.

※ 그렇다면 Service는 database의 IP를 어떻게 받아오는가?
Kube-proxy를 통해 받아올 수 있다.
Kube-proxy는 k8s의 Cluster의 각각의 Node에서 동작하면서 항상 새로운 Service가 생성된게 있는지를 확인한다. 만약, 생성된게 있다면 적절한 IP Table Rules를 각각의 Node에 생성한다.

· Kube-proxy 설치
wget https://storage.googleapis.com/kubernetes-release/release/v1.13.0/bin/linux/amd64/kube-proxy

Kubeadm tool은 kube-proxy를 pods의 각각의 Node에 daemon set으로 deploy한다. 따라서 single Pod는 항상 cluster의 각각의 node에 배포된다.

✔ POD
POD는 application의 작은 instance이다.
POD는 k8s에서 생성할 수 있는 가장 가장 작은 단위의 object이다.

k8s는 worker node에 직접적으로 컨테이너를 deploy하지 않는다.

컨테이너는 POD라 알려진 k8s 오브젝트로 압축되서 들어간다.

새로운 컨테이너 instance를 같은 POD로 불러올 수 없다.

POD는 컨테이너와 1:1 관계이다.

싱글 POD는 멀티플 컨테이너를 가질 수 있다.

새로운 application 컨테이너가 POD에 생성되면 helper 컨테이너도 동시에 생성되고, 컨테이너가 삭제되면 helper 컨테이너도 삭제된다.

두개의 컨테이너는 같은 network namespace에 위치한다면 서로 직접적으로 localhost를 통해서 통신할 수 있다. 또한, storage도 공유할 수 있다.

· POD를 Deploy하는 방법
1. kubectl run nginx -> POD를 생성함으로써 docker 컨테이너를 deploy한다.
이 과정에서 POD를 자동으로 생성하고 nginx docker images의 instance를 deploy한다.

2. 1을 실행하게되면 다양한 application이 저장되어있는 docker hub(Public repository)로부터 최신 버전의 images를 가져온다.

3. kubectl get pods → 사용 가능한 POD를 확인하는 방법.

✔ POD with YAML

· YAML based configuration file을 사용해서 POD를 생성하는 방법

k8s definition files는 항상 4가지의 top-level field를 갖는다.
1. apiVersion : k8s api version을 말한다. → String
- POD를 생성하고자 할 때 : v1
- Service를 생성하고자 할 때 : v1

- ReplicaSet을 생성하고자 할 때 : apps/v1
- Deployment를 생성하고자 할 때 : apps/v1

2. kind : 생성하고자 하는 object의 type을 말한다. → String

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

3. metadata : name labels같은 object에대한 data를 말한다. → Dictionary
metadat의 child에는 k8s에서 인지 가능한 metadata만 넣을 수 있다. 하지만, labels의 child에는 내가 생각하기에 적합하다고 생각하는 것들을 넣을 수 있다.

4. spec : → Dictionary

ex)
apiVersion: v1 → String
kind: Pod → String
metadata: → Dictionary
    name: myapp-pod
    labels:
        app: myapp
        type: front-end

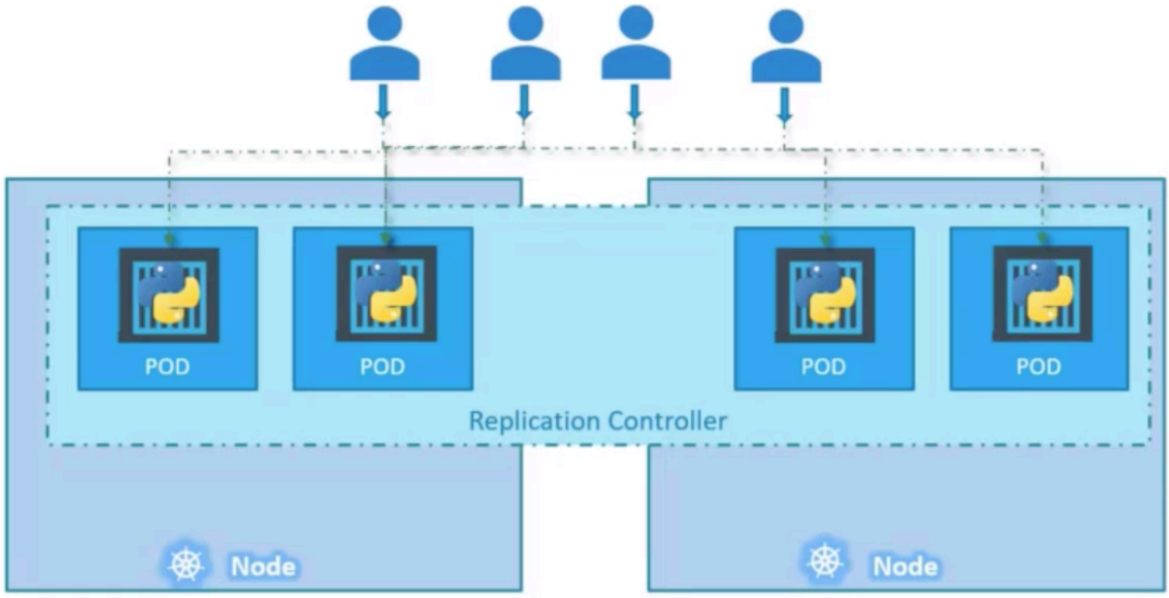
spec: → Dictionary
    containers: → List/Array
    - name: nginx-container → "-" 는 첫번째 list item을 말한다.
      image: nginx
kubectl create -f pod-definition.yml

· kubectl get pods : Pods 확인하는 방법
· kubectl describe pod myapp-pod : 특정 pod의 정보를 보여주는 것

✔Recap - ReplicaSets

· 어떠한 이유로 Application이 Crash되서 Pod가 fails할 때.
유저는 Application에 접근할 수 없게 되는데, 유저가 Application에 대한 접근에 실패하는 것을 방지하기 위해 Replication Controllers가 쿠버네티스 클러스터 내의 파드내에 Multiplle Instance가 돌아갈 수 있도록 한다. 이는 고가용성 (High Availability) 을 보장해준다.
이때 Replication Controllers가 기존과 동일한 Pod를 실행시켜주는 것이다.

※ Node1의 POD1에 한 사용자가 접근하고 있을 때, 다른 추가적인 사용자가 들어오게되어 접근 용량을 초과하게되면 Node1 내에 동일한 Pod를 Replica로 만들어주게되고, 만약 추가적으로 Node 용량을 넘어서는 사용자가 들어오게되면 다른 Node인 Node2의 Pod에 Node1과 동일한 Pod를 Replica로 만들어준다.



· Replication Controller VS Replica Set
둘 다 Replication을 Setup한다는 동일한 목적을 갖고 있지만 차이점이 존재한다.

- Replication Controller
Replication을 Setup하기 위한 오래된 기술로, 현재는 Replica Set이 대체하고 있다.
- Replica Set

· **Replication Controller 설치하는 방법**

- Definition File을 통한 설치.

1. Definition file 작성하기

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: myapp
    type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: frontend   → 검정 글씨는 모두 Replica하고자 하는 POD의 정보이다.
      spec:
        containers:
        - name: nginx-container
          image: nginx

  replicas: 3
```

2. kubectl create -f rc-definition.yaml

※ spec → replication controller는 Multiple Instance를 갖게 해주므로 여러개의 Pod를 생성해야 한다. 따라서 ' Pod template ' 을 사용해야한다. 여기서의 Pod template은 내가 Replicate하고 싶은 POD의 metadata와 spec을 붙여주면된다.

※ replicas → replication controller에서 얼만큼의 replicas가 필요한지에 대한 정보로 ReplicationController의 spec의 template과 sibling 관계여야 한다.

※ 생성된 ReplicationController의 list를 확인하기 위해서는 kubectl get replicationcontroller 를 치면 됨.

• **Replica Set을 설치하는 방법**

- Definition File을 통한 설치

Replication Controller와 유사함.

Replication Controller와 가장 큰 차이 : ‘selector’

1. Definition File 작성하기

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end

spec:
  template:
    name: myapp-pod
    labels:
      app: myapp
      type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

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

2. 생성하기

```
kubectl create -f replicaset-definition.yaml

kubectl get replicaset

kubectl get pods
```

※ selector : Replica set이 어떤 Pod 아래 위치할지를 정해준다?

※ Replica Set 역할은 Pod를 지속적으로 monitor하면서 혹시라도 실패한다면 새로운 Pod를 Deploy해준다.

※ Replica Set이 Pod를 monitor 하는 역할이라 한다면, 동일 클러스터 내에 수많은 Pod가 존재할텐데 뭐를 monitor해야하는지 Replica Set이 어떻게 아냐? -> 이게 바로 selector 란이 있는 이유.
그래서 Replica Set의 selector와 Pod의 metadata가 일치해야되는 것임.

Replication controllers

· Replica Set의 replicas를 수정하고자 할 때.
- 1번 방법 : definition file을 수정하고 업데이트하기
1. replicaset-definition.yaml file의 replicas: 란을 바꿔준다.
2. kubectl replace -f replicaset-definition.yaml → 업데이트해준다.

- 2번 방법 : kubectl scale 명령어 사용하기
3. kubectl scale - -replicas=[변경 숫자] -f [변경하고자 하는 yaml 파일 명]
Ex) kubectl scale - -replicas=6 -f replicaset-definition.yaml
4. kubectl scale - -replicas=6 [Type] [Name]
Ex) kubectl scale - -replicas=6 replicaset myapp-replicaset

※ 2번 명령어를 쓸 경우에는 kubectl scale 커맨드로 replica를 6으로 바꿔줬지만, 실질적으로 yaml 파일은 3으로 그대로 남아있다. 즉, 자동으로 바꿔주지 않는다는 말이다.
→ 자동적으로 바꿔주는 커맨드가 있긴한데 나중에 할거임.

kubectl create -f replicaset-definition.yaml

kubectl get replicaset

kubectl delete replicaset [name]

kubectl replace -f replicaset-definition.yaml : 업데이트 방법