ASC KHU Hands on

Contents

Container ?

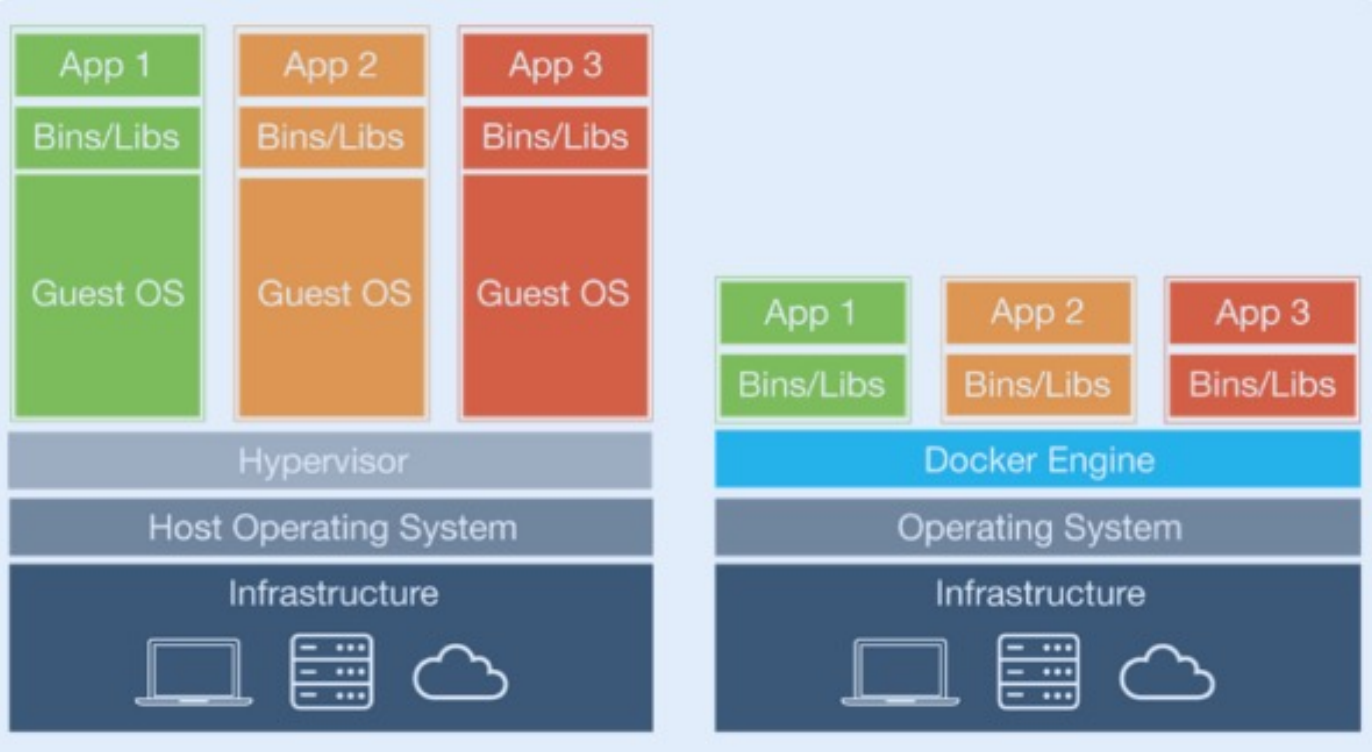
OS 수준에서 가상화

OS 커널 공유

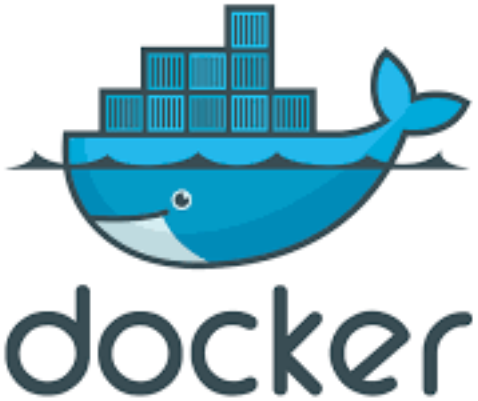
경량 패키지

프로세스 격리

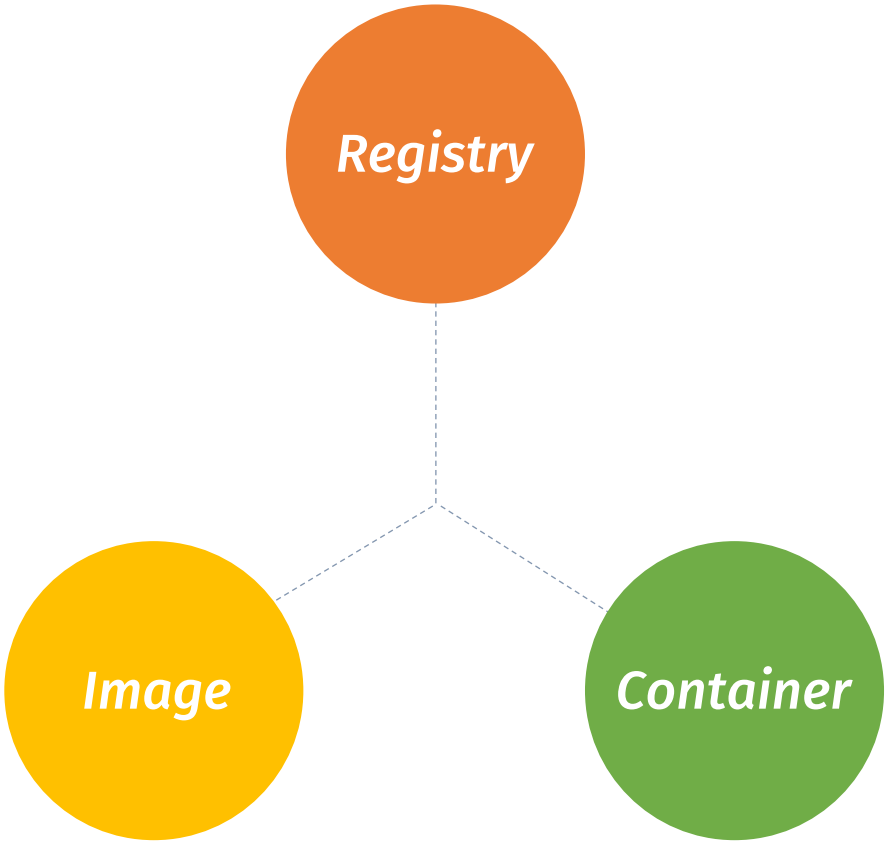
Container ?


App 1
Bins/Libs
Guest OS
App 2
Bins/Libs
Guest OS
App 3
Bins/Libs
Guest OS
Hypervisor
Host Operating System
Infrastructure
App 1
Bins/Libs
App 2
Bins/Libs
App 3
Bins/Libs
Docker Engine
Operating System
Infrastructure

Docker ?



컨테이너를 이미지화해서
간편한 이식 가능한 패키지로
패키징하는 컨테이너 플랫폼



Namespace & cgroups

name space

MNT namespace - 파일 시스템의 마운트 지점을 분할 격리

PID namespace - process ID를 분할 관리

NET namespace - 네트워크 인터페이스, iptables 등의 네트워크 리소스와 관련된 정보를 분할

IPC namespace – IPC 리소스를 분할 격리

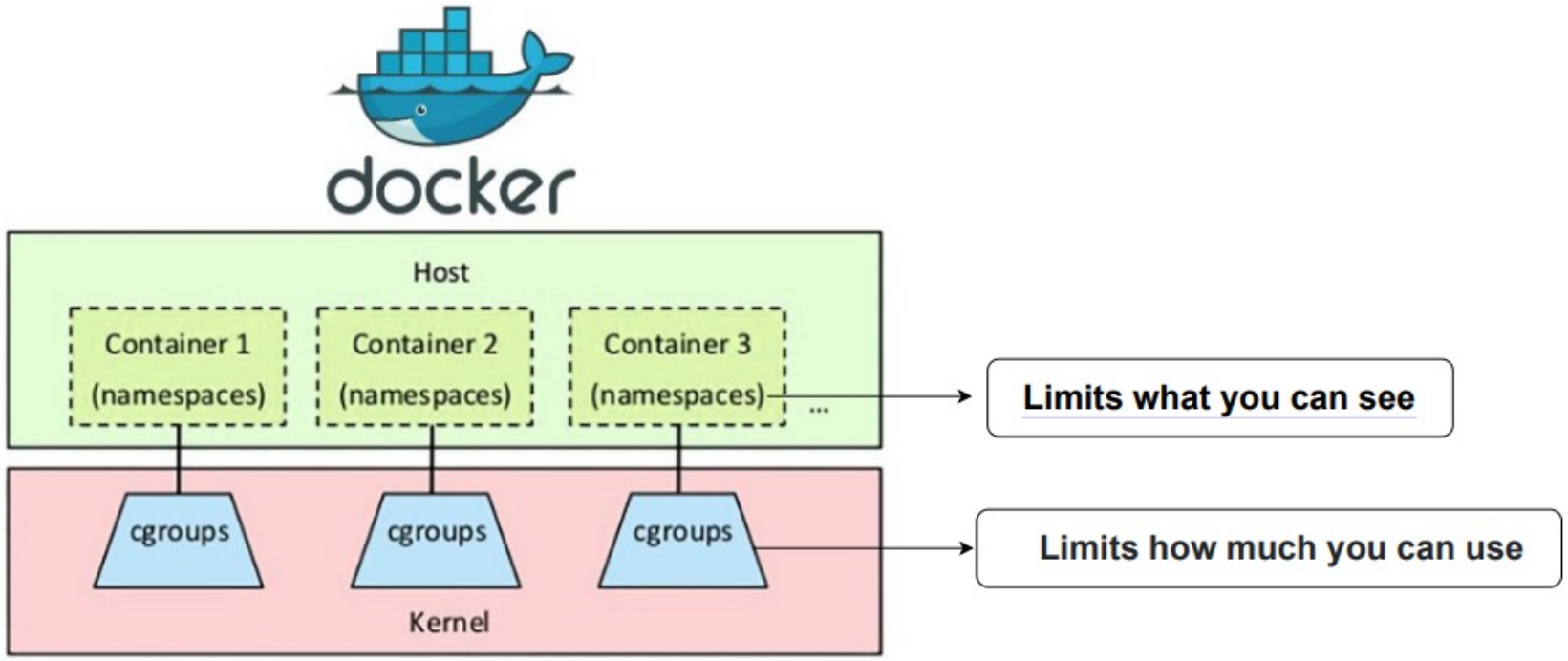
UTS namespace - 호스트와 도메인 네임 별로 격리

USER namespace - user와 group id를 분할하고 격리

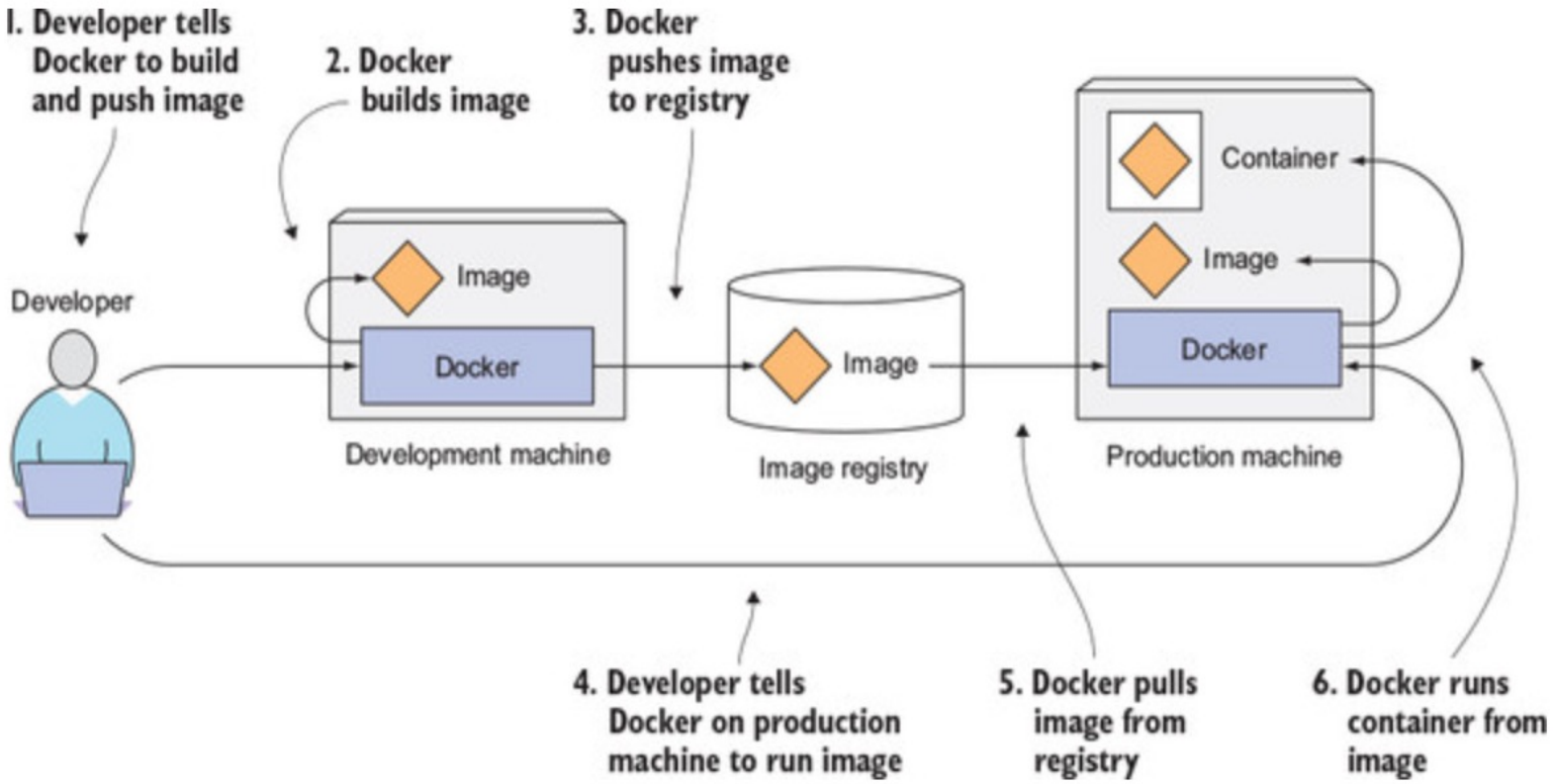
cgroups

물리적 자원을 어디까지 허용할지에 대한 리소스 관리 역할

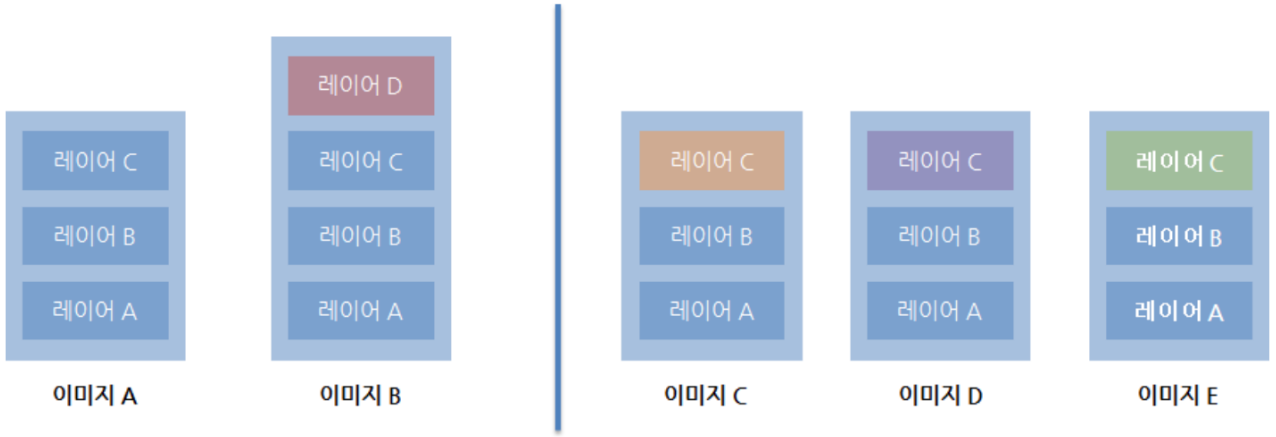
Namespace & cgroups



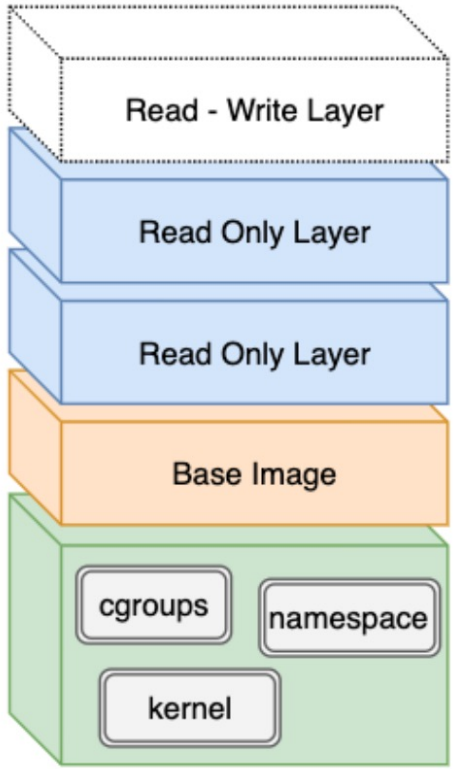
Docker


1. Developer tells Docker to build and push image
2. Docker builds image
3. Docker pushes image to registry
Developer
Image
Docker
Development machine
Image
Image registry
Container
Image
Docker
Production machine
4. Developer tells Docker on production machine to run image
5. Docker pulls image from registry
6. Docker runs container from image

Docker Image Layer



이미지는 여러 개의 **Layer**로 구성
=> 같은 **Layer**가 **local**에 존재하면
저장 시 다른 **Layer**만 **pull**



컨테이너 실행 시 읽기 전용 위에
새로운 쓰기 가능한 레이어가 생성

Dockerfile

```dockerfile
# 베이스 이미지 선택
FROM ubuntu:latest

# 작성자 정보
LABEL maintainer="ASC dusdjhyeon <dusdj0813@khu.ac.kr>"

# ubuntu 패키지 메니저 업데이트 & nginx 설치
# -y를 통해 설치 도중 발생하는 확인메시지에 자동으로 yes 응답
RUN apt-get update
RUN apt-get isntall -y nginx

# this is a ubuntu container라는 메시지 출력
RUN echo "this is a ubuntu container"

# 현재 디렉토리를 /etc/nginx로 설정
# 이 이후 실행되는 명령어는 해당 디렉토리에서 실행
WORKDIR /etc/nginx

# 컨테이너 시작 시 실행할 명령어 (nginx를 백그라운드에서 실행)
CMD ["nginx", "-g", "daemon off;"]

# 포트 80 노출
EXPOSE 80
```

FROM – 베이스 이미지 설정

COPY – 파일 또는 디렉토리를 컨테이너에 복사

CMD – 컨테이너가 **default**로 수행할 동작 기술

EXPOSE – 노출할 컨테이너 포트

WORKDIR – 컨테이너 상의 작업 디렉토리 전환

ENTRYPOINT – 컨테이너가 수행될 때 실행할 명령 기술

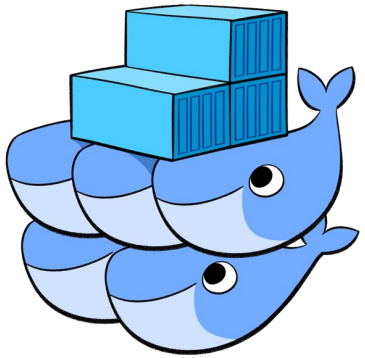
RUN – 레이어 위에 실행할 명령으로 주로 패키지 설치 시 사용

Docker-compose


```yaml
services:
  server:
    image: eeap/server:v1
    ports:
      - "12345:12345"
  client:
    depends_on:
      - server
    image: eeap/client:v1
```

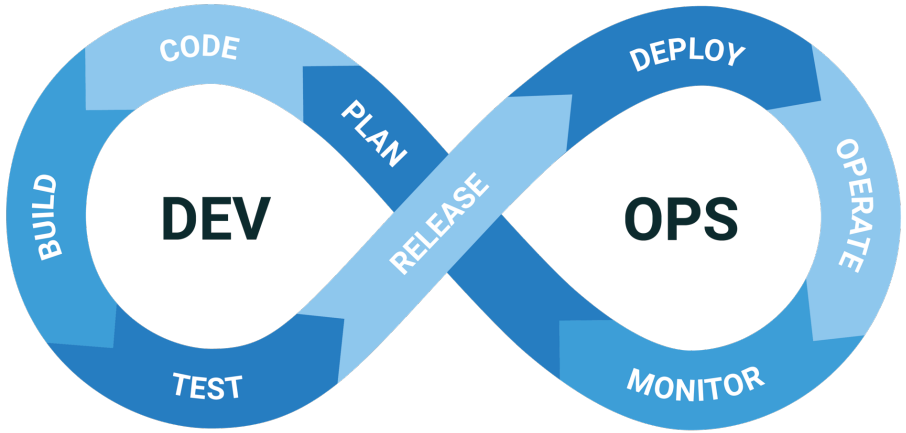
다중 컨테이너를 **yaml**파일을 통해 배포

Container Orchestration



Docker Swarm

CI/CD ?



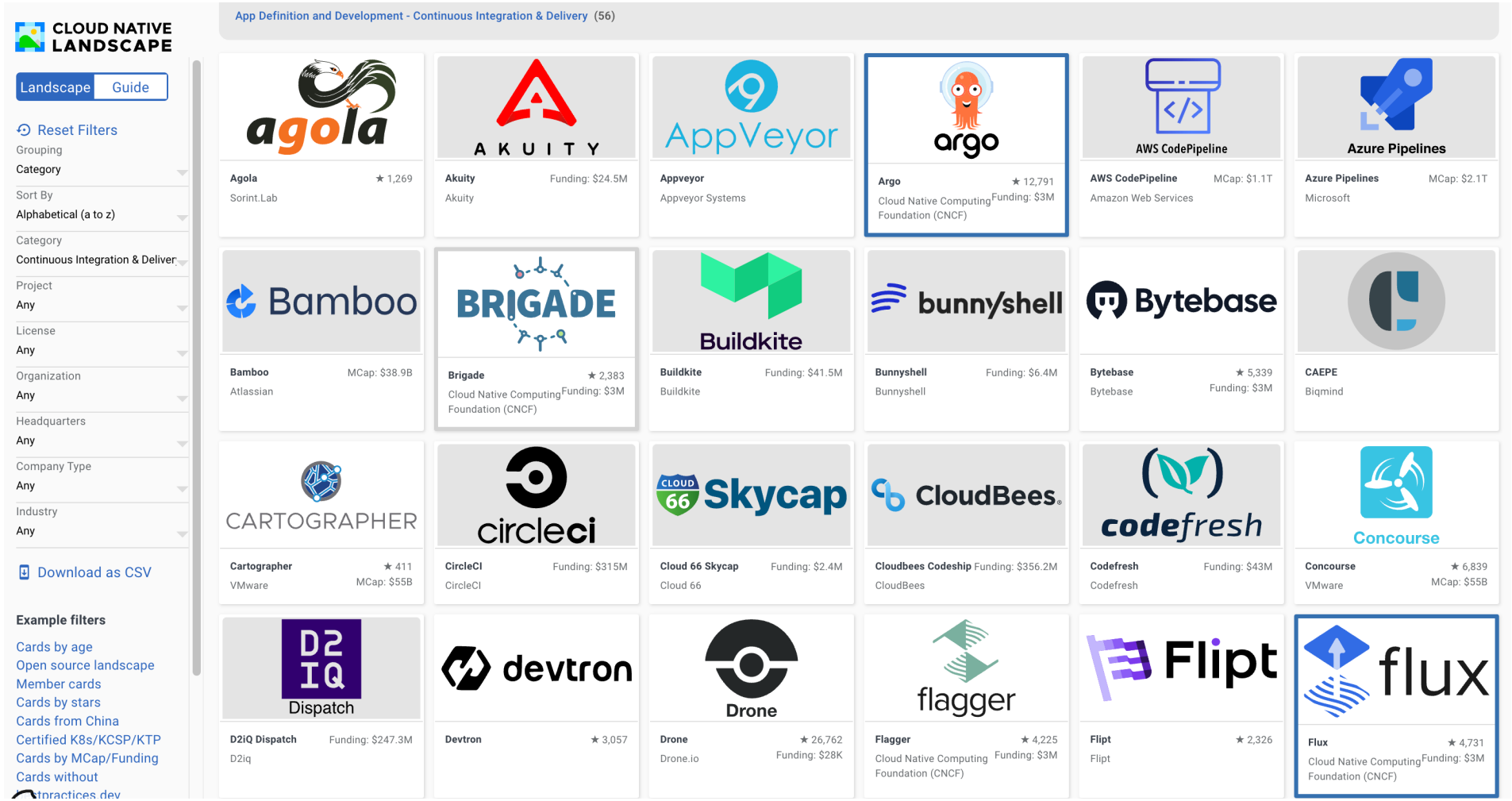
CI (Continuos Integration) 지속적 통합

=> build 자동화, 유닛 및 통합 테스트

CD (Continuos Delivery) 지속적 배포

=> build된 코드를 **release**하고 **production**으로 **deploy**

CI/CD ?



Jenkins

GitHub Actions

CI/CD ?

AWS CodeCommit

Securely host highly scalable private Git repositories and collaborate on code

AWS CodeDeploy

Automate deployments for developers to securely and quickly release new features

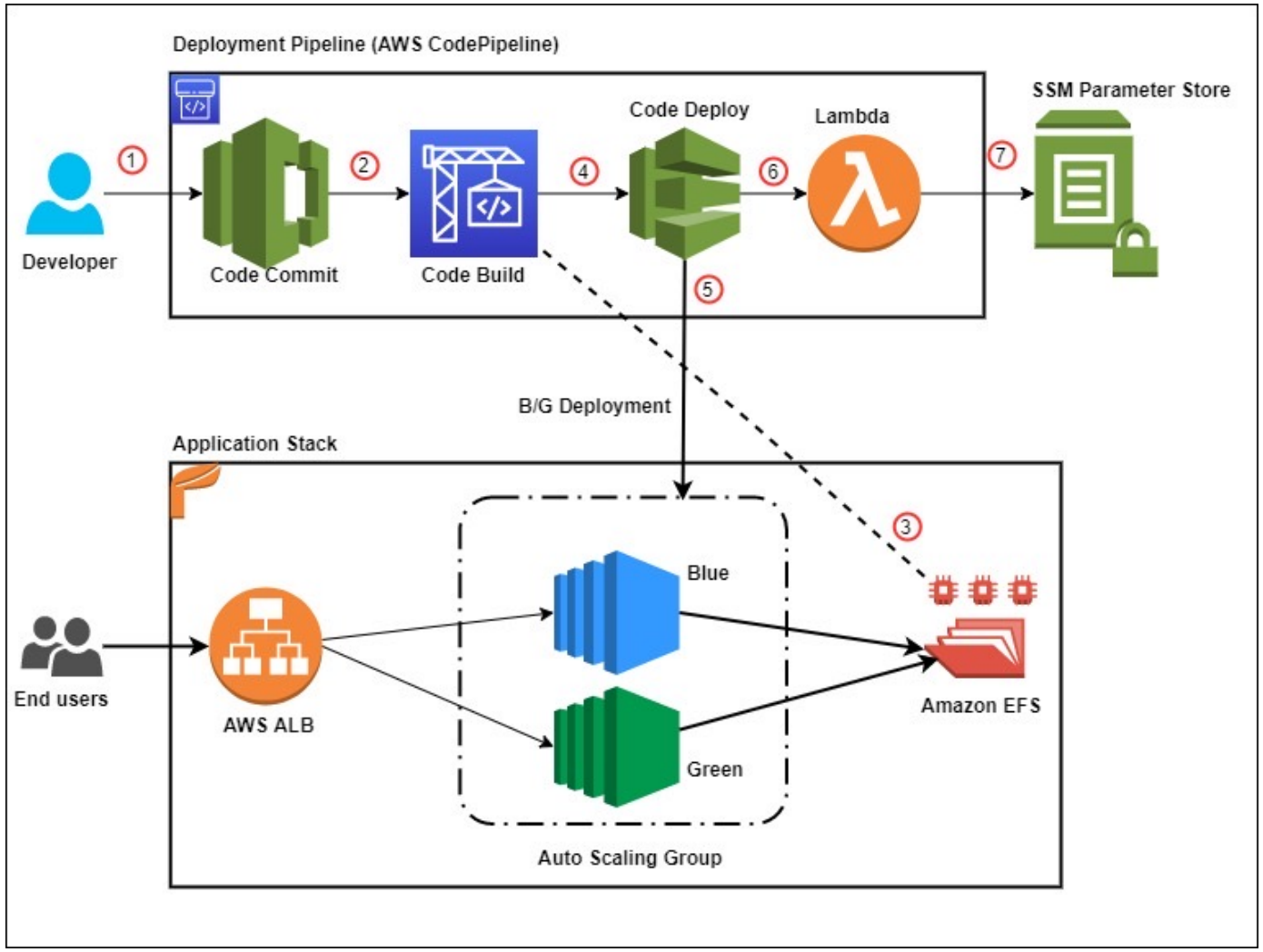
AWS CodeBuild

Compile source code, run tests, and generate artifacts without managing build servers

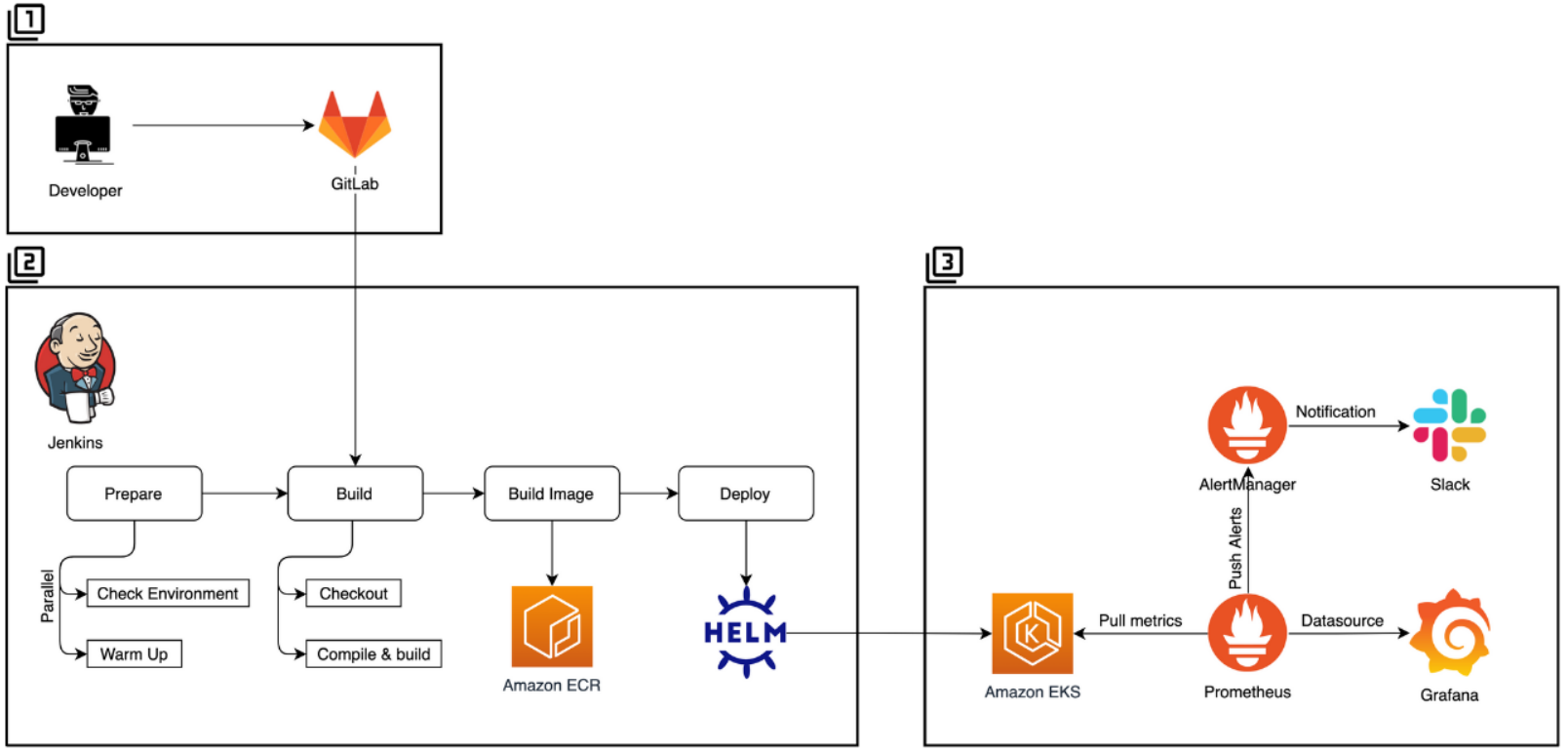
AWS CodePipeline

Example pipeline

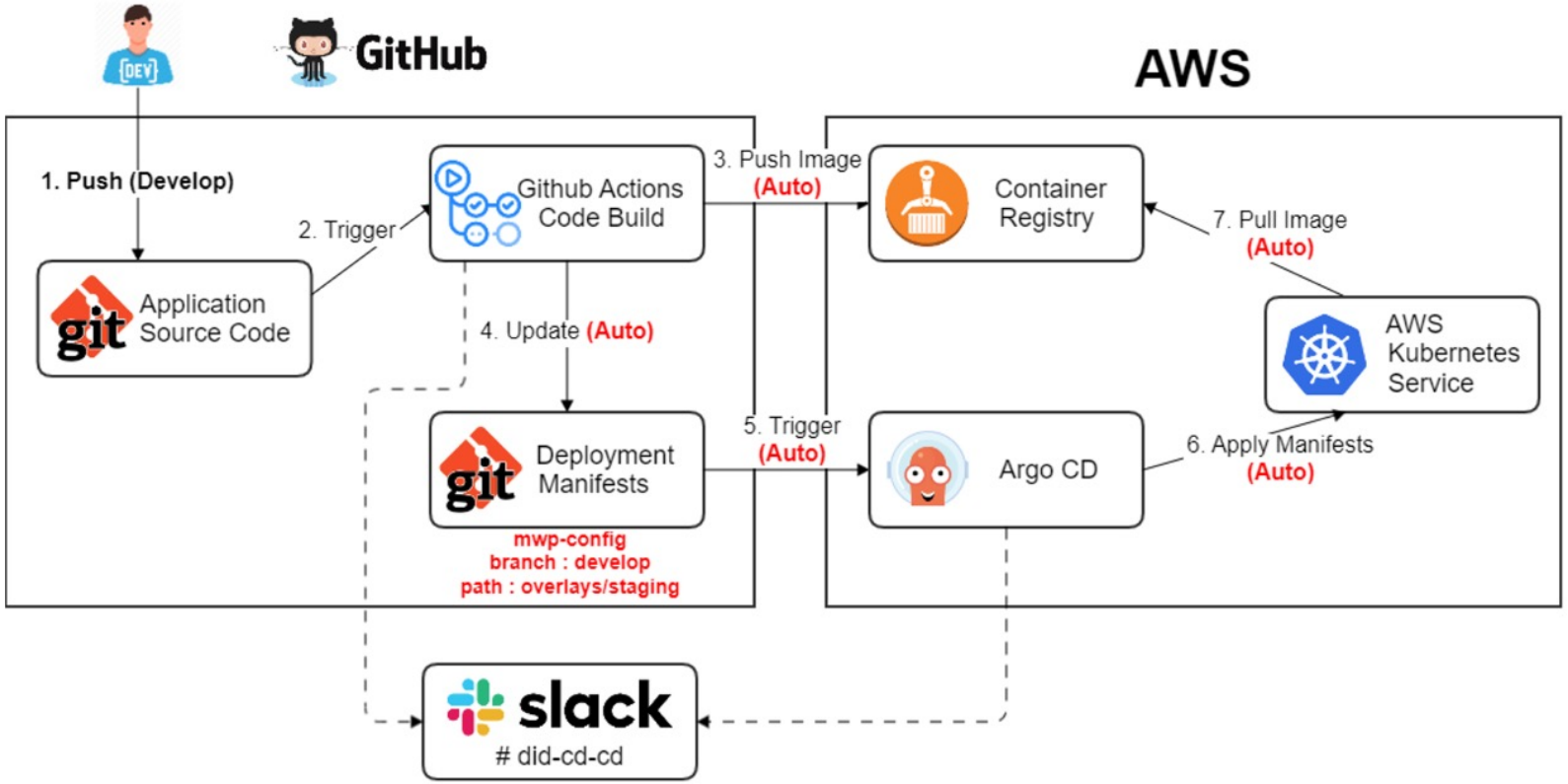
CI/CD 실제 사례


Deployment Pipeline (AWS CodePipeline)
SSM Parameter Store
Code Deploy
Lambda
Developer
Code Commit
Code Build
B/G Deployment
Application Stack
Blue
End users
AWS ALB
Amazon EFS
Green
Auto Scaling Group

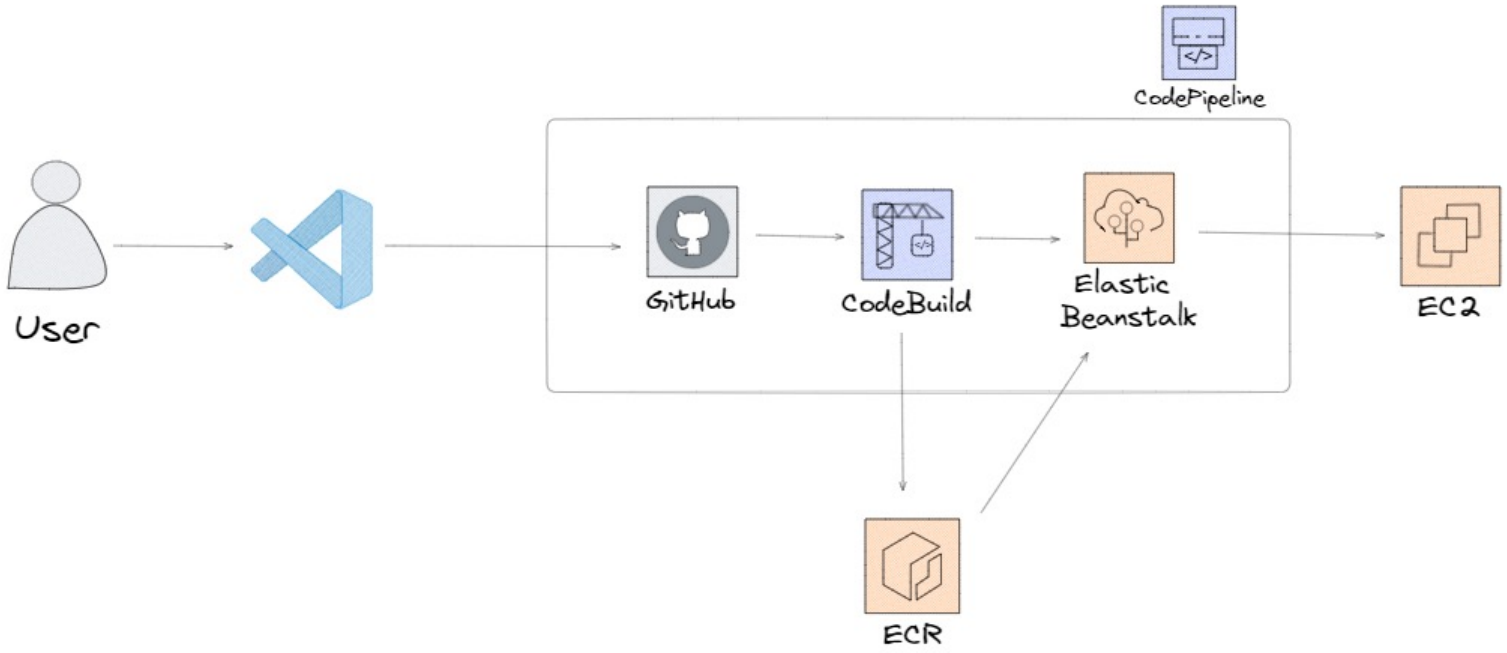
CI/CD 실제 사례


1
Developer
GitLab
2
Jenkins
Prepare
Build
Build Image
Deploy
Parallel
Check Environment
Warm Up
Checkout
Compile & build
Amazon ECR
HELM
3
Notification
AlertManager
Slack
Push Alerts
Pull metrics
Datasource
Amazon EKS
Prometheus
Grafana

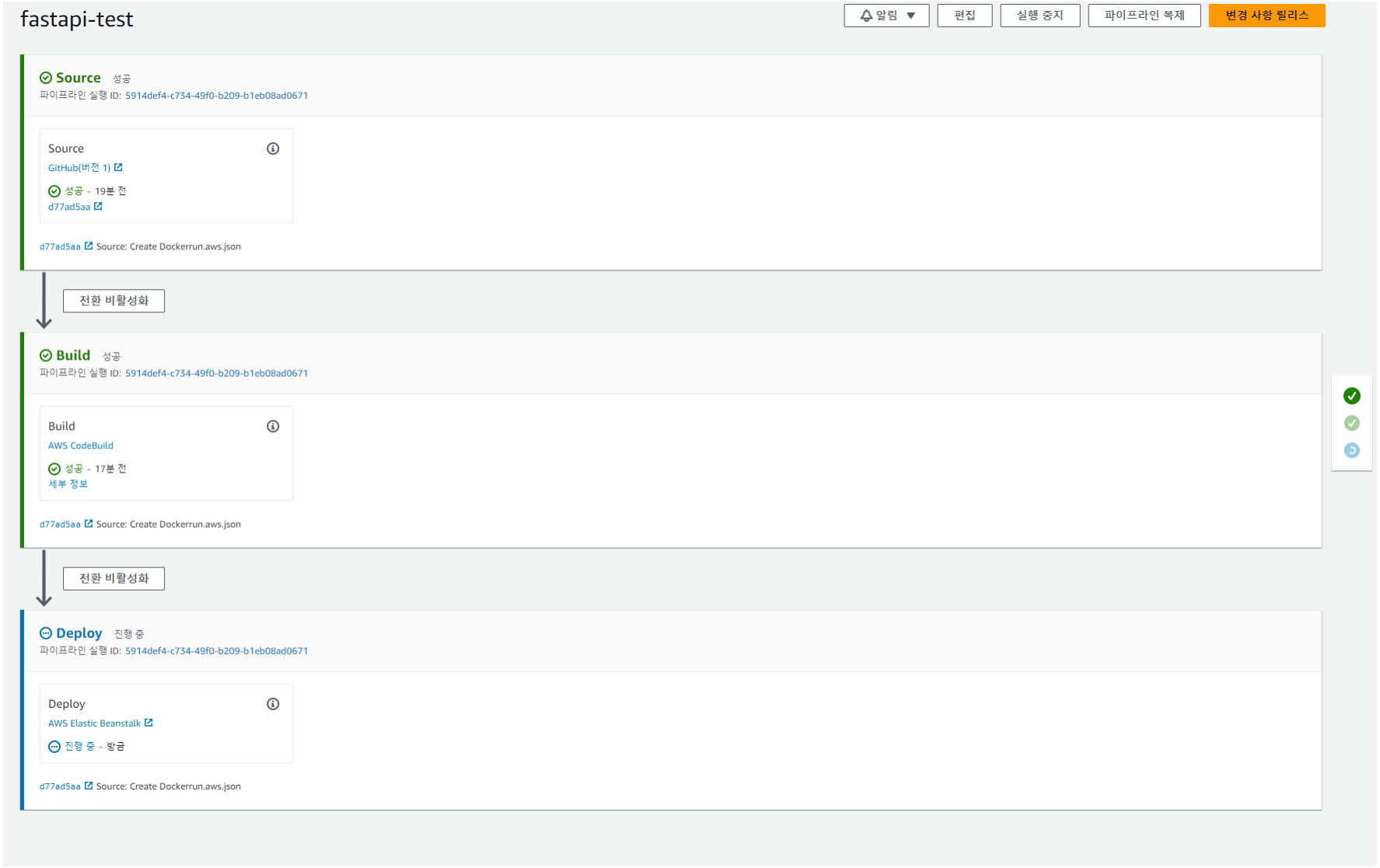
CI/CD 실제 사례


GitHub
AWS
1. Push (Develop)
2. Trigger
Github Actions
Code Build
3. Push Image
(Auto)
Container
Registry
7. Pull Image
(Auto)
Application
Source Code
4. Update (Auto)
AWS
Kubernetes
Service
Deployment
Manifests
5. Trigger
(Auto)
Argo CD
6. Apply Manifests
(Auto)
mwp-config
branch : develop
path : overlays/staging
slack
did-cd-cd

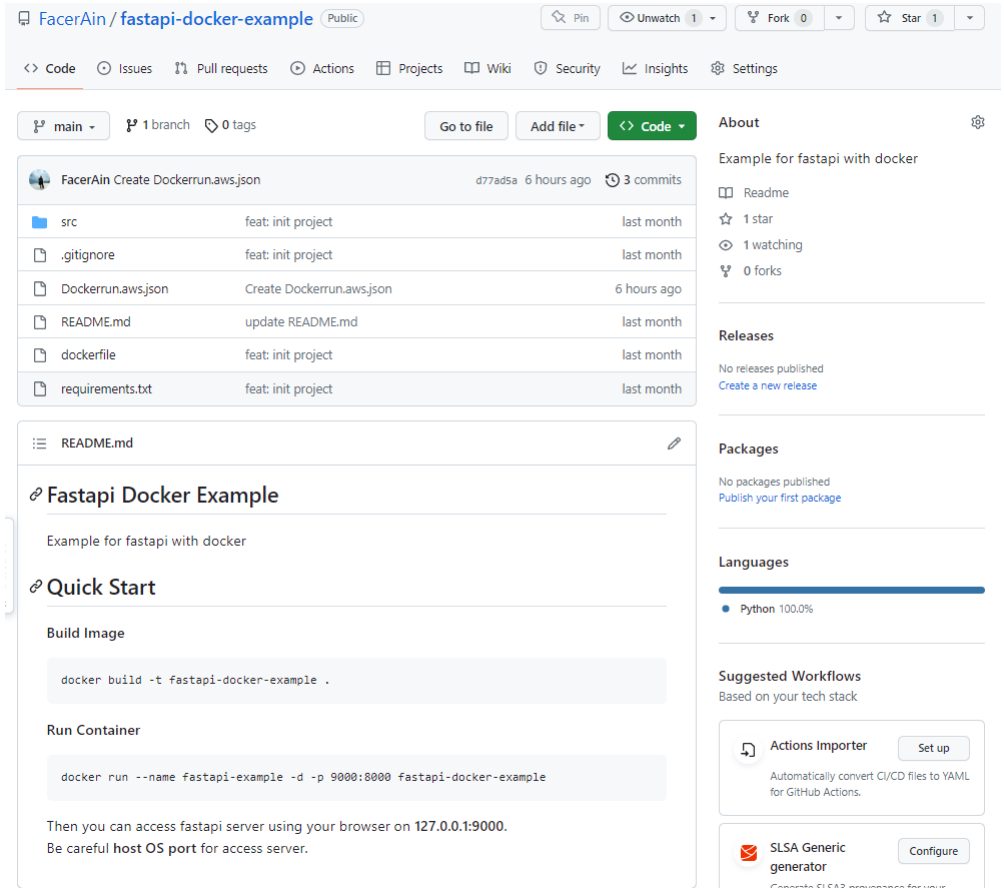
AWS 서비스로 한번 만들어보자



CodePipeline 살펴보기



Source: Github



FacerAin / fastapi-docker-example Public

Pin | Unwatch 1 | Fork 0 | Star 1

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 1 branch | 0 tags | Go to file | Add file | Code

FacerAin Create Dockerrun.aws.json d77ad5a 6 hours ago 3 commits

src	feat: init project	last month
.gitignore	feat: init project	last month
Dockerrun.aws.json	Create Dockerrun.aws.json	6 hours ago
README.md	update README.md	last month
dockerfile	feat: init project	last month
requirements.txt	feat: init project	last month

README.md

Fastapi Docker Example

Example for fastapi with docker

Quick Start

Build Image

```
docker build -t fastapi-docker-example .
```

Run Container

```
docker run --name fastapi-example -d -p 9000:8000 fastapi-docker-example
```

Then you can access fastapi server using your browser on **127.0.0.1:9000**.
Be careful **host OS port** for access server.

About

Example for fastapi with docker

Readme
1 star
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

Python 100.0%

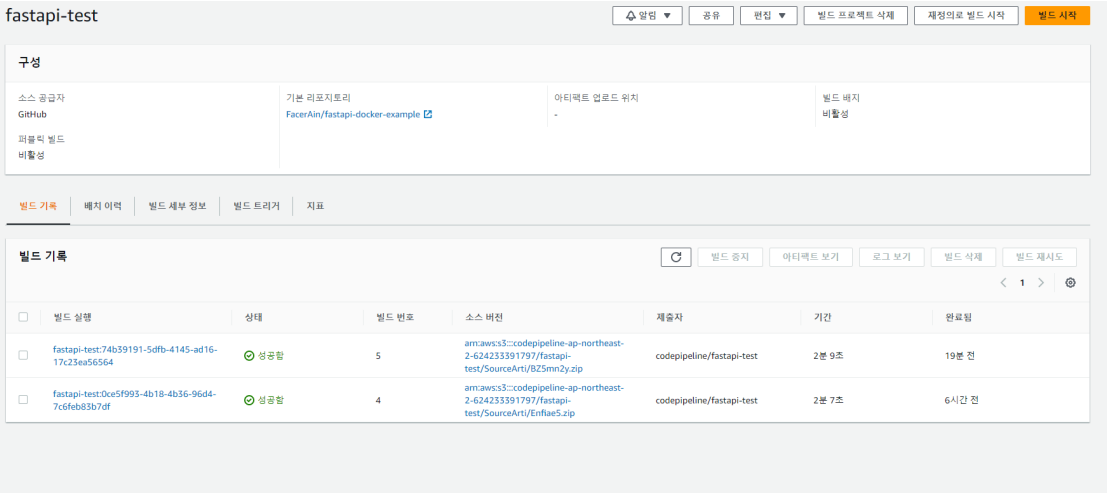
Suggested Workflows

Based on your tech stack

Actions Importer — Set up
Automatically convert CI/CD files to YAML for GitHub Actions.

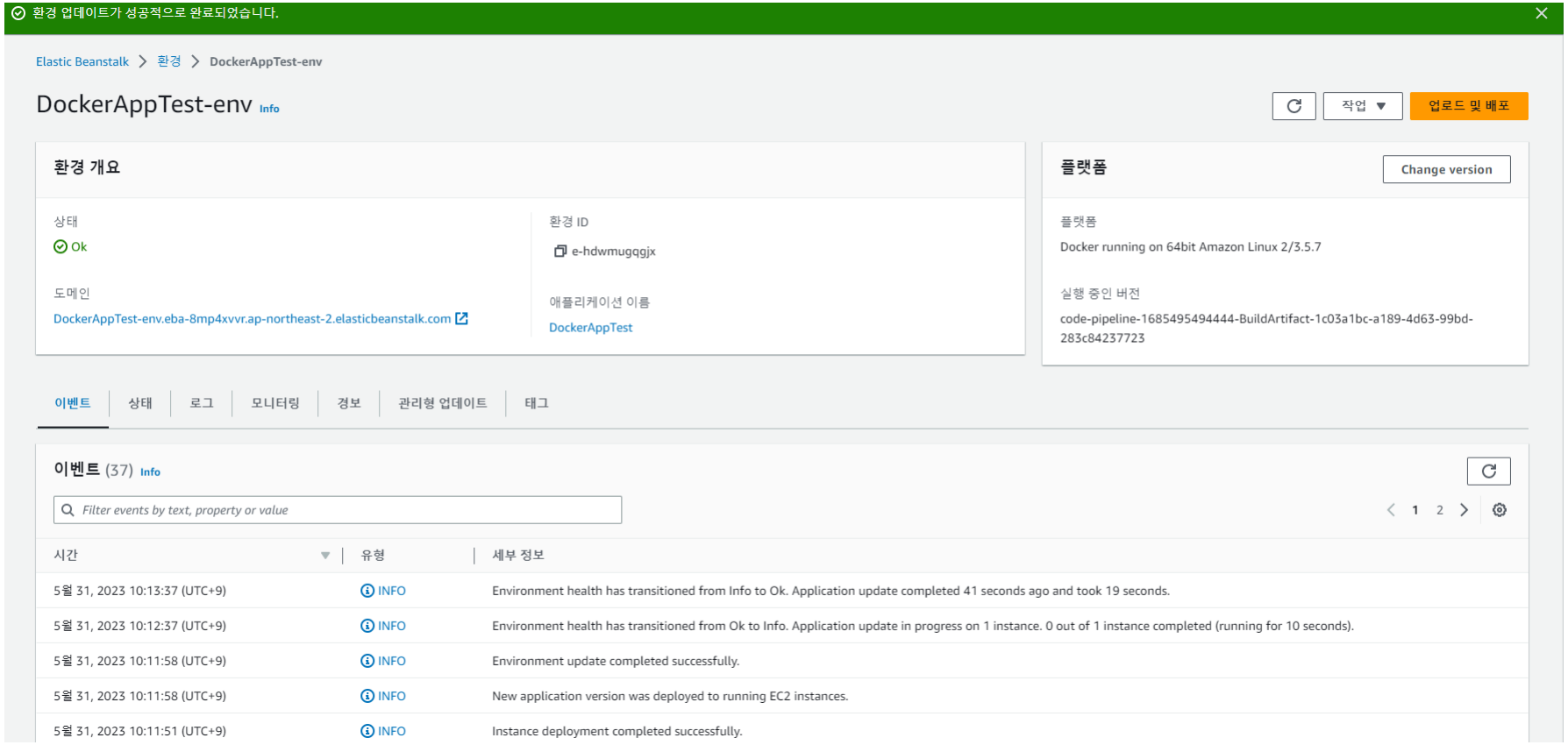
SLSA Generic generator — Configure

Build: CodeBuild




```yaml
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws ecr get-login-password --region ap-northeast-2 | docker login --username AWS --password-stdin
067220025245.dkr.ecr.ap-northeast-2.amazonaws.com

  build:
    commands:
      - echo Building the Docker image...
      - docker build -t fastapi-test .
      - docker tag fastapi-test:latest 067220025245.dkr.ecr.ap-northeast-2.amazonaws.com/fastapi-
test:latest


  post_build:
    commands:
      - docker push 067220025245.dkr.ecr.ap-northeast-2.amazonaws.com/fastapi-test:latest

artifacts:
  files:
  - Dockerrun.aws.json
```

Deploy: Beanstalk



무엇을 더 해볼 수 있을까요?

1. 구성 요소 추가 / 변경 **(Beanstalk -> EKS)**
2. 관리자 리뷰 단계 추가
3. 모니터링 시스템 추가 **(CloudWatch)**

.
.
.

CI/CD ?

GitHub Actions

Github를 이용해서 빌드, 테스트 및 배포 파이프라인을 자동화할 수 있는 **CI/CD** 플랫폼

Workflows – 하나 이상의 작업을 실행하는 구성 가능한 자동화 프로세스

Events – **workflow** 실행을 트리거하는 **repository**의 특정 활동

Jobs – **workflow**에서 하나의 처리 단위

Steps – 작업의 구성 단위로, **Job**은 하나 이상의 **step**으로 구성

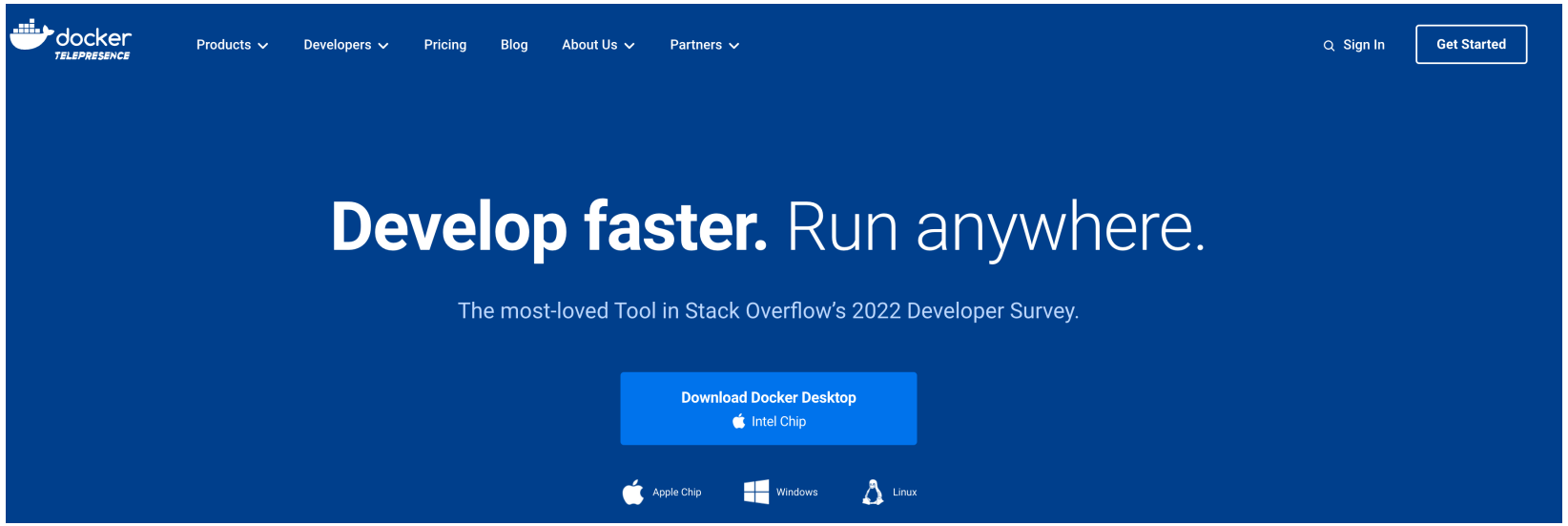
Actions – 자주 반복되는 작업을 용이하도록 하기 위한 작업 공유 메커니즘

Runners – 트리거될 때 **workflow**를 실행하는 서버

Docker 실습

Docker image를 만들고 **DockerHub**에 올려서 **Docker compose** 까지 진행하기**!**

1. Docker Desktop 설치



Mac 같은 경우엔 [docker homepage](#) 들어가서 다운로드

Window는 해당 [가이드](#) 참고

Docker 실습

Docker image를 만들고 **DockerHub**에 올려서 **Docker compose** 까지 진행하기**!**

2. Dockerfile 생성 후 **build & run**

```
FROM ubuntu:latest
LABEL MAINTAINER="sumink0903@gmail.com"

RUN apt-get update
RUN apt-get install -y nginx
RUN echo "nginx container"
WORKDIR /etc/nginx
CMD [ "nginx","-g","daemon off;" ]
EXPOSE 80
```

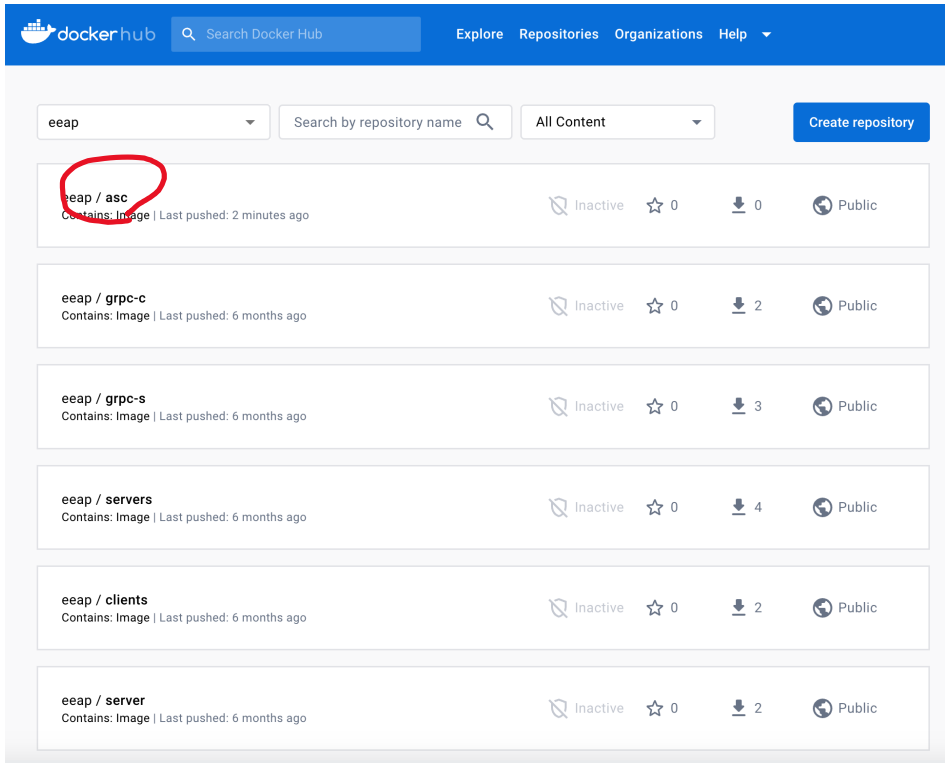
```
~/Desktop/asc-githubAction git:(main) (0.456s)
docker build -t asc:v1 .

[+] Building 0.1s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 242B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/ubuntu:latest
 => [1/5] FROM docker.io/library/ubuntu:latest
 => CACHED [2/5] RUN apt-get update
 => CACHED [3/5] RUN apt-get install -y nginx
 => CACHED [4/5] RUN echo "nginx container"
 => CACHED [5/5] WORKDIR /etc/nginx
 => exporting to image
 => => exporting layers
 => => writing image sha256:38d4920b122ef488cab82c4f72f585d082955de70b796b9313b4a8579b50cf4
 => => naming to docker.io/library/asc:v1

~/Desktop/asc-githubAction git:(main) (0.457s)
docker run -p 8080:80 --name practice -d --rm asc:v1
508dd7fb9107fef3e72a0380e721f0f9e2700be816b6ed57f76014957dd16338
```

Docker 실습

Docker image를 만들고 **DockerHub**에 올려서 **Docker compose** 까지 진행하기**!**

3. dockerHub에 login 후 이미지 push 후 pull 실행

```
~/Desktop/asc-githubAction git:(main) (2.316s)
docker login
Authenticating with existing credentials...
Login Succeeded

~/Desktop/asc-githubAction git:(main) (0.204s)
docker tag asc:v1 eeap/asc:v1

~/Desktop/asc-githubAction git:(main) (28.739s)
docker push eeap/asc:v1
The push refers to repository [docker.io/eeap/asc]
5f70bf18a086: Mounted from spack/centos7
3a0704d1976d: Pushed
f71597669906: Pushed
98fcde1c6e0f: Mounted from library/ubuntu
v1: digest: sha256:da35f8c8bc687cc4f037053eb88509b15c36c0b9d24c3af066e4797e432b7734 size: 1365

~/Desktop/asc-githubAction git:(main) (0.124s)
docker rmi eeap/asc:v1
Untagged: eeap/asc:v1
Untagged: eeap/asc@sha256:da35f8c8bc687cc4f037053eb88509b15c36c0b9d24c3af066e4797e432b7734
```




```
~/Desktop/asc-githubAction git:(main) (3.941s)
docker run -p 8080:80 --name practice -d --rm eeap/asc:v1
Unable to find image 'eeap/asc:v1' locally
v1: Pulling from eeap/asc
Digest: sha256:da35f8c8bc687cc4f037053eb88509b15c36c0b9d24c3af066e4797e432b7734
Status: Downloaded newer image for eeap/asc:v1
028204ac5a8b186aa896e507848596eaaf3bf60c8d2c14e9b59e0b47faca1a1d
```

Docker 실습

Docker image를 만들고 **DockerHub**에 올려서 **Docker compose** 까지 진행하기!

4. docker-compose.yml 파일 생성 후 **up & down** 진행

```yaml
services:
  nginx:
    image: eeap/asc:v1
    ports:
      - "8080:80"
```

```
docker-compose up -d

[+] Running 1/1
 ⠿ Container asc-githubaction-nginx-1  Started
```

```
~/Desktop/asc-githubAction git:(main) (0.328s)
docker-compose config
name: asc-githubaction
services:
  nginx:
    image: eeap/asc:v1
    networks:
      default: null
    ports:
    - mode: ingress
      target: 80
      published: "8080"
      protocol: tcp
networks:
  default:
    name: asc-githubaction_default
```

```
~/Desktop/asc-githubAction git:(main) (0.589s)
docker-compose down

[+] Running 2/2
 ⠿ Container asc-githubaction-nginx-1  Removed
 ⠿ Network asc-githubaction_default    Removed
```

CI/CD 실습

Github Action을 이용한 **AWS ECR**에 이미지 올리기

```yaml
name: Image push Amazon ECR

# Controls when the workflow will run
on:
  # Triggers the workflow on push or pull request events but only for the "main" branch
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

env:
  AWS_REGION: us-east-1
# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a single job called "build"
  deploy:
    name: Deploy image
    # The type of runner that the job will run on
    runs-on: ubuntu-latest
    environment: production

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
      - name: Checkout
        uses: actions/checkout@v3

      - name: Config AWS credentials
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ${{ env.AWS_REGION }}
```

```yaml
      - name: Login to Amazon ECR
        id: login-ecr-public
        uses: aws-actions/amazon-ecr-login@v1
        with:
          registry-type: public

      - name: Build, tag, and push image to Amazon ECR
        id: build-image
        env:
          REGISTRY: ${{ steps.login-ecr-public.outputs.registry }}
          REGISTRY_ALIAS: your-registry-alias
          REPOSITORY: your-repository-name
          IMAGE_TAG: ${{ github.sha }}
        run: |
          # Build a docker container and
          # push it to ECR so that it can
          # be deployed to ECS.
          docker build -t $REGISTRY/$REGISTRY_ALIAS/$REPOSITORY:$IMAGE_TAG .
          docker push $REGISTRY/$REGISTRY_ALIAS/$REPOSITORY:$IMAGE_TAG
          echo "::set-output name=image::$REGISTRY/$REGISTRY_ALIAS/$REPOSITORY:$IMAGE_TAG"
```

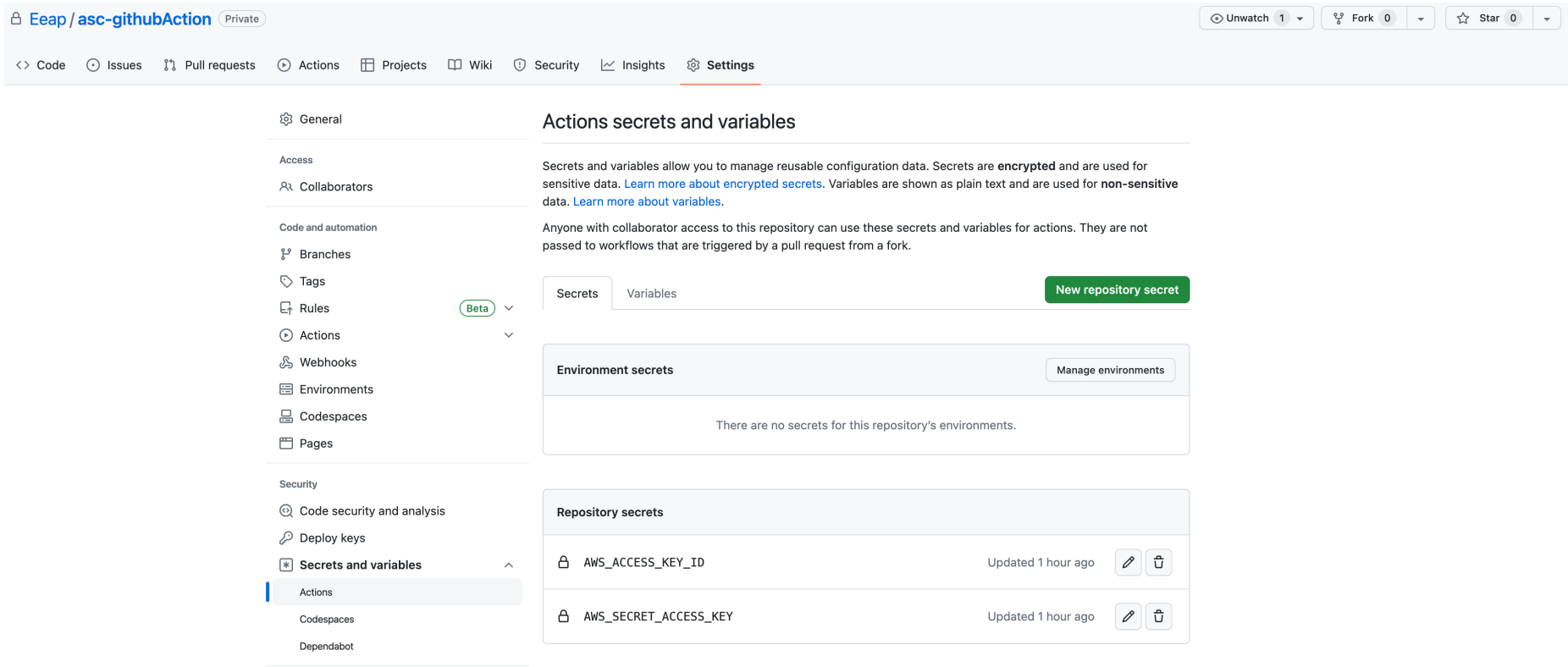
https://github.com/Eeap/asc-githubAction

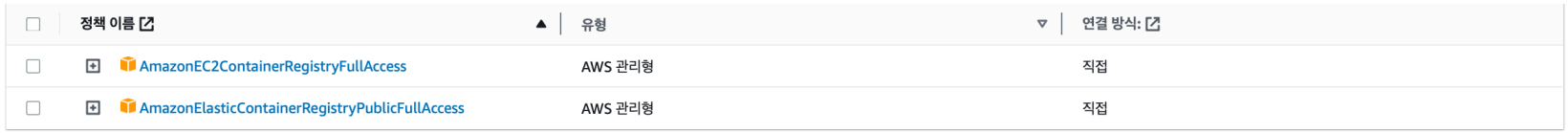
https://github.com/aws-actions/amazon-ecr-login

CI/CD 실습

Github Action을 이용한 AWS ECR에 이미지 올리기

1. AWS IAM user key github repo secret 등록 & IAM 생성 (아래 정책을 적용)





CI/CD 실습

Github Action을 이용한 AWS ECR에 이미지 올리기

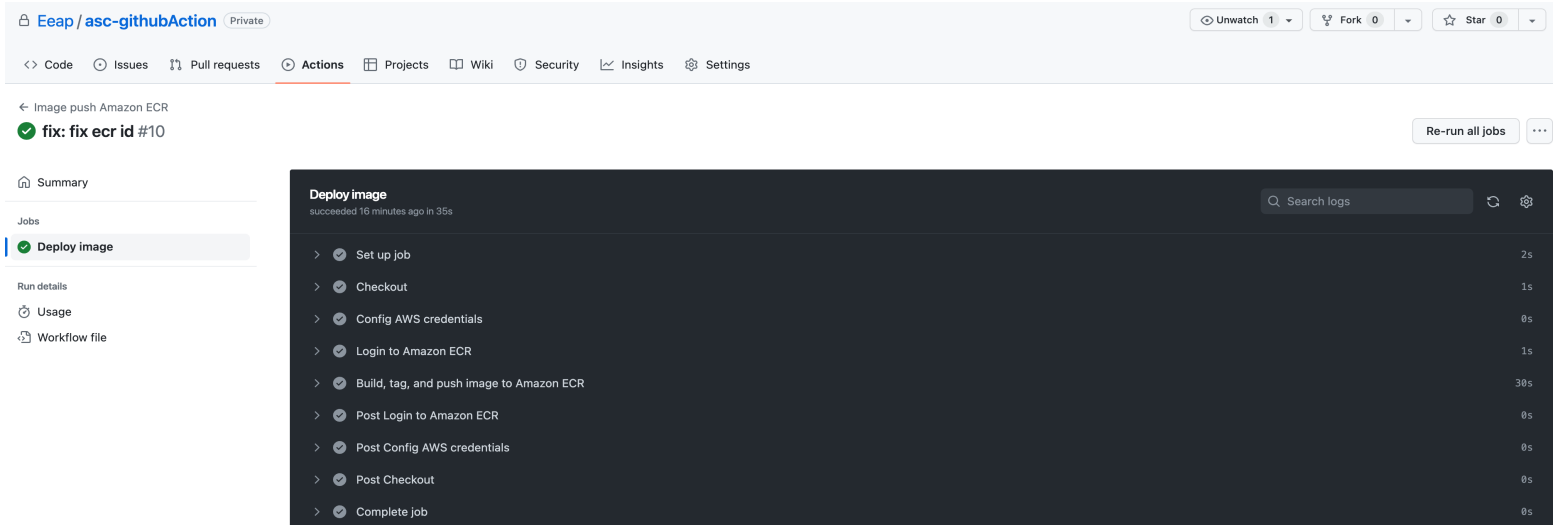
2. AWS Elastic Container Repository 생성

CI/CD 실습

Github Action을 이용한 **AWS ECR**에 이미지 올리기

3. Dockerfile 생성 후 commit, push

```
FROM ubuntu:latest
MAINTAINER sumin "sumink0903@gmail.com"

RUN apt-get update
RUN apt-get install -y nginx
RUN echo "nginx container"
WORKDIR /etc/nginx
CMD [ "nginx","-g","daemoe off;" ]
EXPOSE 80
```




thanks!