**Container Orchestration (Docker & Kubernetes)**

Copyright © 2020. uEngine-solutions All rights reserved.

1

Container Orchestration (Docker & k8s) **Table of content**

1. MSA, Container, and Container Orchestration

2. Docker Hands-on Lab

3. Kubernetes Basic Object Model : Pod, Label, ReplicaSet, Deployment, Service, Volume, Configmap, Secret, Liveness/Readiness

4. Kubernetes Advanced Lab : Ingress, Job, Cron Job, DaemonSet, StatefulSet

5. Real MSA Application Deployment

6. Kubernetes Architecture

7. Service Mesh: Istio for Advanced Services Control

8. Course Test

2

**Monolithic Architecture**

→ **모든 서비스가 한번에 재배포**

→ **한팀의 반영을 위하여 모든 팀이 대기**

→ **지속적 딜리버리가 어려워**

3

**MSA Architecture**

**Contract based, Polyglot Programming**

→ **Separation of Concerns, Parallel Development, Easy Outsourcing**

Written in Java

Written in Node

Written in PHP

4

**서비스가 죽지 않으면 어떨까?**

5

5

**항상성이 (Self-Healing) 자동으로 유지되면 어떨까?**

6

**Kubernetes**

Master

Workers

7

Container-based Deployment

Java Module System

Hypervisors Containers

**Tools: Evolution of Module Systems**

8

**Principles of Container-based Application Design**

• Image Immutability

• High Observability

• Process Disposability

• Lifecycle Conformance

• Runtime Confinement

• Single Concern

• Self-Containment

https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

9

**Image Immutability Principle (IIP)**

Snowflakes Server\* pattern → Phoenix Server pattern

• Immutable container images must be used across all environments

• For each environment, Container must use externalized configuration way

https://martinfowler.com/bliki/SnowflakeServer.html, 2012 https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

10

**High Observability Principle (HOP)**

• Containerized application must provide APIs for health checks and readiness

• Application should log important events for log aggregation by tool (ex. Fluentd)

https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

11

**Process Disposability Principle (PDP)**

• A containerized application with quick start up and shut down for easy replaceability

• Container replace reasons : Failing a health check, scaling down the application, migrating the containers to a different host, etc

https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

12

Pet and Cattle Strategy

**Lifecycle Conformance Principle (LCP)**

• A container providing APIs and conforming to platform events

• Container must conforms Managing Platform(Docker Swarm, Kubernetes) events

https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

13

**Runtime Confinement Principle (RCP)**

• A container that declares its runtime resource requirements and pass them to the platform and respects them at runtime

• The application stay within the indicated resource requirements is important

https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

14

**Single Concern Principle (SCP)**

• Every Container should address a single concern

• If Containerized Microservice needs multiple concerns, it can use sidecar and init- containers patterns, where each container still handles a single concern.

https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

15

**Self-Containment Principle (S-CP)**

• Containers should have baked in all dependencies at build time

• The only exceptions are things such as configurations, which must be provided at runtime; for example, through Kubernetes ConfigMap

https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

16

**Conclusion of Container-based Application Design**

• Cloud native is more than an end state, it is a way of working

• Some container-related best practices

ᄋ Aim for small images ᄋ Support arbitrary user IDs ᄋ Mark important ports ᄋ Use volumes for persistent data ᄋ Set image metadata

https://kubernetes.io/blog/2018/03/principles-of-container-app-design/?fbclid=IwAR2oMrdP0d1Q6LXebtxNPnt-RS5DIlkCIwpaMSL5mmW7VMaQb6hRV8hkd38

17

**Container Orchestrators**

구글에서 개발, 가장 기능이 풍부하며 널리 사용되는 오케스트레이션 프레임워크

베어 메탈, VM환경, 퍼블릭 클라우드 등의 다양한 환경에서 작동하도록 설계

컨테이너의 롤링 업그레이드 지원

여러 개의 Docker 호스트를 함께 클러스터링 하여 단일 가상 Docker 호스트를 생성

호스트 OS에 Agent만 설치하면 간단하게 작동 하고 설정이 용이

Docker명령어와 Compose를 그대로 사용 가능

수만 대의 물리적 시스템으로 확장할 수 있도록 설계

수만 대의 물리적 시스템으로 확장할 수 있도록 설계

Hadoop, MPI, Hypertable, Spark같은 응용 프로그램을 동적 클러스터 환경에서 리소 스 공유와 분리를 통해 자원 최적화 가능

Hadoop, MPI, Hypertable, Spark같은 응용 프로그램을 동적 클러스터 환경에서 리소 스 공유와 분리를 통해 자원 최적화 가능

Hadoop, MPI, Hypertable, Spark같은 응용 프로그램을 동적 클러스터 환경에서 리소 스 공유와 분리를 통해 자원 최적화 가능

Hadoop, MPI, Hypertable, Spark같은 응용 프로그램을 동적 클러스터 환경에서 리소 스 공유와 분리를 통해 자원 최적화 가능

Kubernetes Docker Swarm Apache Mesos

Docker 컨테이너를 적극적으로 지원

18

**Managing Scalability Workload Distribution Engine**

desired=5 desired=4 desired=5

19

**Managing Scalability Workload Distribution Engine**

20

**Managing Scalability Workload Distribution Engine**

desired=5 actual=**5** desired=4 actual=4 desired=5 actual=**5**

21

**Managing Scalability Workload Distribution Engine**

22

**Managing Scalability Workload Distribution Engine**

ERP v2

SW router (HA proxy, Envoy, nginx)

23

**BizDevOps Process**

• **1 Sprint = 1~2 Weeks**

**Cycle 기반**

• **지속적인 Needs 탐색**

24

Container Orchestration (Docker & k8s) **Table of content**

1. MSA, Container, and Container Orchestration

2. Docker Hands-on Lab

3. Kubernetes Basic Object Model : Pod, Label, ReplicaSet, Deployment, Service, Volume, Configmap, Secret, Liveness/Readiness

4. Kubernetes Advanced Lab : Ingress, Job, Cron Job, DaemonSet, StatefulSet

5. Real MSA Application Deployment

6. Kubernetes Architecture

7. Service Mesh: Istio for Advanced Services Control

8. Course Test

25

**Local Setup for Docker Practice**

• Linux(Ubuntu) install

• Docker Daemon, Client Setup & Configuration

26

**Windows 10에 Linux 설치**

• PowerShell을 관리자 권한으로 실행하여 WSL 활성화

Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux

⚫ 컴퓨터 재시작

WSL(Windows Subsystem for Linux) : Windows에서 가상화없이 native Linux 명령을 직접 실행할 수 있는 Windows10 기능 https://docs.microsoft.com/ko-kr/windows/wsl/install-win10

27

**Windows 10에 Linux 설치**

• Ubuntu 설치

ᄋ 시작메뉴 > Microsoft Store에서 ubuntu 검색 후, 18.04 version 설치

https://docs.microsoft.com/ko-kr/windows/wsl/install-win10

28

**Windows 10에 Linux 설치**

• Ubuntu 초기화

ᄋ 시작 메뉴에서 ‘Ubuntu 18.04 LTS’ 실행

ᄋ 초기화 및 새 Linux 계정 설정

ᄋ Ubuntu의 Archive Repository Server를 국내로 설정

ᄋ sudo vi /etc/apt/sources.list ᄋ :%s/archive.ubuntu.com/ftp.daumkakao.com/g ᄋ :wq!

29

**Linux에 JDK 설치**

• 설치 명령

ᄋ $ sudo apt-get update ᄋ $ sudo apt install default-jdk

• JAVA\_HOME 설정

ᄋ ex) .bashrc에 export JAVA\_HOME=‘/usr/lib/jvm/java-11-openjdk-amd64’ 추가

• 실행 path 추가

ᄋ ex) .bashrc에 export PATH=$PATH:$JAVA\_HOME/bin:. 추가

• 수정사항 반영

ᄋ $ source ~/.bashrc

• 설치 확인

ᄋ $ echo $JAVA\_HOME ᄋ java -version

30

**Docker Setup - Daemon**

• Windows 10에 Docker 데몬(Docker for Windows) 설치

ᄋ https://www.docker.com/products/docker-desktop

• 컴퓨터 재시작

Under windows 10 Pro edition, see : https://docs.docker.com/toolbox/toolbox\_install\_windows/

31

**Docker Setup – Daemon config**

• Docker Hub(https://hub.docker.com/)에 접속하여 계정 생성

• Docker Desktop 실행 후, Docker 계정으로 Sign in

• Settings 메뉴 ‘Expose daemon on~~’ 체크 후 재시작

32

**Docker Setup – Client**

• Linux에 Docker Client 설치

ᄋ sudo apt-get update # Install packages to allow apt to use a repository over HTTPS. ᄋ sudo apt install apt-transport-https ca-certificates curl software-properties-common # Add Docker's official GPG key. ᄋ curl -fsSL https*:*//download.docker.com/linux/ubuntu/gpg | sudo apt-key add # Pick the release channel. ᄋ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable“ ᄋ sudo apt update # Install the latest version of Docker CE. ᄋ **sudo apt install docker-ce** # Allow your user to access the Docker CLI without needing root ᄋ sudo usermod -aG docker [*Linux username*]

33

**Docker Setup – connect Client to Daemon**

• Linux 계정에 Docker Daemon 환경변수 설정

ᄋ vi ~/.bashrc ᄋ export DOCKER\_HOST=tcp://0.0.0.0:2375 # 맨 아래에 환경변수 추가 ᄋ source ~/.bashrc # Shell 재 실행

• Docker Setup 확인

ᄋ $ docker run --name nginx –d –p 80:80 nginx ᄋ $ docker images ᄋ $ docker container list

ᄋ 브라우저에서 80포트 접속확인

34

**Docker & Practice**

35

• 가상머신 .vs. Docker

• 2013년 3월, dotCloud 창업자 Solomon Hykes가 Pycon Conference에서 발표

• Go 언어로 작성된 “The future of linux Containers”

• Container based

ᄋ 프로세스 격리 기술 ᄋ 오픈소스 가상화 플랫폼

36

**Docker Image & Container**

• Docker Image

ᄋ 가상머신 생성시 사용하는 ISO 와 유사한 개념의 이미지 ᄋ 여러 개의 층으로 된 바이너리 파일로 존재 ᄋ 컨테이너 생성시 읽기 전용으로 사용 ᄋ 도커 명령어로 레지스트리로부터 다운로드 가능

**apexacme/ubuntu:18.04**

저장소 이름 이미지 이름 이미지 버전 ᄋ 저장소 이름 : 이미지가 저장된 장소, 이름이 없으면 도커 허브(Docker Hub)로 인식 ᄋ 이미지 이름 : 이미지 이름, 생략 불가 ᄋ 이미지 버전 : 이미지 버전정보, 생략 시 latest 로 인식

37

**Docker Image & Container**

• Docker Container

ᄋ 도커 이미지로 부터 생성 ᄋ 격리된 파일시스템, 시스템 자원, 네트워크를 사용할 수 있는 독립공간 생성 ᄋ 이미지를 읽기 전용으로 사용, 이미지 변경 데이터는 컨테이너 계층에 저장

38

**Layered Architecture**

• Image : 여러 개의 읽기 전용(Read Only) 레이어로 구성

• Container : Image 위에 R/W 레이어를 두고, 실행 중 생성 또는 변경 내용 저장

39

**Docker Image Path**

• 이미지 Path는 <URL>/<namespace>/<Image\_name>:<tag> 형식

• library는 도커허브 공식 이미지 Namespace로, 여기에 사용자 이름이 위치

40

FROM openjdk:8-jdk-alpine

RUN apk --no-cache add tzdata && cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime

WORKDIR /app COPY hello.jar hello.jar COPY entrypoint.sh run.sh RUN chmod 774 run.sh

ENV PROFILE=local

ENTRYPOINT ["./run.sh"]

**Dockerfile sample**

▪ **FROM** : 이미지를 생성할 때 사용할 베이스 이미지를 지정한다.

▪ **RUN** : 이미지를 생성할 때 실행할 코드 지정한다. 예제에서는 패키지를 설치하고 파일 권한을 변경하기 위해 RUN을 사용

▪ **WORKDIR** : 작업 디렉토리를 지정한다. 해당 디렉토리가 없으면 새로 생성한다. 작업 디렉토리를 지정하면 그 이후 명령어는 해당 디렉토리를 기준으로 동작

▪ **COPY** : 파일이나 폴더를 이미지에 복사한다. WORKDIR로 지정한 디렉토리를 기준으로 복사

▪ **ENV** : 이미지에서 사용할 환경 변수 값을 지정한다. 컨테이너를 생성할 때 PROFILE 환경 변수를 따로 지정하지 않으면 local을 기본 값으로 사용

▪ **ENTRYPOINT** : 컨테이너를 구동할 때 실행할 명령어를 지정한다.

41

**Docker Image Commands**

• 도커 이미지 목록 확인

ᄋ $ docker images

• 도커 이미지 불러오기

ᄋ 컨테이너 run 시에 이미지가 없으면 Docker Hub로부터 자동으로 Pull ᄋ $ docker pull [ImageName:태그]

• 도커 이미지 삭제

ᄋ docker image rm [ 이미지 ID ] ᄋ docker image rm -f [ 이미지 ID ] # 컨테이너를 삭제하기 전에 이미지 삭제

• 도커 모든 이미지 한 번에 삭제

ᄋ $ docker image rm $(docker images -q)

42

**Docker Container Commands**

• 컨테이너 실행

ᄋ $ docker run [Options] [Image] [Command]

• 실행 중인 컨테이너 확인

ᄋ $ docker ps ᄋ $ docker ps -a # 정지된 컨테이너 포함

• 컨테이너 시작, 재시작, 종료

ᄋ $ docker start / restart / stop [ 컨테이너 이름 ]

• 컨테이너 삭제

ᄋ $ docker container rm [ 컨테이너 ID ]

• 모든 컨테이너 한번에 삭제 (중지 후 삭제) ᄋ $ docker container rm $(docker ps –a –q)

43

**Docker Build & Push Commands**

• Dockerfile로 이미지 생성

ᄋ docker build --tag [ 생성할 이미지 이름 ] **:** [ 태그 이름 ] . ᄋ # 맨 마지막의 .(마침표)은 Dockerfile의 위치 ᄋ 이미지 이름은 URL(Docker hub, Cloud Container registry, Private registry)로 시작

• 이미지 Push

ᄋ $ docker login # 도커 로그인 ᄋ $ docker push [ 이미지 REPOSITORY ]:[ 태그 ]

• Docker Hub에서 확인

ᄋ http://hub.docker.com (login)

44

**Lab. Docker Build & Push**

• Lab Script Location

ᄋ Workflowy :

45

Container Orchestration (Docker & k8s) **Table of content**

1. MSA, Container, and Container Orchestration

2. Docker Hands-on Lab

3. Kubernetes Basic Object Model : Pod, Label, ReplicaSet, Deployment, Service, Volume, Configmap, Secret, Liveness/Readiness

4. Kubernetes Advanced Lab : Ingress, Job, Cron Job, DaemonSet, StatefulSet

5. Real MSA Application Deployment

6. Kubernetes Architecture

7. Service Mesh: Istio for Advanced Services Control

8. Course Test

46

**Container Orchestration Features**

• 컨테이너 자동 배치 및 복제

• 컨테이너 그룹에 대한 로드 밸런싱

• 컨테이너 장애 복구

• 클러스터 외부에 서비스 노출

• 컨테이너 확장 및 축소

• 컨테이너 서비스간 인터페이스를 통한 연결

47

**Container Orchestrators**

구글에서 개발, 가장 기능이 풍부하며 널리 사용되는 오케스트레이션 프레임워크

베어 메탈, VM환경, 퍼블릭 클라우드 등의 다양한 환경에서 작동하도록 설계

컨테이너의 롤링 업그레이드 지원

여러 개의 Docker 호스트를 함께 클러스터링 하여 단일 가상 Docker 호스트를 생성

호스트 OS에 Agent만 설치하면 간단하게 작동 하고 설정이 용이

Docker명령어와 Compose를 그대로 사용 가능

수만 대의 물리적 시스템으로 확장할 수 있도록 설계

수만 대의 물리적 시스템으로 확장할 수 있도록 설계

Hadoop, MPI, Hypertable, Spark같은 응용 프로그램을 동적 클러스터 환경에서 리소 스 공유와 분리를 통해 자원 최적화 가능

Hadoop, MPI, Hypertable, Spark같은 응용 프로그램을 동적 클러스터 환경에서 리소 스 공유와 분리를 통해 자원 최적화 가능

Hadoop, MPI, Hypertable, Spark같은 응용 프로그램을 동적 클러스터 환경에서 리소 스 공유와 분리를 통해 자원 최적화 가능

Hadoop, MPI, Hypertable, Spark같은 응용 프로그램을 동적 클러스터 환경에서 리소 스 공유와 분리를 통해 자원 최적화 가능

Kubernetes Docker Swarm Apache Mesos

Docker 컨테이너를 적극적으로 지원

48

**Kubernetes**

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

**Kubernetes**는 컨테이너화된 어플리케이션을 자동으로 배포, 조정, 관리할 수 있는 오픈소스 플랫폼이다.

- From Kubernetes Website

49

**Kubernetes Origin**

• Borg System 영향을 받아, 2014년 구글의 의해 처음 발표

• 2015년 7월 21일, v1.0 출시

• 리눅스재단과 Cloud Native Computing Foundation(CNCF) 설립

• Kubernetes를 seed 테크놀로지(seed technology)로 제공

https://www.cncf.io/

50

**Kubernetes key Features (1/2)**

• **Automatic binpacking**

ᄋ 각 컨테이너가 필요로 하는 CPU와 메모리(RAM)를 쿠버네티스에게 요청하면, 쿠버네티스

는 컨테이너를 노드에 맞추어서 자동으로 스케줄링

• **Self-healing**

ᄋ Kubernetes는 실패한 노드에 대해, 컨테이너를 자동으로 교체하고, 재 스케줄링하며, 또한

Health check에 반응하지 않는 컨테이너를 정해진 규칙에 따라 다시 시작

• **Horizontal scaling**

ᄋ Kubernetes는 CPU 및 메모리와 같은 리소스 사용량을 기반으로 애플리케이션을 자동으로

확장 할 수 있으며, 메트릭을 기반으로 하는 동적 스케일링도 지원

• **Service discovery and load balancing**

ᄋ Service Discovery 매커니즘을 위해 Application을 수정할 필요가 없으며, K8s는 내부적으로

Pod에 고유 IP, 단일 DNS를 제공하고 이를 이용해 load balancing

https://kubernetes.io/ko/docs/concepts/overview/what-is-kubernetes/

51

**Kubernetes key Features (2/2)**

• **Automated rollouts and rollbacks**

ᄋ Application, Configuration의 변경이 있을 경우 전체 인스턴스의 중단이 아닌 점진적으로

Container에 적용(rolling update) 가능 ᄋ Release revision이 관리되고 새로운 버전의 배포시점에 문제가 발생할 경우, 자동으로 이전

의 상태로 Rollback 진행

• **Secret and configuration management**

ᄋ Kubernetes는 secrets와 Config 정보에 대해 이미지 재빌드없이 변경관리가 가능하고, Github 같

은 저장소에 노출시키지 않고도 어플리케이션 내에서 보안정보 공유 가능

• **Storage Orchestration**

ᄋ Local storage를 비롯해서 Public Cloud(Azure, GCP, AWS), Network storage등을 구미에 맞게 자

동 mount 가능

https://kubernetes.io/ko/docs/concepts/overview/what-is-kubernetes/

52

**Kubernetes: 발음하기**

미국식: 큐브네리스 영국식: 쿠버네티스 인도식: 꾸-버네띡스

53

**Kubernetes: 쓰기**

Kubernetes

K8S =

54

**Cloud Setup**

• Create Azure account and Subscription

• Register DevOps team & Project define

• Resource Group & Kubernetes Cluster creation

• Azure Container Registry creation

55

**Cloud Platform**

• 직관적이고 편한 UI

• 높은 윈도우 서비스 호환

• 국내 2개 리전 보유

• VM 생성 및 실행 속도

• 우세한 시장 점유율

• 높은 서비스 성숙도

• 광범위한 서비스 교육

• UI 및 사용성, 이용 요금

• Cloud-friendly Biz 디자인

• Cloud-friendly Biz 디자인

• 전문적인 DevOps 보유

• 전문적인 DevOps 보유

• 높은 컴퓨터 오퍼링

• 높은 컴퓨터 오퍼링

• 높은 Windows VM 비용

• 높은 Windows VM 비용

56

**“가장 많은 글로벌 지역 보유, 56개 리전 140개 국가”**

**Azure Cloud Platform**

Kubernetes 서비스 Azure SQL

57

**Azure Cloud Setup**

• Azure 계정 생성

ᄋ http://portal.azure.com 접속

평가판 무료서비스 https://portal.azure.com/?quickstart=true#blade/Microsoft\_Azure\_Billing/FreeServicesBlade

58

**Azure Cloud Setup**

• 무료체험 구독(Subscription) 생성

59

**Azure Cloud Setup**

• DevOps 조직 및 Project 생성

ᄋ dev.azure.com 접속 > new organization 클릭

60

**Azure Cloud Setup**

• 리소스 그룹 생성

61

**Azure Cloud Setup**

• Kubernetes 서비스(Cluster) 생성

생성한 리소스그룹 선택

62

**Azure Cloud Setup**

• Container Registry(컨테이너 레지스트리) 생성

생성한 리소스그룹 선택

63

**Azure Cloud Setup**

• Azure Cloud Setup 확인

ᄋ Cloud Shell 클릭

ᄋ az aks get-credentials --resource-group *My\_Resource\_Group* --name *My-cluster* ᄋ Kubectl config current-context

64

**Cloud Client Setup**

• Azure CLI and Kubectl install, AKS connection

• Create AKS Credentials on Local Ubuntu

• Azure Container Registry Login

• Integrate AKS with ACR(Private Container Registry)

• Config other useful Utility

65

**Kubernetes Client Install**

• Kubectl 설치

ᄋ sudo apt-get update ᄋ sudo apt-get install –y kubectl

• 설치 확인

ᄋ kubectl version --client

66

**Azure-Cli tool Install & Config**

• Azure-Cli 설치

ᄋ curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

• Azure Client 인증하기

ᄋ az login # azure Cli authentication

Subscription 정보가 출력

67

**Azure AKS Connection**

• Local에 Azure AKS Token 생성

ᄋ az aks get-credentials --resource-group *My\_Resource\_Group* --name *My-cluster*

• AKS 연결 확인

ᄋ kubectl config current-context ᄋ kubectl get all

Default kubernetes service 확인

68

**Login Azure Container Registry**

• Azure CLI

ᄋ az acr login --name *user35registry* --expose-token

• 생성확인

ᄋ cat ~/.docker/config.json

69

**Integrate AKS with ACR**

• Azure CLI

ᄋ az aks update -n *My-cluster* -g *My\_Resource\_Group* --attach-acr *uengineorg*

70

**Config other useful Utility (skip)**

• 효율적인 Kubernetes 실습을 위한 화면 분할(Screen splitting) 도구

ᄋ kubectl 오퍼레이션에 따른 Pod 인스턴스 동시 모니터링 ᄋ Load Stress 상황과 Pod의 실시간 Scale In/Out 변화량 멀티 뷰

• ‘tmux’ tool Upgrade

ᄋ Github에서 .tmux.conf 파일 C:\ 다운로드 ᄋ Linux Home 디렉토리로 복사 - cp /mnt/c/.tmux.conf ~/

• Upgrade 확인

tmux 실행 후, 터미널 분할 (ctrl + B, %)

71

**Kubernetes Basic Objects**

• “Desired State” declaration

• Pod, Label, ReplicaSet, Deployment, Service, Volume, Configmap, Secret

• Liveness & Readiness

72

**Kubernetes Core Concept : “Desired State”**

• **Kubernetes는 Current State을 모니터링하면서, Desired State를 유지하려는 습성**

• 직접적인 동작을 명령하지 않고, 원하는 상태를 선언(Not Imperative, But Declarative)

ᄋ Imperative – “nginx 컨테이너를 3개 실행해줘, 그리고 80포트로 오픈해줘.” ᄋ Declarative – “80포트를 오픈한 채로, nginx 컨테이너를 3개 유지해줘.”

Declarative Model & Desired States

73

**Kubernetes Object, Controller and Kubectl**

• Object : K8s의 상태를 나타내는 엔티티로 K8s API의 Endpoint

ᄋ 유형 - Pod, Service, Volume, Namespace 등 ᄋ Spec과 Status 필드를 가짐 - Spec(Desired State), Status(Current State)

• Controller : Object의 Status를 갱신하고, Object를 Spec에 정의된 상태로 지속적으 로 변화시켜 주는 주체

ᄋ 유형 – ReplicaSet, Deployment, StatefulSets, DaemonSet, Cronjob 등

• Kubectl : Command CLI에서 Object와 Controller를 제어하는 K8s Client

ᄋ 발음하기 – “큐브시티엘”, “쿠베시티엘”

74

**Pod ; Kubernetes 최소 배포 단위**

• Pod : 미국식 [pɑːd], 영국식 [pɒd]

ᄋ “물고기, 고래” 작은 떼 (Docker의 심볼이 고래 모양에서 유래) ᄋ 발음하기 : “팟”, “파드“, “포드”

Pod Pod

75

**Pod ; Kubernetes 최소 배포 단위**

• 한 Pod은 내부에 여러 컨테이너를 가질 수 있지만 대부분 1~2개의 컨테이너 가짐

ᄋ 스케일링이 컨테이너가 아닌 Pod단위로 수행되기 때문에, Pod내부에 다수의 컨테이너들이

타이트하게 묶여 있으면 스케일링이 쉽지 않거나 비효율적으로 이뤄지는 문제

• 1개의 Pod은 1개의 물리서버(Node) 위에서 실행

• 동일 작업을 수행하는 Pod은 ReplicaSet Controller에 의해 룰에 따라 복제생성

ᄋ 이때 복수의 Pod이 Master의 Scheduler에 의해 여러 개의 Node에 걸쳐 실행

• Pod의 외부에서는 이 ‘Service’ 를 통해 Pod에 접근

ᄋ Service를 Pod에 연결했을 때 Pod의 특정 포트가 외부로 expose

76

**Kubernetes Object Model**

Service (name: foo)

Deployment

ReplicaSet (Blue)

**Responsible for :** http://serviceId:8080

• Dynamic Service Binding

Micro-Service

• e.g. http://foo:8080

• Type :

- LoadBalancer e.g. Ingress (API GW) or front-end - ClusterIP e.g. 내부 마이크로 서비스들 http://external\_ip:8080

Load Balancer from cloud provider

Pod

Container (Main)

Redirects

• Zero-down time deployment

• (Kubernetes default is rolling-update) e.g. Blue / Green

Creates / Manages

ReplicaSet

• Keep replica count as desired (Green) (replicas=2)Pod

Pod

Pod

• Service Hosting

(Side-car) Container

Container Container

Container Container

Container Container

77

**Kubernetes Object Model**

• **apiVersion** : 해당 Object description 을 해석할 수 있는 API server 의 버전

• **kind** : 오브젝트의 타입 – 예제는 **Deployment**

• **metadata** : 객체의 기본 정보. 예) 이름

• **spec (spec and spec.template.spec)** : 원하는 ”Desired State” 의 세부 내역. 예제에서는 3개의 replica를 template 내 의 pod 정의대로 찍어내어 유지하라는 desired state 설정임

• **spec.template.spec** : defines the desired state of the Pod. The example Pod would be created using **nginx:1.7.9**.

Once the object is created, the Kubernetes system attaches the **status** field to the object

apiVersion: apps/v1 kind: Deployment metadata:

name: nginx-deployment labels:

app: nginx spec:

replicas: 3 selector:

matchLabels: app: nginx template:

metadata:

labels:

app: nginx spec:

containers: - name: nginx

image: nginx:1.7.9 ports: - containerPort: 80

Deployment object Example

78

> my-app.yml

**> Desired state**

apiVersion: apps/v1 kind: Deployment metadata:

name: nginx-deployment labels:

app: nginx spec:

replicas: 2 ...template:

metadata:

labels:

app: nginx spec:

containers: - name: nginx

image: nginx:1.7.9 ports: - containerPort: 80 ---apiVersion: apps/v1 kind: Service metadata:

name: nginx-service labels:

app: nginx

---apiVersion: apps/v1 kind: Deployment metadata:

---apiVersion: apps/v1 kind: Deployment metadata:

---apiVersion: apps/v1 kind: Deployment metadata:

name: backend-deployment labels:

app: backend spec:

replicas: 3 ...template:

metadata:

labels:

app: backend spec:

containers: - name: backend

image: backend:latest ports: - containerPort: 8080 ---apiVersion: apps/v1 kind: Service metadata:

name: backend-service labels:

app: backend

---apiVersion: apps/v1 kind: Deployment metadata:

---apiVersion: apps/v1 kind: Deployment metadata:

---apiVersion: apps/v1 kind: Deployment metadata:

---apiVersion: apps/v1 kind: Deployment metadata:

name: db-deployment labels:

name: db-deployment labels:

name: db-deployment labels:

app: db spec:

app: db spec:

app: db spec:

replicas: 2 ...template:

replicas: 2 ...template:

replicas: 2 ...template:

metadata:

metadata:

metadata:

labels:

labels:

labels:

app: db spec:

app: db spec:

app: db spec:

containers: - name: db

containers: - name: db

containers: - name: db

image: mongo ports: - containerPort: 27017 ---apiVersion: apps/v1 kind: Service metadata:

image: mongo ports: - containerPort: 27017 ---apiVersion: apps/v1 kind: Service metadata:

image: mongo ports: - containerPort: 27017 ---apiVersion: apps/v1 kind: Service metadata:

name: mongo-service labels:

name: mongo-service labels:

name: mongo-service labels:

app: mongo

app: mongo

**Declaration based configuration**

79

**My App. 쿠버네티스에 배포해 보기**

• Lab Script Location

ᄋ Workflowy :

80