



# ASC KHU Hands on



# Contents

- 01 Container
- 02 Docker
- 03 CI/CD
- 04 Docker 및 CI/CD 실습

# *Container* ?

OS 수준에서 가상화

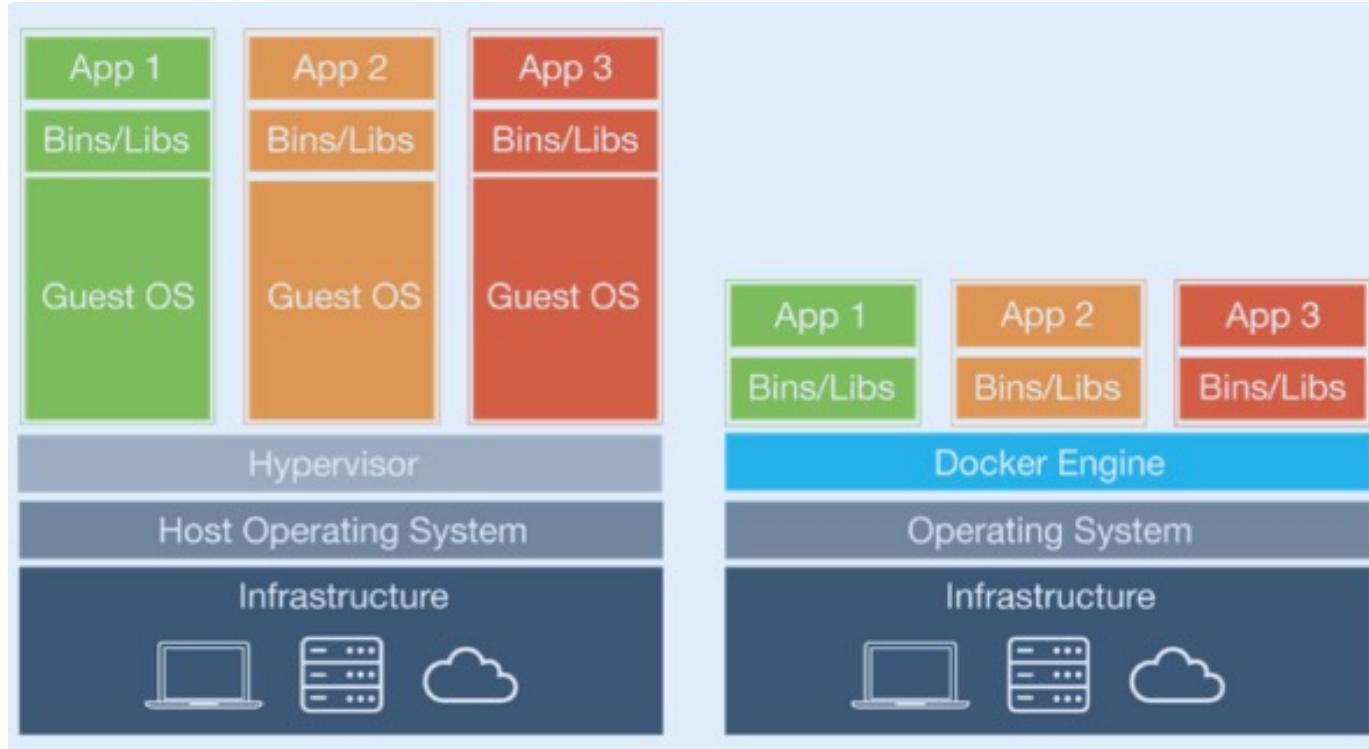
OS 커널 공유

*Container*

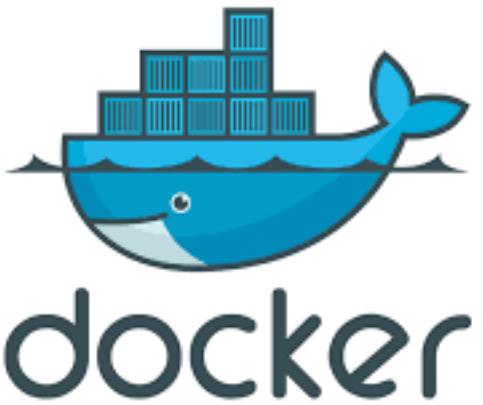
경량 패키지

프로세스 격리

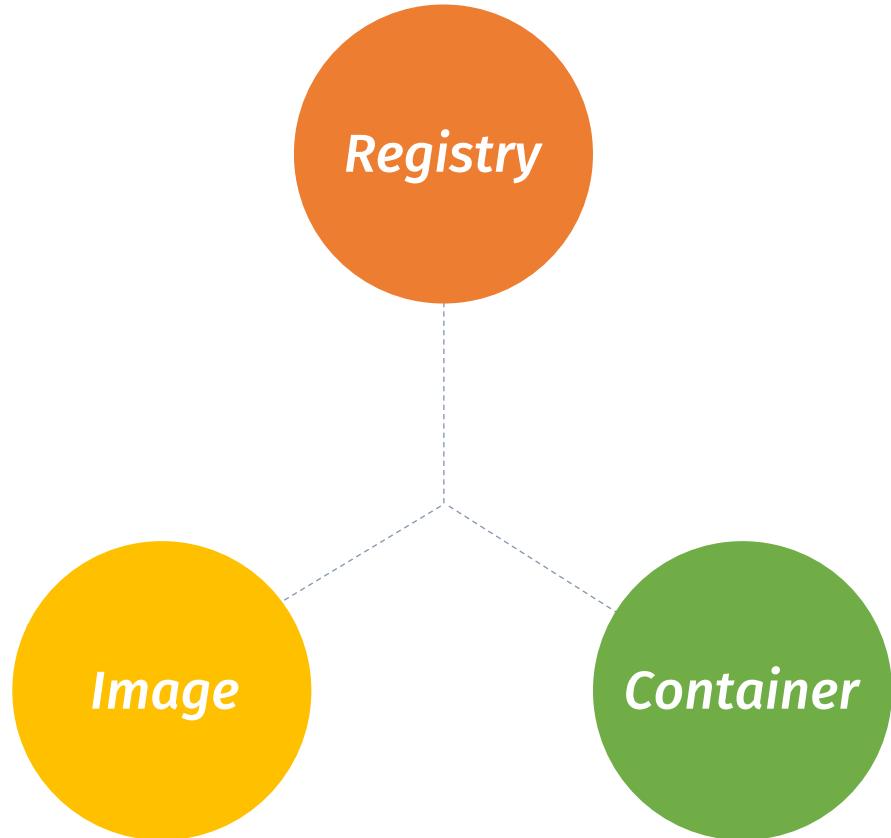
# Container ?



# 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

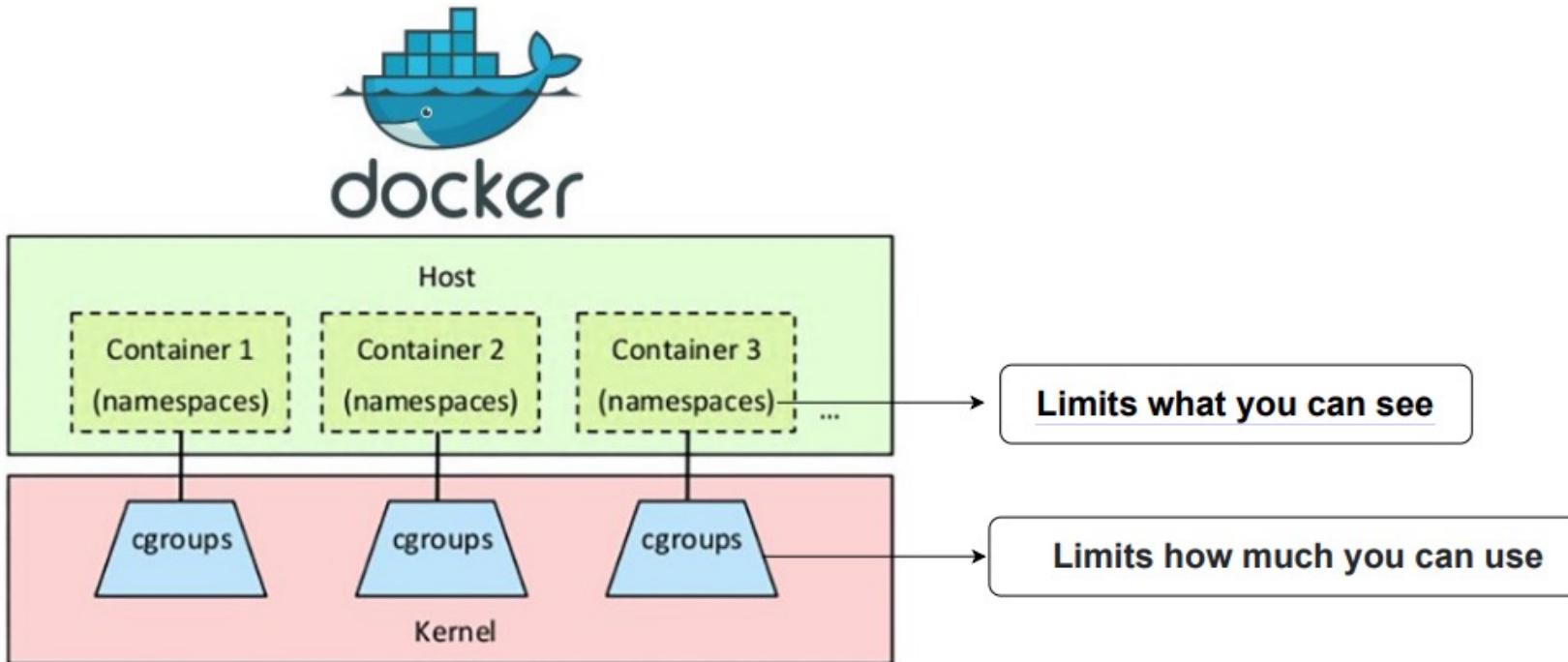
UTS namespace - 호스트와 도메인 네임 별로 격리

USER namespace - user와 group id를 분할하고 격리

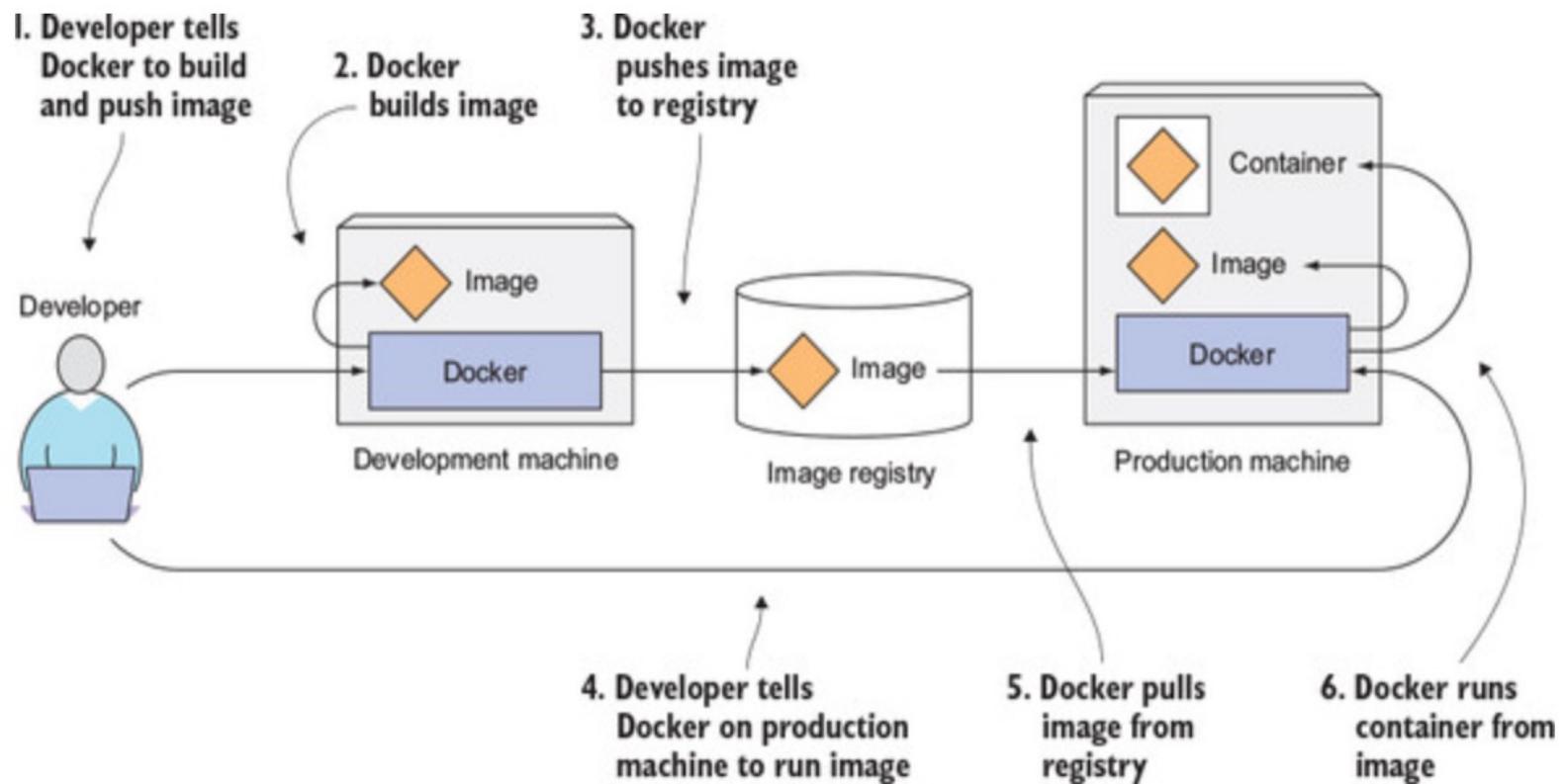
## cgroups

물리적 자원을 어디까지 허용할지에 대한 리소스 관리 역할

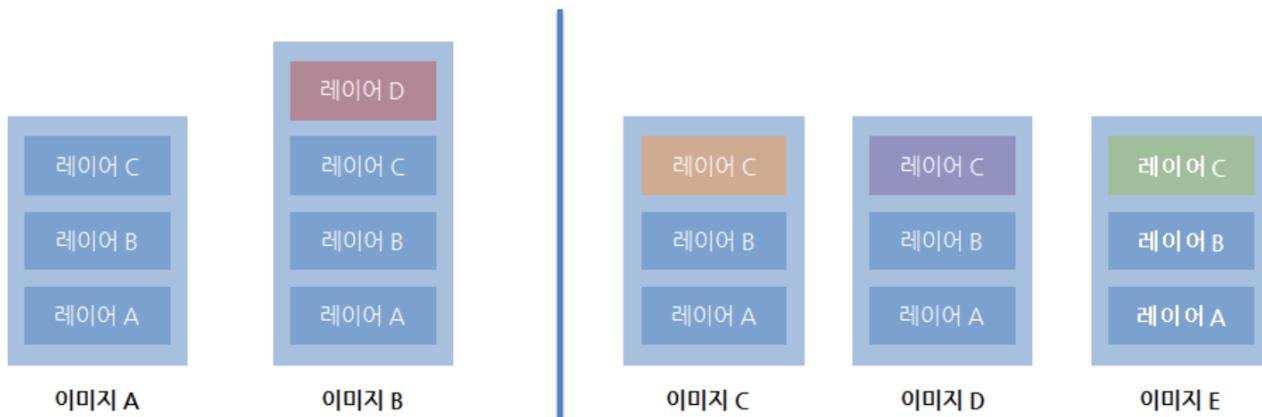
# Namespace & cgroups



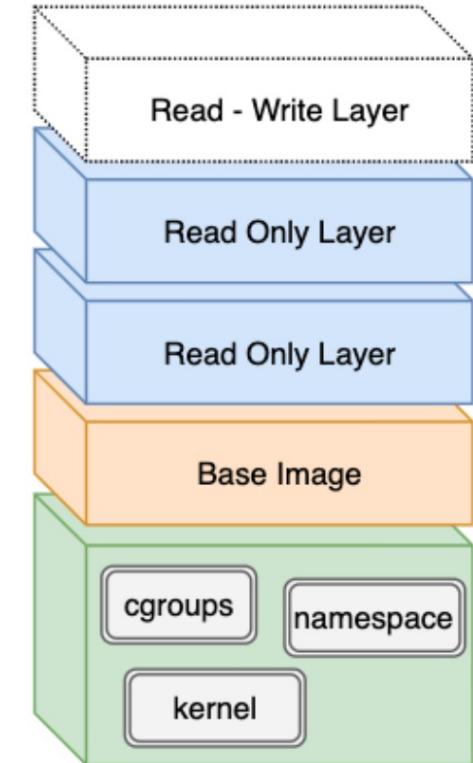
# Docker



# 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 > ...
1  # 베이스 이미지 선택
2  FROM ubuntu:latest
3
4  # 작성자 정보
5  LABEL maintainer="ASC dusdjhyeon <dusdj0813@knu.ac.kr>"
6
7  # ubuntu 패키지 매니저 업데이트 & nginx 설치
8  # -y를 통해 설치 도중 발생하는 확인메시지에 자동으로 yes 응답
9  RUN apt-get update
10 RUN apt-get install -y nginx
11
12 # this is a ubuntu container라는 메시지 출력
13 RUN echo "this is a ubuntu container"
14
15 # 현재 디렉토리를 /etc/nginx로 설정
16 # 이 이후 실행되는 명령어는 해당 디렉토리에서 실행
17 WORKDIR /etc/nginx
18
19 # 컨테이너 시작 시 실행할 명령어 (nginx를 백그라운드에서 실행)
20 CMD ["nginx", "-g", "daemon off;"]
21
22 # 포트 80 노출
23 EXPOSE 80
```

**FROM** – 베이스 이미지 설정

**COPY** – 파일 또는 디렉토리를 컨테이너에 복사

**CMD** – 컨테이너가 **default**로 수행할 동작 기술

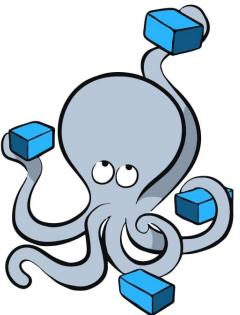
**EXPOSE** – 노출할 컨테이너 포트

**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

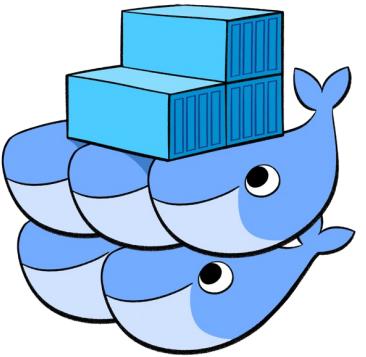


docker  
Compose

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

다중 컨테이너를 yaml파일을 통해 배포

# *Container Orchestration*



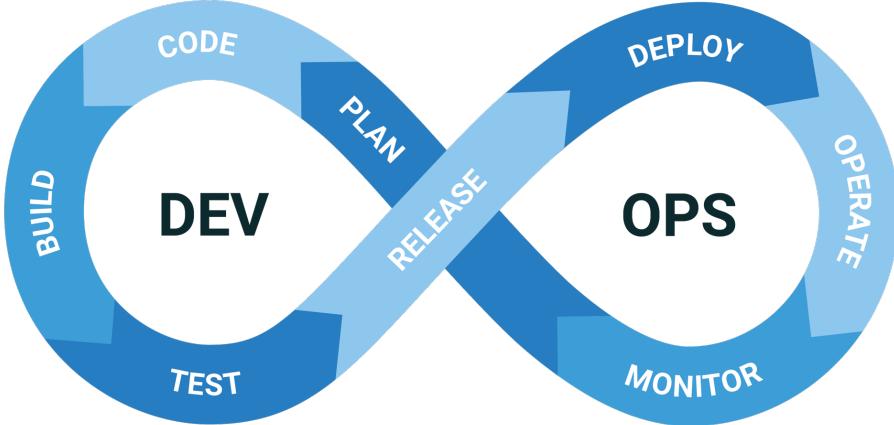
Docker Swarm



**kubernetes**



# CI/CD ?



**CI (Continuos Integration)** 지속적 통합

=> build 자동화, 유닛 및 통합 테스트

**CD (Continuos Delivery)** 지속적 배포

=> build된 코드를 release하고 production으로 deploy

# CI/CD ?

Cloud Native Landscape

App Definition and Development - Continuous Integration & Delivery (56)

Landscape Guide

Reset Filters

Grouping

Category

Sort By

Alphabetical (a to z)

Category

Continuous Integration & Delivery

Project

Any

License

Any

Organization

Any

Headquarters

Any

Company Type

Any

Industry

Any

Download as CSV

Example filters

Cards by age

Open source landscape

Member cards

Cards by stars

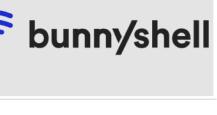
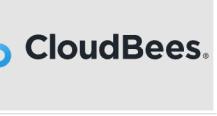
Cards from China

Certified K8s/KCSP/KTP

Cards by MCap/Funding

Cards without

infractices.dev

 Agola Sortin.Lab ★ 1,269	 Akuity Akuity Funding: \$24.5M	 AppVeyor Appveyor Systems	 Argo Cloud Native Computing Foundation (CNCF) Funding: \$3M	 AWS CodePipeline Amazon Web Services MCap: \$1.1T	 Azure Pipelines Microsoft MCap: \$2.1T
 Bamboo Atlassian MCap: \$38.9B	 Brigade Cloud Native Computing Foundation (CNCF) Funding: \$3M	 Buildkite Buildkite Funding: \$41.5M	 Bunnyshell Bunnyshell Funding: \$6.4M	 Bytebase Bytebase Funding: \$3M	 CAEPE Biqmind
 Cartographer VMware MCap: \$55B	 CircleCI CircleCI Funding: \$315M	 Cloud 66 Skycap Cloud 66 Funding: \$2.4M	 CloudBees CloudBees Funding: \$356.2M	 Codefresh Codefresh Funding: \$43M	 Concourse VMware ★ 6,839 MCap: \$55B
 D2IQ Dispatch D2iq Funding: \$247.3M	 Devtron Funding: \$3,057	 Drone Drone.io ★ 26,762 Funding: \$28K	 Flagger Cloud Native Computing Foundation (CNCF) Funding: \$3M	 Flipt Flipt ★ 2,326	 flux Cloud Native Computing Foundation (CNCF) Funding: \$3M



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

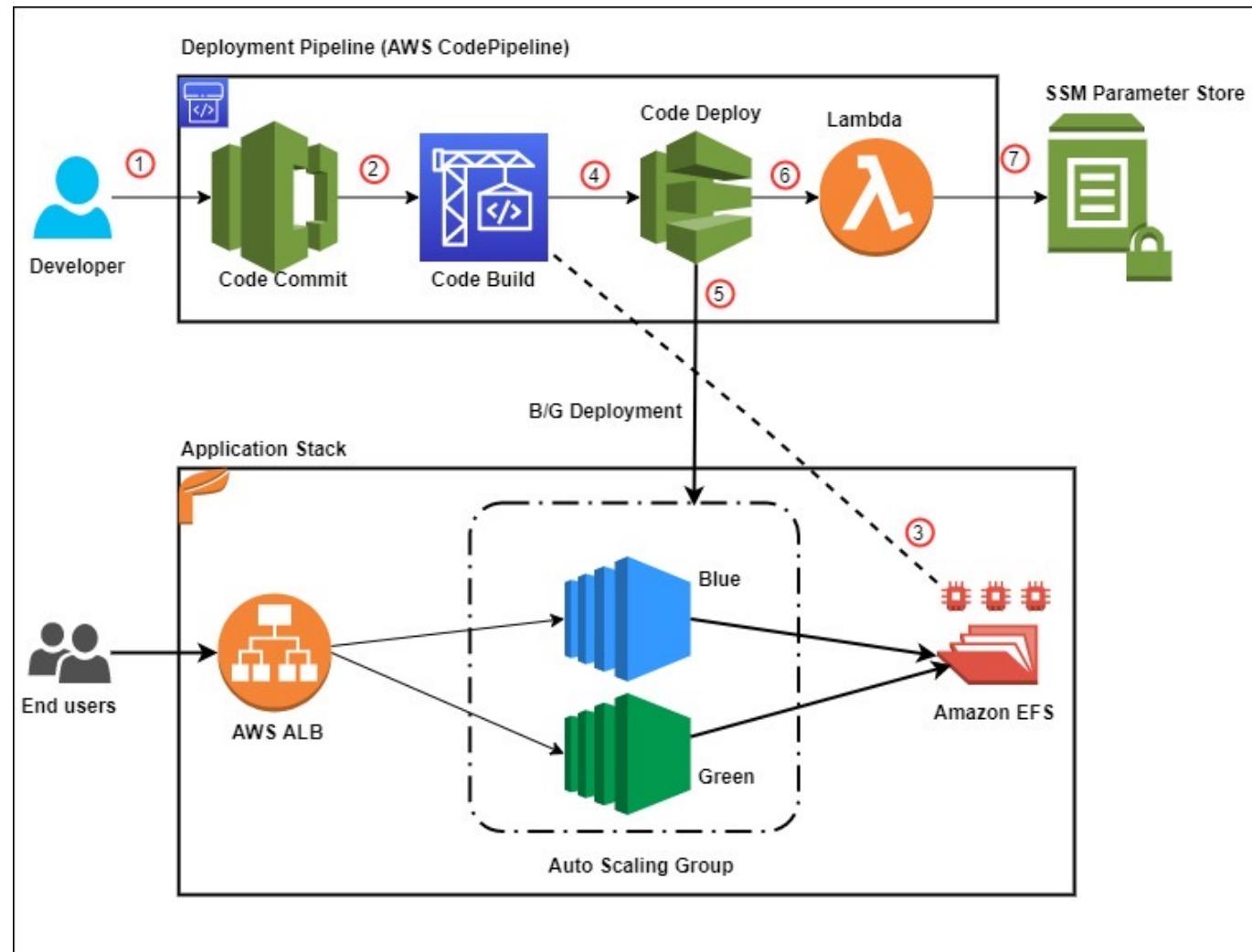
Compile source code, run tests, and generate artifacts without managing build servers



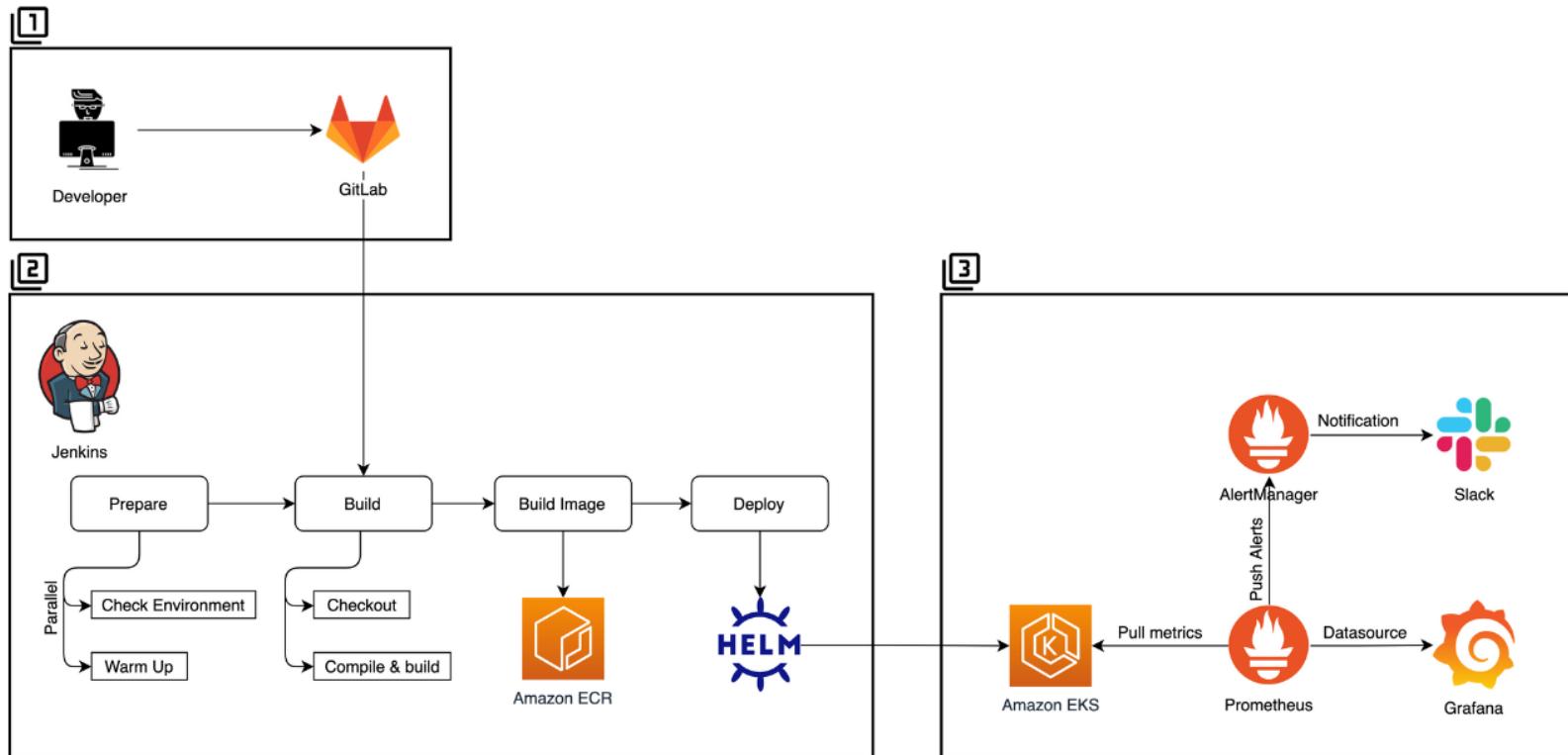
## AWS CodePipeline

Example pipeline

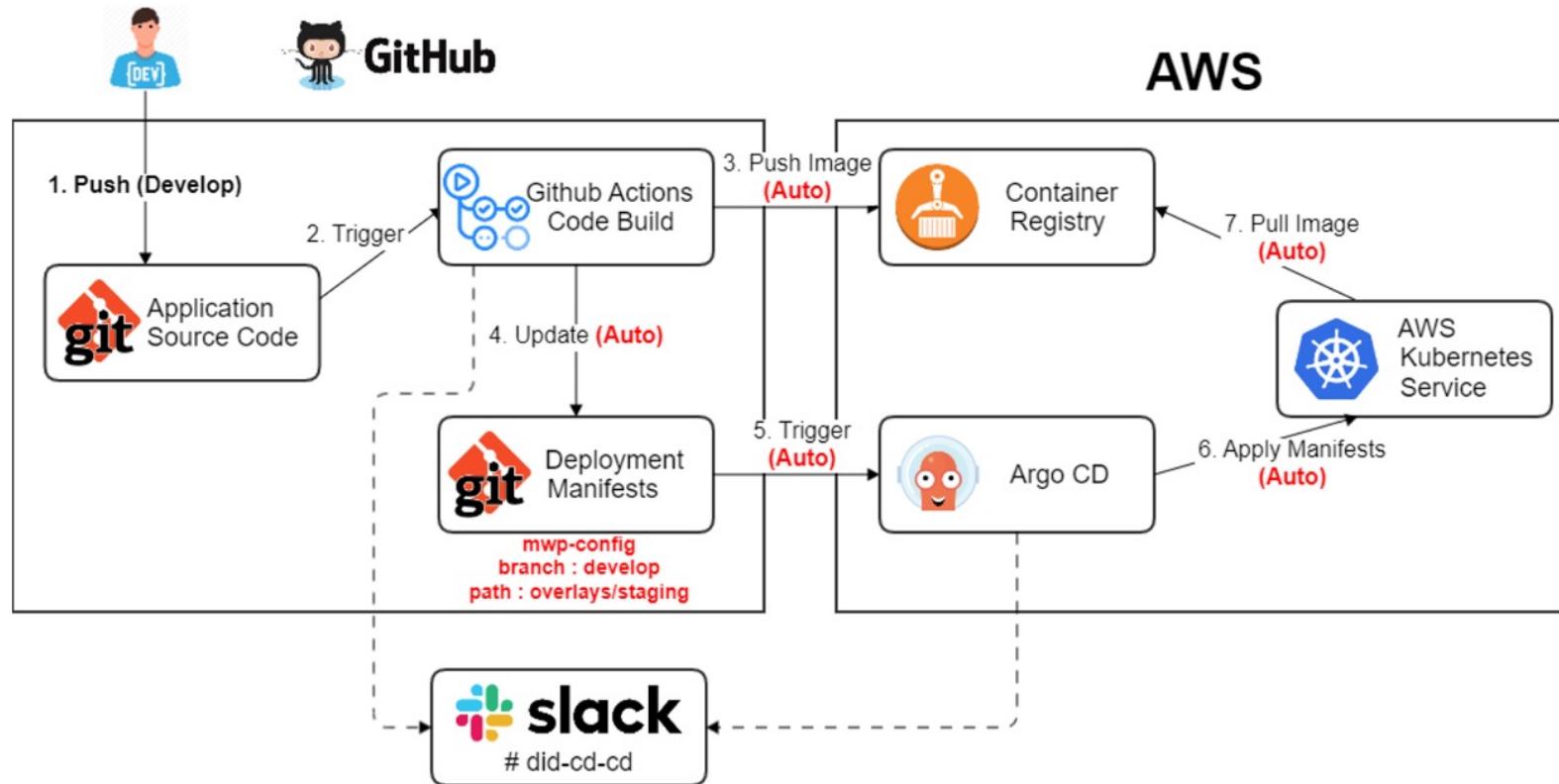
# CI/CD 실제 사례



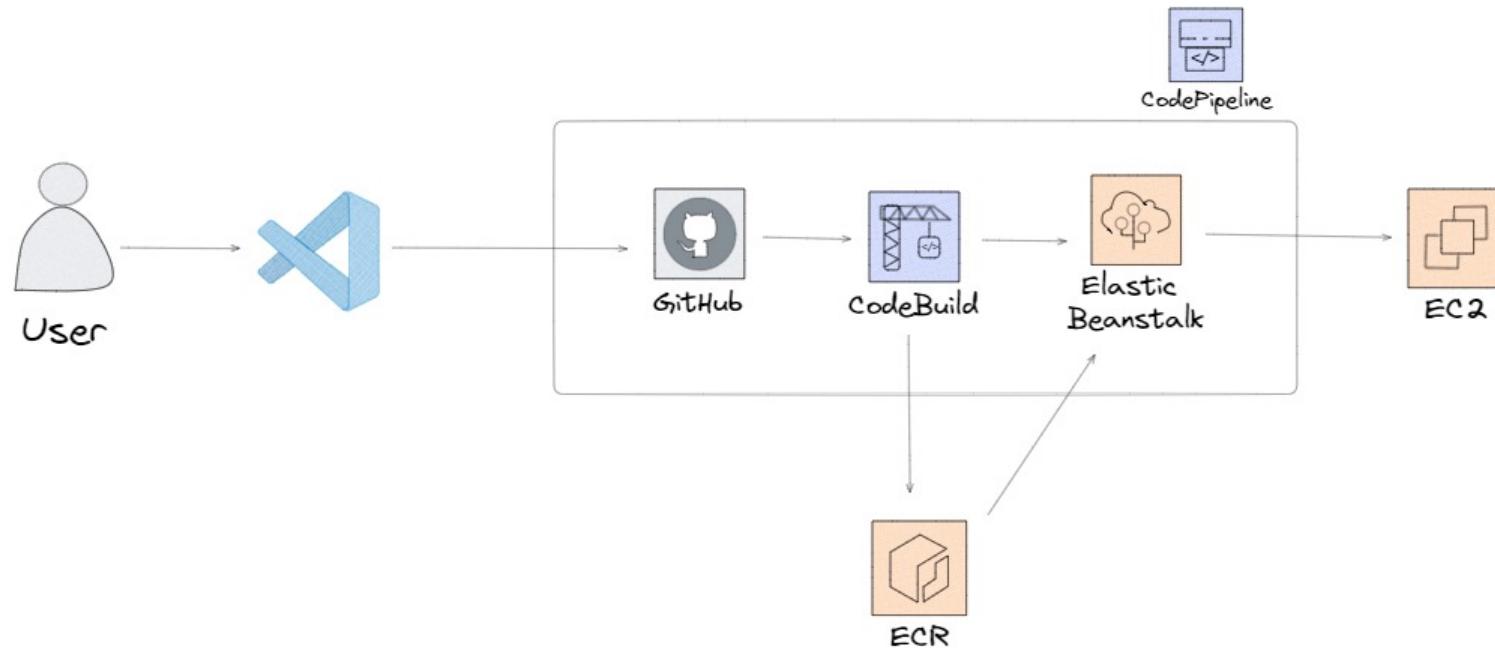
# CI/CD 실제 사례



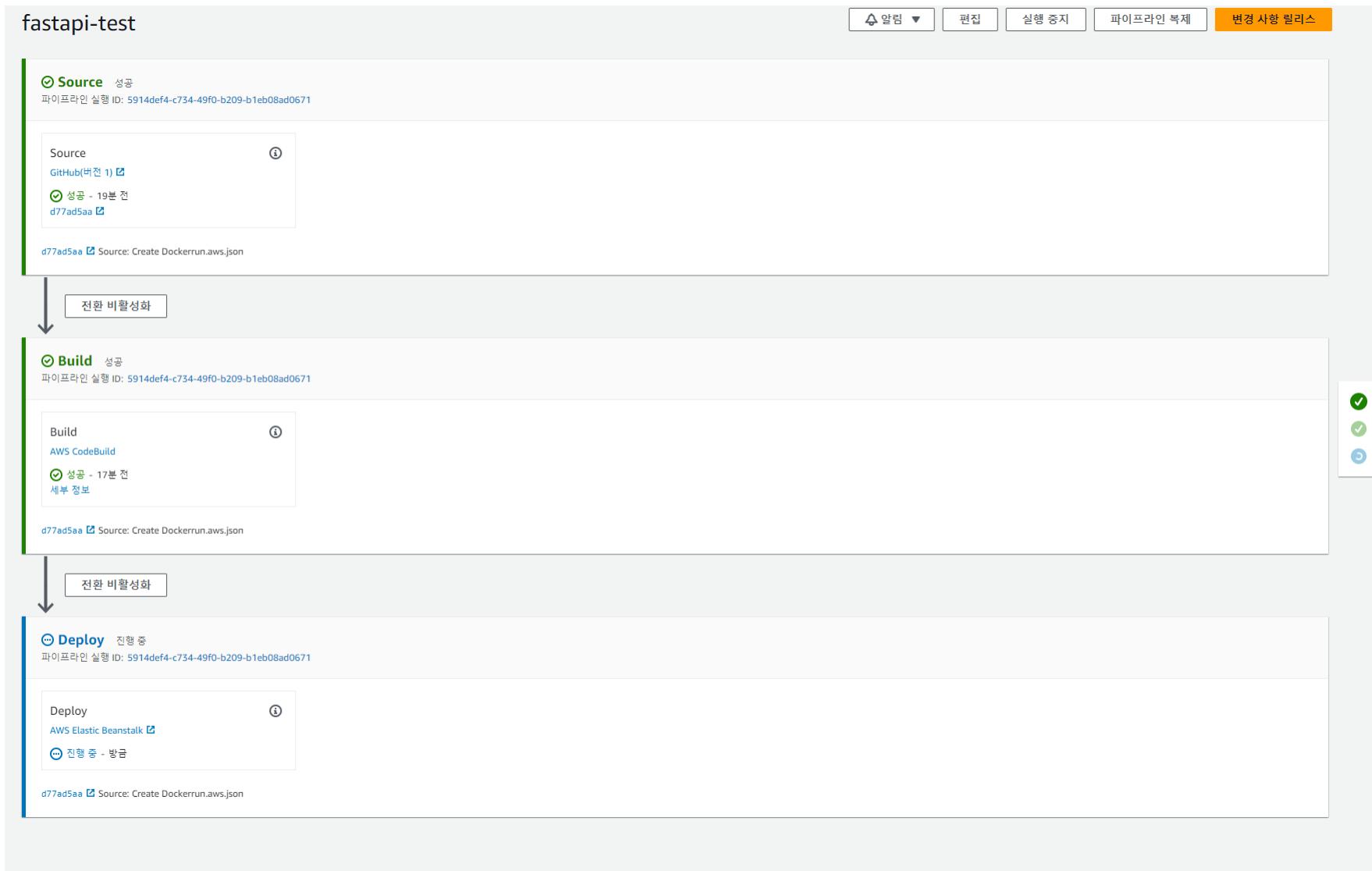
# CI/CD 실제 사례



# 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 6 hours ago 3 commits

File	Commit Message	Time
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

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 Generate SLSA compliance for your

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.

# Build: CodeBuild



fastapi-test

구성

소스 출처 GitHub 퍼블릭 빌드 비활성

기본 리포지토리 FacerAin/fastapi-docker-example

아티팩트 업로드 위치

빌드 버전

비활성

빌드 기록 배치 이력 | 빌드 세부 정보 | 빌드 트리거 | 자료

빌드 기록

빌드 번호	상태	빌드 번호	소스 버전	제출자	기간	원로필
fastapi-test:74b39191-5dfb-4145-ad16-17c23e56564	성공함	5	am.aws:3:codepipeline-ap-northeast-2-624233391797/fastapi-test/SourceArtifacts5mn2y.zip	codepipeline/fastapi-test	2분 9초	19분 전
fastapi-test:0ce5f993-4b18-4b36-96d4-7c6feb83b7df	성공함	4	am.aws:3:codepipeline-ap-northeast-2-624233391797/fastapi-test/SourceArtifactsEnfue5.zip	codepipeline/fastapi-test	2분 7초	6시간 전

```
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:
    - Dockerfile.aws.json
```

# Deploy: Beanstalk



ⓘ 환경 업데이트가 성공적으로 완료되었습니다.

Elastic Beanstalk > 환경 > DockerAppTest-env

DockerAppTest-env [Info](#)

환경 개요

상태	Ok	환경 ID	e-hdwmugqgjx
도메인	<a href="https://DockerAppTest-env.eba-8mp4xvvr.ap-northeast-2.elasticbeanstalk.com">DockerAppTest-env.eba-8mp4xvvr.ap-northeast-2.elasticbeanstalk.com</a>	애플리케이션 이름	DockerAppTest

플랫폼

플랫폼	Docker running on 64bit Amazon Linux 2/3.5.7
실행 중인 버전	code-pipeline-1685495494444-BuildArtifact-1c03a1bc-a189-4d63-99bd-283c84237723

이벤트 상태 로그 모니터링 경보 관리형 업데이트 태그

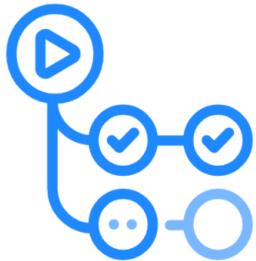
이벤트 (37) [Info](#)

시간	유형	세부 정보
5월 31, 2023 10:13:37 (UTC+9)	INFO	Environment health has transitioned from Info to Ok. Application update completed 41 seconds ago and took 19 seconds.
5월 31, 2023 10:12:37 (UTC+9)	INFO	Environment health has transitioned from Ok to Info. Application update in progress on 1 instance. 0 out of 1 instance completed (running for 10 seconds).
5월 31, 2023 10:11:58 (UTC+9)	INFO	Environment update completed successfully.
5월 31, 2023 10:11:58 (UTC+9)	INFO	New application version was deployed to running EC2 instances.
5월 31, 2023 10:11:51 (UTC+9)	INFO	Instance deployment completed successfully.

# 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** – workflow에서 하나의 처리 단위

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

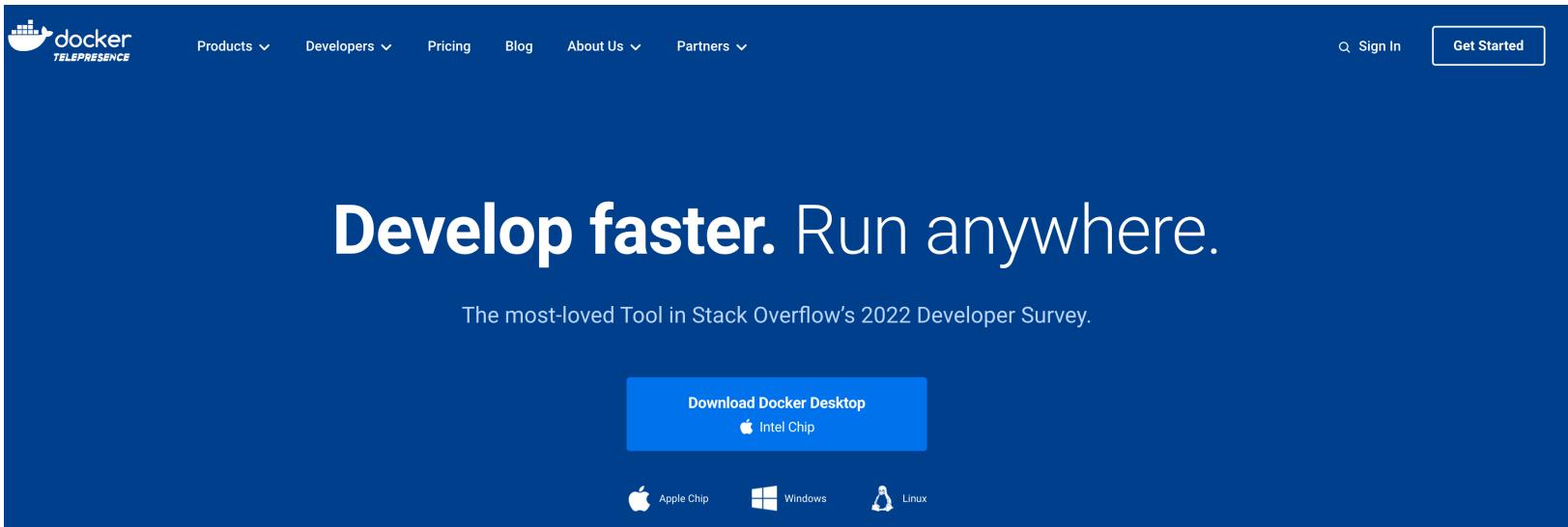
**Actions** – 자주 반복되는 작업을 용이하도록 하기 위한 작업 공유 메커니즘

**Runners** – 트리거될 때 workflow를 실행하는 서버

# Docker 실습

Docker image를 만들고 DockerHub에 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 DockerHub에 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
```

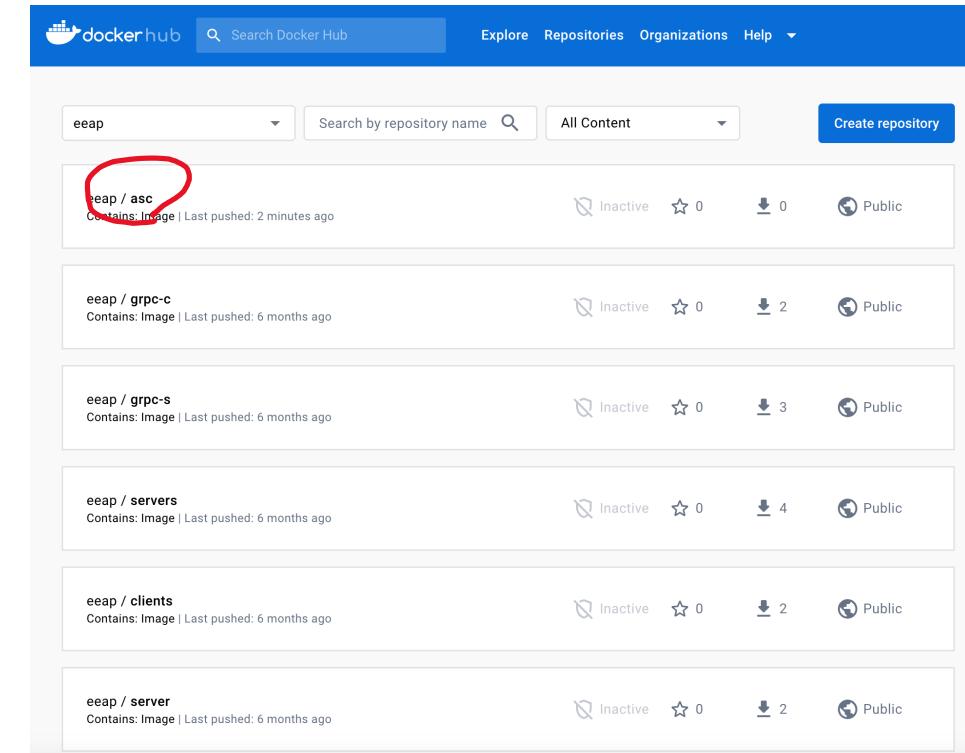
# Docker 실습

Docker image를 만들고 DockerHub에 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 DockerHub에 올려서 Docker compose 까지 진행하기!

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

```
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 ECR에 이미지 올리기

```
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 }}
```

```
  - 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)

The screenshot shows the GitHub repository settings for 'Eeap / asc-githubAction'. The 'Actions secrets and variables' section is displayed, showing two secrets: 'AWS\_ACCESS\_KEY\_ID' and 'AWS\_SECRET\_ACCESS\_KEY', both updated 1 hour ago. The 'Repository secrets' section is also visible. On the left, the 'Secrets and variables' section is selected in the sidebar. At the bottom, a table shows the policy assignments for these secrets, including the 'AmazonEC2ContainerRegistryFullAccess' and 'AmazonElasticContainerRegistryPublicFullAccess' policies.

정책 이름	유형	연결 방식
AmazonEC2ContainerRegistryFullAccess	AWS 관리형	직접
AmazonElasticContainerRegistryPublicFullAccess	AWS 관리형	직접

# CI/CD 실습

## Github Action을 이용한 AWS ECR에 이미지 올리기

### 2. AWS Elastic Container Repository 생성



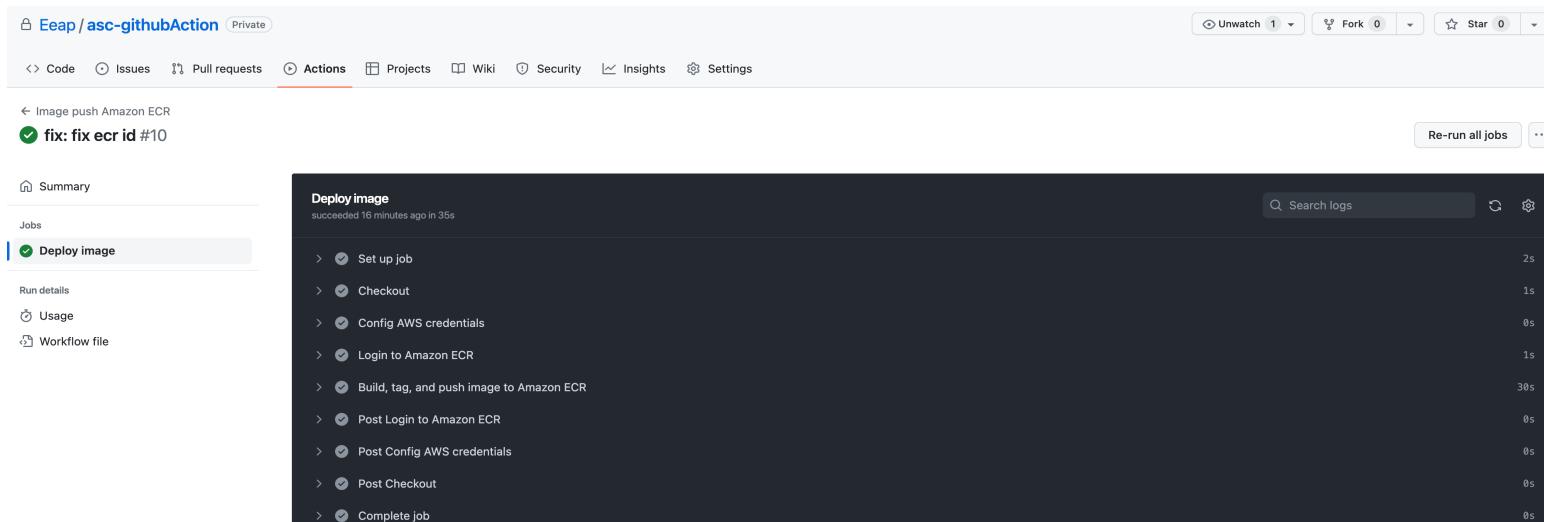
# CI/CD 실습

Github Action을 이용한 AWS ECR에 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","daemon off;" ]
EXPOSE 80
```



The screenshot shows a GitHub repository named 'Eeap / asc-githubAction'. The 'Actions' tab is selected, showing a single workflow run titled 'fix: fix ecr id #10'. The run status is 'succeeded' 16 minutes ago. The workflow steps are:

- Set up job (0s)
- Checkout (1s)
- Config AWS credentials (0s)
- Login to Amazon ECR (1s)
- Build, tag, and push image to Amazon ECR (30s)
- Post Login to Amazon ECR (0s)
- Post Config AWS credentials (0s)
- Post Checkout (0s)
- Complete job (0s)

At the bottom right, there is a 'Search logs' input field and a 'Re-run all jobs' button.



thanks!

