Container Overview

안준환
Solutions Architect
Amazon Web Services

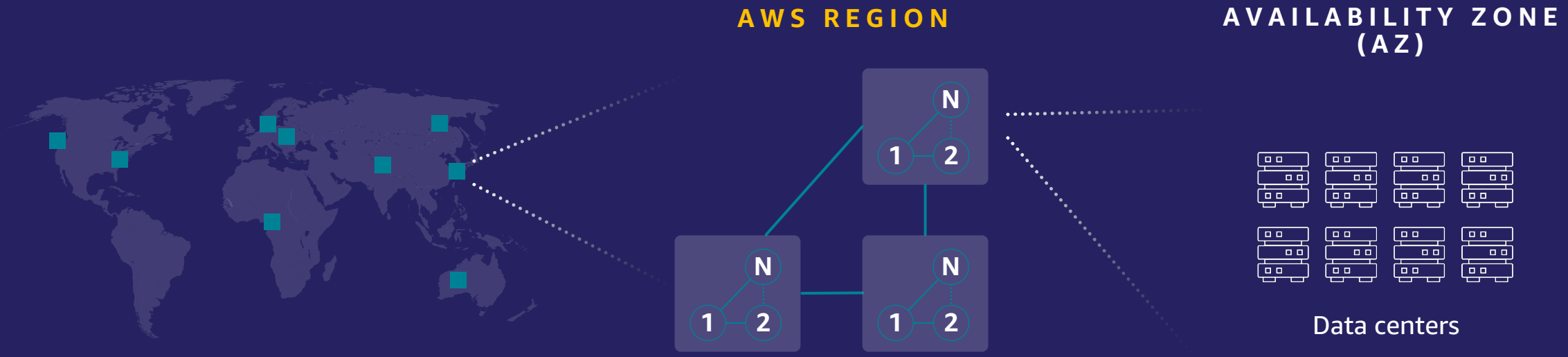
Agenda

- AWS 클라우드 용어 정리
- Container Component
- Docker Component
- Docker Workflow
- Summary

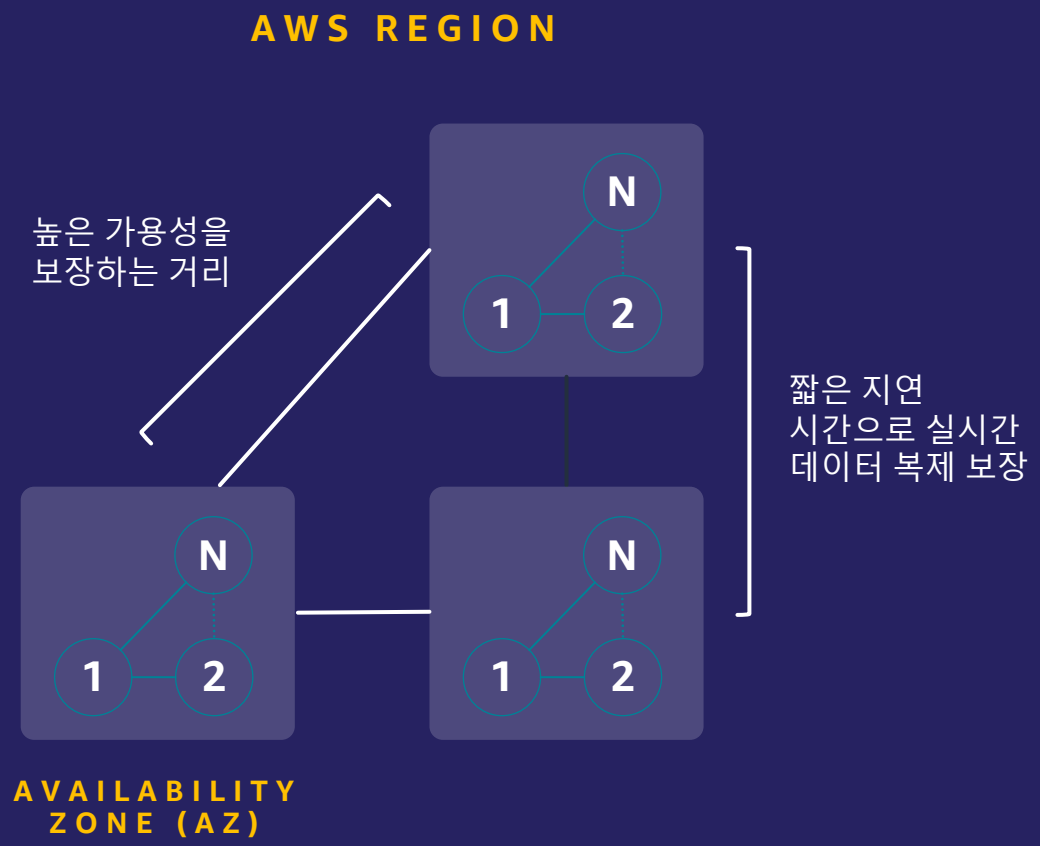
AWS 클라우드 용어 정리

AWS 리전 설계

AWS 리전은 더 높은 가용성, 확장성, 내결함성을 위해서 다중의 가용영역으로 구성됩니다. 어플리케이션과 데이터는 다른 가용영역 간에 실시간 복제가 되며 일관성을 가집니다.



AWS 가용 영역 (AZ) 설계



100,000대 이상의 서버 규모

격리된 파티션

하나 이상의 데이터센터로 완전히 격리됨

전력

높은 가용성, 내결함성, 확장성

거리

물리적으로 의미 있는 거리로 떨어져 있으며 모두 서로 60마일(100km) 이내

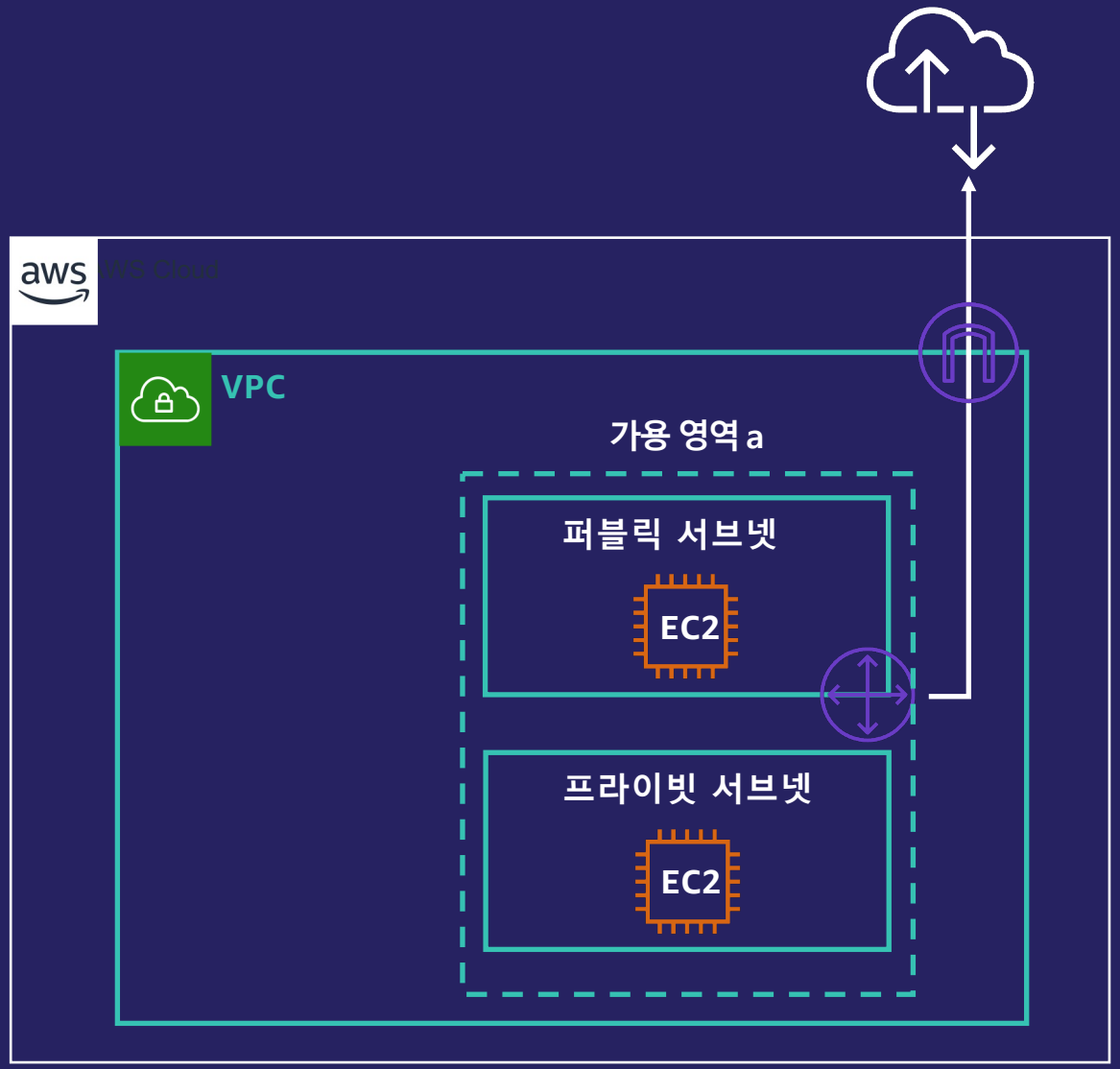
상호 연결

완전 이중화 및 절연된 메트로 파이버를 통해 연결된 데이터센터

Amazon VPC 정의

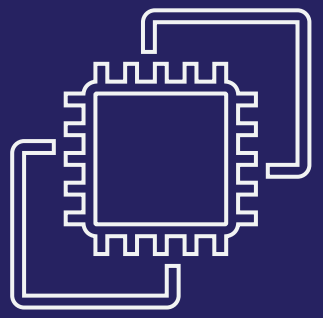
Virtual Private Cloud

- 사용자가 정의한 가상의 네트워크 공간
- 완전한 네트워크 제어 가능
  - IP 범위
  - Subnet
  - Route Table
  - Network ACL, 보안 그룹
  - 다양한 게이트웨이
- VPC 내의 모든 EC2 인스턴스들은 사설 IP가 부여됨
- 개별 인스턴스에 공인 IP 할당 가능 (Public IP/Elastic IP)



Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud (EC2)



Amazon EC2

안전하고 크기 조정이 가능한 컴퓨팅 파워

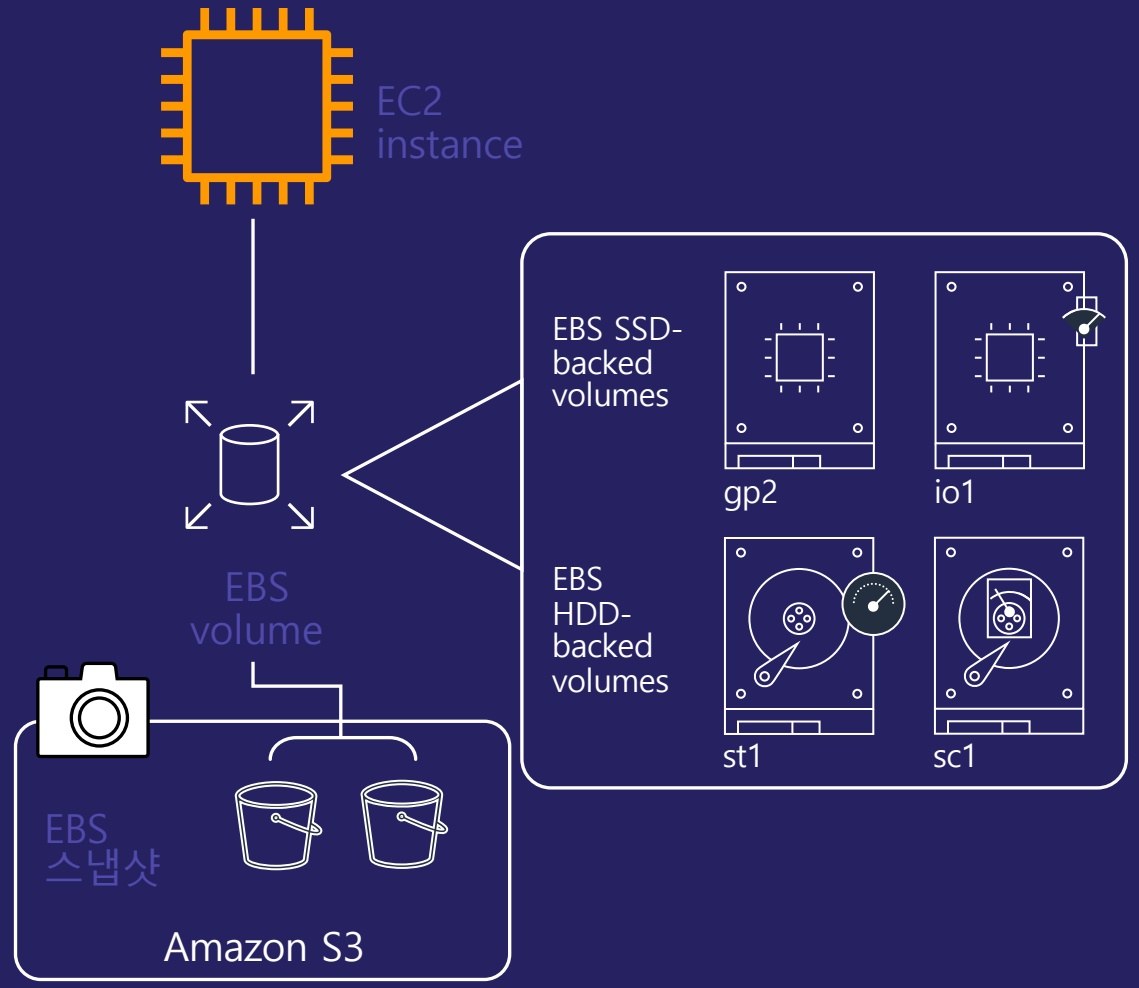
컴퓨팅 요구 사항 변화에 따라 신속하게 용량을 확장하거나 축소

실제 사용한 만큼 요금을 지불하면 되므로, 컴퓨팅 비용이 절약

OS : Linux | Windows | MAC

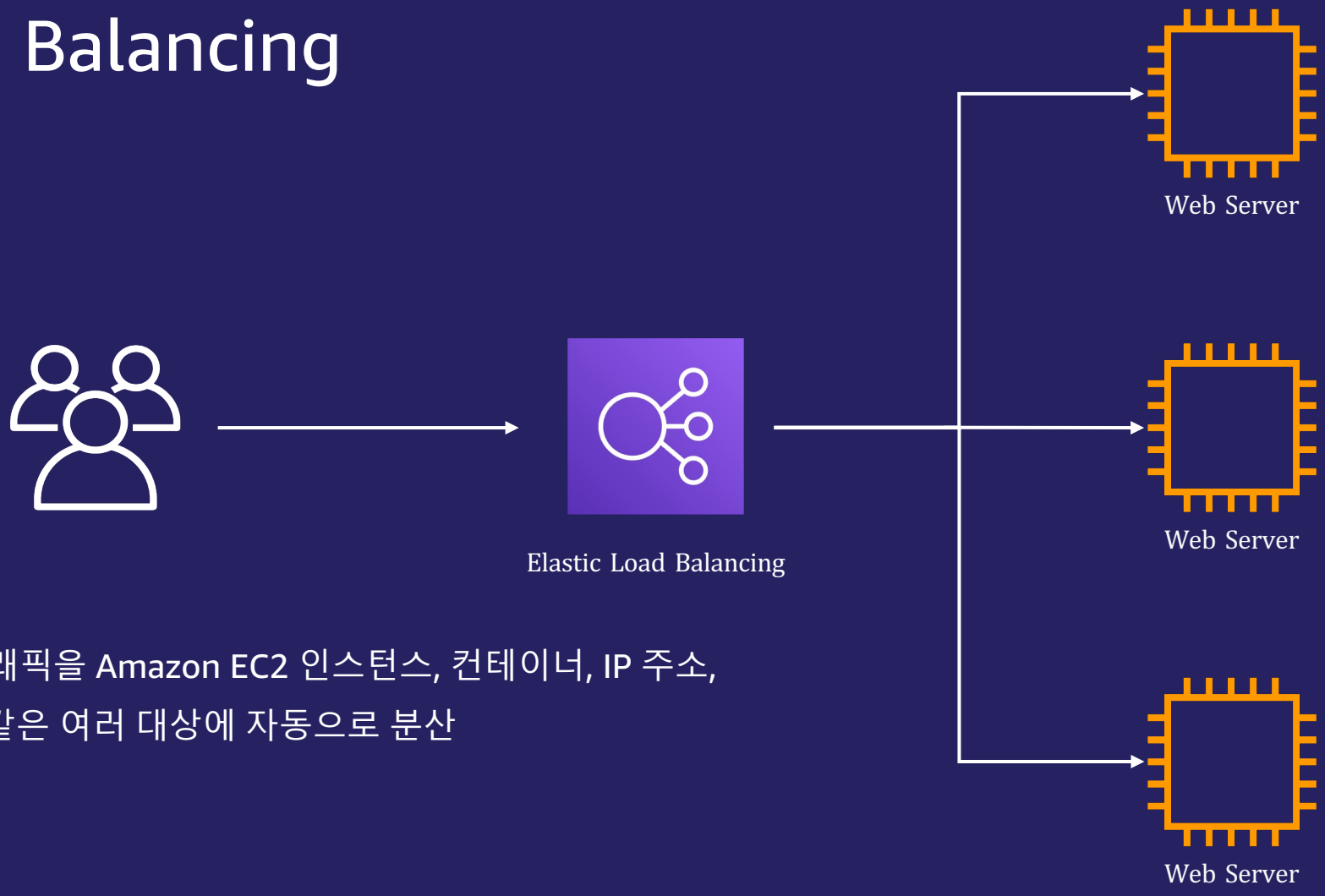
여러가지 구매 옵션 : 온디맨드, 예약 인스턴스, 스팟, 세이빙 플랜

Amazon EBS (Elastic Block Store)



- 블록 스토리지 (영구저장)
- API를 이용하여 생성, 연결, 수정
- 워크로드에 따라 스토리지 및 컴퓨팅 선택
- 하나의 EBS 볼륨은 하나의 인스턴스에만 연결
- 마그네틱 및 SSD 기반 볼륨 유형 선택
- 스냅 샷 지원 : 특정 시점 백업

Elastic Load Balancing



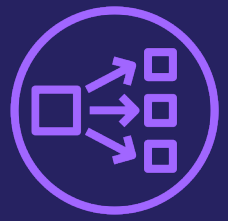
애플리케이션 트래픽을 Amazon EC2 인스턴스, 컨테이너, IP 주소,
Lambda 함수와 같은 여러 대상에 자동으로 분산

ELB: Options

Application Load Balancer

- IPv4, Dualstack front-end
- Layer 7
- HTTP, HTTPS
- Host-, Path-based routing
- Integrated authentication
- Supported Targets
  - EC2 instances
  - Containers
  - AWS Lambda
  - Private IP addresses

Network Load Balancer



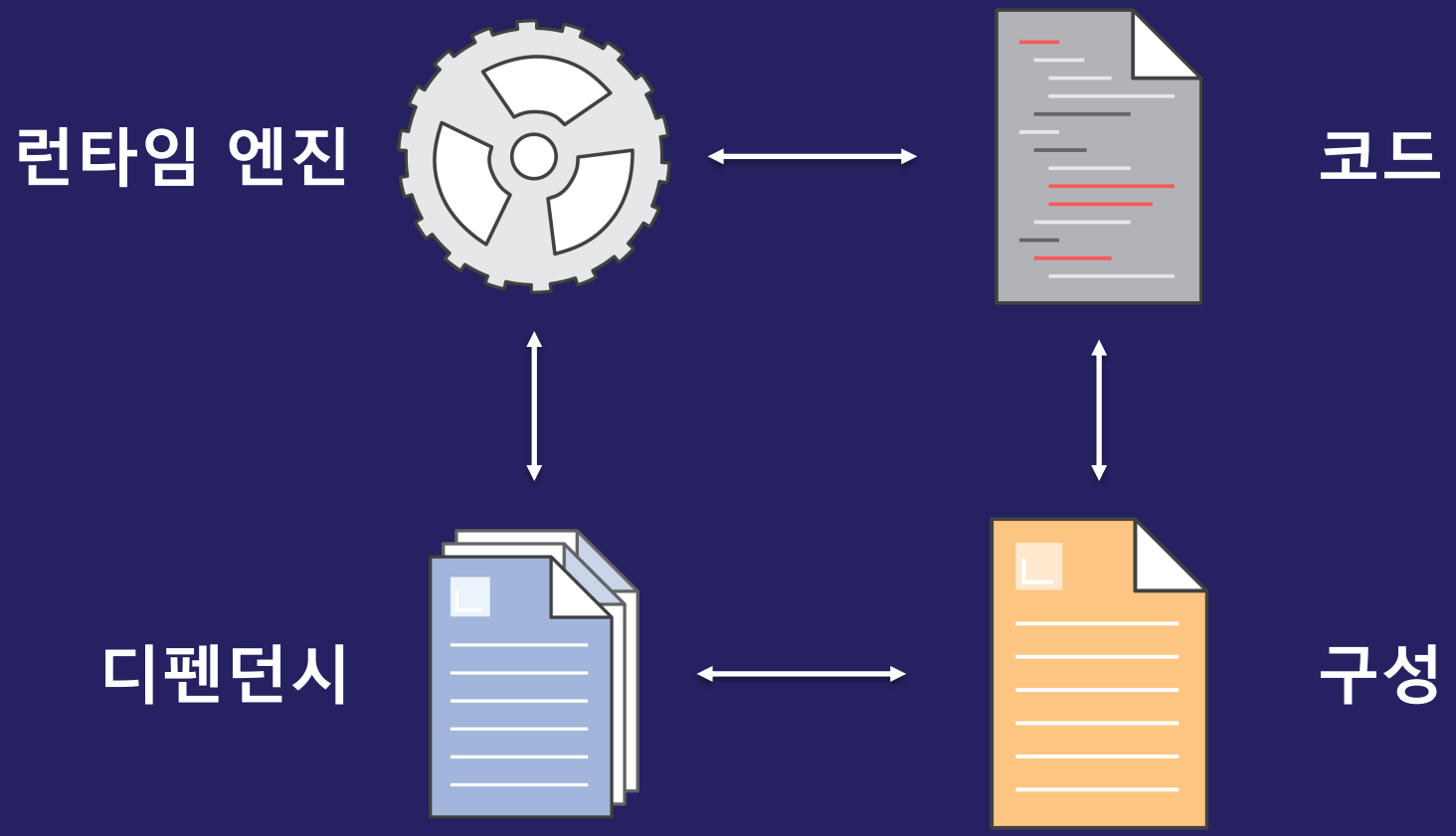
- IPv4
- Layer 4
- TCP, UDP, TLS
- Supported Targets
  - EC2 instances
  - Containers
  - Private IP addresses

Classic Load Balancer

- IPv4, Dualstack front-end
- Layer 4/7
- HTTP, HTTPS, TCP, TLS
- Supported Targets
  - EC2 Instances

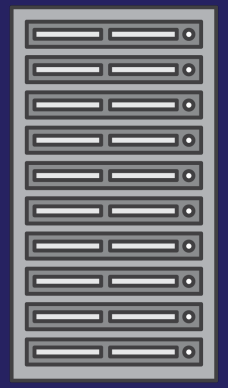
Container Component

애플리케이션 주요 구성 요소

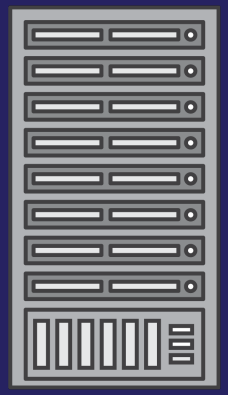


서로 다른 환경으로의 배포

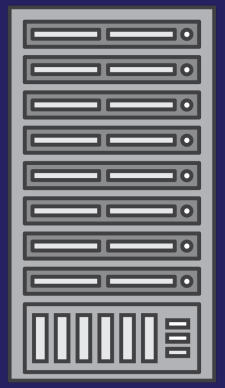
Local Laptop



Staging / QA

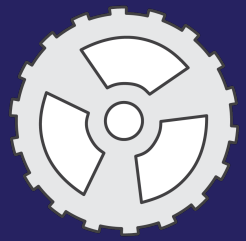


Production

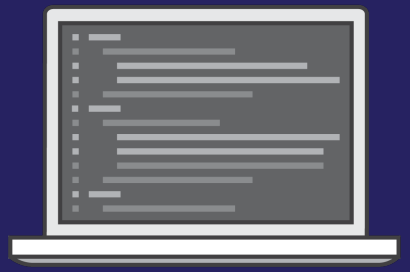


On-Prem

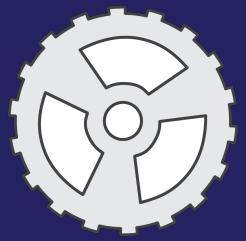
내 컴퓨터에서는 잘 되는데?



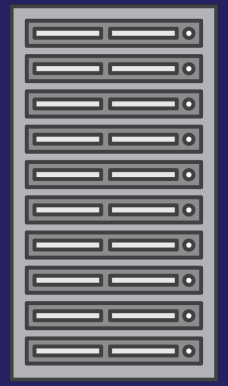
v6.0.0



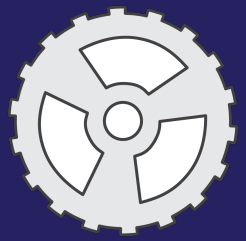
Local Laptop



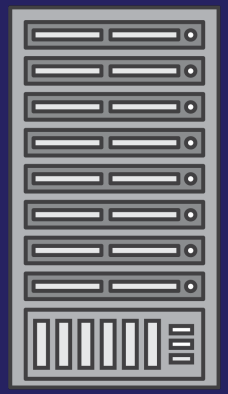
v7.0.0



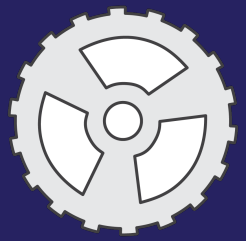
Staging / QA



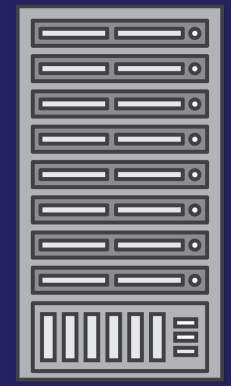
v4.0.0



Production

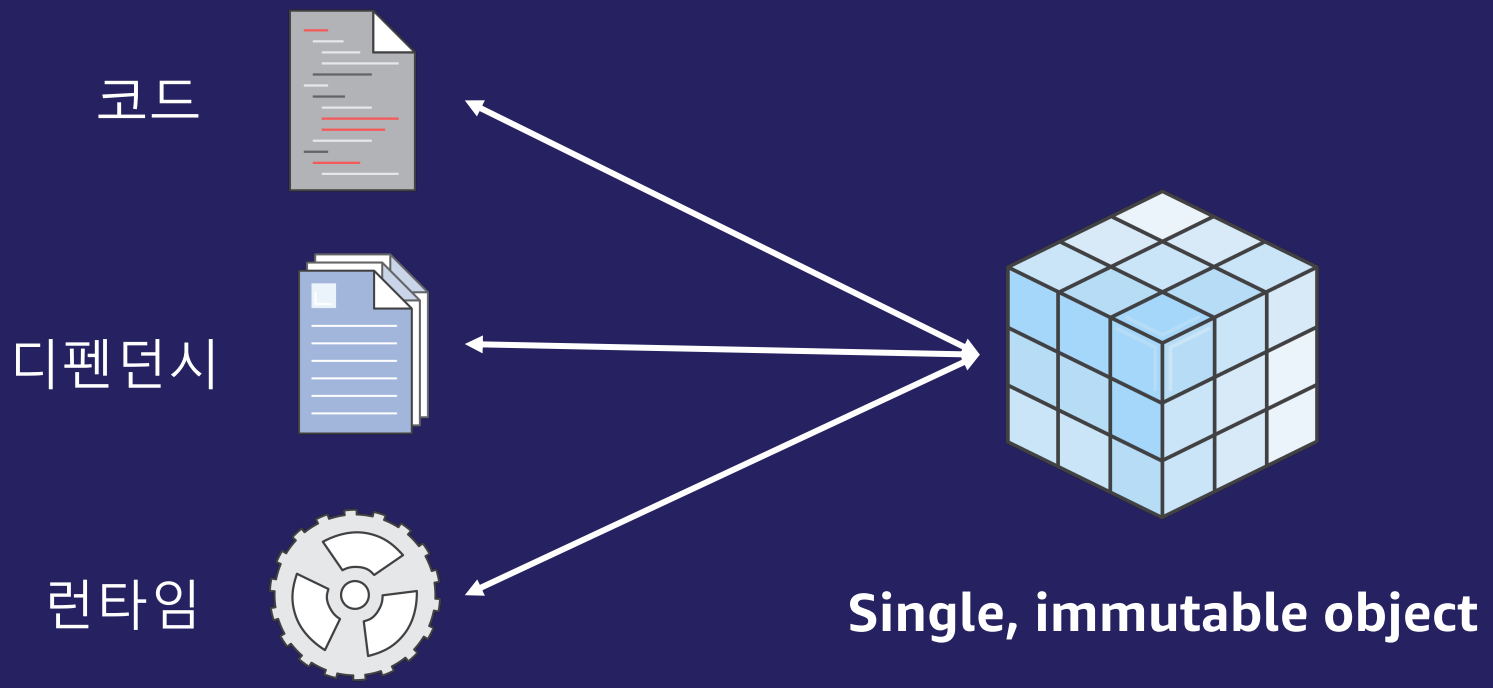


v7.0.0



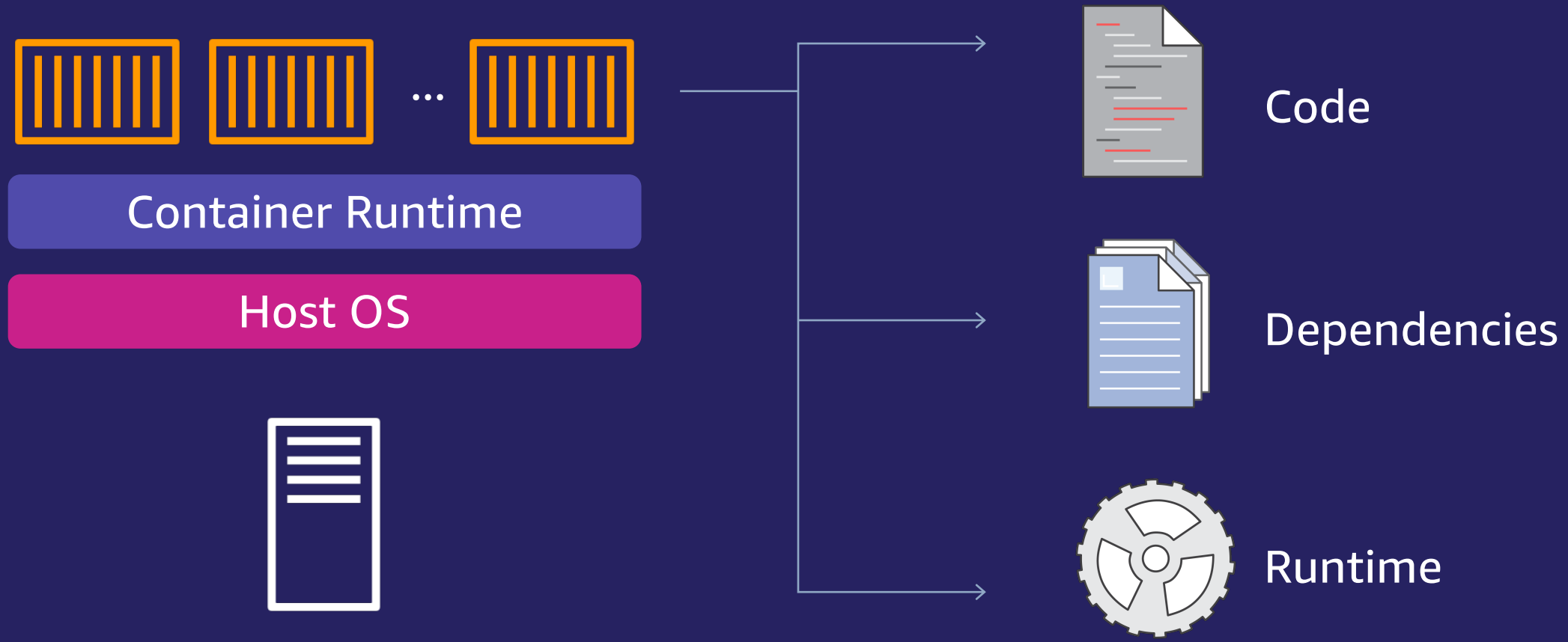
On-Prem

컨테이너 이미지

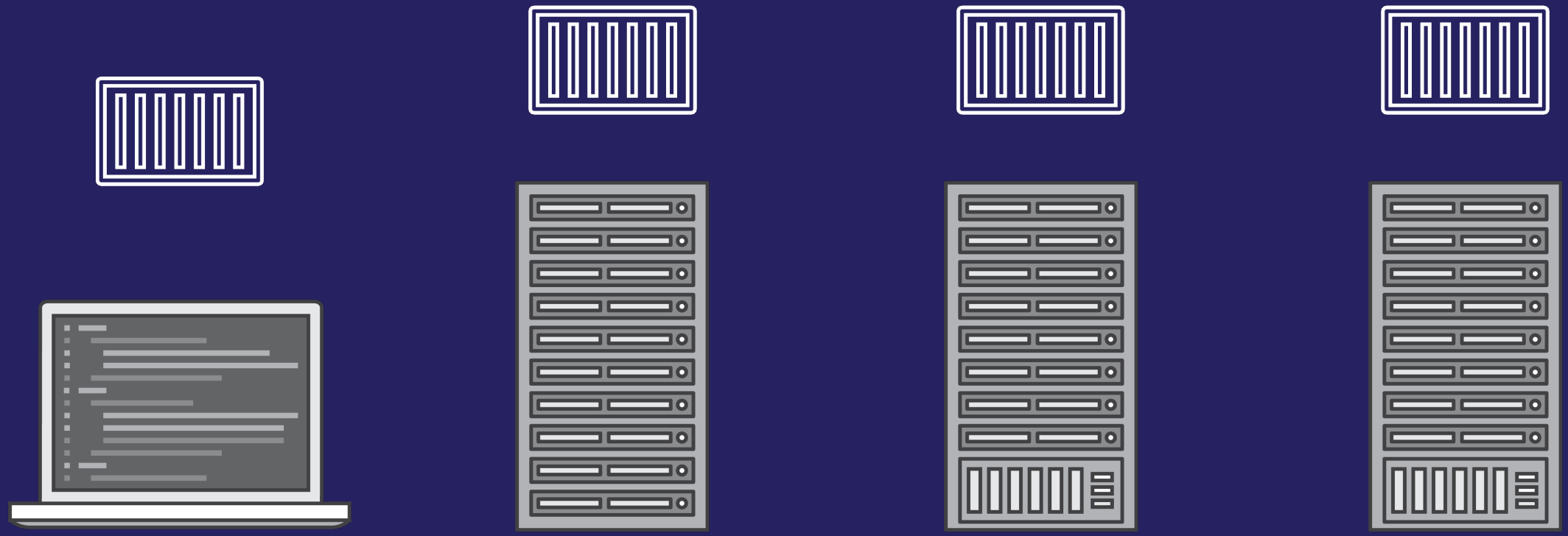


애플리케이션이 컴퓨팅 환경에서 실행되기 위해 필요한 코드와 모든 의존성을 패키징한 소프트웨어 단위

컨테이너



서로 다른 환경, 하지만 일관된 배포



Local Laptop

Staging / QA

Production

On-Prem

Docker Component

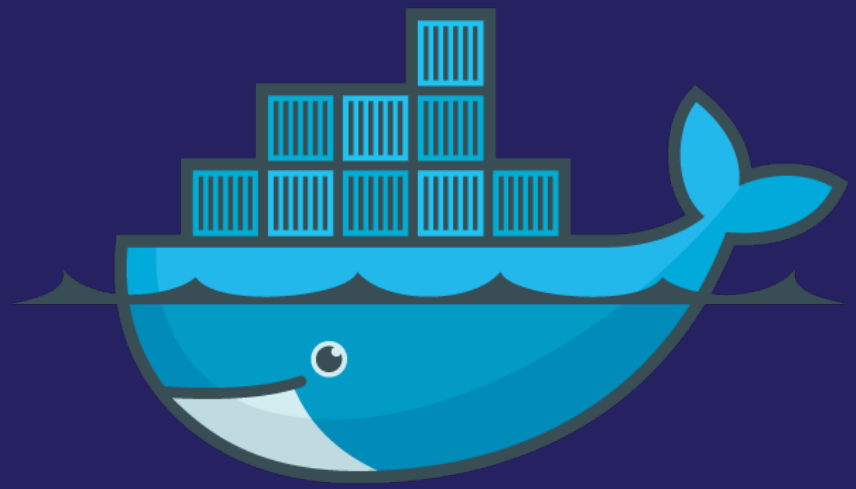
Docker

경량의 컨테이너 가상화 기술

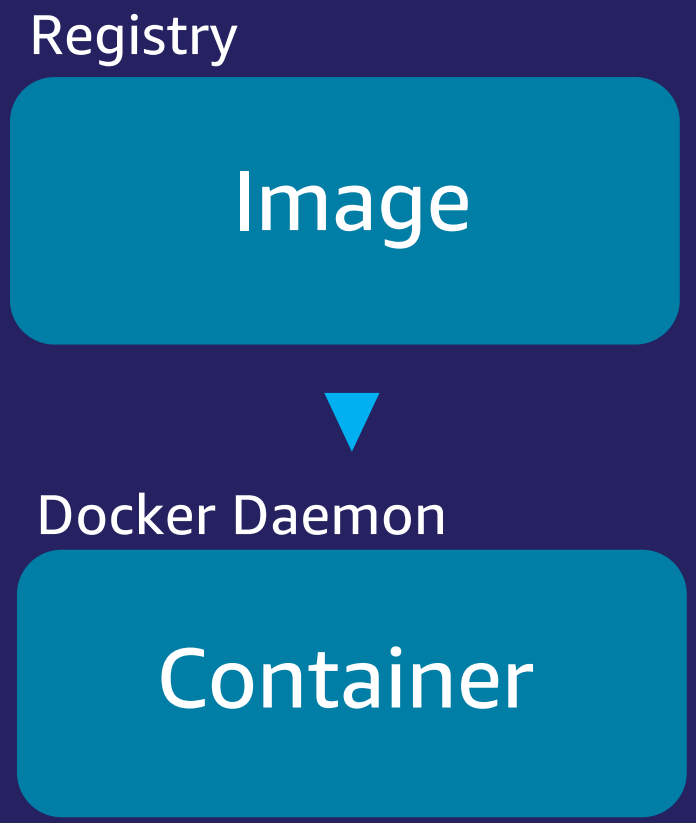
애플리케이션을 배포하고 관리하는 도구

Apache 2.0 license

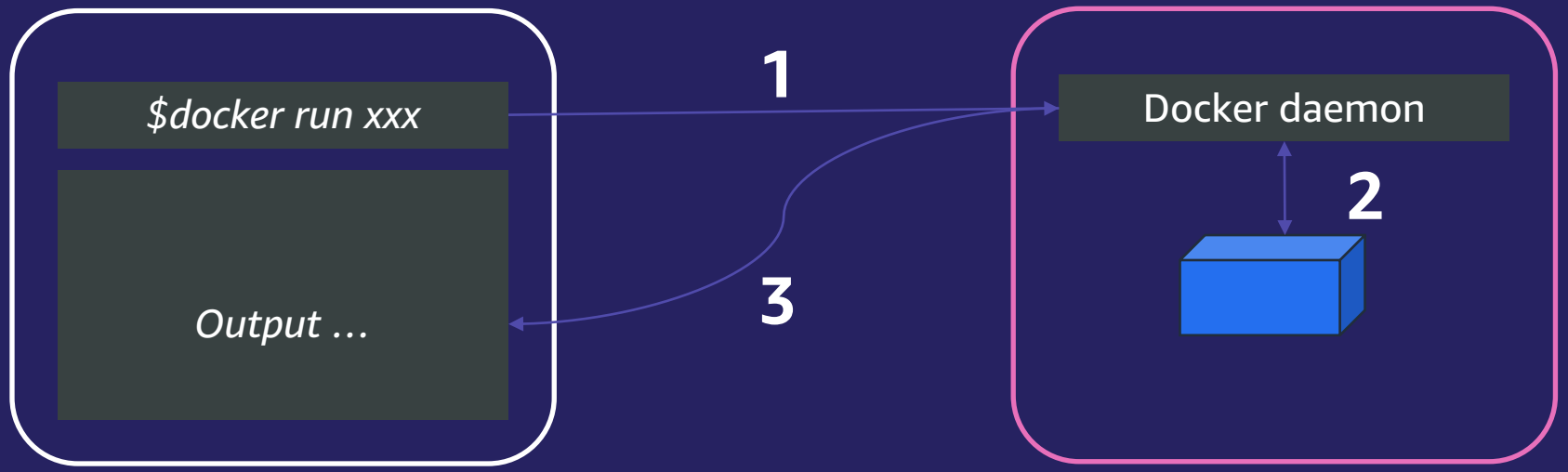
Built by Docker, Inc.



Docker 주요 용어



Docker Client and Daemon



Client

Docker 데몬 API를 이용 할 수 있도록 CLI를 제공

Host

Docker 데몬 API를 이용 할 수 있도록 CLI를 제공

Docker Image

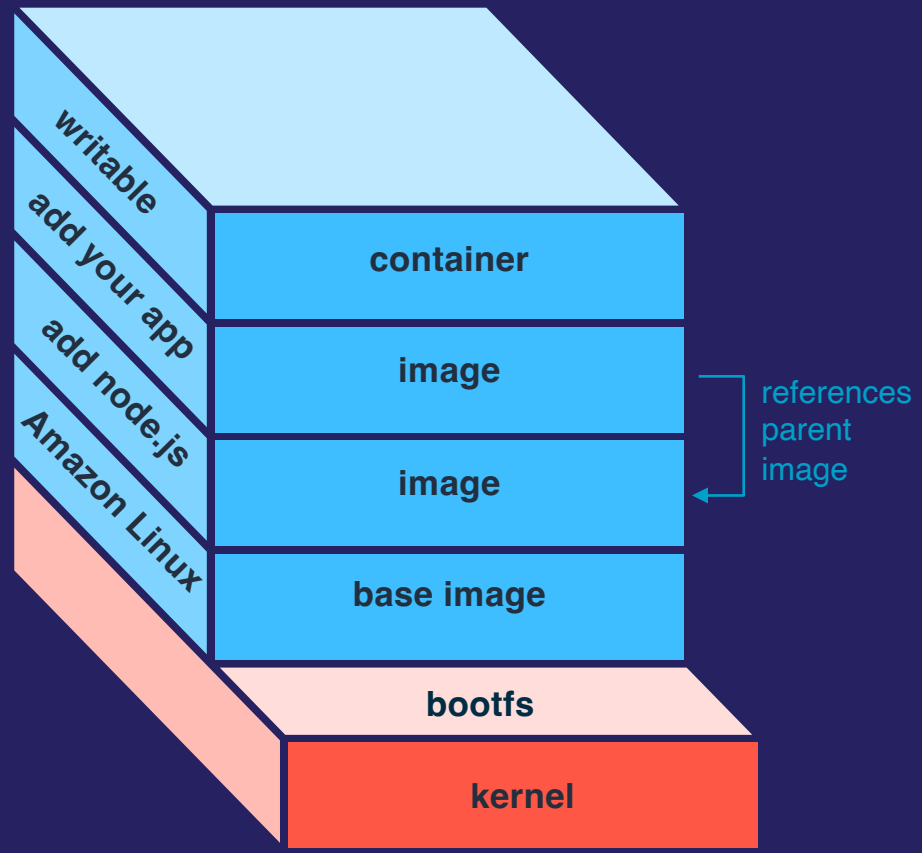
Read-only template

컨테이너를 구동하는데 사용됨

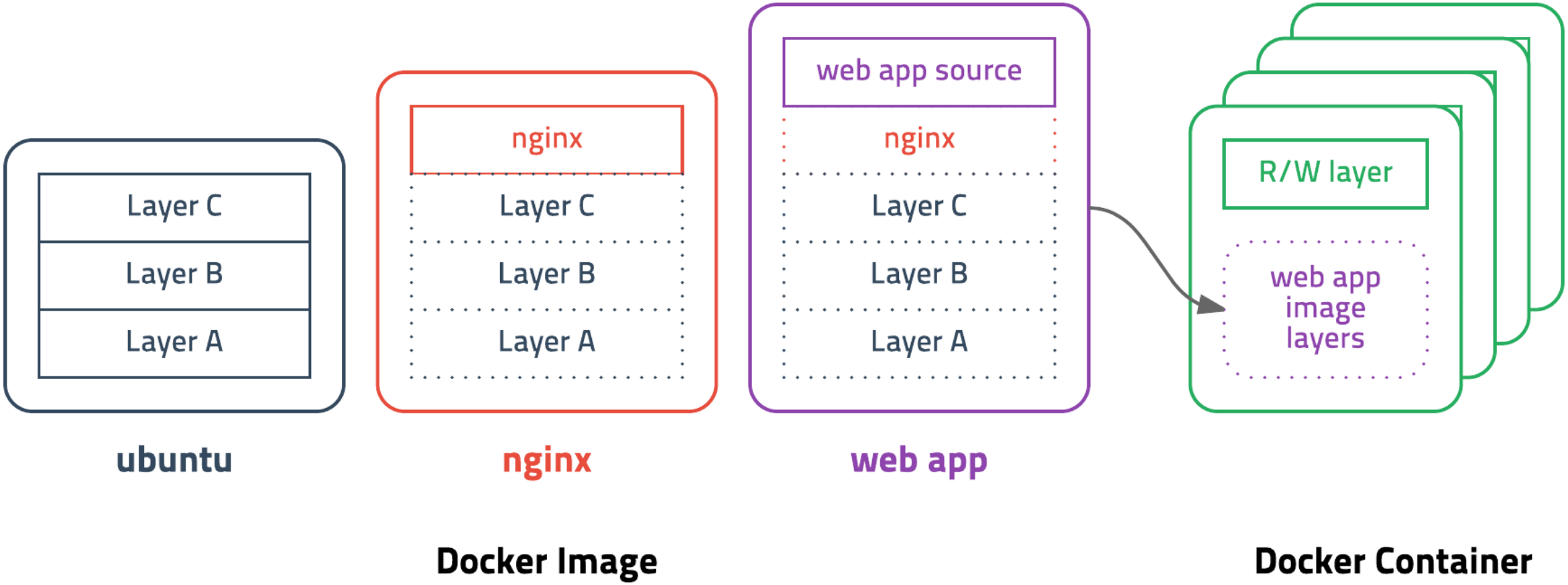
Union F/S를 이용, 다른 레이어들을 하나의 이미지로 합침

Base image위에 사용자 instruction 레이어들이 올라감

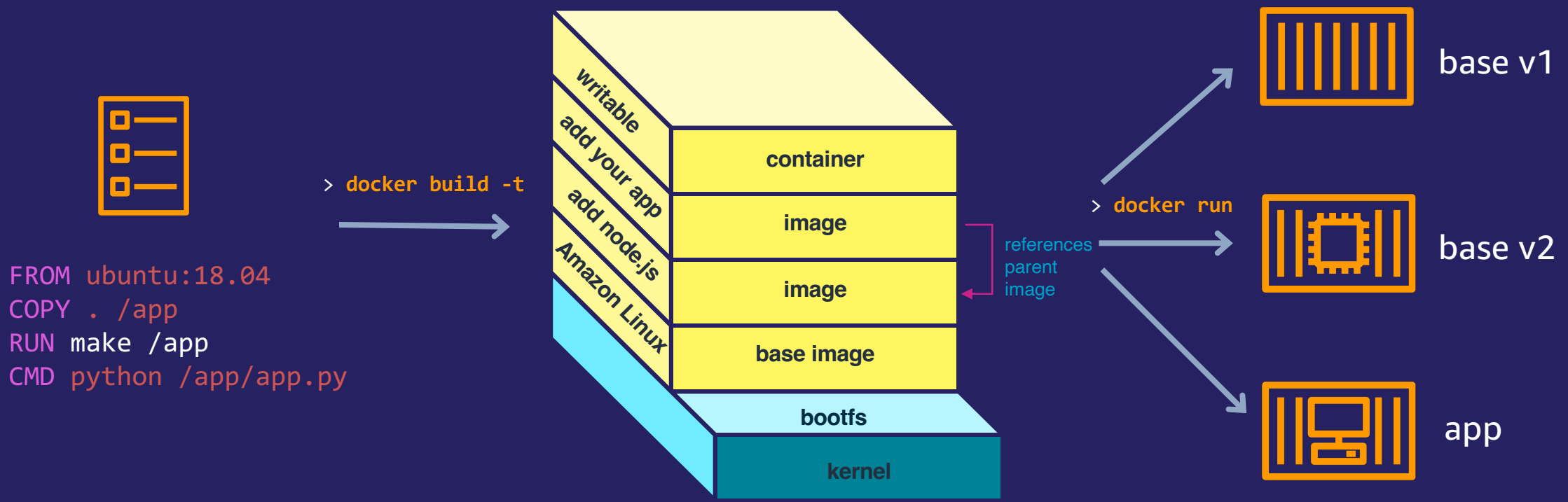
사용자 Instruction은 Dockerfile로 작성



Docker layer



Docker Image 생성 - Dockerfile



See also “Best practices for writing Dockerfiles” by Docker, Inc.:
https://docs.docker.com/build/building/best-practices/

Container Registry – Docker Hub

https://hub.docker.com

my.priviate.image:5000/hello-app:123 } 사용자가 관리하는 저장소

docker.io/library/ubuntu:18.04 } Docker 기본 저장소

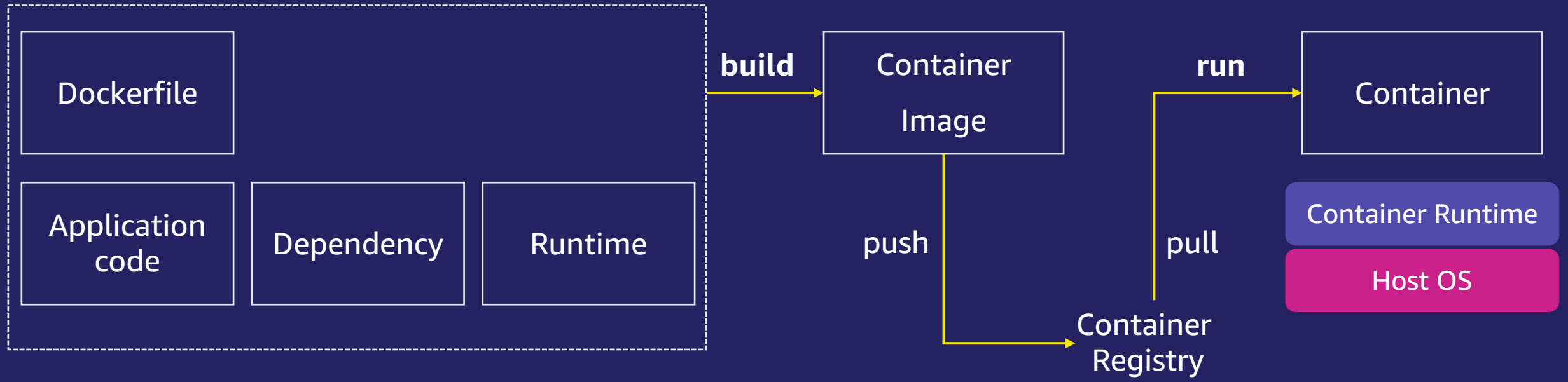
Image name

Image tag

0123456789123.dkr.ecr.us-west-2.amazonaws.com/test-repo/test-img:1

Docker Workflow

Workflow



Docker Workflows Example

Pull an image from Docker Hub

Build an image from a Dockerfile

Push an image

Run a container

Inspect a container

Stop a container

Remove a container

Remove an image

Pull an Image From Docker Hub

```
# Get the latest Ubuntu image
$ docker pull ubuntu

# Get a specific version (18.04) of the Ubuntu image
$ docker pull ubuntu:18.04

# List local images
$ docker images
```

Run a Container - Interactive

```
# Run a container from the Ubuntu image and execute bash
$ docker run -t -i ubuntu /bin/bash

# Run a container with a specific name
$ docker run -t -i --name my_container ubuntu /bin/bash

# List running containers
$ docker ps
```

Run a Container - Daemon

Run a container from the Ubuntu image and execute an infinite loop

```
$ docker run -d --name my_daemon_container ubuntu /bin/bash -c "while true; do echo hello world; sleep 1; done"
```

Get logs for daemonized container

```
$ docker logs -f my_daemon_container
```

Build an Image From a Dockerfile

```
FROM python:3.8-slim
# Set the working directory to /app
WORKDIR /app
# Copy the current directory contents into the container at /app
ADD . /app
# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt
# Make port 80 available to the world outside this container
EXPOSE 80
# Define environment variable
ENV NAME World
# Run app.py when the container launches
CMD ["python", "app.py"]
```

Build an Image From a Dockerfile

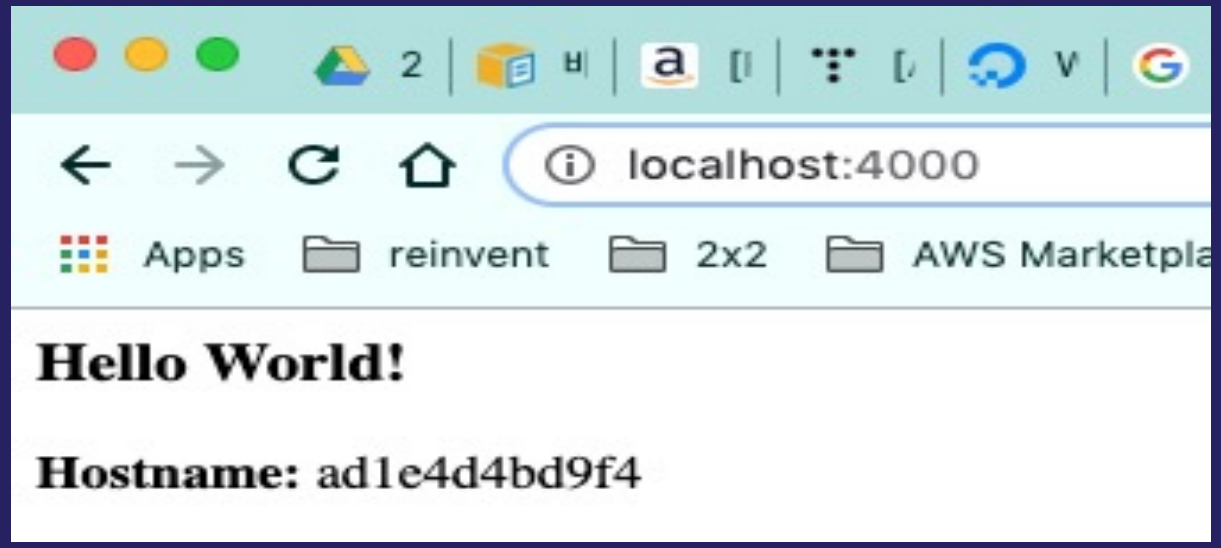
```
# Build an image from a Dockerfile and tag it
$ docker build -t <username>/mypython:v1 /path/to/Dockerfile

# Log into Docker Hub
$ docker login

# Publish my image to Docker Hub
$ docker push <username>/mypython:v1
```

Run a Container – user app

Run a container from the user custom image and execute

```
$ docker run -p 4000:80 --name mypython
<username>/mypython:v1
```

Access url in browser



Inspect a Container

```
# Return low-level information on a container
$ docker inspect mypython
```

```
# Return true if the container is running
$ docker inspect --f '{{.State.Running}}' mypython
```

Stop a container and Remove a container

```
# Stop a container
$ docker stop mypython
```

```
# Remove a container
$ docker rm mypython
```

Dockerfile 문법 알아보기 (1)

FROM	베이스 이미지 지정
RUN	베이스 이미지에 새로운 레이어를 추가해 커맨드를 실행하고 결과를 빌드 이미지에 반영
CMD	컨테이너 시작시 실행할 커맨드 설정
LABEL	이미지에 레이블 지정
EXPOSE	컨테이너에서 노출하는 포트 번호 설정
ENV	환경 변수 설정
ADD	이미지에 파일 복사
COPY	이미지에 파일 복사
ENTRYPOINT	컨테이너 시작시 실행할 커맨드 설정

https://docs.docker.com/engine/reference/builder/

Dockerfile 문법 알아보기 (2)

VOLUME	볼륨이 마운트 될 위치 설정
USER	커맨드를 실행할 때 사용자 ID 설정
WORKDIR	커맨드를 실행할 때 작업 디렉터리 설정
ARG	빌드 시에만 사용되는 변수 설정
ONBUILD	이 이미지를 베이스로 빌드할 때 커맨드가 실행되도록 하기
STOPSIGNAL	컨테이너를 중지시킬 때의 시그널 번호 설정
HEALTHCHECK	헬스 체크를 위한 커맨드 설정
SHELL	커맨드 실행할 때 설정

https://docs.docker.com/engine/reference/builder/

Docker CLI 살펴 보기 – 이미지 관리

docker build	Dockerfile로 이미지 빌드
docker images	이미지 리스트 보기
docker image inspect	이미지 상세 정보 보기
docker pull	레지스트리로부터 이미지 가지고 오기
docker push	레지스트리에 이미지 전송하기
docker history	이미지 생성 기록 보기
docker rmi	이미지 삭제
docker import	Docker container export로 가지고 온 것으로 이미지 만들기
docker save	이미지 tar 아카이브로 출력
docker load	Docker image save로 출력한 것(tar 아카이브)으로 이미지 로드
docker tag	기존의 이미지에 태그 붙이기
docker image prune	불필요한(예: 태그가 없는) 이미지 삭제

https://docs.docker.com/engine/reference/commandline/docker/

Docker CLI 살펴 보기 – 컨테이너 관리

docker ps	컨테이너 목록 보기
docker logs	컨테이너 로그 취득
docker exec	실행 중인 컨테이너 내부에서 커맨드 실행
docker container inspect	컨테이너의 자세한 정보 보기
docker port	컨테이너 포트 맵핑 보기
docker rm	컨테이너 삭제
docker run	새로운 컨테이너로 커맨드 실행
docker start	정지 중인 컨테이너 기동
docker stop	실행 중인 컨테이너 중지
docker restart	컨테이너 다시 시작
docker kill	실행 중인 컨테이너(Docker가 만든 PID 1 프로세스)에 신호 보내기
docker container prune	정지 중인 모든 컨테이너 삭제

https://docs.docker.com/engine/reference/commandline/docker/

Thank you!