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(영어)
2009. 03 ~ 2012. 02 : 서울 창동고등학교 졸업
2014. 03 ~ 2019. 02 : 청주대학교 컴퓨터정보공학과 학사(중퇴)
2014. 06 ~ 2014. 08 : 에이치엘비제약 근무
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 : 서울 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(퇴사)
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..

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

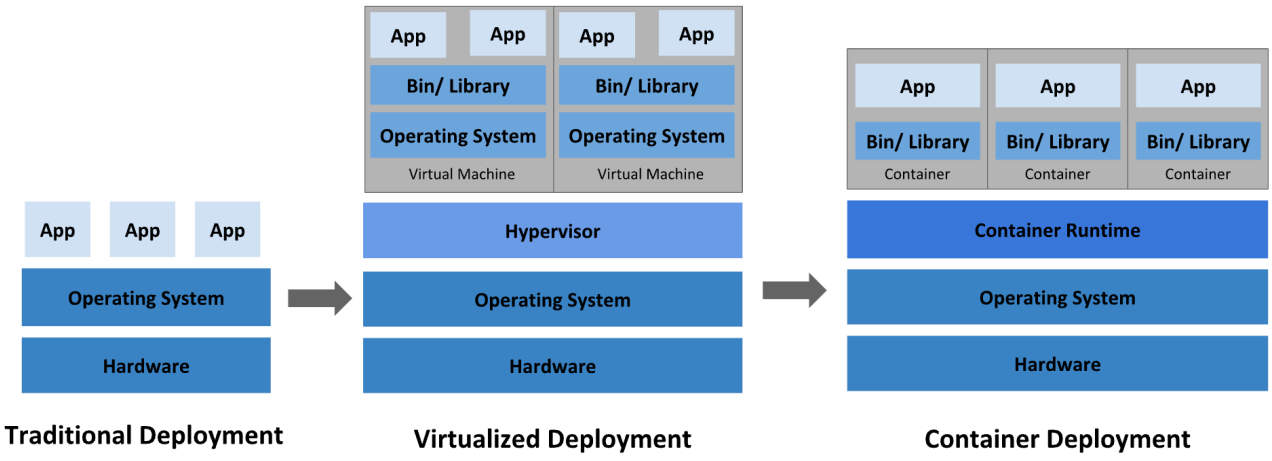
Chapter 1.

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.

Namespace

네임스페이스는 networking Solution, DNS services 등의 서로 다른 규칙을 적용하기 위해 사용한다. 예를 들어 2명의 병욱이라는 같은 이름을 가진 학생이 있을 때, 이 둘을 구분하기 위해서는 last name(ex. 현병욱, 최병욱)이 필요하다. 이 둘은 서로 다른 집과 가족을 갖고 있을 것이고 각 가족은 서로 다른 규칙을 갖고 있을 것이다. 같은 병욱이지만 서로 다른 현병욱과 최병욱의 차이를 위해서는 단순성으로 구분하기 보다는 거주중인 집으로 구분하게되면 이름에 대한 구별이 가능할 뿐만 아니라, 각 가족이 갖고있는 서로 다른 규칙에 대해서도 보존해줄 수 있다. 이때의 집이 네임스페이스인 것이다. 이를 통해 두 컨테이너가 같은 네트워크 네임스페이스에 위치하게 되면 서로 직접적으로 localhost를 통해서 통신할 수 있고 storage도 공유할 수 있다.

Replication-controller

Replication-controller는 노드를 새로운 클러스터에 가져오거나 사용 불가 상태의 노드를 삭제할 수 있는 작업을 수행한다. 이 내용만 보자면 kubelet과 차이가 없는 것 처럼 보일 것이다. 하지만 엄연히 kubelet과 replication-controller는 다른 작업을 수행하게 된다. replication-controller는 레플리카셋의 상태를 모니터링하고 필요한 수의 파드를 사용 가능하도록 만들어주는 기능을 수행한다. 즉, 레플리카셋을 수행하는데 있어서 4개의 파드가 필요한데 파드 1개가 죽게되어 현재 3개의 파드만 존재한다

면 replication-controller는 다른 파드 하나를 추가적으로 생성하게 된다. 이처럼 kubelet과는 다르게 운영 단계에서의 자동성이 보장된 작업을 수행하는 것이 replication-controller이다.

Replica set

레플리카셋은 replication을 setup하기 위한 오래된 기술인 replicatoin-controller를 대체하기 위한 기술로 파드를 지속적으로 모니터링하면서 혹시라도 동작에 실패하여 죽은 상태의 파드가 발생하게 되면 새로운 파드를 deploy시켜준다. 이때 레플리카셋이 파드를 모니터링을 수행하는데 동일 클러스터 내에 존재하는 수많은 파드 중에서 어떤 파드를 모니터링해야 하는지 레플리카셋이 알 수 있도록 지정해주는 것이 'selector'이다. 이를 위해서 레플리카셋의 definition file의 셀렉터와 파드의 메타 데이터가 반드시 일치해야 한다.

Service

서비스는 파드, 레플리카셋 등과 마찬가지로 하나의 object이다. 애플리케이션의 다양한 components에 대한 내부 혹은 외부 간의 통신을 가능하게 하는 기능을 수행하게 되는데, 서비스는 actual thing이 아닌 memory의 cabinet에 위치한 가상의 요소로 파드 네트워크에는 집적 접근할 수는 없지만 간접적으로 파드간의 통신이 가능하도록 상대방의 IP를 외부로 노출시켜주는 역할을 수행한다.

Kube-apiserver

Kube-apiserver는 kubernetes를 전반적으로 관리하는 메인 서버로 다음과 같은 대표적인 역할을 수행한다.

첫 번째, 클러스터 내에 접근하고자 하는 사용자의 인증 단계를 처리한다. 이는 흔히 알려진 key-box와 같은 인증 프로토콜과 유사한 프로세스로 수행되어 클러스터 내의 노드간의 통신 접근성에 대한 인증을 수행하여 보완성을 책임진다.

두 번째, 요청에 대한 타당성을 검증한다. kube-apiserver는 kubernetes 내에 존재하는 다양한 프로세스와 직·간접적으로 연결되어 있는데, 사용자가 kubernetes를 운용하고자 할 때 가장 많이 사용하는 kubectl을 통한 커맨드에 대한 타당성 및 운용 가능성을 검증하게 된다.

세 번째, 데이터를 검색한다. kubernetes에서 동작중인 클러스터와 내부 파드 단위의 동작을 수행하는데 있어서 손상된 components가 있는지 등에 대한 검증을 수행하여 두 번째 기능이였던 요청에 대한 수행 가능성을 검증하는데 도움을 준다. kubernetes의 데이터 검색 대표 예시로 kube-apiserver와 연결된 ETCD에 back-up된 정보 검색이 있다.

네 번째, ETCD를 업데이트 한다. 세 번째 역할에서 언급한 바와 같이 kube-apiserver는 ETCD와 연결되어있다고 볼 수 있다. 이로 인해서 ETCD 버전이나 back-up된 데이터에 대한 업그레이드를 수행할 수 있다.
다섯 번째, 스케줄링을 수행한다. kubernetes의 강력한 기능인 자동 스케줄링(Auto-Scheduling)은 동작중인 애플리케이션 단위의 레벨에 대한 모니터링 기능을 수행하여 서비스의 downtime 없는 동작, 우선 처리 파드 등에 대한 자동 스케줄링 기능을 제공한다.

여섯 번째, kubelet을 지원한다. 위에서 언급한 바와 같이 kube-apiserver는 클러스터 안의 모든 동작에 대한 오케스트레이션을 관리할 수 있는데, 이때 오케스트레이션 관리의 기준을 kubelet으로부터 노드 단위의 모니터링 정보를 보고받게 된다. 이를 통해 kubelet에서 컨테이너의 상태를 점검하여 kube-apiserver 단위에서의 통합 정보 분석이 가능하도록 kubelet을 지원한다.

Kubelet

Kubelet은 kube-apiserver로 부터 명령어를 받아 클러스터 내에 존재하는 모든 컨테이너, 파드, 노드 등에 대한 상태 정보를 kube-apiserver로 보낸다. 또한, 노드에 컨테이너를 파드 단위로 배포 혹은 삭제하는 기능을 수행한다. 사실 kubelet은 kube-apiserver, ETCD 클러스터, kube-scheduler가 없더라도 독립적으로 노드를 관리할 수는 있다. 하지만, kube-apiserver가 없는 상황에서 파드를 생성하게 되면 파드의 details 정보를 제공받지 못하므로 파드 definition file을 kubelet에 직접 전달해줘야 하는 불편함이 발생하게 된다. 이로 인해서 종합적으로 말하자면 kubelet은 노드에 load된 파드에서의 Instruction을 위해 kube-apiserver에 의존한다 할 수 있다.

※ Static POD : kube-apiserver없이 kubelet이 생성한 파드를 Static POD라 부른다.

Kube-proxy

Kube-proxy는 서비스와 클러스터 사이의 통신을 보조한다. 세부적으로 설명하자면 kubernetes의 클러스터는 모든 파드가 서로 다른 파드에 접근할 수 있어야 microservice 단위로 구성된 애플리케이션의 동작을 수행할 수 있다. 이를 위해서 파드 networking solution을 클러스터에 deploy해줄 필요가 있는데, 이를 kube-proxy가 수행하게 된다. 예를 들어 단일 클러스터 내에 노드1, 노드2가 존재하는 상황에서 노드1에 웹 애플리케이션이 deploy되어있고 노드2에 데이터베이스 애플리케이션이 deploy 되어있다고 가정해보자. 이러한 경우 웹 애플리케이션은 데이터베이스 애플리케이션 파드의 IP를 통해 데이터베이스에 접근할 수 있어야 하는데, 데이터베이스 애플리케이션 파드의 IP가 항상 같은 상태로 고정되어있는다는 보장이 없다. 따라서 service를 이용하여 데이터베이스 애플리케이션을 클러스터 밖에 노출시켜줄 수 있는 service:db가 필요하게 된다. 이러한 기능을 종합적으로 수행하는 것이 kube-proxy이다.

Kube-scheduler

kubeadm이 kube-scheduler를 Master 노드의 kube-system namespace에 파드로 deploy하게 되는 Kube-scheduler는 현재 필요한 컨테이너를 가져오는 작업을 수행한다. kubelet을 통해 클러스터 내의 모든 정보를 kube-apiserver가 받고 스케줄링 우선 순위 및 처리 상황에 대한 분석을 통해 kube-scheduler가 기능을 수행하여 스케줄링을 진행하게 된다.

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 있는 기능을 제공한다. 여기서의 백업은 ETCD 클러스터에 저장되게 되는데, ETCD 클러스터에는 노드, 파드, config, secret, accounts, roles, bindings 등의 클러스터 내의 종합적인 정보들이 모두 저장되게 된다. kube-apiserver에서 설명한 바와 같이, ETCD는 kube-apiserver와 유일하게 직접적으로 상호작용한다.

Chapter 3.

Monitoring, Logging, and Troubleshooting

Monitoring

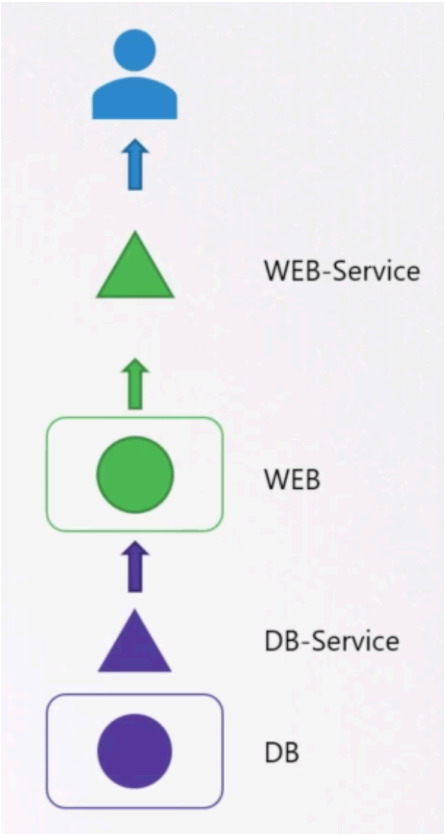
모니터링이란 운영중인 애플리케이션 레벨에 대한 백데이터를 수집하여 문제 해결 능력을 보장하거나 elastic한 자원 할당, 문제 감지 등의 작업을 수행하는 것을 의미한다. 이는 클라우드 네이티브 컴퓨팅의 도입에 있어서 가장 중요한 장점 및 특징으로 꼽힌다. 이를 위해 kubernetes에서는 kubelet에서 클러스터 내의 모든 구성 요소들의 상태 정보를 받아옴으로써 kube-apiserver가 스케줄링을 수행할 수 있도록 지원하는 것이다. 이 과정을 kubernets에서의 모니터링 과정이라 할 수 있다. 한가지 문제점은 kubelet을 통해 생성된 static-pod는 kube-apiserver 없이 생성된 details 정보가 없는 파드이므로 모니터링 과정에서 제외될 수 있다는 단점이 존재한다. 이를 위해서 kubeadm에서 보조적인 기능을 지원받거나 기타 open-source 툴을 통해 보완하기도 한다.

Logging

로깅은 단어가 갖는 의미 그대로 기록하는 것을 말한다. 로깅이 필요한 이유는 시스템 동작시 시스템의 상태 및 작동 정보를 시간의 경과에 따라 기록함으로써 문제 발생시에 신속 정확한 대처가 가능하기 때문이다. 방금 전에 배운 모니터링과 연결해서 생각해보자. 클러스터 내에서 동작중인 애플리케이션의 프로세스에 로깅 작업을 수행했다고 하자. 이러한 경우 동작 과정에서 문제가 발생한다면 시스템 전체를 볼 필요 없이 문제가 발생한 로그만 찾아내면 된다. 이는 형상관리 툴인 'Git'과 유사한 특징을 갖는다고 보면 되는데, 팀단위의 프로젝트의 형상관리를 git을 통해서 진행했다고 해보자. 각 팀원이 공유하여 사용중인 리파지토리에 대해 각자의 작업 환경인 branch에서 push, merge를 수행하는 과정에서 서로 다른 수정 사항이 생기게되면 git 시스템 단에서 confict가 발생하게 된다. 이를 git에서는 log로 기록하여 어느 부분에서 발생한 문제인지 가시적으로 쉽게 확인하고 조치할 수 있도록 되어있다. 이러한 방법과 유사한 것이 바로 kubernetes의 로깅 방식인 것이다.

Troubleshooting

kubernetes 운용에 있어서 트러블 슈팅은 가장 핵심이 되는 작업이라 할 수 있다. 트러블 슈팅은 말 그대로 문제를 해결하는 과정을 의미한다. 아래의 사진을 보면 이해가 쉬울 것이다.



파드 단위에서 동작중인 웹 애플리케이션에 대한 운영을 시작하여 사용자가 확인할 수 있도록 하는 과정을 보면 DB 파드에서 인증 단계를 거쳐서 DB-service를 통해 web 파드와 통신하고 들어온 정보를 토대로 사용자에게 Web-service를 통해 통신하여 데이터를 전달하게 된다. 이러한 경우가 가장 일반적인 파드 단위 레벨에서의 애플리케이션 운영 과정이다. 해당 과정을 보면 상당히 이해하기 쉬운 것처럼

보이지만 사실은 저 단순한 과정 내에도 상당히 복잡한 문제들이 존재한다. 예를 들어보자. web 파드의 web server가 노드 단위의 포트의 IP에 접근 가능한지를 점검해야 한다. 만약 노드 단위 포트 IP에 대한 접근 가능 여부를 Web-service의 Definition File에 지정해주지 않았거나 잘못 지정해놓게 되면 DB 파드로부터 제대로된 정보를 받아왔다 하더라도 실질적으로 사용자에게 해당 정보를 전달할 수 없게 된다. 이 외에도 파드가 running state 여부에 따라서도 각 파드간의 통신이 가능하거나 불가능한 경우가 발생하게 된다. 또한 kubelet의 상태, kube-apiserver의 연결 상태, 각 components 간의 버전 관계 등 상당히 복잡한 양상을 보인다. 이처럼 kubernetes의 도입은 단순한 과정이 아니라 전문적으로 이를 관리할 수 있는 전문 엔지니어가 필요한 복잡한 과정이다. 이를 위해 ‘Beginning kubernetes’를 통해 여러분들은 kubernetes를 학습하고 있는 것이다.