

DEVOPS_PORTFOLIO

컴퓨터 공학과 박종하

목차

1. 1주차

1.1 실습 이론

1.1.1 Git Flow/Github Flow의 차이

1.1.2 Docker 와 Container

1.2 실습 내용

1.2.1 Python+Flask 를 통한 간단한 웹 어플리케이션 작성

1.2.2 웹어플리케이션을 Docker로 빌드하여 DockerHub 이미지 등록

1.2.3 소스코드를 Github에 등록

1.3 실습 결과

2. 2주차

2.1 실습 이론

2.1.1 CI의 정의

2.1.2 CI의 목적

2.1.3 CI의 장점

2.1.4 CI의 솔루션

2.2. 실습 내용

2.2.1 GitHub 레포지토리에 Continuous integration 연동

2.2.2 소스코드가 변경이 되면 CI도구(AWS CodeBuild)가 자동으로 도커로 빌드

2.2.3 빌드 된 도커 이미지를 자동으로 DockerHub 레포지토리에 등록

2.3 실습 결과

3. 3주차

3.1 실습 이론

3.1.1 수동으로 배포 및 codepipeline으로 자동으로 배포할 때 차이점

3.1.2 CI/CD pipeline의 흐름(pathway), CodePipeline의 목적과 장점

3.1.3 ElasticBeanstalk

3.2 실습 내용

3.2.1 CI도구와 클라우드(AWS)를 연동

3.2.2 CI/CD Pipeline도구를 통해 자동으로 클라우드로 배포되도록 구성

3.3.3 코드를 업데이트 하고 Git으로 merge후 자동으로 클라우드에 서비스가 배포되는 것을 확인

3.3 실습 결과

4. 4주차

4.1 실습 이론

4.1 kubernetes의 정의 및 필요성

4.2 AWS에서 컨테이너를 사용하기 위한 방법

4.2 실습 내용

4.1 AWS를 통해 쿠버네틱스 인프라 환경 구성

4.2 웹 어플리케이션, DB 어플리케이션을 쿠버네틱스에 배포

4.3 구축된 쿠버네틱스에 Grafana 모니터링 환경 연동

4.3 실습 결과

1. 1주차

1.1 실습 이론

1.1.1 Git Flow/Github Flow의 차이

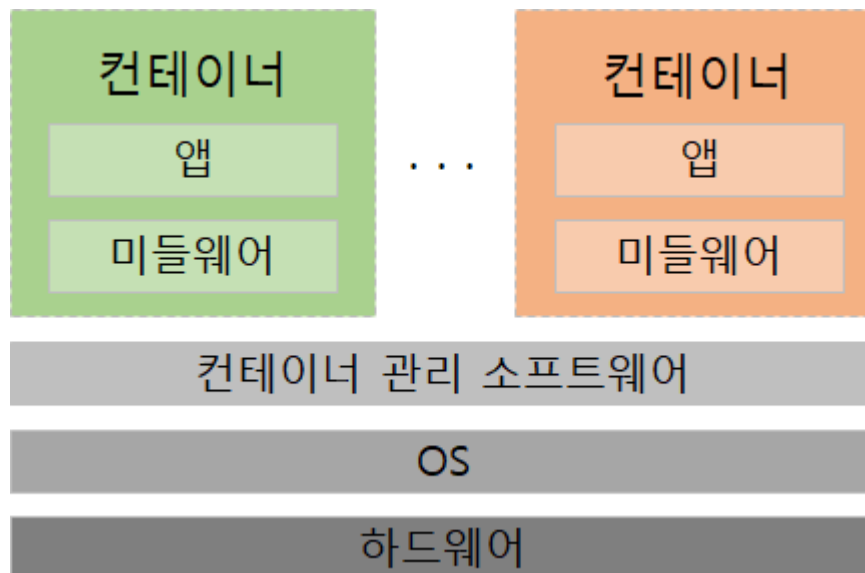
1) Git Flow

- feature → develop → release → hotfix → main 5가지 branch 구조
- 대규모, 정기적인 배포가 필요한 프로젝트를 효율적으로 관리
- 규모가 커질 수록 branch구조가 복잡해지고 개발 정책에 따른 branch 관리 전략이 필요

2) GitHub Flow

- Git flow를 개선하기 위해 나온 방식이며 단순함
- Commit→pull request→main branch로 병합→ 자동 배포
- Main branch가 주요 역할을 하며 항상 배포가 가능한 상태로 운영됨

1.1.2 Docker 와 Container



<DOCKER 의 구조>

컨테이너란 호스트 OS상에 논리적인 구획(컨테이너)을 만들고, 어플리케이션을 작동시키기 위해 필요한 라이브러리나 어플리케이션 등을 하나로 모아, 마치 별도의 서버인 것처럼 사용할 수 있게 한다. 호스트 OS의 리소스를 논리적으로 분리시키고, 여러 개의 컨테이너가 공유하여 사용한다. 컨테이너는 오버헤드가 적기 때문에 가볍고 고속으로 작동하는 것이 특징이다.

Docker(도커)는 어플리케이션의 실행에 필요한 환경을 하나의 이미지로 모아두고, 그 이미지를 사용하여 다양한 환경에서 어플리케이션 실행 환경을 구축 및 운용하기 위한 컨테이너 기반의 오픈소스 플랫폼이다.

1.2 실습 내용

1.2.1 docker 설치

```
yum install docker // docker 설치
systemctl start docker // Docker 실행
systemctl status docker // docker status 확인
```

1.2.2 docker 이미지 빌드

<https://docs.docker.com/language/python/build-images/>

1.2.3 docker 시작

<https://docs.docker.com/language/python/run-containers/>

1.2.4 Dockerhub 에 등록

1.3 실습 결과

Github : <https://github.com/jongha2788/devops>

Dockerhub : <https://hub.docker.com/repository/docker/jongha2788/flask-app>

```
---> cc857b649f6d
Removing intermediate container 66d0007b9164
Step 5/6 : COPY . .
---> 53c461f53dc9
Removing intermediate container 37c9bf6c3603
Step 6/6 : CMD python3 -m flask run --host=0.0.0.0
---> Running in 471b97a931be
---> 29e7e9f0df7e
Removing intermediate container 471b97a931be
Successfully built 29e7e9f0df7e
[root@jongha flask-app]# docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
jongha2788/flask-app v1              29e7e9f0df7e   4 seconds ago  56.8 MB
docker.io/python     3.9.5-alpine   25d7d3156527   9 days ago     45 MB
[root@jongha flask-app]# docker run --publish 5000:5000 jongha2788/flask-app:v1
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://172.17.0.2:5000/ (Press CTRL+C to quit)
172.17.0.1 - - [08/Jul/2021 14:50:02] "GET / HTTP/1.1" 200 -
172.17.0.1 - - [08/Jul/2021 14:50:05] "GET / HTTP/1.1" 200 -
[root@jongha flask-app]# docker push docker.io/jongha/flask-app
```

root@jongha:~/flask-app

```
<h1>Welcome to DevOps</h1>
</body>
</html>[root@jongha flask-app]# ^C
[root@jongha flask-app]# curl localhost:5000
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="/static/css/style.css">
  <title>DevOps Tutorial</title>
</head>
<body>
  <h1>Welcome to DevOps</h1>
</body>
</html>[root@jongha flask-app]# curl localhost:5000
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="/static/css/style.css">
  <title>DevOps Tutorial</title>
</head>
<body>
  <h1>Welcome to DevOps</h1>
</body>
</html>[root@jongha flask-app]#
```

다음과 같이 성공적으로 image 가 빌드 된 것을 확인할 수 있다

2. 2주차

2.1 실습 이론

2.1.1 CI의 정의

지속적 통합. 소프트웨어 개발에서 각 소프트웨어 개발자가 작업한 변경점을 프로젝트의 원래 소스 코드에 자주, 빠르게 통합하는 것이다.

2.1.2 CI의 목적

규모가 커지고 여러 사람이 작업하면 지속적으로 통합 작업을 해야 하는데 이를 자동화하기 위해 만들어졌다.

2.1.3 CI의 장점

애플리케이션에 대한 새로운 코드 변경 사항이 정기적으로 빌드 및 테스트되어 공유 레포지토리에 통합되므로, 여러 명의 개발자가 동시에 애플리케이션 개발과 관련된 코드 작업을 할 경우 서로 충돌할 수 있는 문제를 해결 가능하다.

https://docs.aws.amazon.com/ko_kr/whitepapers/latest/practicing-continuous-integration-continuous-delivery/practicing-continuous-integration-continuous-delivery.pdf

2.1.4 CI의 솔루션

Aws 은 CodeCommit , Codebuild, Codepipeline, CodeDeploy 등 과 같은 Developer tool을 제공하며 CI/CD pipeline 을 구축한다. 이외에도, Jenkins, GitLab CI, Travis 등이 존재한다.

https://docs.aws.amazon.com/ko_kr/whitepapers/latest/introduction-devops-aws/introduction-devops-aws.pdf#welcome

1) AWS CodeCommit

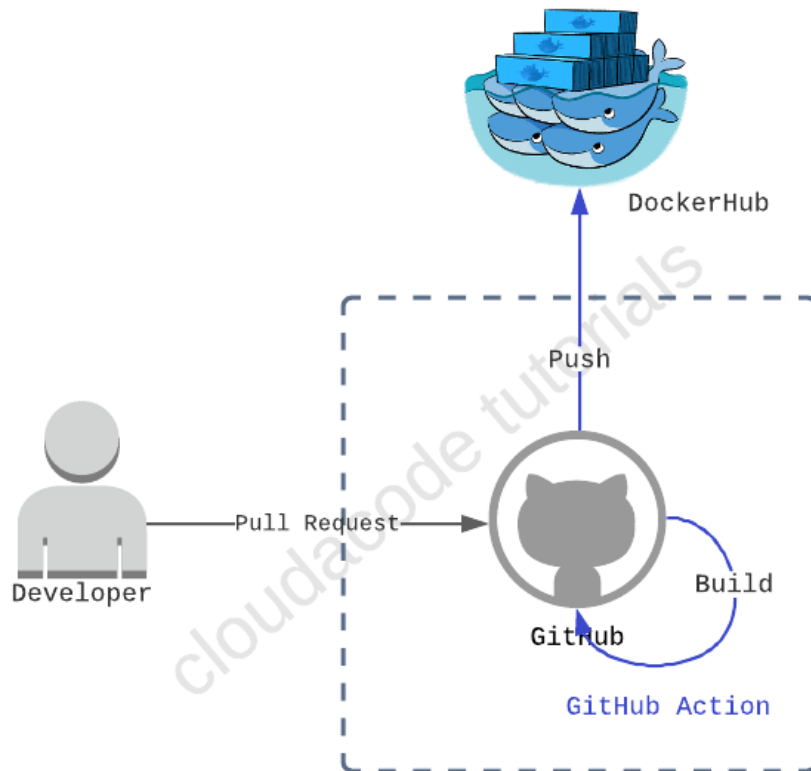
Amazon Web Services에서 호스팅하는 버전 관리 서비스로서 자산 (예: 문서, 소스 코드, 바이너리 파일)을 클라우드에서 비공개로 저장하여 관리할 수 있다.

2) AWS CodeBuild

클라우드의 종합 관리형 빌드 서비스로서 소스 코드를 컴파일하고 단위테스트를 실행하며 배포 준비가 된 소프트웨어 패키지를 생성한다.

2.2. 실습 내용

실습의 구조는 다음과 같다.



master로 코드가 커밋이 되면 어플리케이션을 도커로 빌드 자동화 및 도커 레지스트리 (hub.docker.com)에 이미지 등록하는 과정이다.

즉 1) GitHub 레포지토리에 Continuous integration 연동하고 ,2) 소스코드가 변경이 되면 CI도구(AWS CodeBuild)가 자동으로 도커로 빌드한다. 3) 빌드된 도커 이미지를 자동으로 DockerHub 레포지토리에 등록하는 과정을 진행한다.

2.2.1 AWS Codebuild Project를 생성

1) Make a BuildSpec file for CodeBuild

최상위 디렉토리에 CodeBuild의 작업을 정의한 buildspec.yml를 생성한다

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Docker Hub...
      - docker login -u $DOCKERHUB_USER -p $DOCKERHUB_PW
      - TAG="$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | head -c 8)"
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $IMAGE_REPO_NAME:$TAG .
      - docker tag $IMAGE_REPO_NAME:$TAG $IMAGE_REPO_NAME:$TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      - docker push $IMAGE_REPO_NAME:$TAG
```

2) codebuild 에 필요한 환경을 Setup 한다.

<https://ap-northeast-2.console.aws.amazon.com/codesuite/codebuild/projects>

위의 링크를 들어가서, 다음과 같이 환경을 세팅해준다.

-소스: Github, 내 GitHub 계정의 리포지토리

-GitHub Personal access token 생성 필요

-권한은 repo, admin:repo_hook

-Webhook: 코드 변경이 이 리포지토리에 푸시될 때마다 다시 빌드

-이벤트유형: PULL_REQUEST_CREATED, PULL_REQUEST_UPDATED, PULL_REQUEST_REOPENED

-특정 Branch 이름이나 Tag로 이벤트를 감지 하고 싶다면 Start a build under these condition에 필터 추가 참고 문서 e.g., feature/ 브랜치 이벤트만 HEAD_REF: ^refs/heads/feature/*

-환경: 관리형 이미지, Ubuntu, Standard, aws/codebuild/standard:4.0, 권한 승격 활성화

-서비스 역할: 새 서비스 역할 (Name: default e.g., codebuild-[project_name]-service-role)

- Additional configuration 에 환경 변수 설정

1. TAG_VERSION(*일반 텍스트*): latest

2. IMAGE_REPO_NAME(*일반 텍스트*): [Docker Repo Name] e.g.,
cloudacode/devops-flask
 3. DOCKERHUB_USER(*Secrets Manager*): dockerhub:username
 4. DOCKERHUB_PW(*Secrets Manager*): dockerhub:password
- BuildSpec: default(빈칸) 혹은 buildspec.yml 입력
- 상위 디렉토리에 buildspec.yml이라는 파일로 이름을 정했으므로 별도의 입력값 필요 없음
- Artifact: 없음
- CloudWatch: Default(CloudWatch 로그 선택)

2.2.2 Secret Manager 구성

Username 과 password 의 보안을 위해 암호관리가 필요하다.

Store a new secret – Type: Other type of Secrets

- Secret key/value에 username:

DOCKERHUB 계정이름, password: DOCKERHUB 계정패스워드 입력

- Secret Name: dockerhub

2.2.3 Configure IAM policy

SecretManager에서 정의한 dockerhub secret도 읽는 권한을 부여하기 위해 CodeBuildSecretsManagerPolicy-[codebuild project name]-ap-northeast-2의 Resource에 secretsmanager dockerhub ARN 추가 필요

2.3 실습 결과

- 1) AWS CodeBuild 동작 여부 스크린샷, 깃허브에서 Pull Request시 트리거 잘되는지 확인

change index.html #5

Open

jongha2788 wants to merge 1 commit into master from new2

Conversation 0

Commits 1

Checks 0

Files changed 2

jongha2788 commented now

No description provided.

change index.html

d89a81b

Add more commits by pushing to the new2 branch on jongha2788/devops.

Checking for ability to merge automatically...

Hang in there while we check the branch's status.

Merge pull request

You can also open this in GitHub Desktop or view command line instructions.

Write

Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Reviewers

No reviews

Still in progress

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Linked issues

Successfully resolved this issue

None yet

change index.html #5

Merged

jongha2788 merged 1 commit into master from new2 4 minutes ago

Conversation 0

Commits 1

Checks 0

Files changed 2

jongha2788 commented 6 minutes ago

No description provided.

change index.html

d89a81b

jongha2788 merged commit 062484d into master 4 minutes ago

Hide details

Revert

1 check passed

AWS CodeBuild ap-northeast-2 (devops-cicd) Build succeeded for project devops-cicd

Details

jongha2788 deleted the new2 branch 3 minutes ago

Restore branch

Review

No reviews

Assignees

No one assigned

Labels

None yet

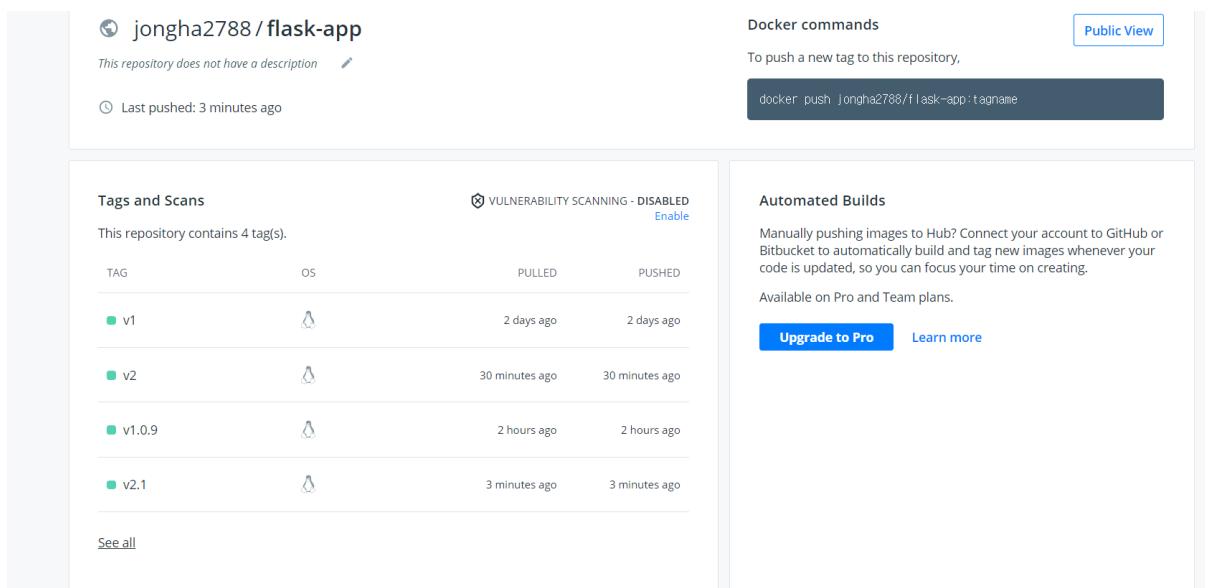
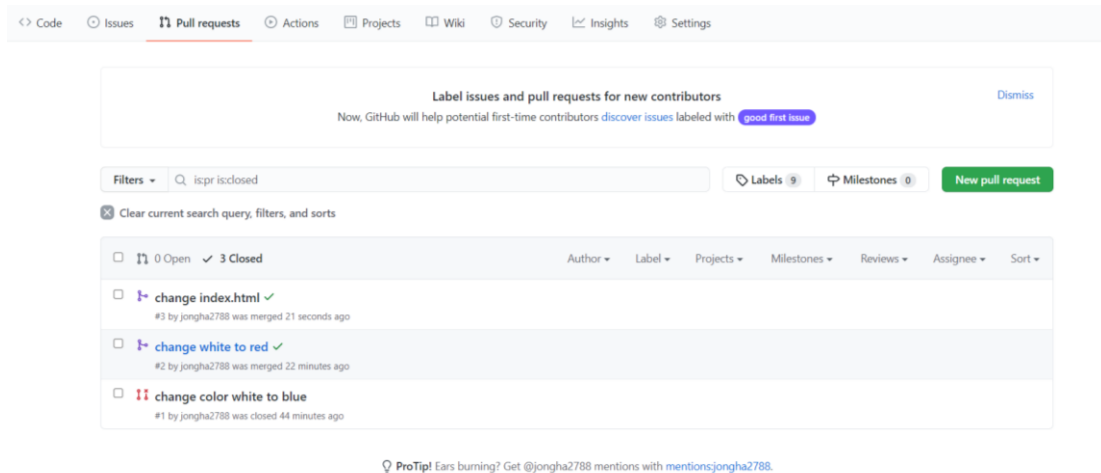
Projects

None yet

Milestone

No milestone

2) Code를 Main으로 Merge 및 DockerHub에 자동으로 image가 업로드 되는지 확인



Commit message “change index.html” 변경사항이 image tag v2.1로 빌드 된 것을 확인할 수 있다.

깃헙: <https://github.com/jongha2788/devops/pulls?q=is%3Apr+is%3Aclosed>

Dockerhub : <https://hub.docker.com/repository/docker/jongha2788/flask-app>

3. 3주차

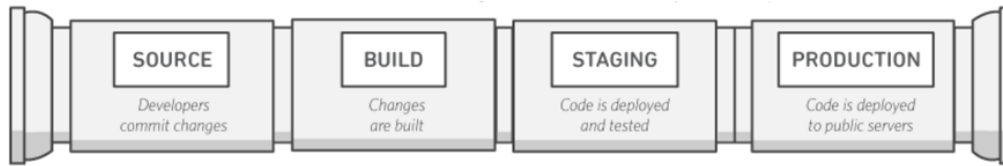
3.1 실습 이론

3.1.1 수동으로 배포 및 codepipeline으로 자동으로 배포할 때 차이점

수동으로 배포하였을 때는 만약 코드가 변경되어 이미지가 다시 빌드 되었다면, 일일히 배포를 다시 진행하여야 한다는 특성이 있다. 하지만 codepipeline을 통해 배포를 하게 되면, 코드가 변경되어 이미지가 재생성 되어도, 자동으로 배포가 되기 때문에 팀 단위의 프로젝트를 진행할 때 용이하게 사용할 수 있다.

3.1.2 CI/CD pipeline의 흐름(pathway), CodePipeline의 목적과 장점

1) CI/CD pipeline pathway



CI/CD 파이프라인의 각 단계는 위와 같이 구성되는데, 코드가 파이프라인을 통해 진행됨에 따라 코드의 더 많은 측면이 계속 검증된다. 만약 초기에 문제가 발견되면, 코드가 파이프라인을 통해 진행되는 것을 멈추고, 테스트 결과는 즉시 팀에 전송되며 소프트웨어가 단계를 통과하지 못하면 모든 추가 빌드 및 릴리스가 중지된다.

2) CodePipeline 목적

코드가 변경될 때마다 정의된 워크플로우에 따라 애플리케이션을 빌드, 테스트 및 배포를 자동화할 수 있게 한다.

3) CodePipeline 장점

Build, test, release 과정을 자동화함으로써 소프트웨어 업데이트의 속도와 품질을 향상시킬 수 있다. (rapid delivery, improved quality, easy to integrate, configurable workflow)

3.1.3 ElasticBeanstalk

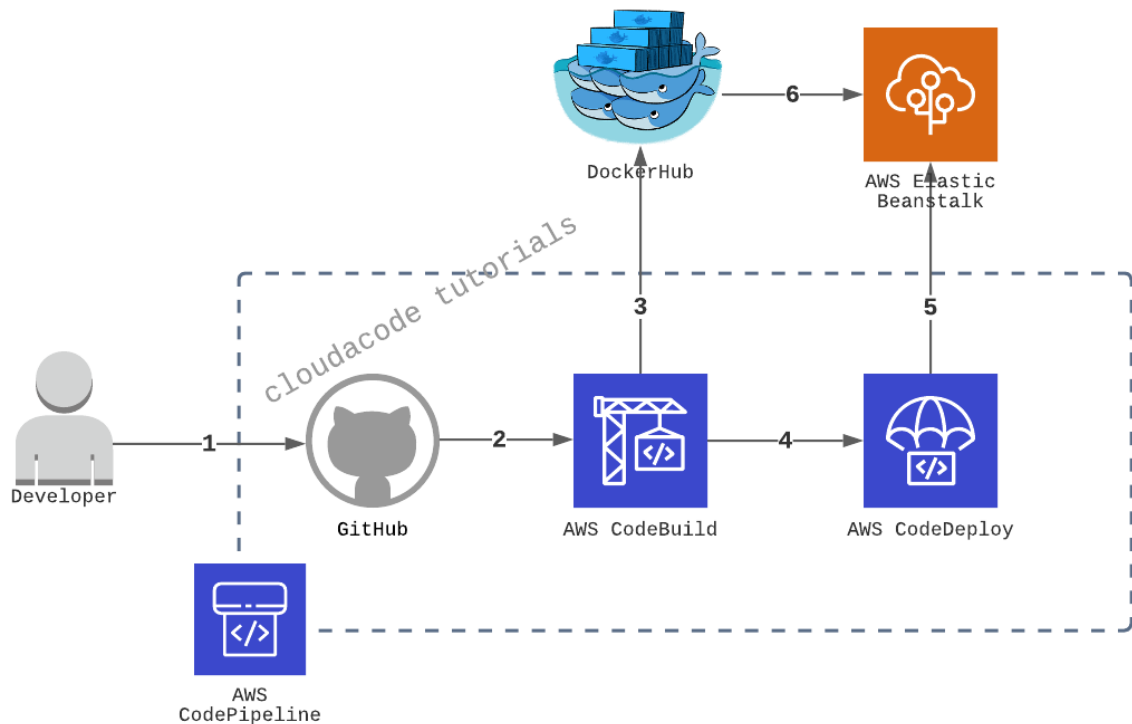
- Docker 컨테이너를 기반으로 애플리케이션을 쉽게 배포하고, 운영하고, 관리하는걸 도와주는 AWS 서비스로, 애플리케이션을 업로드하면 실행하는 데에 필요한 리소스를 프로비저닝한다.

- 프로비저닝: 시스템 자원을 할당, 배치, 배포해두었다가 필요 시 시스템을 즉시 사용할 수 있는 상태로 준비해두는 것

-오토 스케일링, 로깅, 모니터링, 보안 등 지원

3.2 실습 내용

실습의 개요는 다음과 같다.



따라서, 1) CI도구와 클라우드(AWS)를 연동하고, 2) CI/CD Pipeline도구를 통해 자동으로 클라우드로 배포되도록 구성 3) 코드를 업데이트 하고 Git으로 merge후 자동으로 클라우드에 서비스가 배포되는 것을 확인한다.

3.2.1 ElasticBeanstalk 환경을 세팅

Create Application(Create a Web app)

1. Application Name
2. Platform: Docker, Platform Branch: Docker running...Amazon Linux 2, Platform version: Recommended
3. Application Code: Sample application

3.2.2 ElasticBeanstalk 를 위해 Buildspec 을 update

```

version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Docker Hub...
      - docker login -u $DOCKERHUB_USER -p $DOCKERHUB_PW
      - TAG_VERSION="latest"
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
  
```

```

    - docker build -t $IMAGE_REPO_NAME:$TAG_VERSION .
    - docker tag $IMAGE_REPO_NAME:$TAG_VERSION
$IMAGE_REPO_NAME:$TAG_VERSION
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
      - docker push $IMAGE_REPO_NAME:$TAG_VERSION
      - echo Writing image definitions file...
      - printf
'{"AWSEBDockerrunVersion":"1", "Image":{"Name":"%s"}, "Ports":[{"ContainerPort":
8000}]}' $IMAGE_REPO_NAME:$TAG_VERSION > Dockerrun.aws.json
  artifacts:
    files: Dockerrun.aws.json

```

3.2.3 codepipeline 셋업

1) pipeline setting

1. Pipeline Name
2. Service Role: New Service Role
3. Role Name: AWSCodePipelineServiceRole-ap-northeast-2-[Pipeline Name]
4. AWS CodePipeline 이 이 새 파이프라인에 사용할 서비스 역할을 생성하도록 허용
활성화

2) Source stage

1. 소스: Github (Version 1), 내 GitHub 계정의 리포지토리
2. Github v1
3. Repository, Branch: 본인의 Repo, 원하는 Branch name e.g., main, dev, release
4. Detection option: GitHub Webhook (recommended)

3) build stage

3주차 때 이용했던 codebuild project 를 활용한다.

4) Deploy stage

1. Provider: AWS Elastic Beanstalk

2. Application Name, Environment Name: 위에서 자동 생성한 [EB 정보](#)

5) IAM에 권한 부여

3.3 실습 결과

1) AWS Codepipeline

The screenshot shows the AWS CodePipeline console for a pipeline named 'flask-pipeline'. The pipeline consists of three stages: Source, Build, and Deploy. Each stage is marked as 'Succeeded'.

- Source Stage:** Succeeded. Pipeline execution ID: ff7e0ff3-48a0-4aef-bce2-12b320d265d1. Action: Source (Merge pull request #5 from jongha2788/new4).
- Build Stage:** Succeeded. Pipeline execution ID: ff7e0ff3-48a0-4aef-bce2-12b320d265d1. Action: Build (AWS CodeBuild).
- Deploy Stage:** Succeeded. Pipeline execution ID: ff7e0ff3-48a0-4aef-bce2-12b320d265d1. Action: Deploy (AWS Elastic Beanstalk).

2) ElasticBeanStalk

The screenshot shows the AWS Elastic Beanstalk console for an application named 'flaskapp'. It displays a table of environments.

Environment name	Health	Date created	Last modified	URL	Running versions	Platform	Platform state	Tier name
flaskapp-env	Healthy	2021-07-19 13:36:18 UTC+0900	2021-07-19 15:13:51 UTC+0900	flaskapp-env.ubla.elpivoff.ap-northeast-2.elb.amazonaws.com	code-pipeline-7628675190205-BuildK0fac-4769234-413b-4038-8149-65a48221711	Docker running on 64bit Amazon Linux 2	-	WebServer

3) Github 주소 , dockerhub 주소 및 PR history

<https://github.com/jongha2788/devops/pulls?q=is%3Apr+is%3Aclosed>

<https://hub.docker.com/repository/docker/jongha2788/flask-app>

4. 4주차

4.1 실습 이론

4.1.1 kubernetes의 정의 및 필요성

Kubernetes는 컨테이너형 워크로드 및 서비스를 관리하기 위한 확장 가능한 휴대용 오픈 소스 플랫폼으로 declarative configuration과 자동화를 모두 지원한다. 프로덕션 환경에서는 애플리케이션을 실행하는 컨테이너를 관리하고 다운타임이 없도록 해야 한다. 따라서, 시스템 적으로 처리될 수 있도록, Kubernetes는 분산 시스템을 탄력적으로 실행할 수 있는 프레임워크를 제공하고, 애플리케이션의 확장 및 failover를 관리 및 배포 패턴을 제공하는 등의 작업을 수행한다.

4.1.2 AWS에서 컨테이너를 사용하기 위한 방법

4.1.2.1 서버 관리 여부를 고려할 때

컨테이너용 서버리스 컴퓨팅의 경우 AWS Fargate를 선택하고, 컴퓨팅 환경의 설치, 구성 및 관리를 제어하려면 Amazon Elastic Compute Cloud(Amazon EC2)를 선택한다.

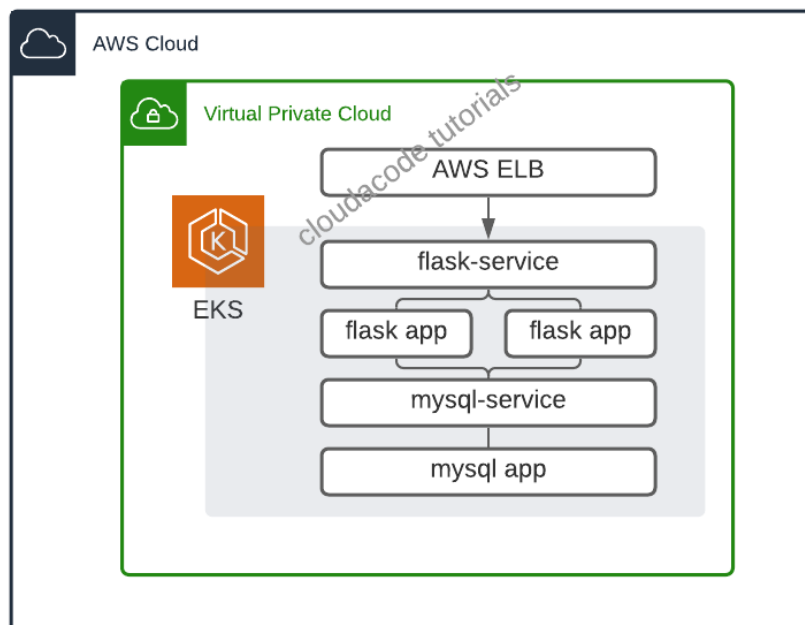
4.1.2.2 사용할 컨테이너 오케스트레이터를 고려할 때

Amazon ECS는 AWS 플랫폼의 나머지 기능과 긴밀하게 통합되고 AWS가 완벽하게 제어하므로, 출시하는 모든 기능이 ECS와 긴밀하게 작동하도록 보장할 수 있다.

Amazon EKS는 Kubernetes 관리 서비스로, 업스트림 Kubernetes를 운영하고 있기 때문에, 커뮤니티에서 제공하는 오픈 소스 툴링의 이점을 활용할 수 있다. 코드를 리팩터링하지 않고도 표준 Kubernetes 어플리케이션을 EKS로 migration 가능하다.

4.2 실습 내용

실습에 대한 개요는 다음과 같다.



- 1) AWS를 통해 쿠버네틱스 인프라 환경 구성하고, 2) 웹 어플리케이션, DB 어플리케이션

션을 쿠버네틱스에 배포하는 과정이다

4.2.1 EKS 구성하기

1) IAM user for EKS

사용중인 IAM 엔터티(유저, Role)가 있다면 eksctl 권한이 있는지 검토. 원활한 실습을 위해 AdministratorAccess policy 부여한다.

2) Install eksctl and kubectl

EKS 생성을 위해 eksctl을 설치하고 추후 kubernetes 관리를 위해 kubectl도 사전에 설치한다.

3) Deploy EKS Cluster

EKS 배포를 위한 구성 정보 파일 (eks-cluster-config.yml) 작성

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: cloud-eks-cluster
  region: ap-northeast-2

availabilityZones: ["ap-northeast-2a", "ap-northeast-2c"]

iam:
  withOIDC: true

managedNodeGroups:
- name: cloud-eks-workers
  desiredCapacity: 1
  iam:
    withAddonPolicies:
      albIngress: true
  instanceTypes: ["c4.large", "c5.large"]
  spot: true
# instanceType: t3.small
# ssh:
#   publicKeyName: "<your key pair name>"
#   https://ap-northeast-2.console.aws.amazon.com/ec2/v2/home?region=ap-northeast-2#KeyPairs:

cloudWatch:
  clusterLogging:
    enableTypes: ["audit", "authenticator", "controllerManager"]
```

정의한 구성 정보 대로 cluster 생성한다.

```
eksctl create cluster -f ./eks-cluster-config.yml
```

4) EKS Cluster 접속 확인

kubectl 을 통해 추가된 node 확인한다.

```
kubectl get nodes
```

4.2.2 Application 배포

1) Database 배포

쿠버네틱스에 DB app을 배포하기 위해 Deployment manifest 파일 작성

mysql-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
spec:
  selector:
    matchLabels:
      app: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - image: cloudacode/mariadb:v1.1.0
          name: mysql
          ports:
            - containerPort: 3306
              name: mysql
```

Deploy the contents of the deployment file: `kubectl apply -f mysql-deployment.yaml`

Display information about the Deployment: `kubectl describe deployment mysql`

정상적으로 mysql 이 Deploy 가 되었다면 flask app 에서 mysql 로 접속을 위해 service 를 deployment 에 매핑한다

mysql-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: mysql
spec:
  ports:
    - port: 3306
  selector:
```

```
app: mysql
clusterIP: None
```

Deploy the contents of the service file: `kubectl apply -f mysql-service.yaml`

배포가 정상적으로 완료가 되면 Pod 정보를 찾을 수 있다

```
kubectl get pods -l app=mysql
```

2) FLASK APP 배포

쿠버네틱스에 Flask app을 배포하기 위해 Deployment manifest 파일 작성

flask-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cloud-flask
  labels:
    app: cloud-flask
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: cloud-flask
  strategy:
    rollingUpdate:
      maxSurge: 20%
      maxUnavailable: 20%
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: cloud-flask
    spec:
      containers:
        - image: cloudacode/cloudflask:v1.1.0
          imagePullPolicy: Always
          name: cloud-flask
          ports:
            - containerPort: 5000
              protocol: TCP
          env:
            - name: DB_USER
              value: root
            - name: DB_PASSWORD
              value: mysecret
            - name: DB_NAME
              value: cloud_user
            - name: DB_HOST
```

```
value: mysql
```

Deploy the contents of the deployment file: `kubectl apply -f flask-deployment.yaml`

Display information about the Deployment: `kubectl describe deployment cloud-flask`

정상적으로 Flask app 이 배포되었다면 외부에서 flask app 으로 접속을 위해 service 를 deployment 에 매핑한다.

flask-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: cloud-flask-svc
spec:
  selector:
    app: cloud-flask
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
  type: LoadBalancer
```

Deploy the contents of the service file: `kubectl apply -f flask-service.yaml`

배포가 정상적으로 완료가 되면 Pod 정보를 찾을 수 있다.

`kubectl get pods -l app=cloud-flask`

flask app 의 경우는 Service 타입을 LB 로 외부 노출을 시켰으므로 다음과 같이 LB Endpoint 를 확인 가능하다.

`kubectl get svc cloud-flask-svc`

+구축된 쿠버네틱스에 Grafana 모니터링 환경 연동

Kubernetes에 ArgoCD 연동 및 Grafana, Prometheus로 모니터링 파이프라인을 구성한다.



1) AWS EKS 구성

4.2에서 실습한 AWS EKS를 사용한다.

2) ArgoCD 연동

ArgoCD CLI, ArgoCD 를 설치한다. ArgoCD Server 를 접속한다.

-In order to access server via URL, need to expose the Argo CD API server. Change the argocd-server service type to LoadBalancer:

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
```

-Check the LB Endpoint

```
kubectl get -n argocd svc argocd-server
```

-Also available to get the external LB endpoint as a raw value:

```
kubectl get -n argocd svc argocd-server --output
jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

-초기 admin 패스워드 확인

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath='{.data.password}' |
base64 -d
```

브라우저를 통해 LB Endpoint 에 접속한다.

3) ArgoCD를 통해 모니터링 App(Prometheus, Grafana) 배포

- 웹 콘솔에 접속후 + New App 클릭하여 신규 애플리케이션(Prometheus) 생성

- GENERAL
- Application Name: prometheus
- Project: default
- Sync Policy: Manual
- SOURCE
- Repo URL: <https://prometheus-community.github.io/helm-charts> HELM
- Chart: prometheus 13.6.0
- DESTINATION
- Cluster URL: <https://kubernetes.default.svc>
- Namespace: default

Create 진행 후, 화면을 새로고침 하면 다음과 같이 앱이 하나 등록되어 Sync 가 아직 되지 않은 OutOfSync 상태로 확인된다. Sync 정책을 Manual 로 하였기 때문에 초기에 OutOfSync 상태는 정상이다. **SYNC** 수행 후, 레포지토리 URL 이 올바르게 되어 있다면 문제없이 sync 가 완료된다.

-웹 콘솔에서 + New App 클릭하여 신규 애플리케이션(Grafana) 생성

- GENERAL
- Application Name: grafana
- Project: default
- Sync Policy: Manual
- SOURCE
- Repo URL: <https://grafana.github.io/helm-charts> HELM
- Chart: prometheus 6.6.2
- DESTINATION
- Cluster URL: <https://kubernetes.default.svc>
- Namespace: default

아래 HELM 변수값 탭에서 service.type 검색 후 값을 LoadBalancer 로 변경한다. Create 와 SYNC 수행 후 정상적으로 애플리케이션이 만들어졌다면 admin password 를 조회한다. Endpoint 확인을 위해 ArgoCD 화면에서 Grafana 선택 후 상세페이지 Service(화면에서는 svc grafana)의 Hostnames 항목에 앱에 접속 가능한 LB URL 이 생성되었는지 확인 후 웹 브라우저를 통해 해당 URL 로 접속하여 Grafana 에 접근한다.

- Monitoring Dashboard 구성

왼쪽 탭 Configuration 에서 Data Sources 선택 후 위에서 조회한 Prometheus Cluster IP

를 HTTP URL로 지정한다. SAVE & TEST 하여 Data Source 등록한다. 왼쪽 탭 + Import
-> Upload Json File: kubernetes-cluster-prometheus_rev1.json 업로드한다.

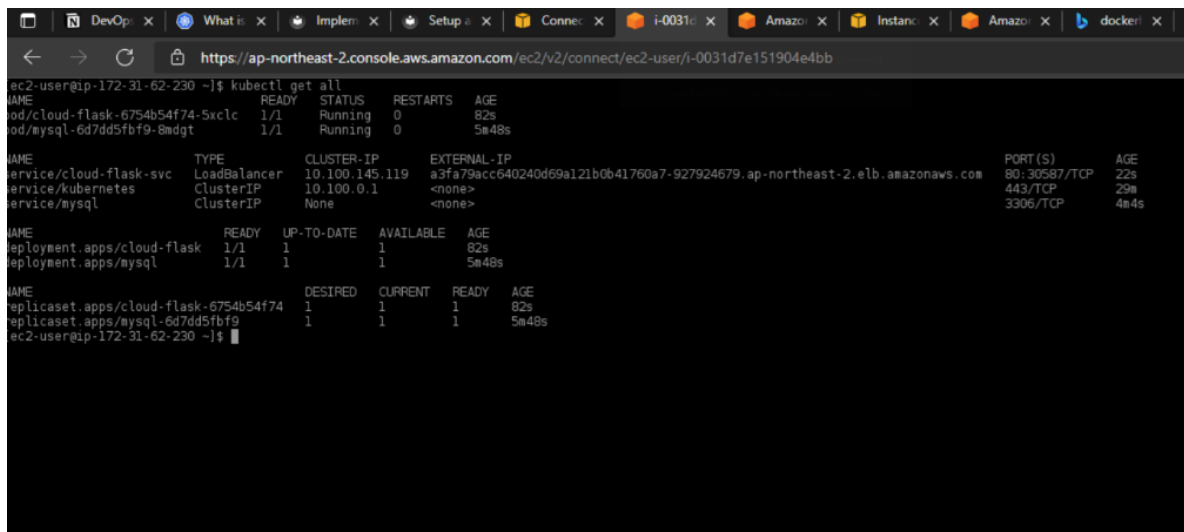
Options - prometheus: Prometheus

4) EKS 리소스를 정리

```
eksctl delete cluster --region=ap-northeast-2 --name=<your eks cluster name>
```

4.3 실습 결과

1) 쿠버네티스에 어플리케이션 배포해서 접근 확인한 화면

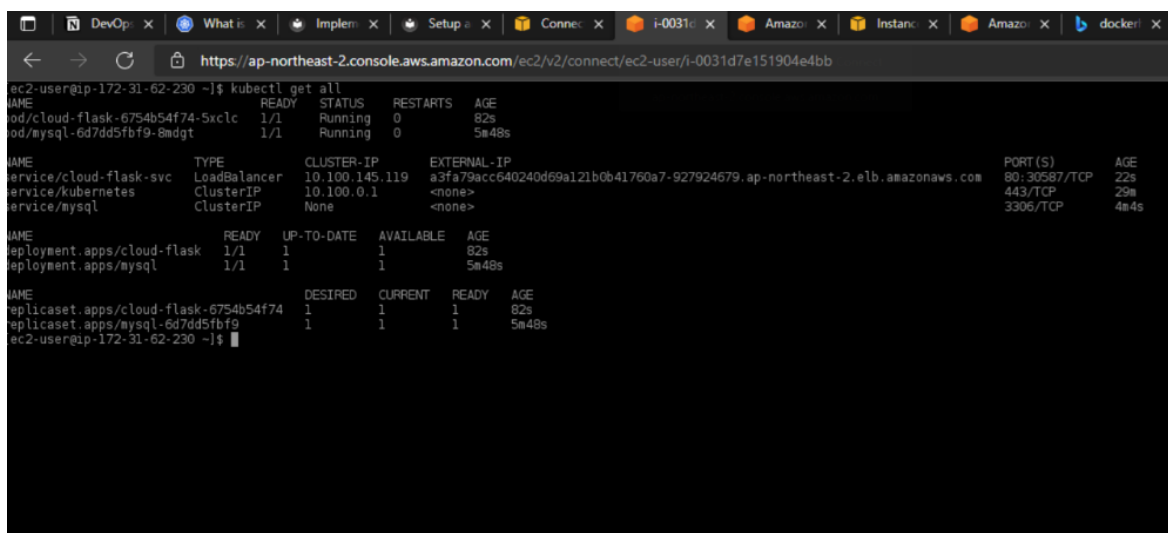


```
ec2-user@ip-172-31-62-230 ~]$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/cloud-flask-6754b54f74-5xclc    1/1     Running   0           82s
pod/mysql-6d7dd5fbf9-8mdgt         1/1     Running   0           5m48s

NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/cloud-flask-svc             LoadBalancer        10.100.145.119  a3fa79acc640240d69a121b0b41760a7-927924679.ap-northeast-2.elb.amazonaws.com  80:30587/TCP     22s
service/kubernetes                  ClusterIP            10.100.0.1      <none>           443/TCP          29m
service/mysql                       ClusterIP            10.100.0.1      <none>           3306/TCP         4m4s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cloud-flask         1/1     1             1           82s
deployment.apps/mysql               1/1     1             1           5m48s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/cloud-flask-6754b54f74 1         1         1       82s
replicaset.apps/mysql-6d7dd5fbf9       1         1         1       5m48s
ec2-user@ip-172-31-62-230 ~]$
```



```
ec2-user@ip-172-31-62-230 ~]$ kubectl get all
NAME                                READY   STATUS    RESTARTS   AGE
pod/cloud-flask-6754b54f74-5xclc    1/1     Running   0           82s
pod/mysql-6d7dd5fbf9-8mdgt         1/1     Running   0           5m48s

NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/cloud-flask-svc             LoadBalancer        10.100.145.119  a3fa79acc640240d69a121b0b41760a7-927924679.ap-northeast-2.elb.amazonaws.com  80:30587/TCP     22s
service/kubernetes                  ClusterIP            10.100.0.1      <none>           443/TCP          29m
service/mysql                       ClusterIP            10.100.0.1      <none>           3306/TCP         4m4s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cloud-flask         1/1     1             1           82s
deployment.apps/mysql               1/1     1             1           5m48s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/cloud-flask-6754b54f74 1         1         1       82s
replicaset.apps/mysql-6d7dd5fbf9       1         1         1       5m48s
ec2-user@ip-172-31-62-230 ~]$
```

```
af39d060da24a45bab5397c71a8 x +
< > ↻ ⚠ 안전하지 않음 | af39d060da24a45bab5397c71a8511b7-511147149.ap-northeast-2.elb.amazonaws.com/user

[
  {
    "user_bio": "mento",
    "user_email": "cloudacode@gmail.com",
    "user_id": 1,
    "user_name": "kc chang"
  }
]
```

```
Flask app - CRUD x +
< > ↻ ⚠ 안전하지 않음 | a3fa79acc640240d69a121b0b41760a7-927924679.ap-northeast-2.elb.amazonaws.com
```

Hello Flask

2) ArgoCD를 연동 및 모니터링 파이프라인 연동

