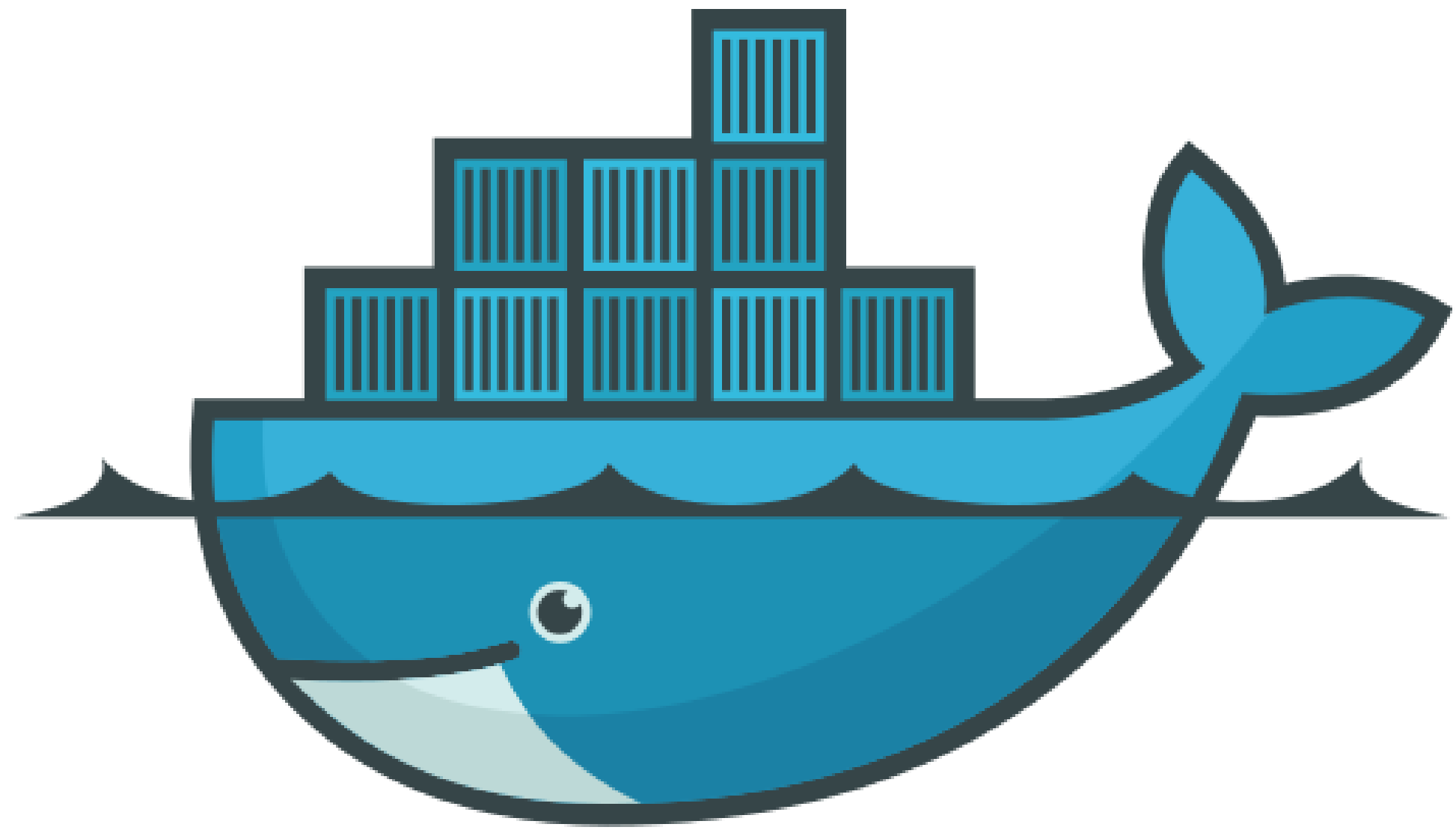


도커의 개념

도커란 무엇인가?

- 리눅스의 응용 프로그램들을 프로세스 격리 기술들을 사용해 컨테이너로 실행하고 관리하는 오픈 소스 프로젝트
- 다양한 이유로 계속 바뀌는 서버 환경과 개발 환경 문제를 해결하기 위해 등장




docker

도커의 개념

도커란 무엇인가?

- 리눅스의 응용 프로그램들을 프로세스 격리 기술들을 사용해 컨테이너로 실행하고 관리하는 오픈 소스 프로젝트
- 다양한 이유로 계속 바뀌는 서버 환경과 개발 환경 문제를 해결하기 위해 등장

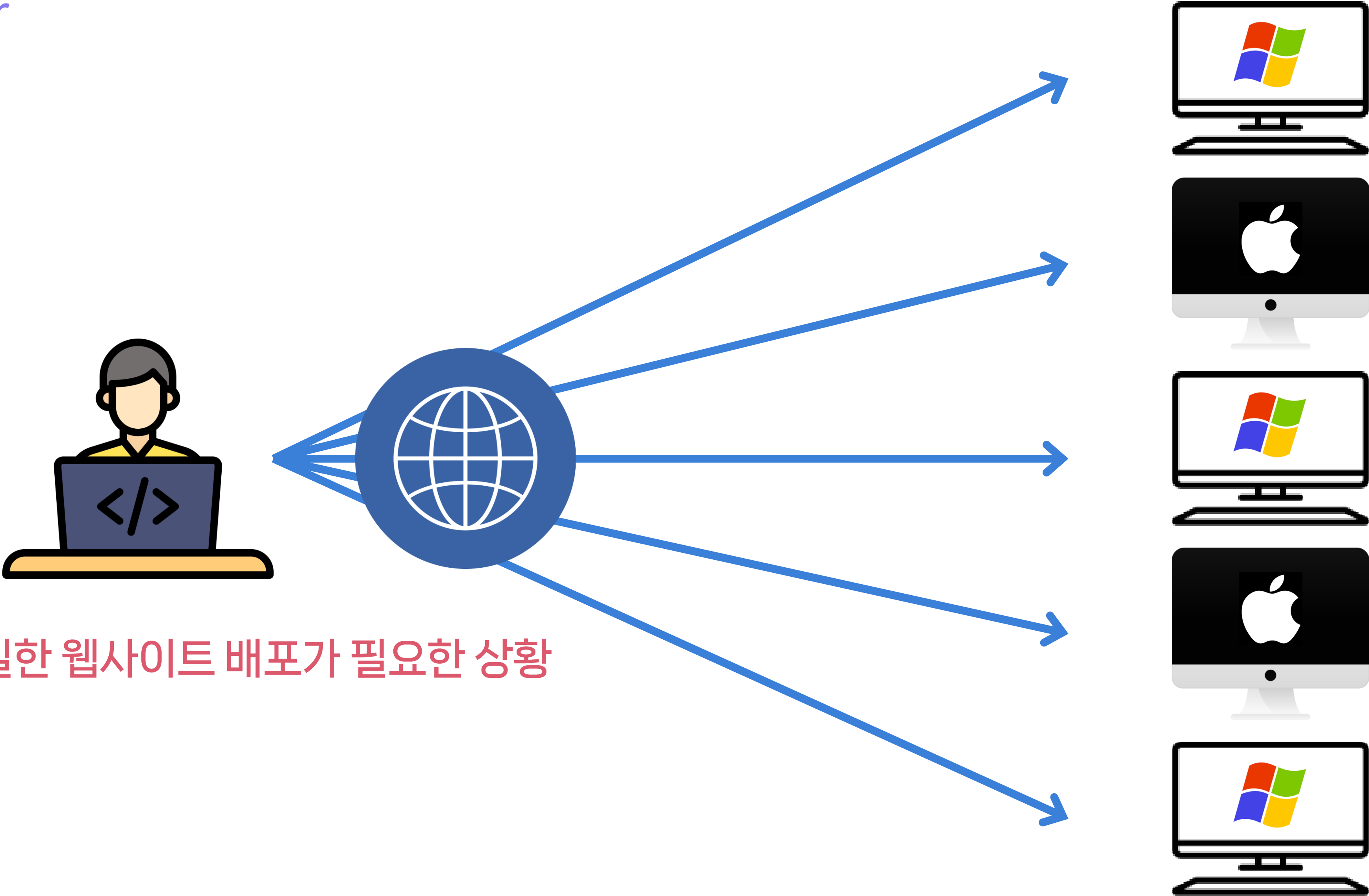


원할 때 빌려 쓰고 반납할 수 있는 작은 컴퓨터를 관리해주는 도구

docker

도커의 개념

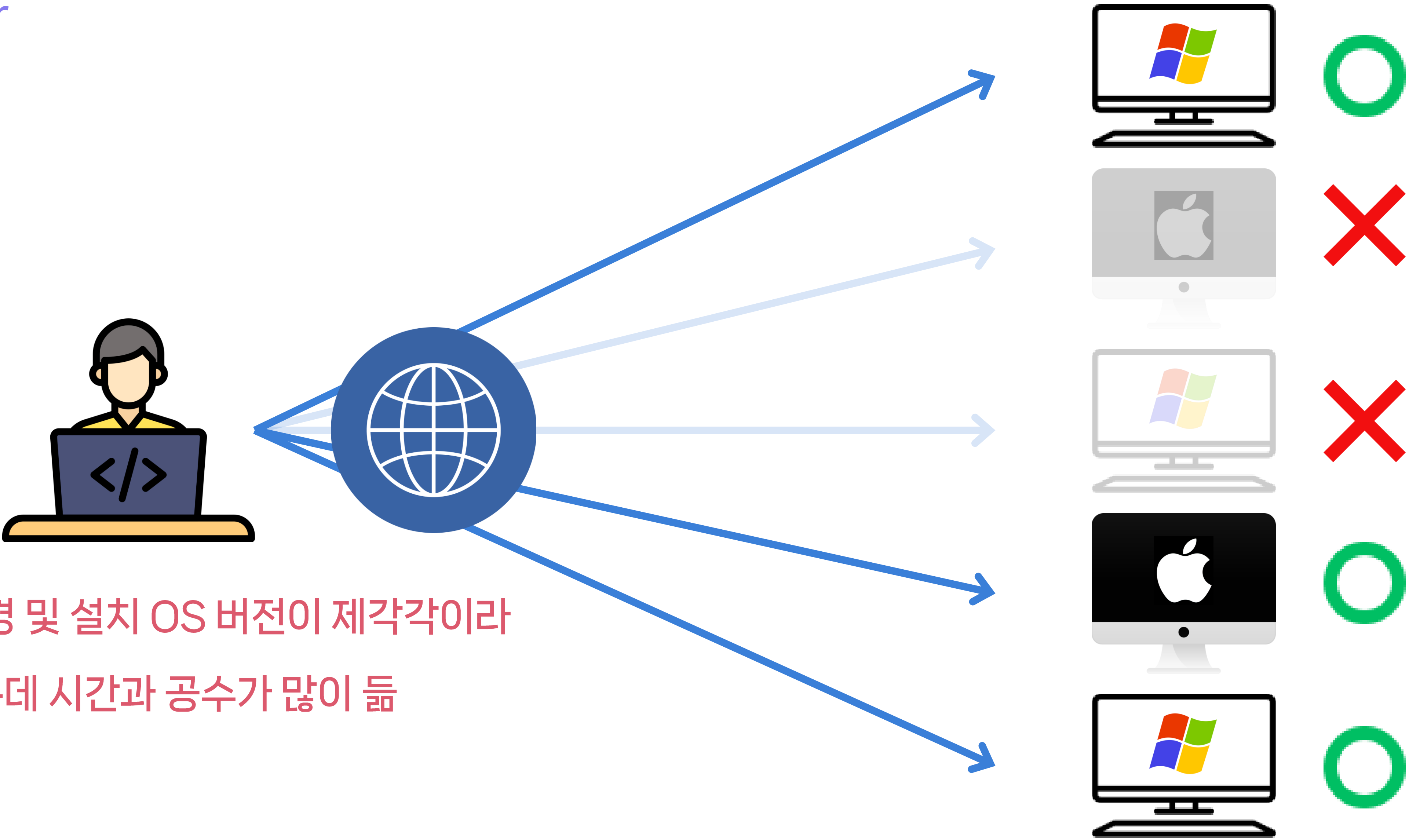
Before Docker



각 컴퓨터에 동일한 웹사이트 배포가 필요한 상황

도커의 개념

Before Docker

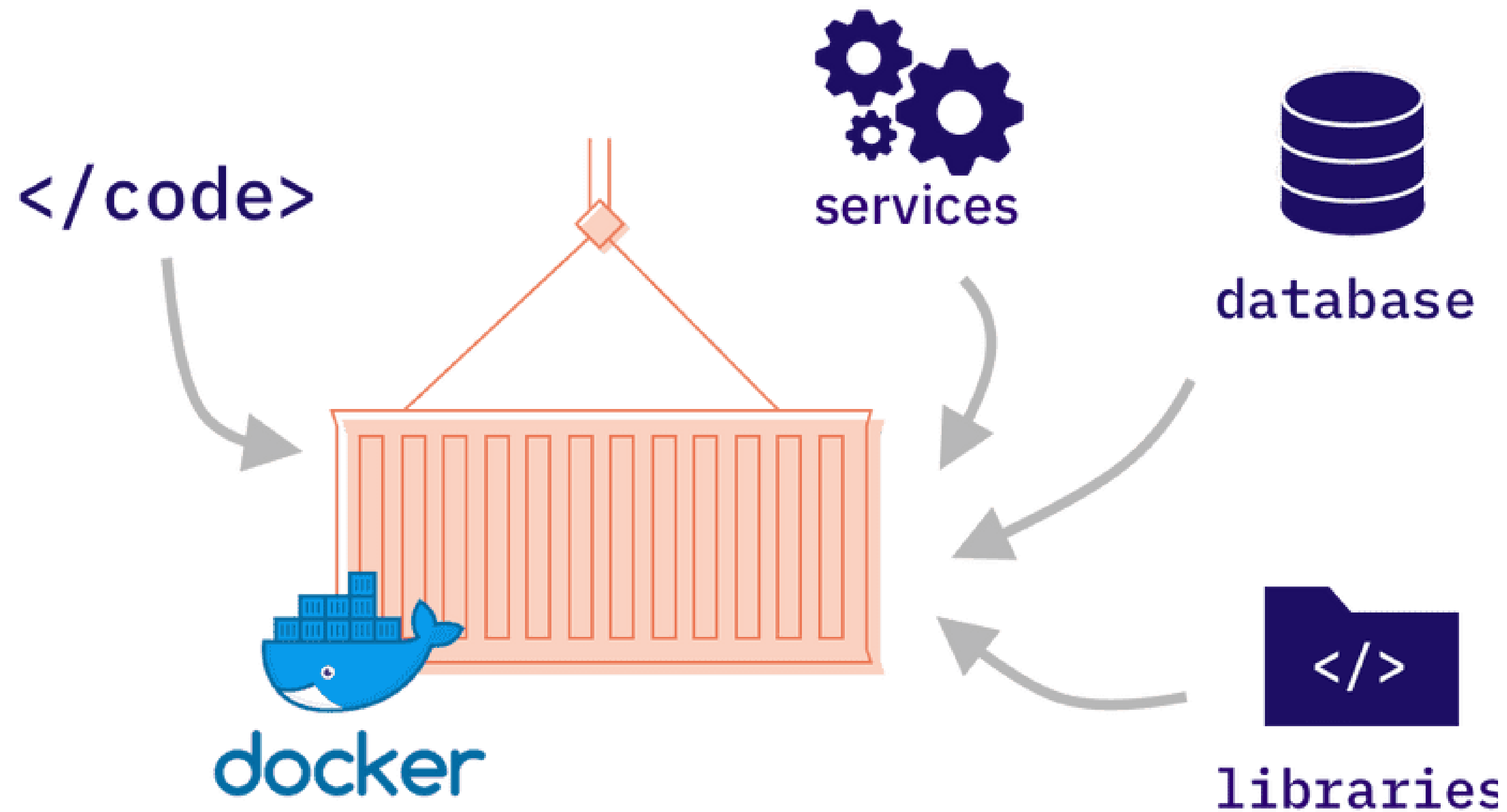


컴퓨터마다 환경 및 설치 OS 버전이 제각각이라
배포하는데 시간과 공수가 많이 듭

도커의 개념

도커 컨테이너란?


- 애플리케이션과 그 실행에 필요한 모든 것을 포함한 가볍고 독립적인 실행 환경
- 가상의 리눅스 서버라고 생각하면 이해가 조금 더 쉽다.



도커의 개념

도커란 무엇인가?

- 리눅스의 응용 프로그램들을 프로세스 격리 기술들을 사용해 컨테이너로 실행하고 관리하는 오픈 소스 프로젝트
- 다양한 이유로 계속 바뀌는 서버 환경과 개발 환경 문제를 해결하기 위해 등장



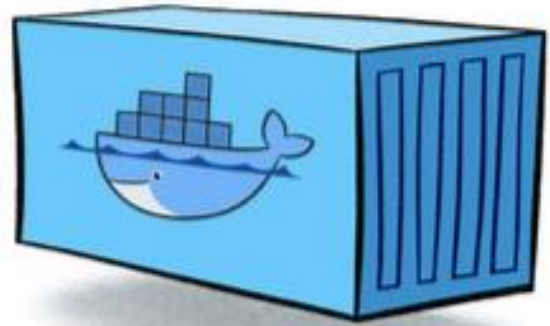
원할 때 빌려 쓰고 반납할 수 있는 작은 컴퓨터를 관리해주는 도구

-> 컨테이너!

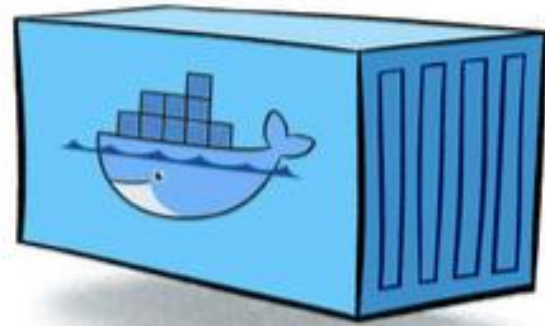
docker

도커의 개념

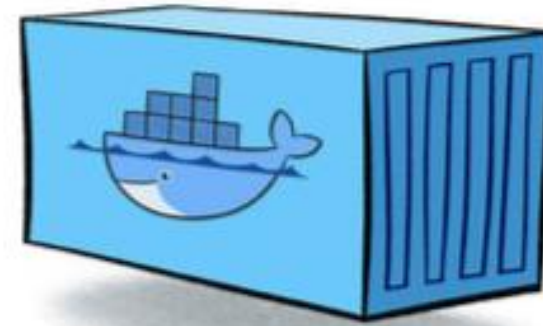
After Docker



mysql



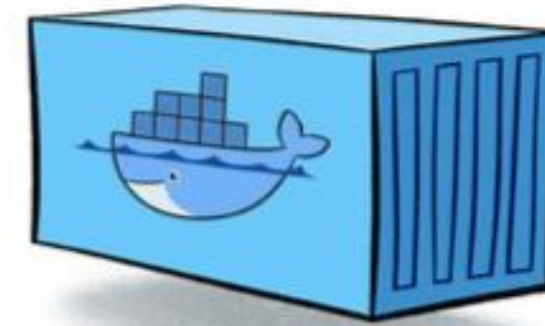
redis



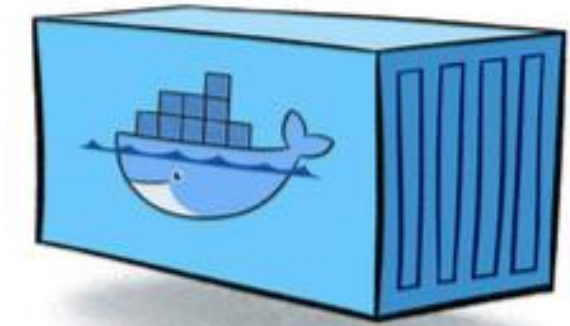
rebbitmq



Jenkins



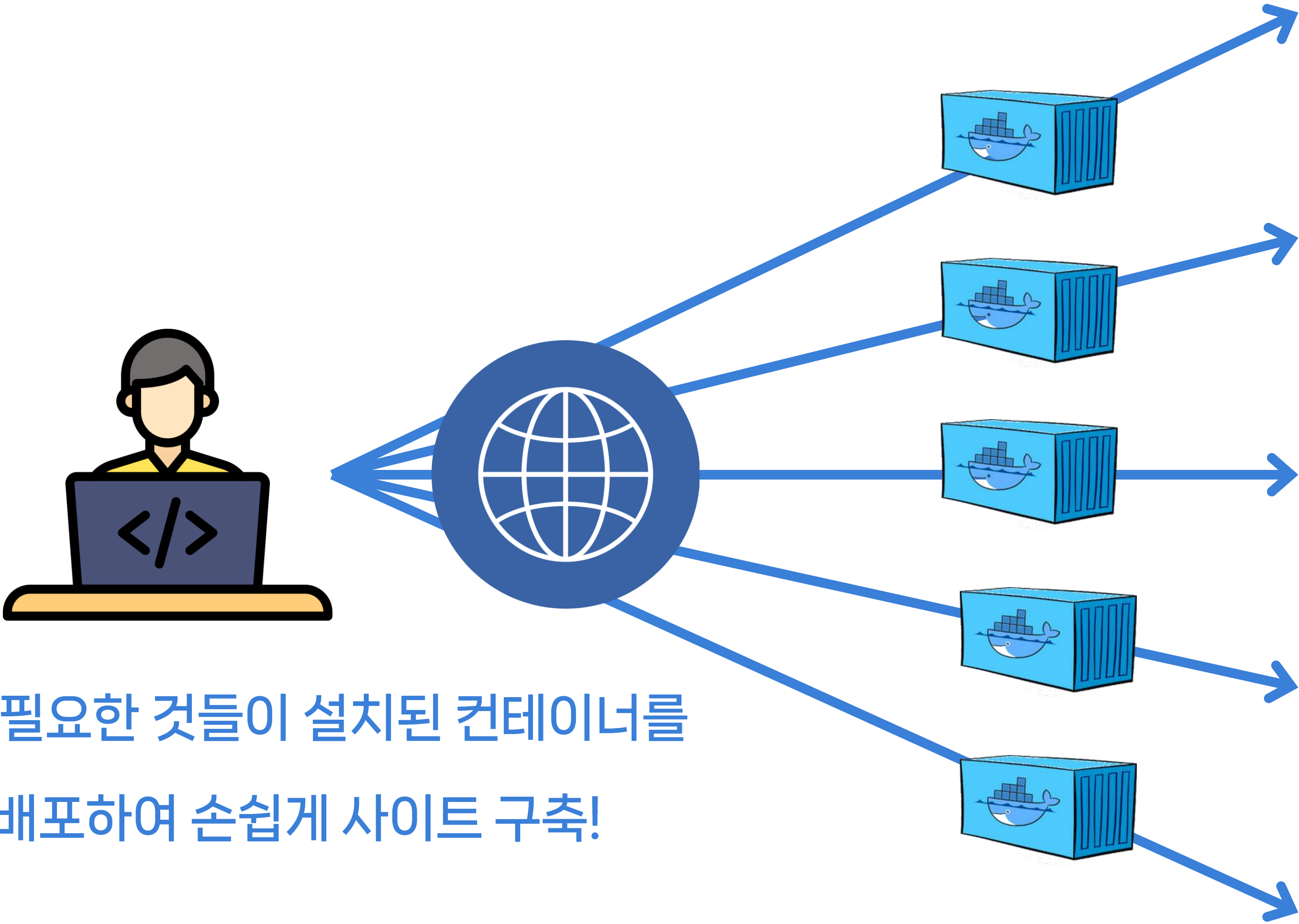
jenkins



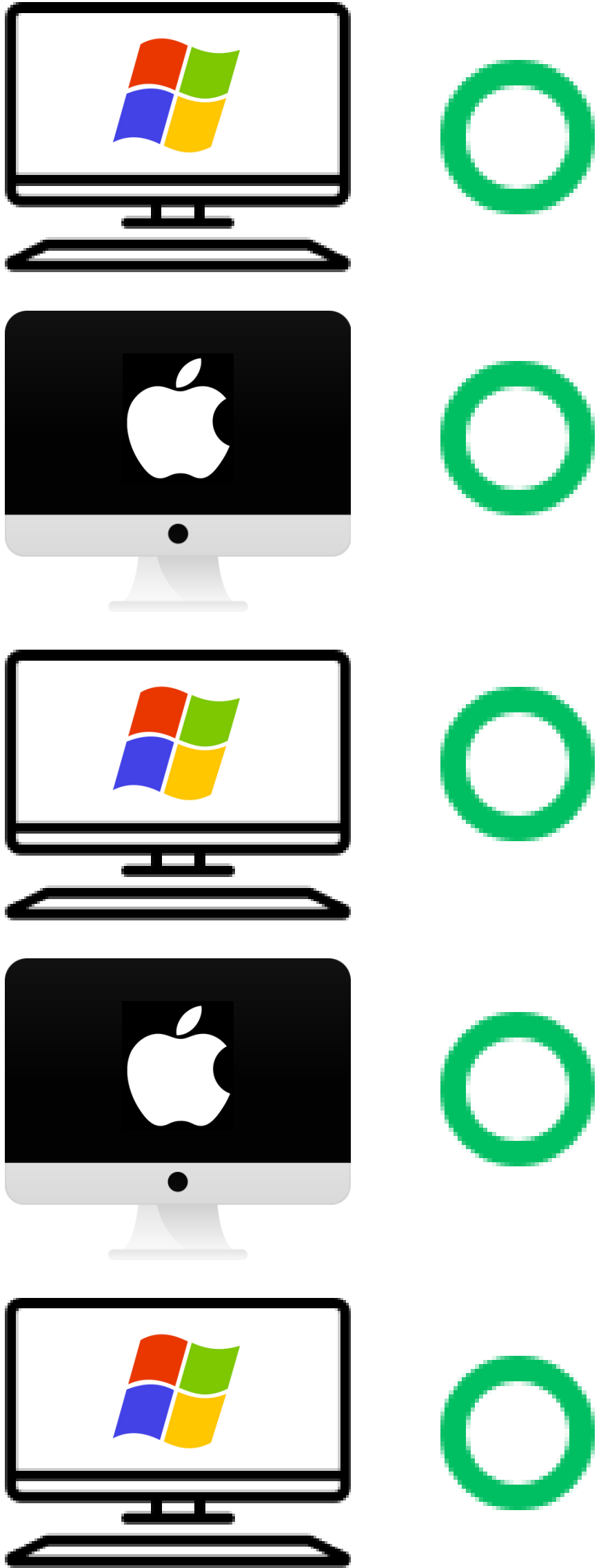
wordpress

도커의 개념

After Docker



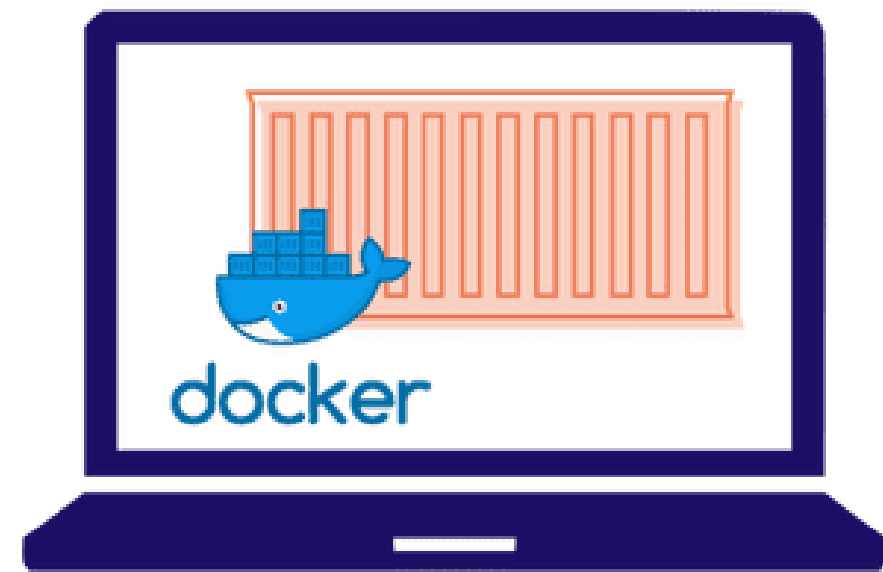
웹사이트 배포에 필요한 것들이 설치된 컨테이너를
각 컴퓨터에 배포하여 손쉽게 사이트 구축!



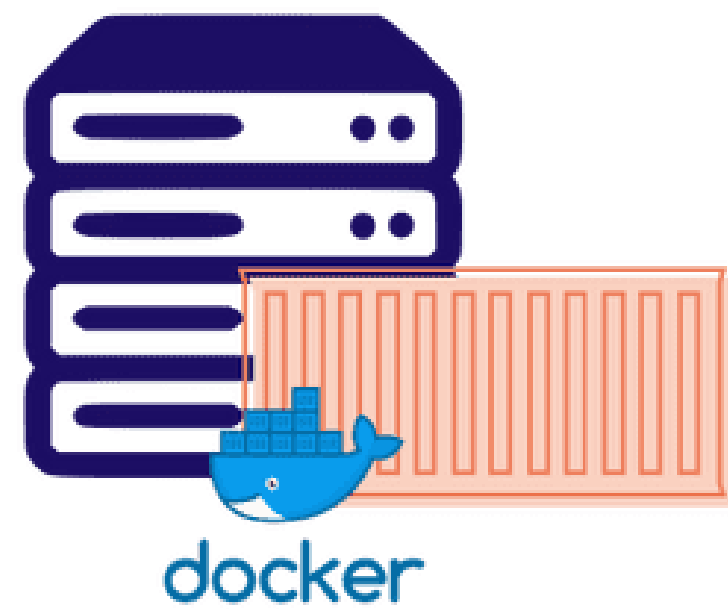
도커의 개념

After Docker

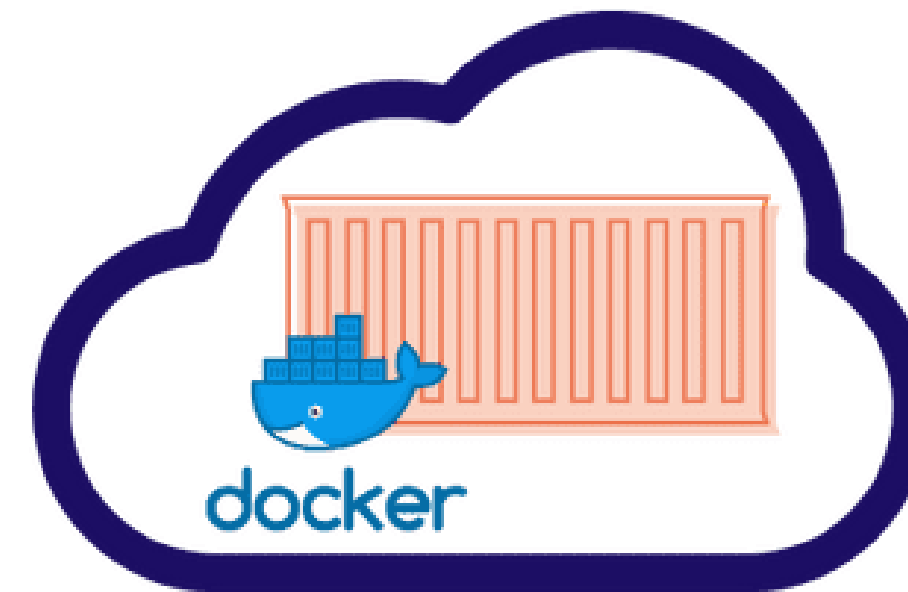
- 기기나 환경에 상관 없이 동일한 로직으로 서비스 배포가 가능



” Works well
on my laptop



” All good also on server

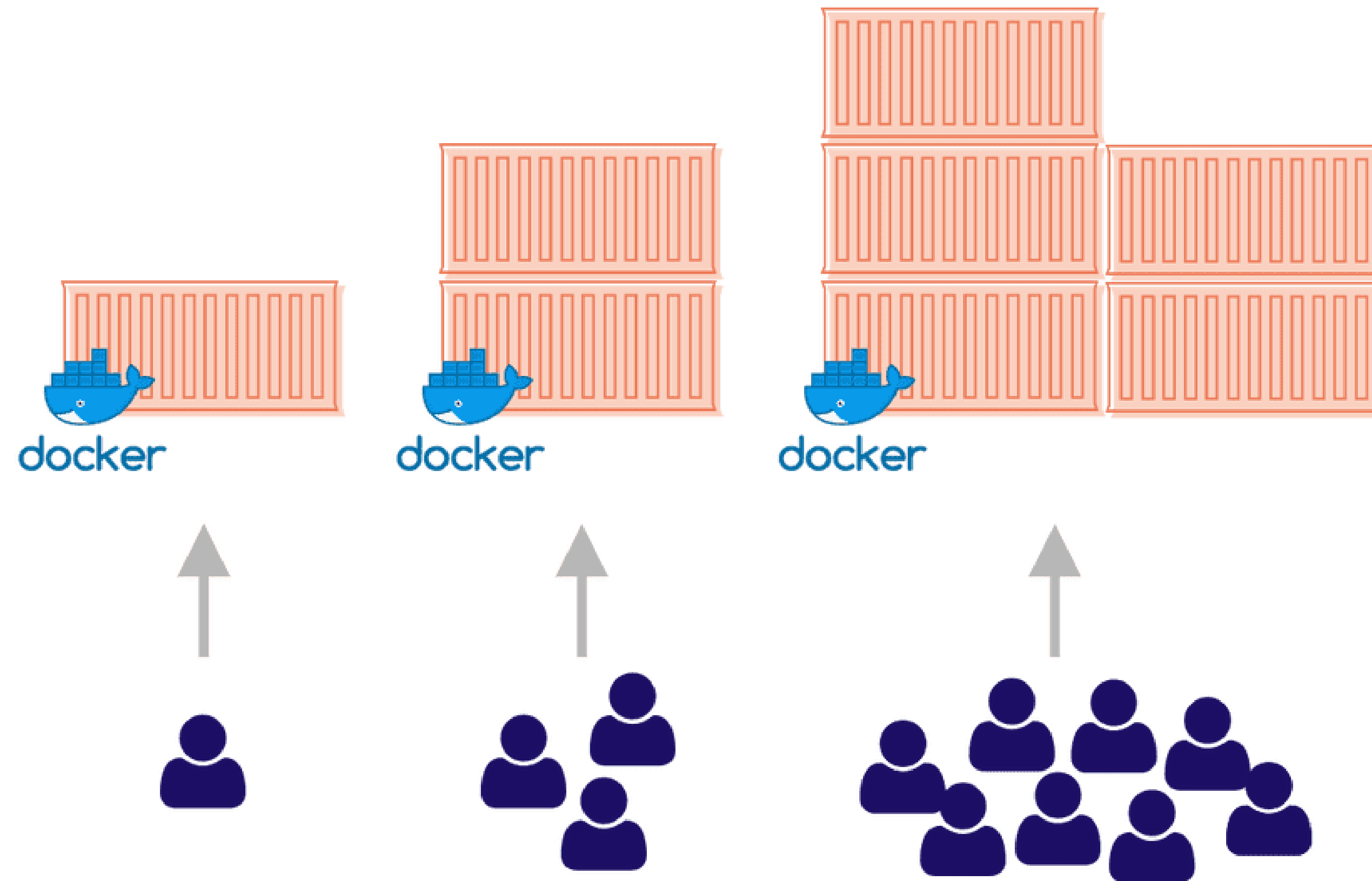


” No issues on cloud!

도커의 개념

After Docker

- 기기나 환경에 상관 없이 동일한 로직으로 서비스 배포가 가능
- 트래픽이 많아져도 빠른 대응이 가능



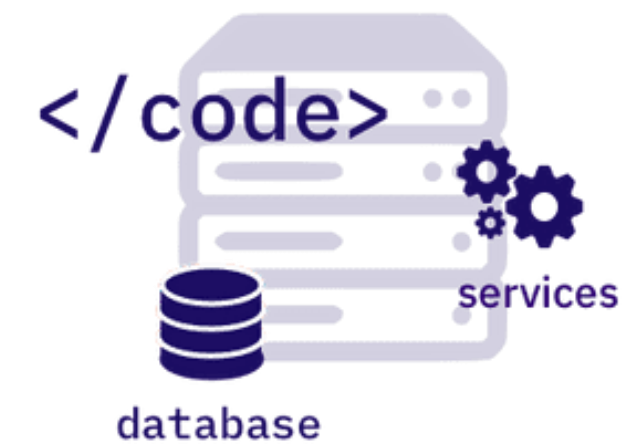
도커의 개념

컨테이너가 해결할 수 있는 것

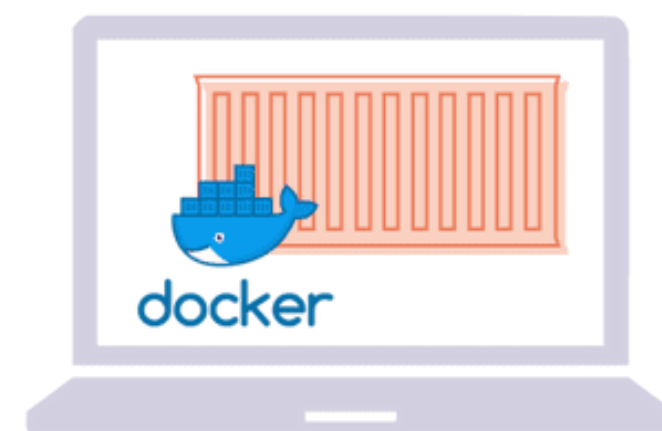
- 어느 기기에서든 동일한 개발 환경 제공



local status: OK



server status: ERROR
E: library XYZ missing



local status: OK

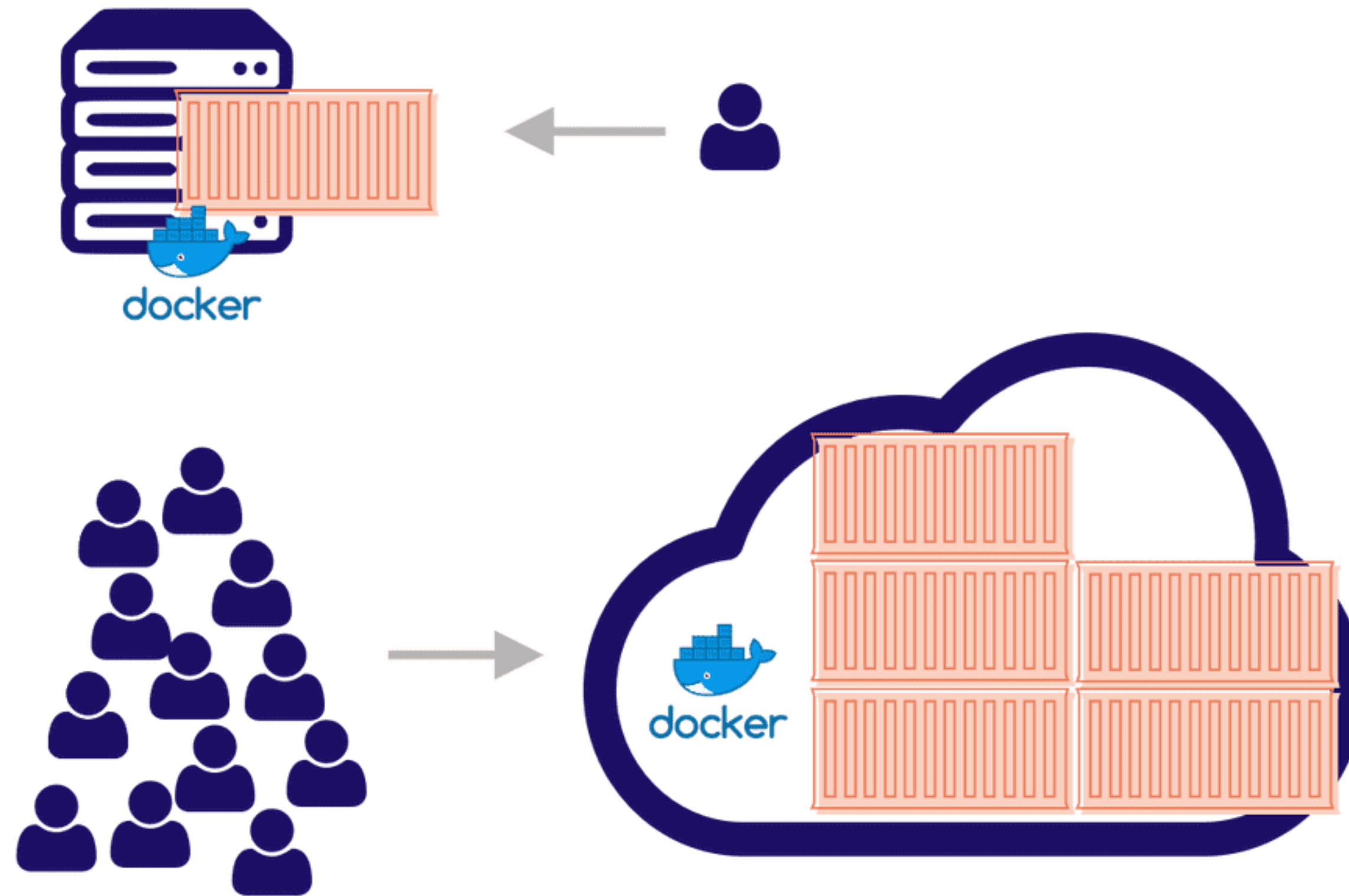


server status: OK

도커의 개념

컨테이너가 해결할 수 있는 것

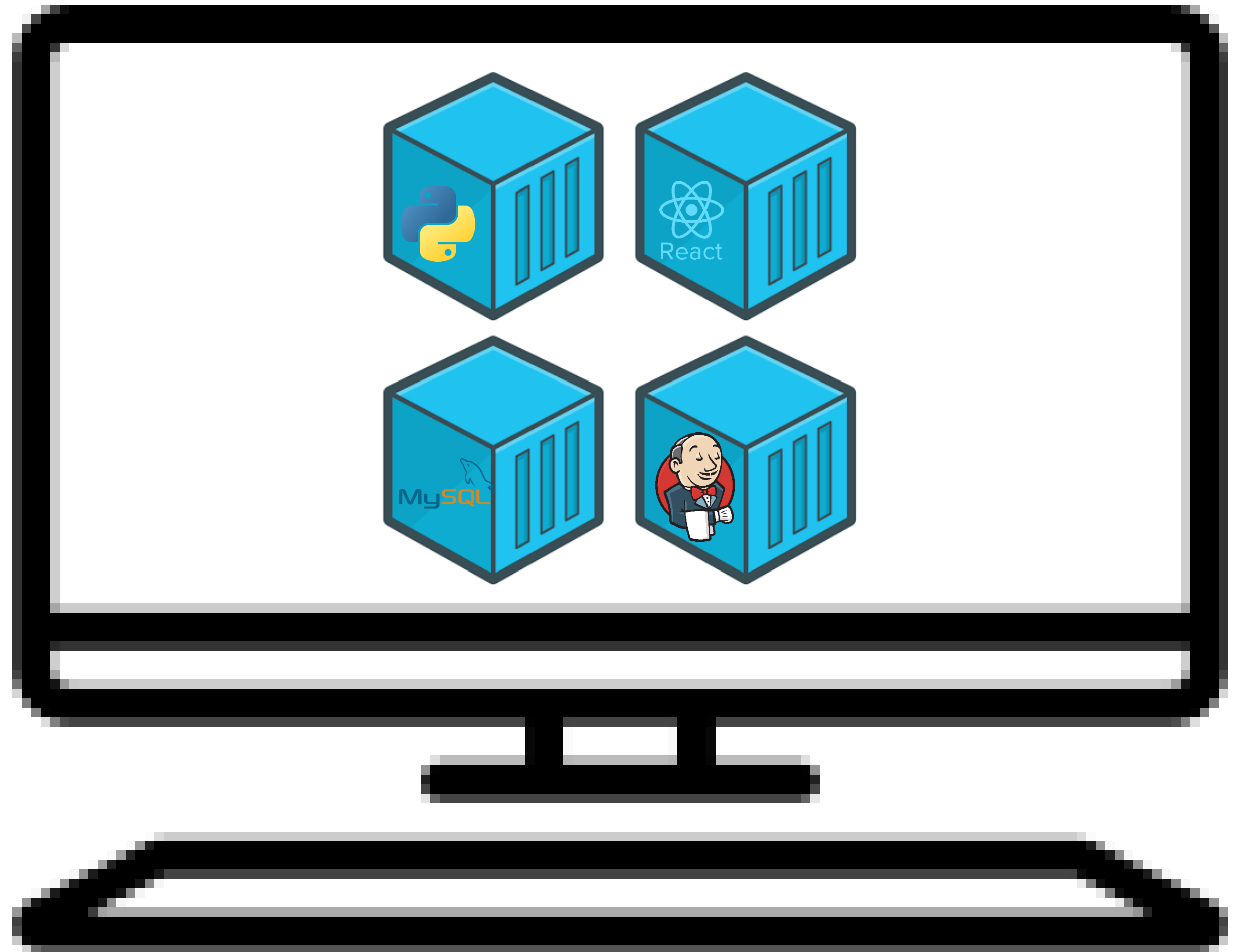
- 어느 기기에서든 동일한 개발 환경 제공
- 서비스 리소스에 대한 확장 가능성 및 유연성 보장



도커의 개념

컨테이너가 해결할 수 있는 것

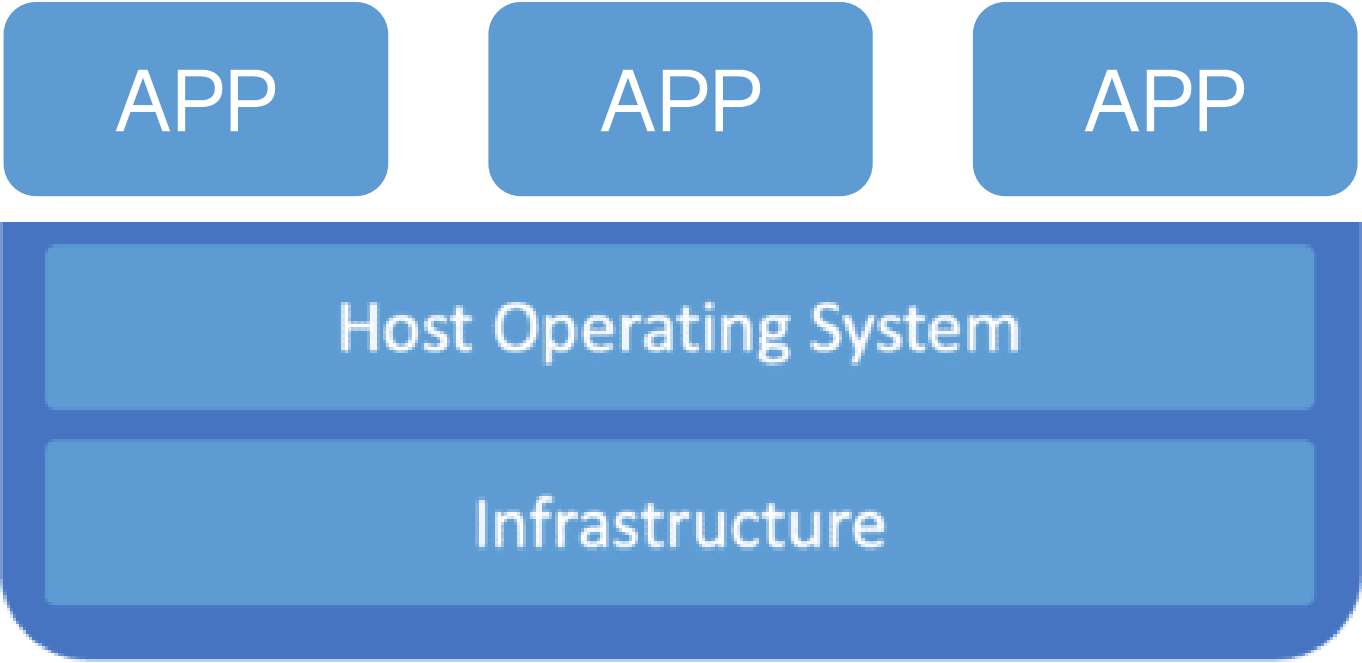
- 어느 기기에서든 동일한 개발 환경 제공
- 서비스 리소스에 대한 확장 가능성 및 유연성 보장
- 한 대의 물리 서버에 여러가지 서비스 배포 가능



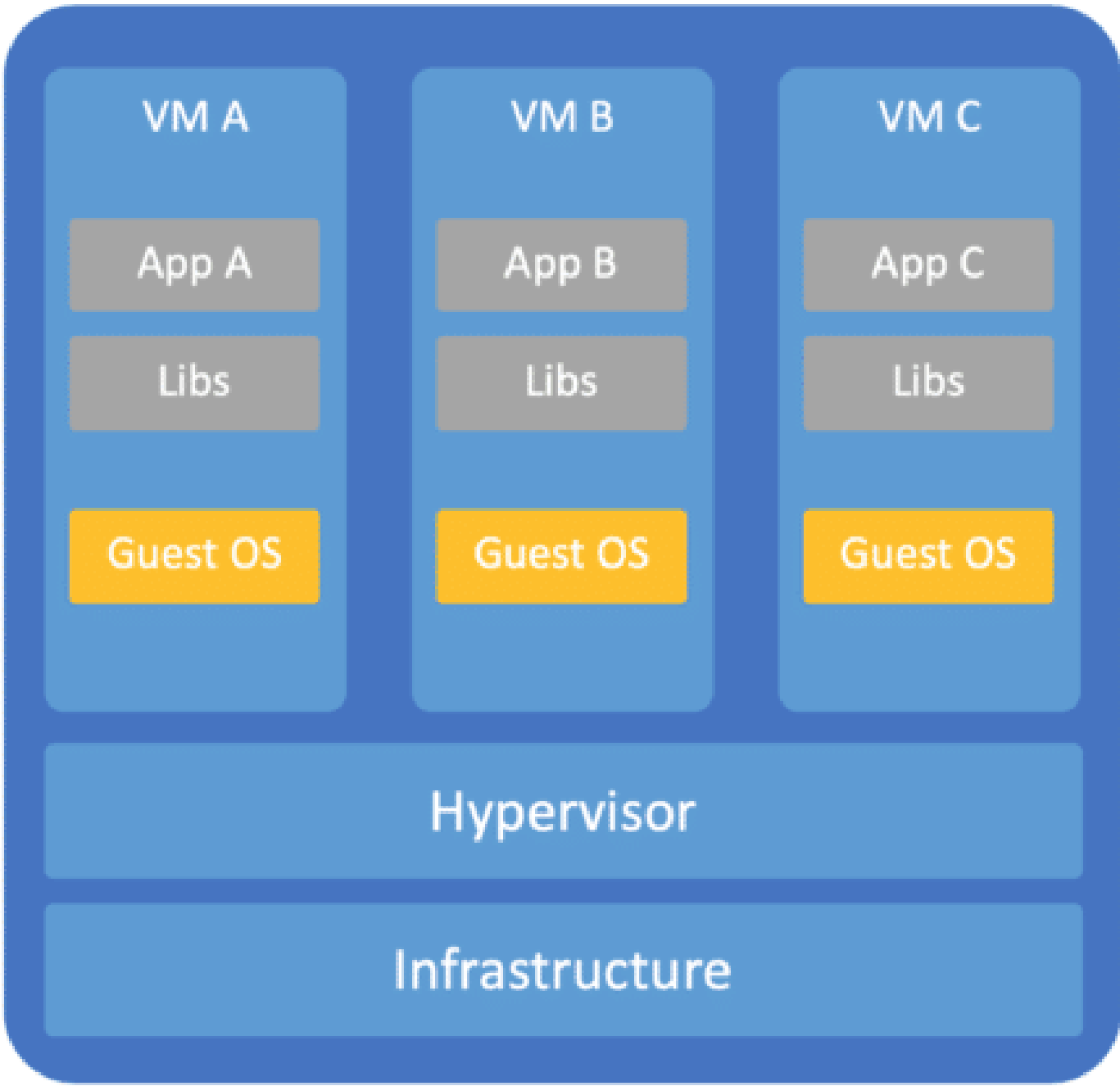
도커의 개념

도커의 구성

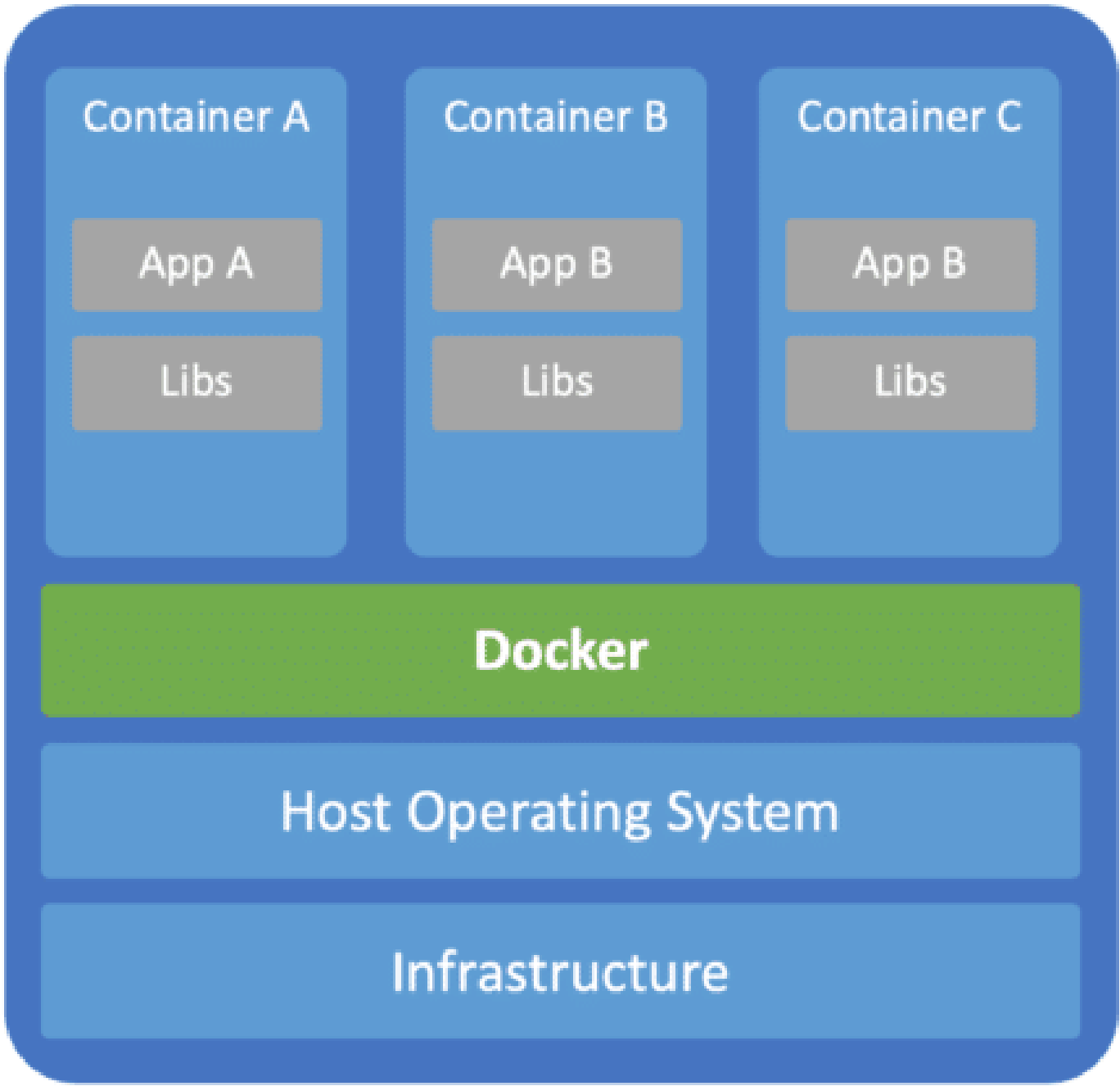
Traditional System



Virtual Machines

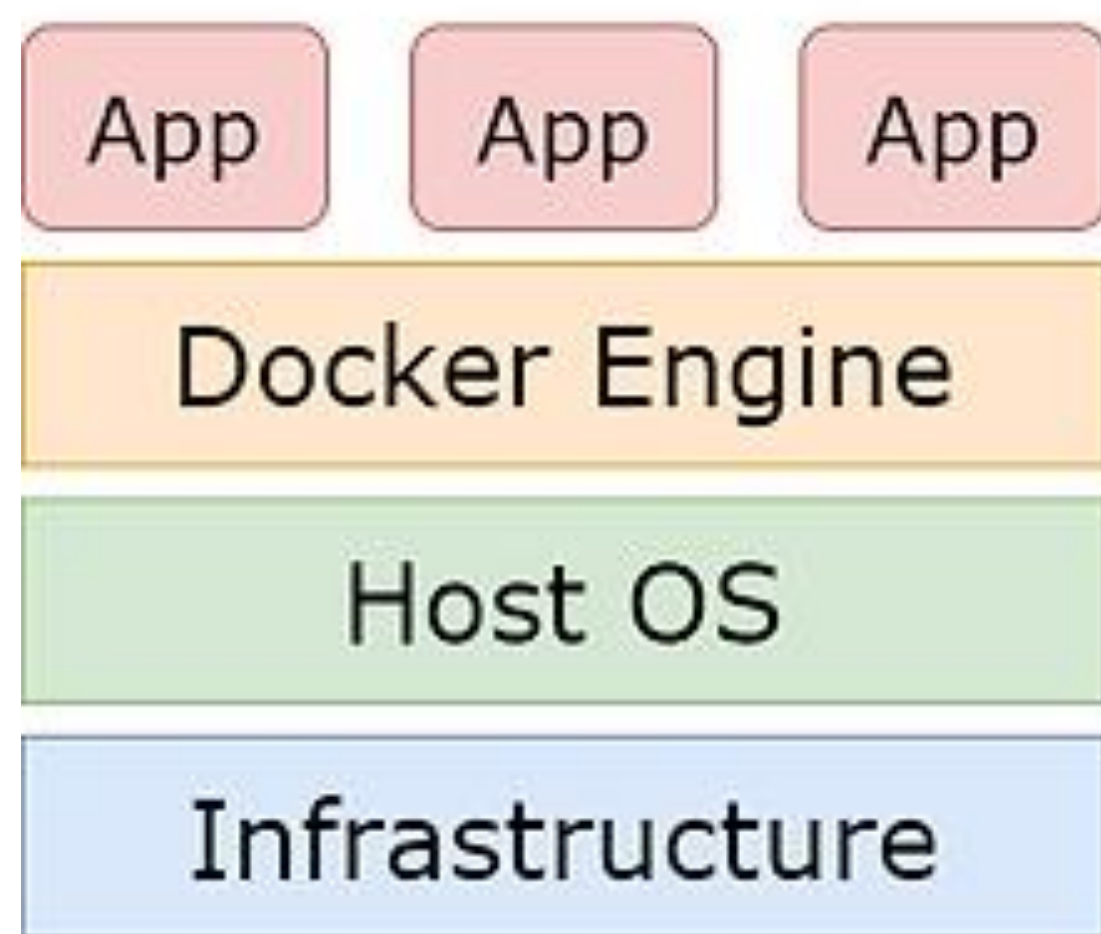


Container

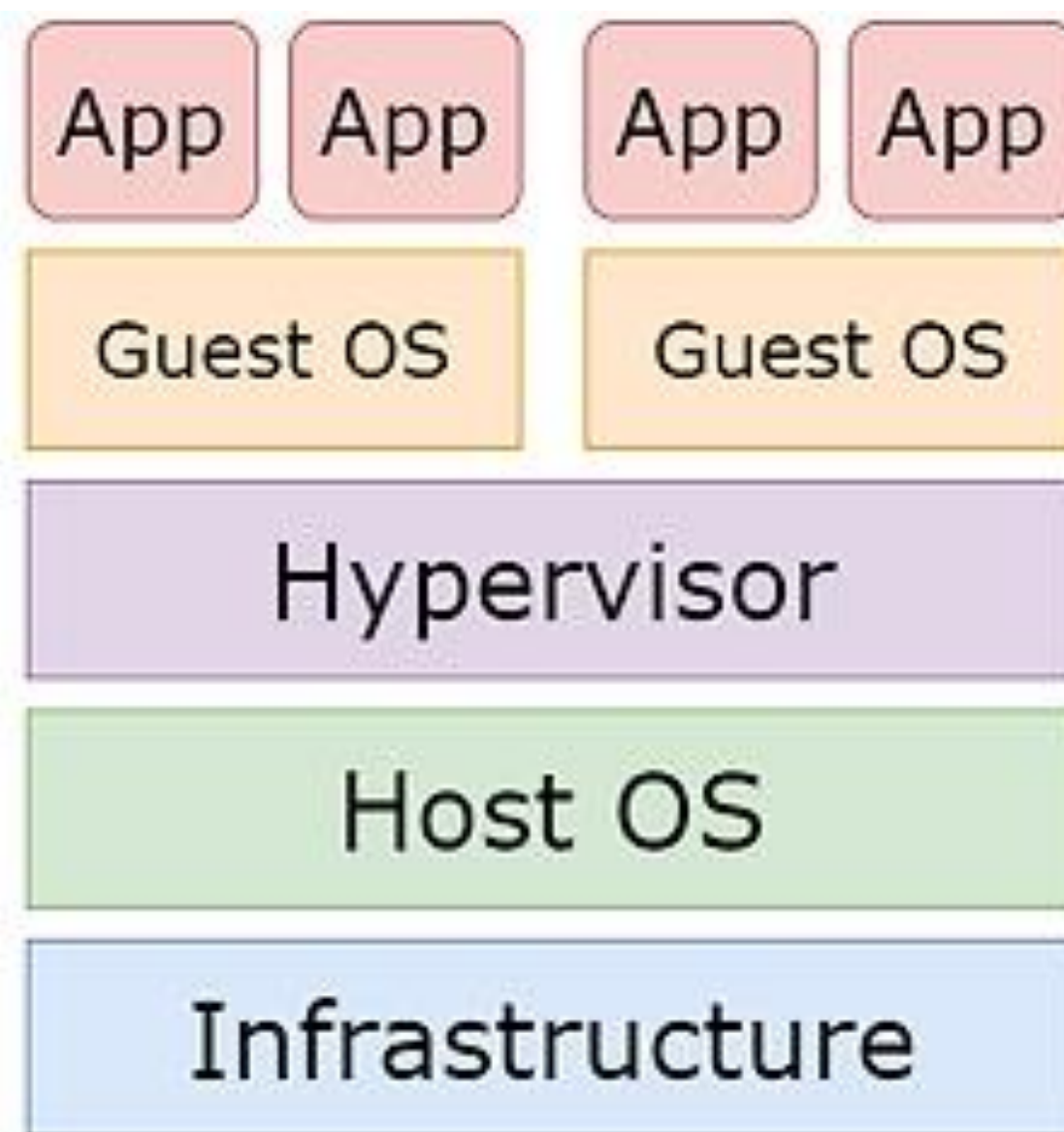


도커의 개념

도커의 구성



Container



Virtual Machine

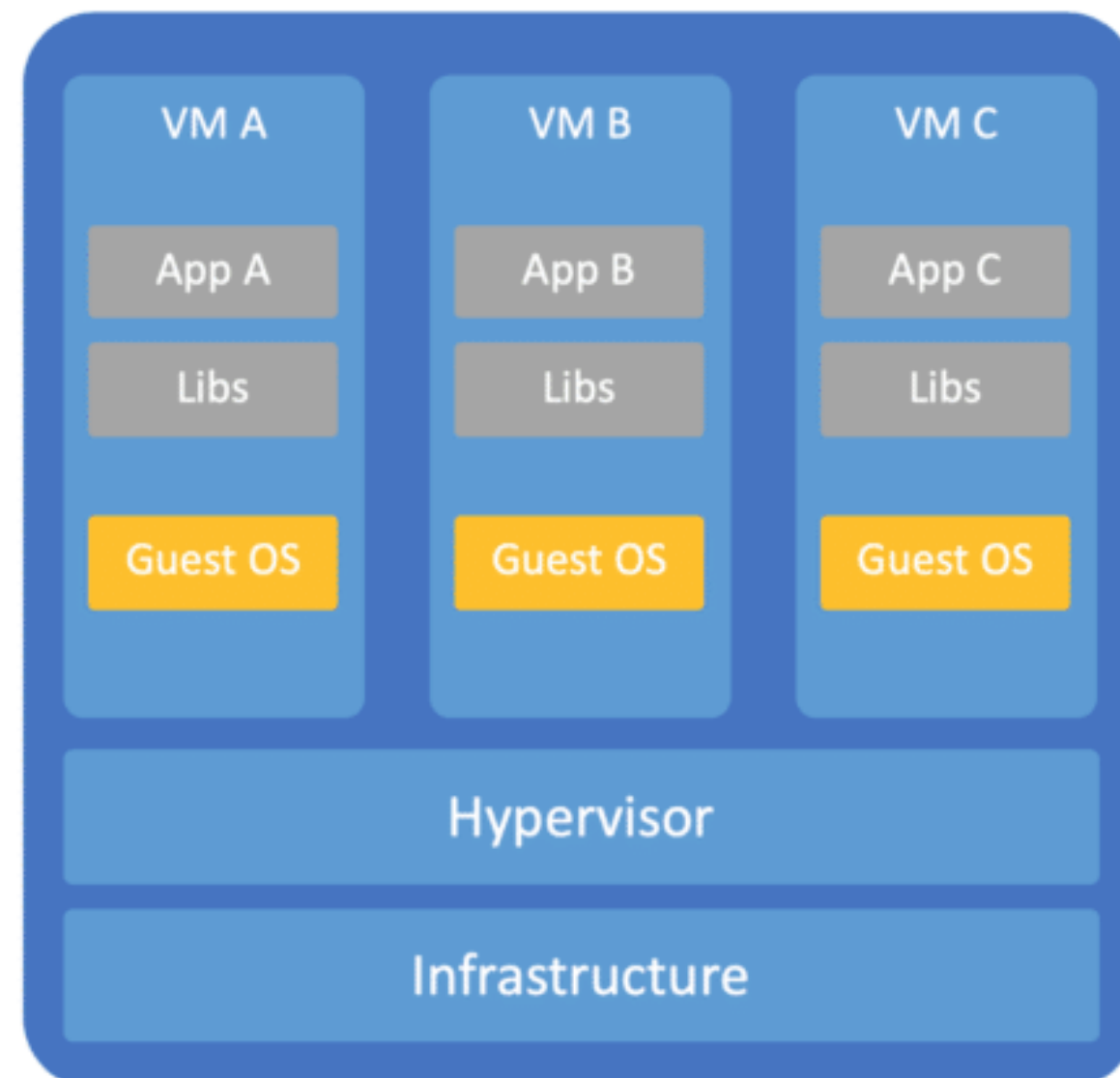


도커의 개념

도커의 구성

- 하이퍼바이저 : 하나의 물리적 호스트에서 여러 개의 가상 머신을 생성하고 관리할 수 있게 해주는 소프트웨어
 - Windows/Linux 등 다양한 선택 가능
 - init 시간이 길고, GB 단위
 - 각 머신마다 OS가 존재하여 무겁고, Host 컴퓨터에 많은 부하를 줄 수 있다.

Virtual Machines

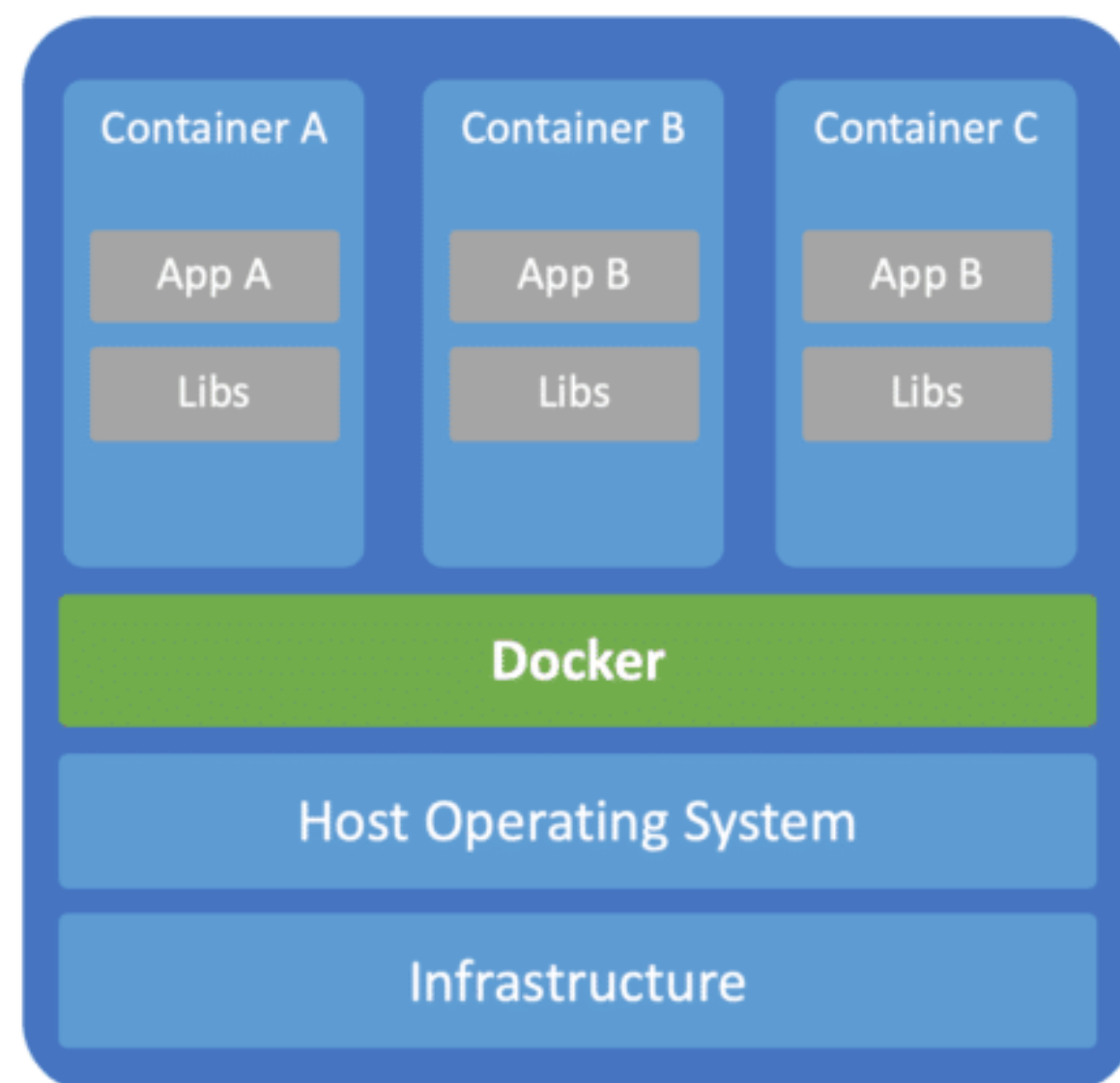


도커의 개념

도커의 구성

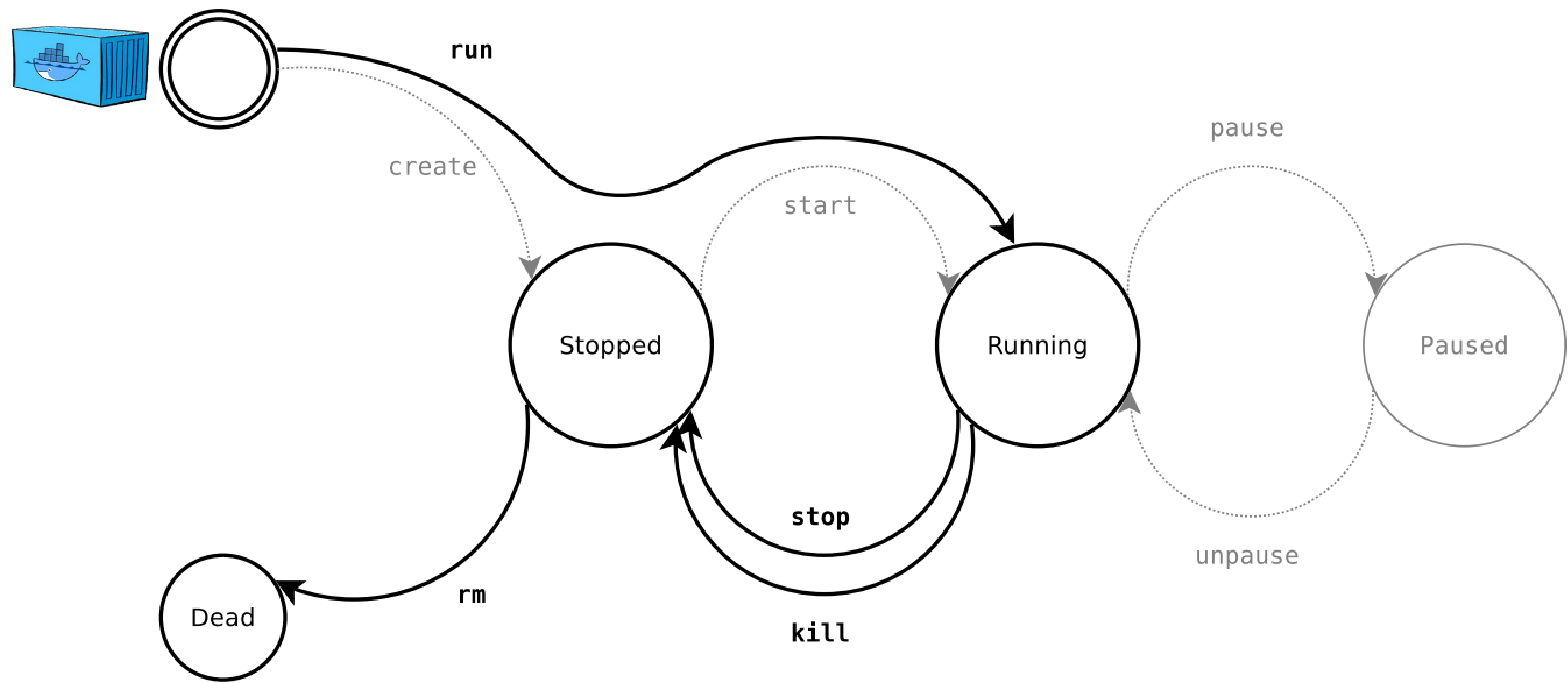
- 도커 컨테이너
 - 일반적으로는 Linux 환경
 - init 시간이 짧고, MB 단위
 - Host 위에 Docker 엔진이 컨테이너를 커널 수준으로 분리하여 사용하기 때문에 가볍다.

Container



도커의 개념

컨테이너 수명주기



도커의 활용

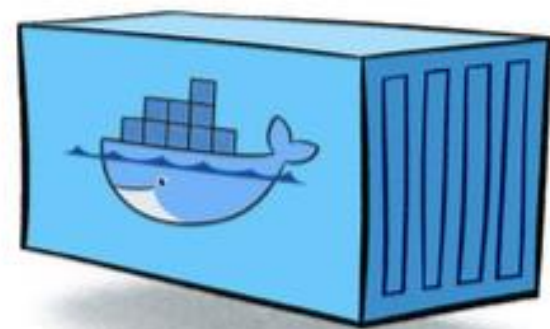
Docker Image

- 코드, 라이브러리, 런타임 환경 등 컨테이너 실행에 필요한 모든 파일과 설정들을 포함한 템플릿

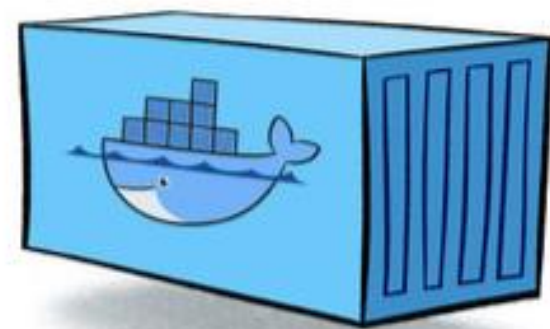
Docker Image



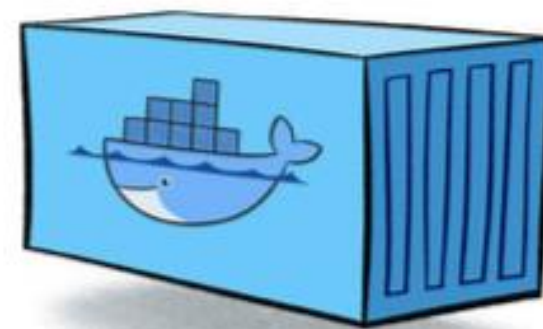
Jenkins



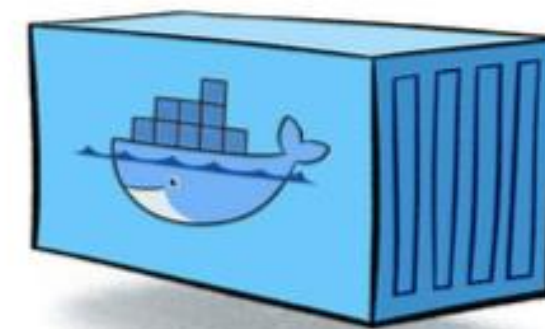
mysql



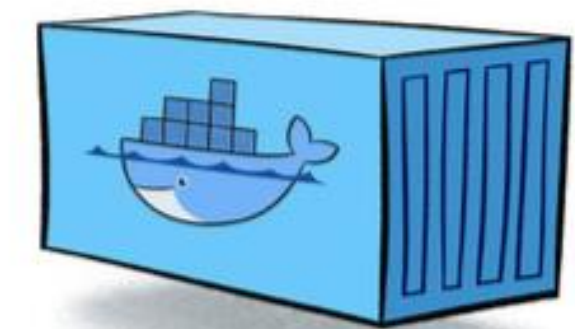
redis



rebbitmq



jenkins

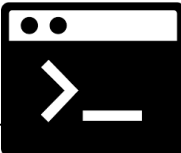


wordpress

도커의 활용

Dockerfile

- 도커에서 이미지를 생성하기 위한 용도로 작성하는 파일
- 베이스 이미지, 구성 설치, 환경 설정, 명령어 등의 단계가 순차적으로 정의되어, 도커 이미지를 일관되게 빌드 가능



Terminal

```
export GO_VERSION="1.20"

sudo rm -rf /usr/local/go

wget https://golang.org/dl/go$GO_VERSION.linux-amd64.tar.gz
sudo tar -C /usr/local -xzf go$GO_VERSION.linux-amd64.tar.gz

echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.profile
echo "export GOPATH=$HOME/go" >> ~/.profile
echo "export PATH=$PATH:$GOPATH/bin" >> ~/.profile

source ~/.profile

rm go$GO_VERSION.linux-amd64.tar.gz

cd /src
go mod download
go build -o /bin/client ./cmd/client
go build -o /bin/server ./cmd/server
```



Dockerfile

```
FROM golang:1.20-alpine

WORKDIR /src

COPY . .

RUN go mod download

RUN go build -o /bin/client ./cmd/client

RUN go build -o /bin/server ./cmd/server

ENTRYPOINT [ "/bin/server" ]
```



Builder



Layers

```
FROM golang:1.20-alpine

WORKDIR /src

COPY . .

RUN go mod download

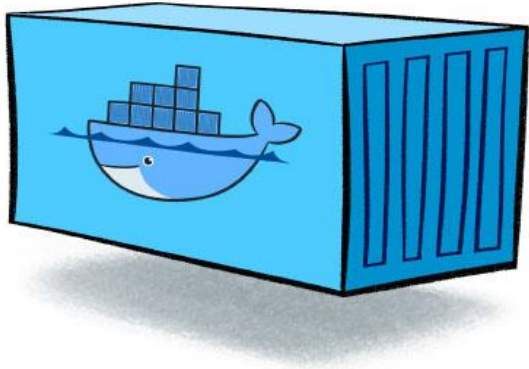
RUN go build -o /bin/client ./cmd/client

RUN go build -o /bin/server ./cmd/server

ENTRYPOINT [ "/bin/server" ]
```



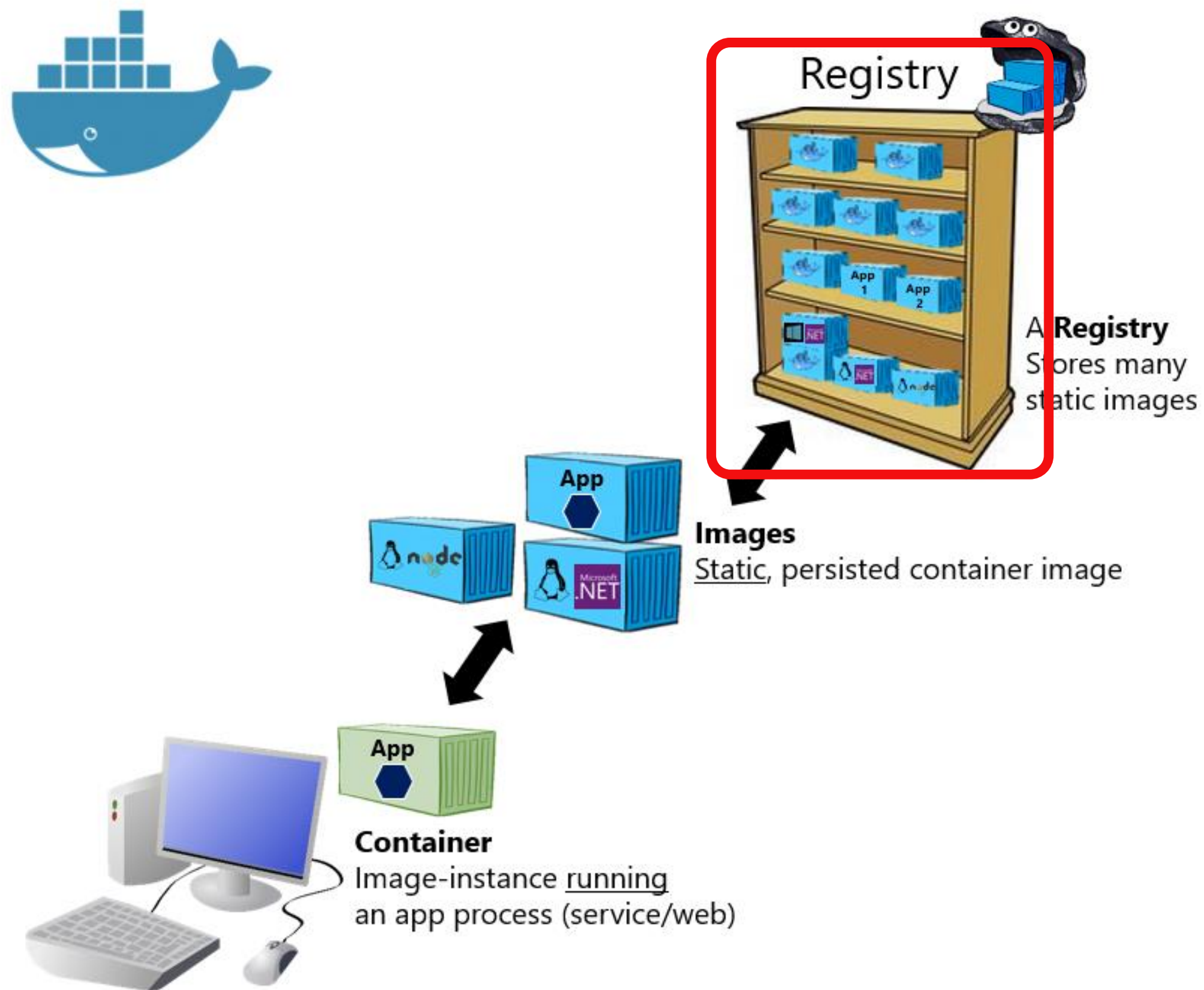
Image



도커의 활용

Docker Hub

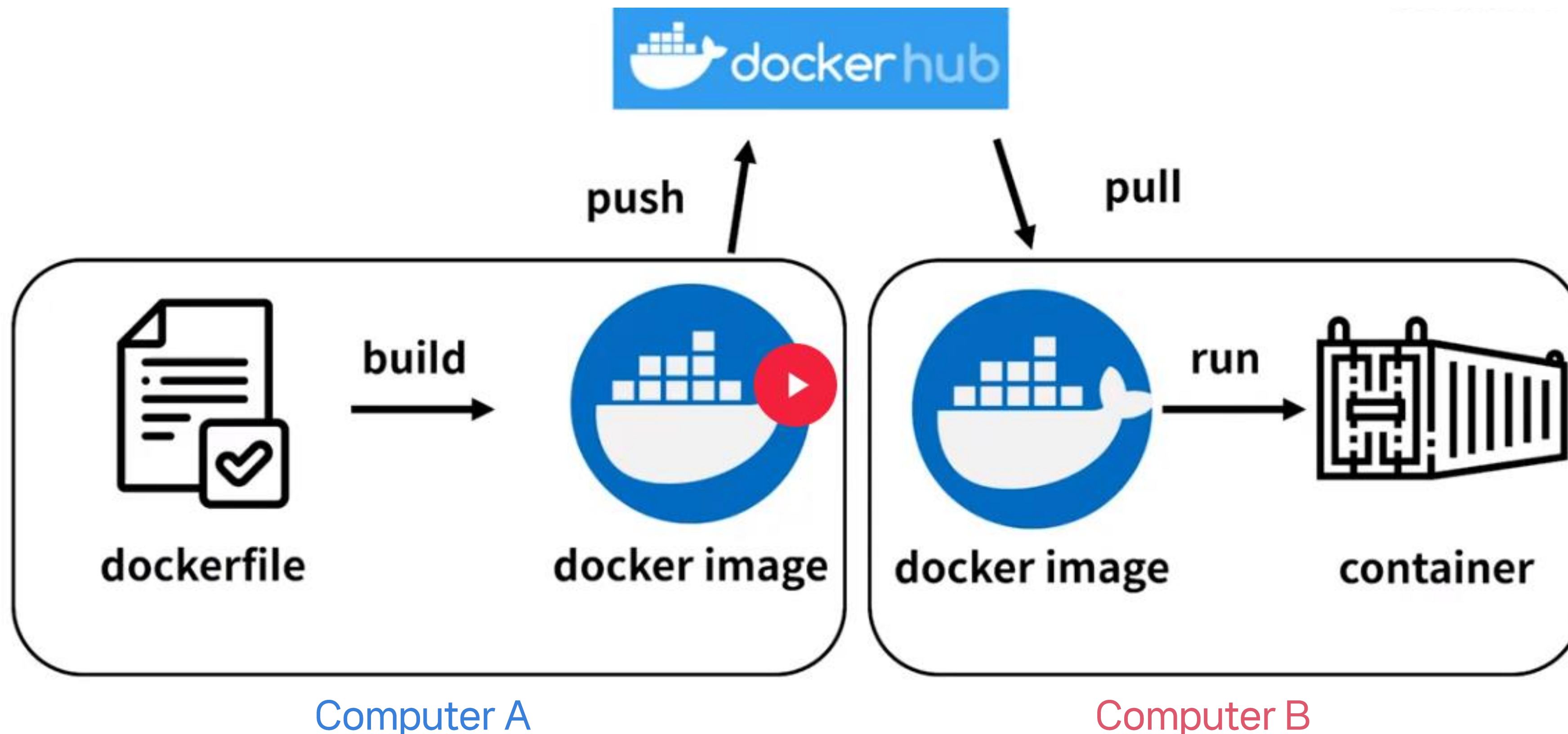
- 도커 이미지를 저장하고 배포하는 중앙 저장소(Registry)
- Docker Hub는 공개 저장소이며, 개인 또는 조직의 프라이빗 저장소 구축도 가능하다.



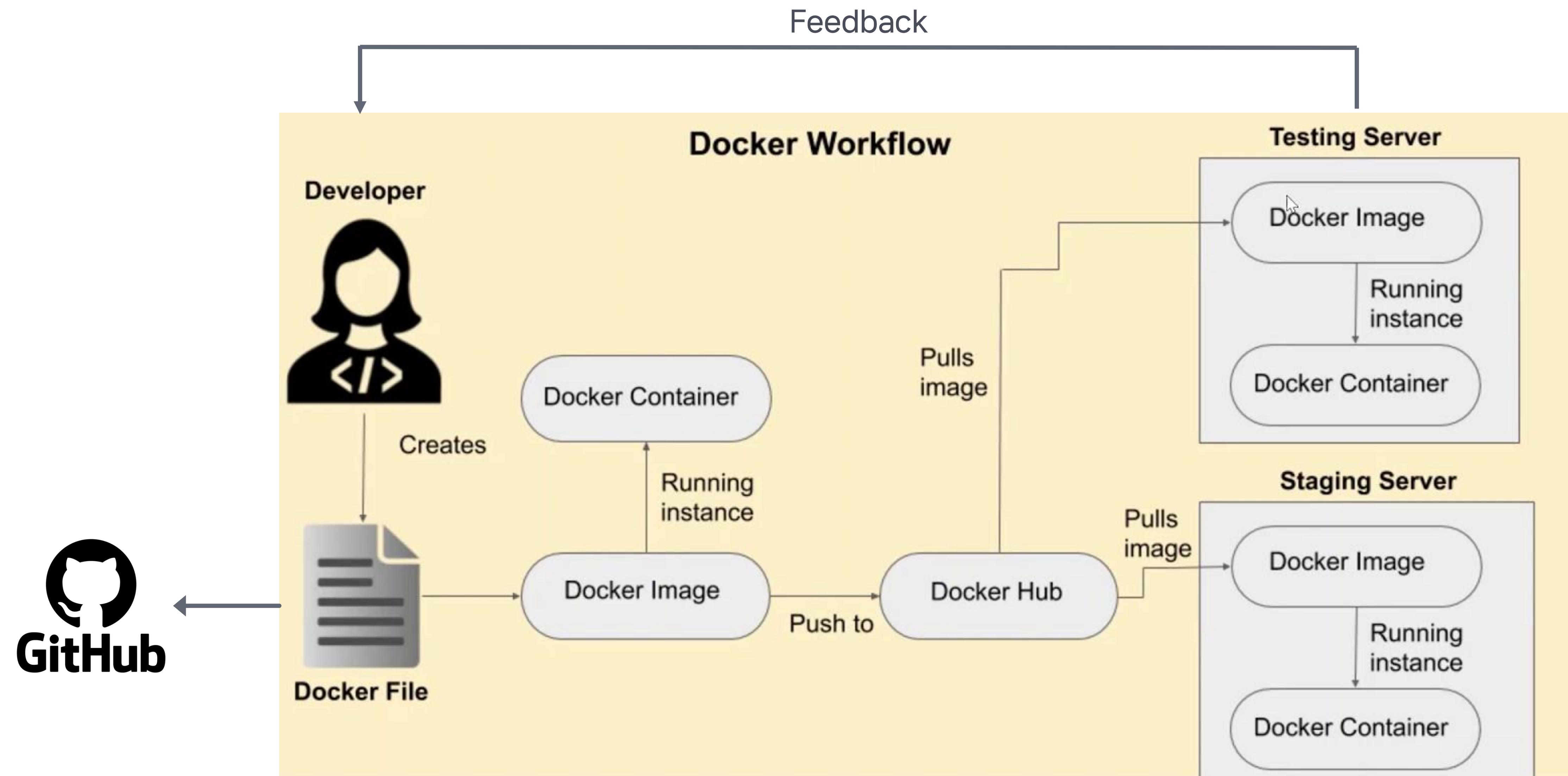
도커의 활용

Docker Hub

- 도커 이미지를 저장하고 배포하는 중앙 저장소(Registry)
- Docker Hub는 공개 저장소이며, 개인 또는 조직의 프라이빗 저장소 구축도 가능하다.



Docker Workflow



도커의 활용

도커의 활용 사례

- 같은 팀 내에 협업이 필요하고, 서로 다른 환경에서 프로그램이 실행될 때
- 확장 가능한 App 배포를 원할 때
- 마이크로서비스 아키텍처 구축을 고려할 때
- 호스팅된 컴퓨터에 머지않아 많은 환경이 변경될 때 (데이터 이행, OS 업그레이드, 등)

하지만...

- 리눅스용 소프트웨어만 지원 가능
- 호스트 서버에 문제 생기면 모든 컨테이너에 영향을 미침
- 서비스가 작은 경우, 컨테이너 하나만 장기간 걸쳐 사용할 경우 큰 장점을 느끼기 어려움

도커의 옵션들

옵션	설명
--name	컨테이너 이름 설정
-d	detached mode (백그라운드 실행 모드)
-p	호스트와 컨테이너의 포트를 연결
-v	호스트와 컨테이너의 폴더를 연결
-it	컨테이너 터미널 입력을 위한 옵션
-e	컨테이너 내에서 사용할 환경변수 설정

도커의 옵션들

port forwarding (-p)

`docker run -d -p {host port}:{cnt port} --name nginx_cnt nginx`

➤ `docker run -d -p 8080:80 --name nginx_cnt nginx`

Port Forwarding

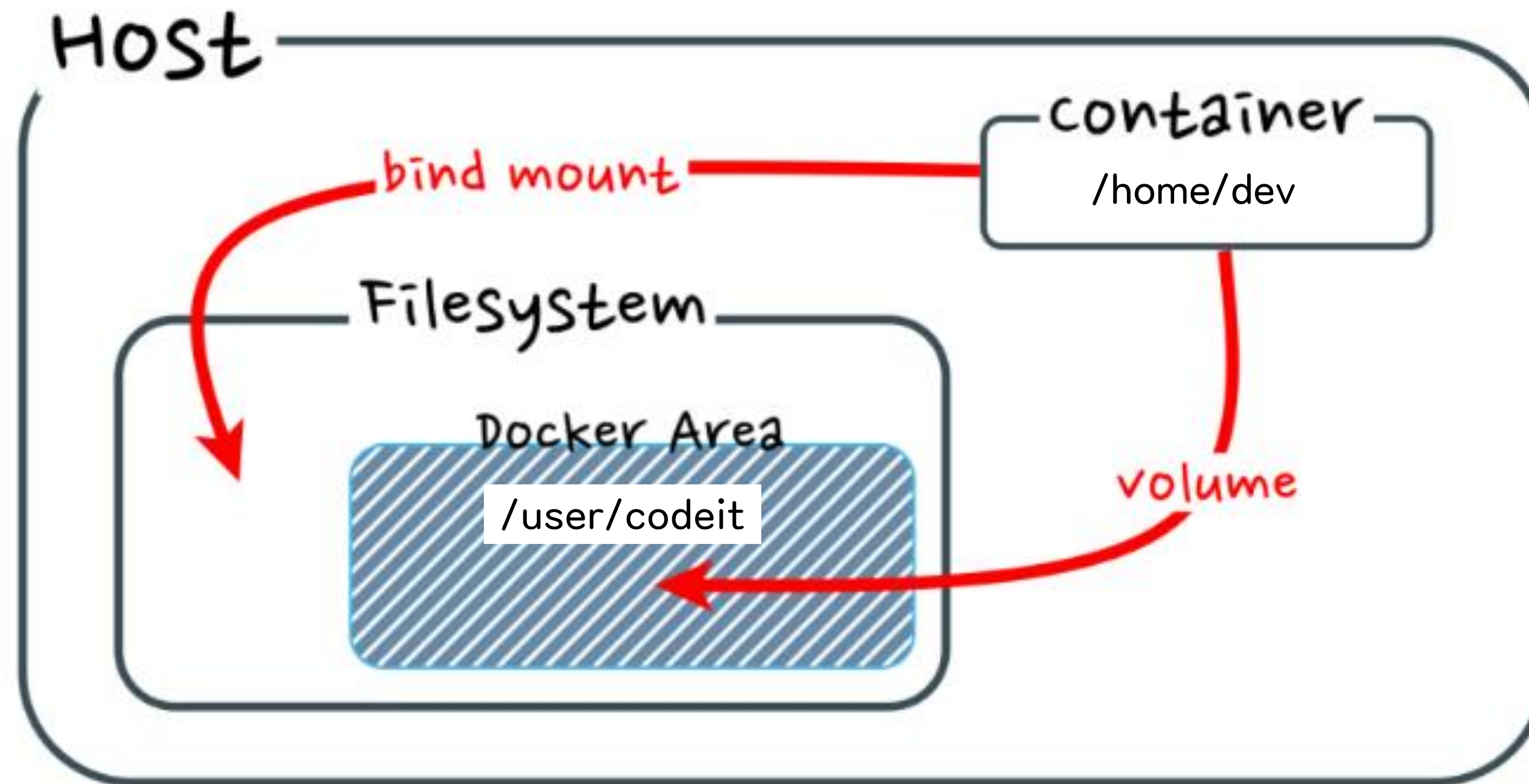


도커의 옵션들

volume mount (-v)

`docker run -d -p 8080:80 -v {host_dir}:{cnt_dir} --name nginx_cnt nginx`

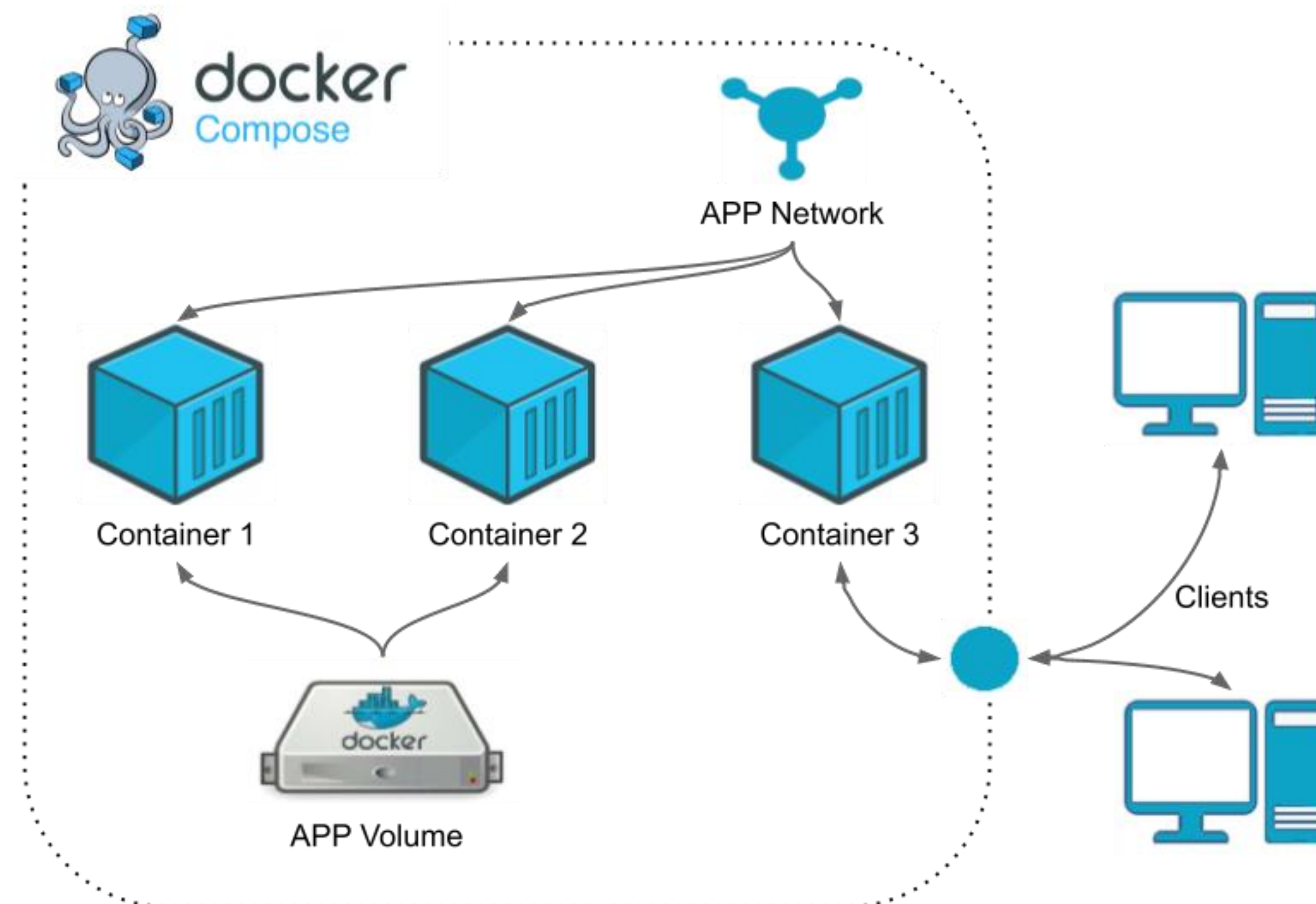
➤ `docker run -d -p 8080:80 -v /user/codeit:/home/dev --name nginx_cnt nginx`



도커 컴포즈

도커 컴포즈의 개념

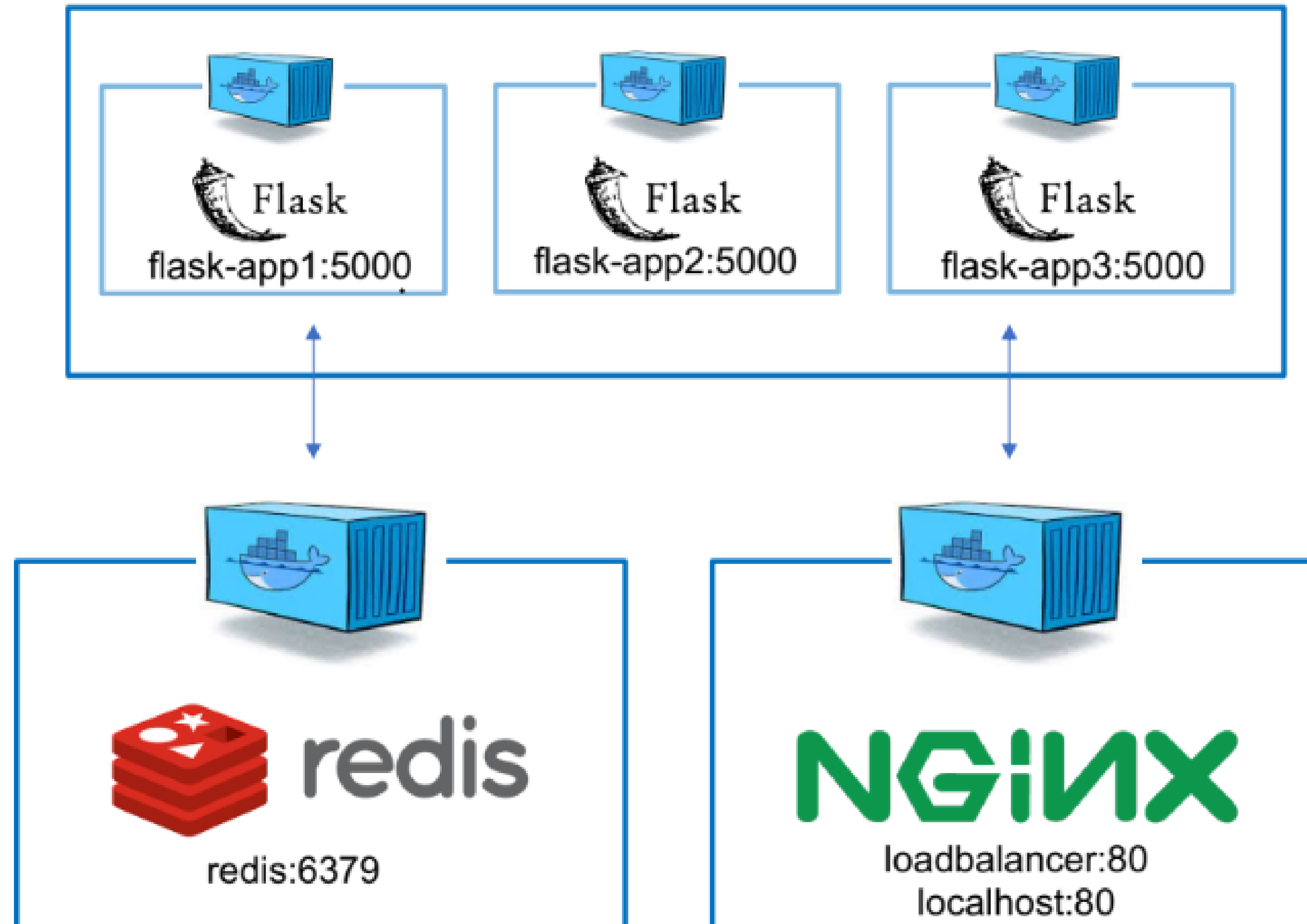
- 여러 개의 도커 컨테이너를 정의하고 관리하기 위한 도구
- 컨테이너 생성을 위한 정보들을 YAML 파일에 정의하여 여러 컨테이너들을 한 번에 실행
- 다수의 컨테이너들이 필요한 마이크로 서비스를 쉽게 관리할 수 있도록 도와줌
- 여러 Docker 어플리케이션 간 상호작용 혹은 의존적일 때 관리 가능
 - 웹 서비스 + DB



도커 컴포즈

도커 컴포즈의 사용 사례 1

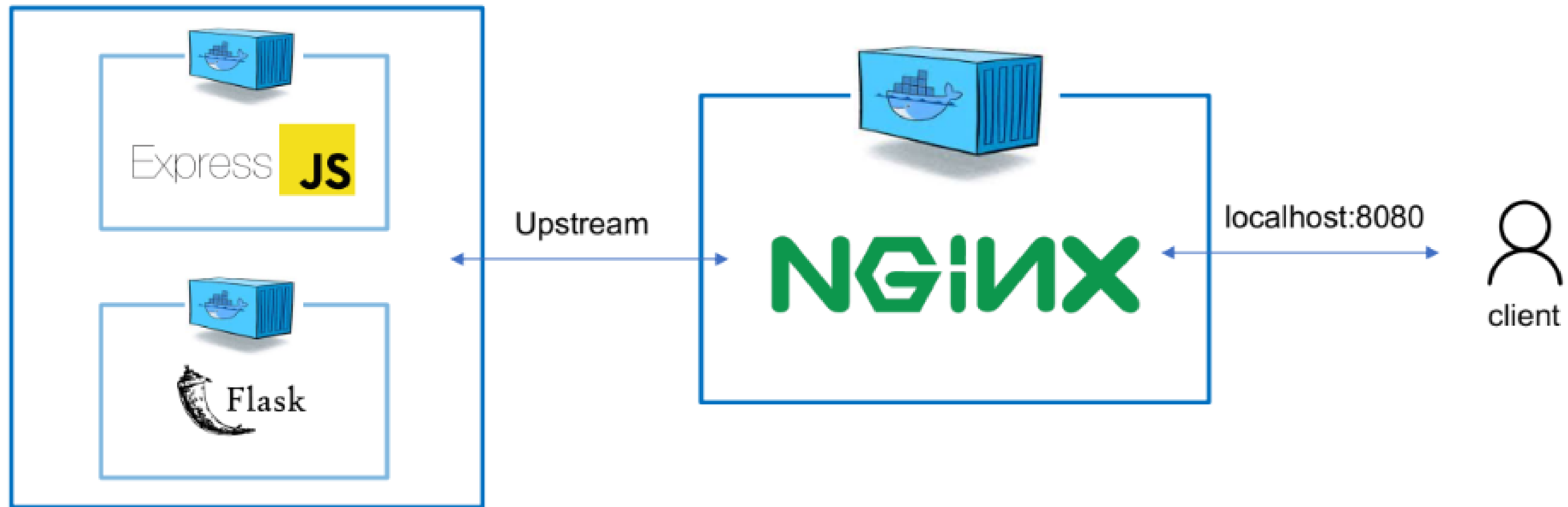
- 백엔드 : Flask
- 데이터베이스 : Redis
- 웹서버 : Nginx



도커 컴포즈

도커 컴포즈의 사용 사례 2

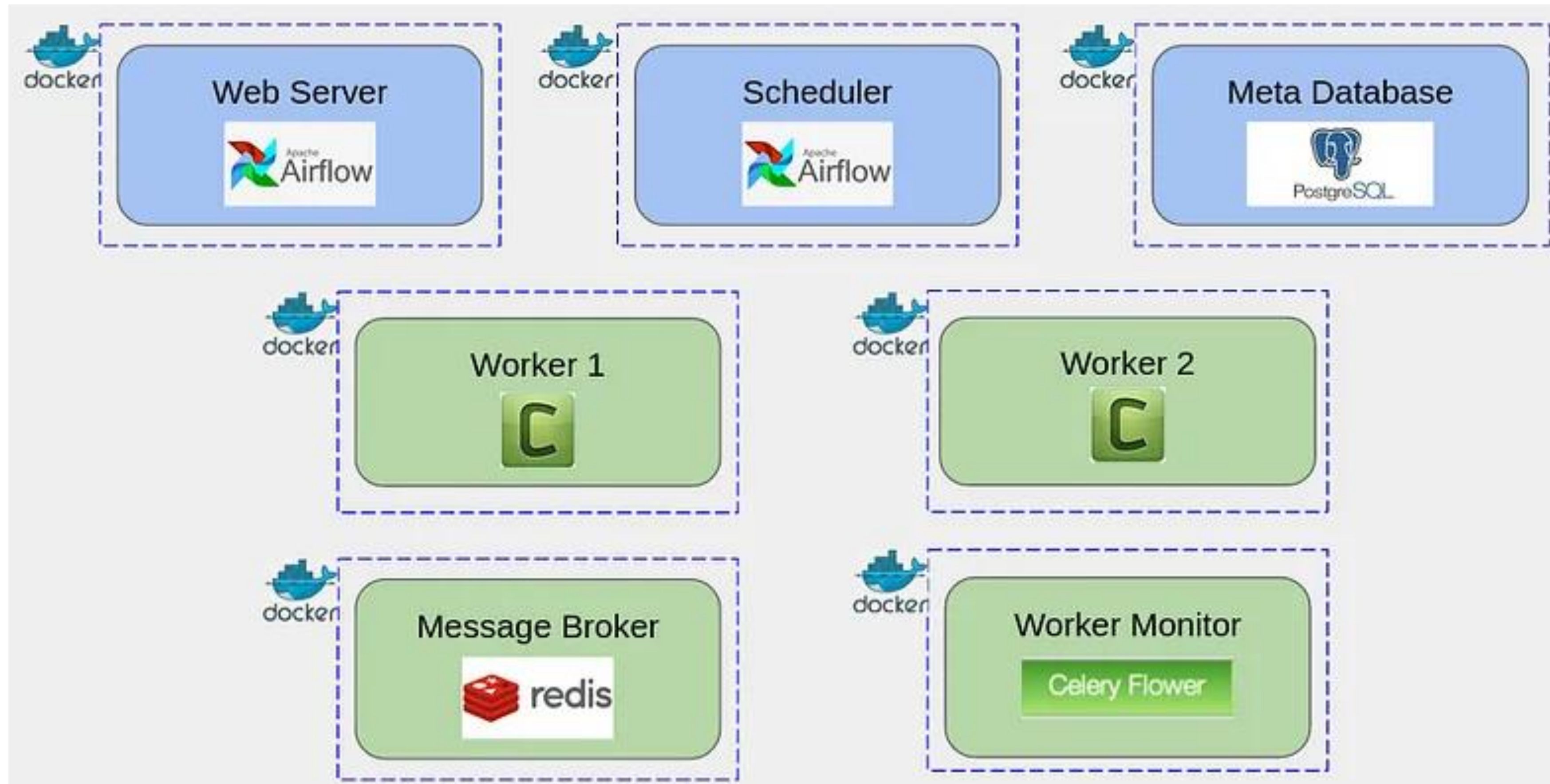
- 백엔드 : Flask
- 프론트엔드 : ExpressJS
- 웹서버 : Nginx



도커 컴포즈

도커 컴포즈의 사용 사례 3

- Airflow



쿠버네티스

쿠버네티스의 개념

- 다수의 컨테이너화 된 워크로드와 서비스를 효율적으로 다루기 위한 오픈소스 기술/플랫폼
- 일반적으로 여러대의 서버가 하나의 컴퓨터처럼 동작되는 '클러스터 환경'에서 동작
- 컨테이너의 배포와 확장, 로드 밸런싱, 자동 복구 등의 기능을 제공하여 복잡한 분산 시스템의 운영을 단순화, 효율화



kubernetes

쿠버네티스

파드, 노드, 클러스터

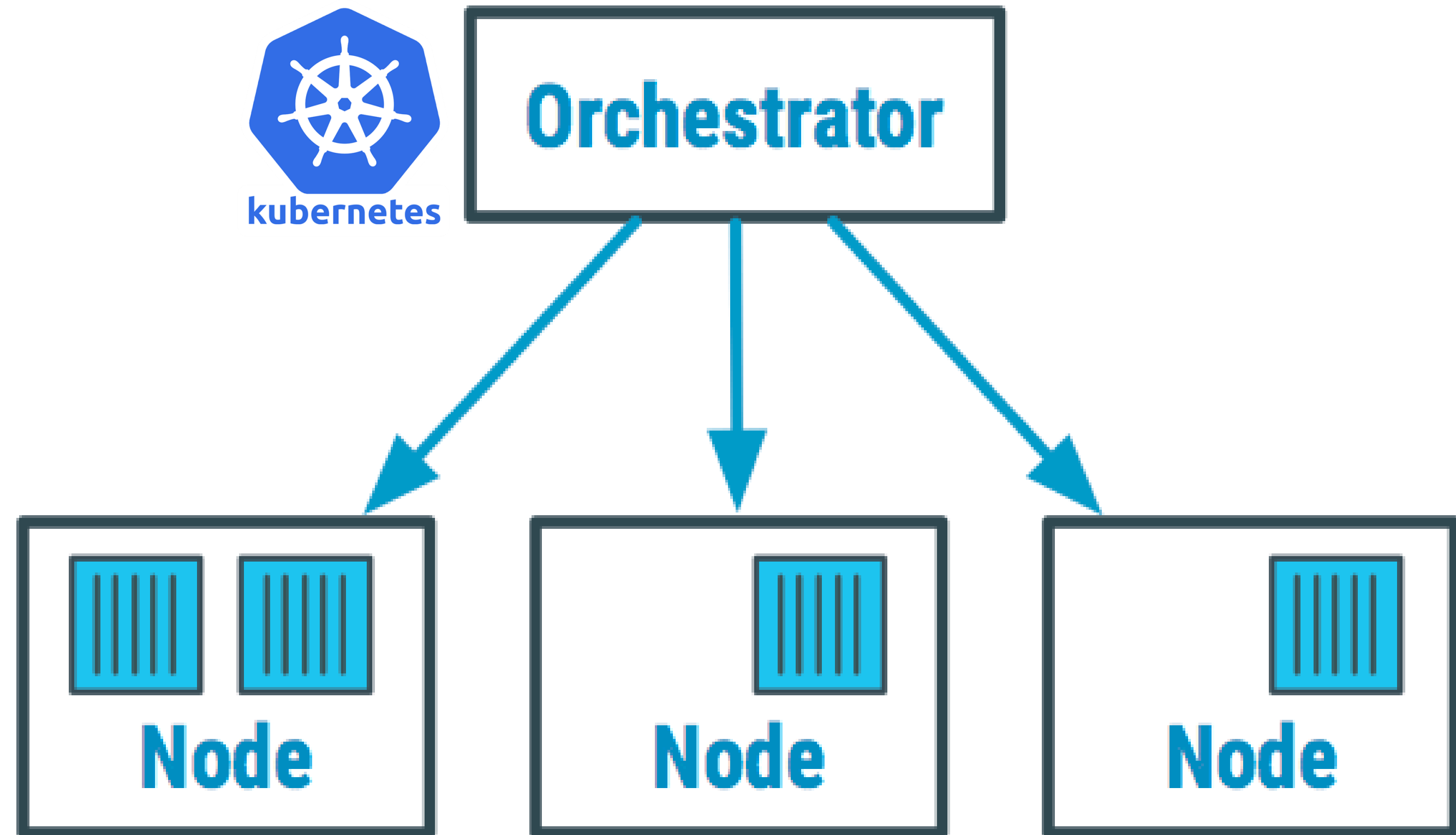
- Pod : 하나 이상의 컨테이너를 그룹화한 최소 배포 단위로, 쿠버네티스에서 관리되는 가장 작은 단위
- Node : 파드가 동작하고 실행되는 서버
- Cluster : 여러 노드로 구성된 집합체
 - Pod + Pod = Node
 - Node + Node = Cluster



쿠버네티스

파드, 노드, 클러스터

- Pod : 하나 이상의 컨테이너를 그룹화한 최소 배포 단위로, 쿠버네티스에서 관리되는 가장 작은 단위
- Node : 파드가 동작하고 실행되는 서버
- Cluster : 여러 노드로 구성된 집합체
 - Pod + Pod = Node
 - Node + Node = Cluster



쿠버네티스

컨테이너 오케스트레이션 도구들

