Spring Cloud로 개발하는 마이크로서비스 애플리케이션

Section 2.

Docker Essentials

- Docker와 컨테이너 가상화 기술
- Docker Image
- Docker Container로 구축하는 IT 인프라
- Dockerfile

Docker와 컨테이너 가상화 기술

Docker Architecture

- **CNCF** https://landscape.cncf.io/

Docker Architecture

- **DevOps Tools**

  https://survey.stackoverflow.co/2023/#technology

Docker와 컨테이너 가상화 기술

Docker Architecture

- **2014년 6월 Docker 1.0 발표**
- **컨테이너 기반의 오픈소스 가상화 플랫폼**
  - 일반PC, AWS, Azure, Google cloud 등에서 실행 가능
  - OS, Backend, Frontend, Database, Message Queue → 컨테이너로 추상화 가능
  - 2019.11 Mirantis에 인수 (Docker Enterprise)
  - https://www.docker.com/blog/updating-product-subscriptions/

20M+	16M+	15M+	16B+
Monthly Active Users	Registered Developers	Docker Hub Repositories	Monthly Image Pulls

Docker와 컨테이너 가상화 기술

Docke

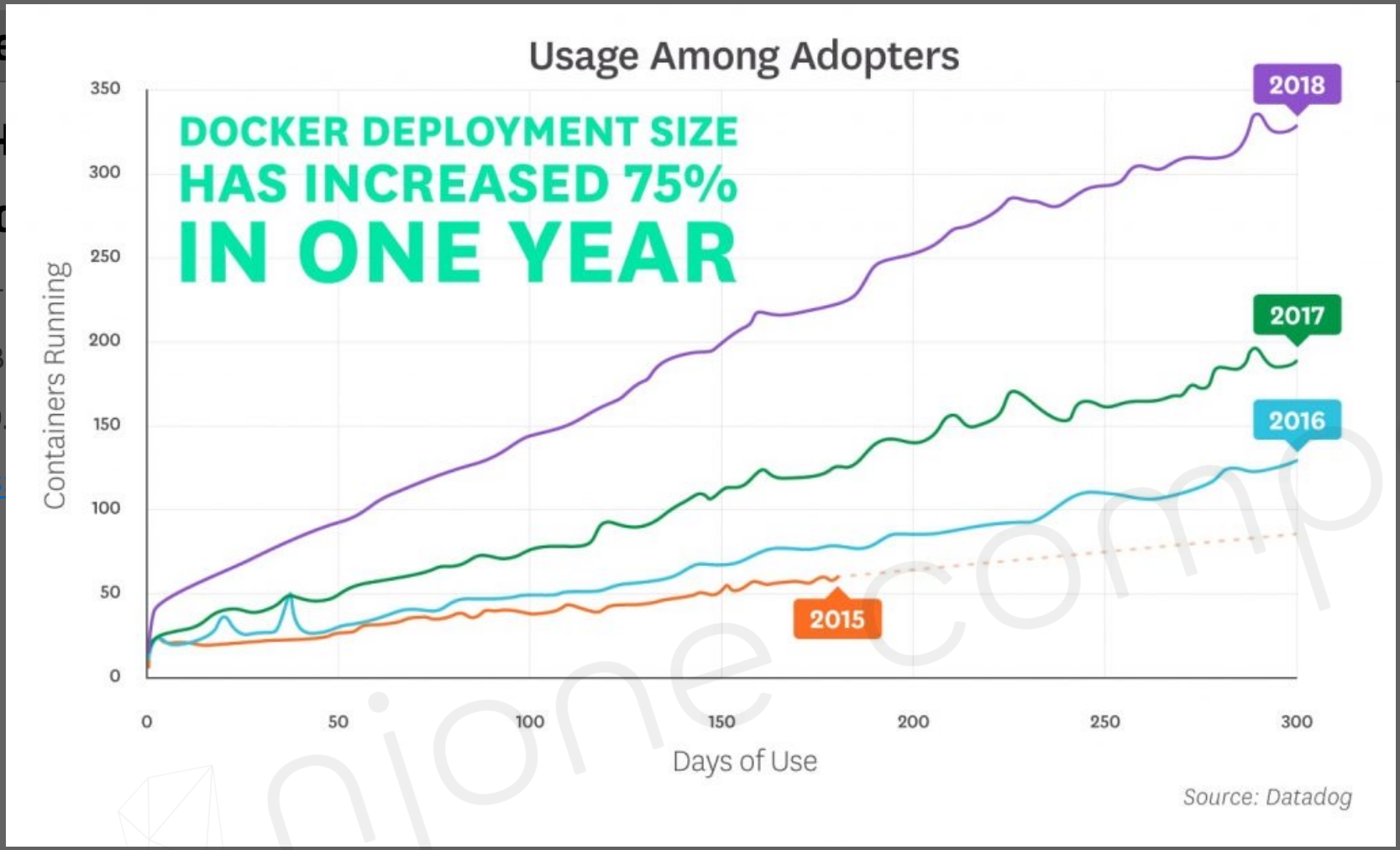
- 2014
- 컨테이

일반

OS, B

2019.

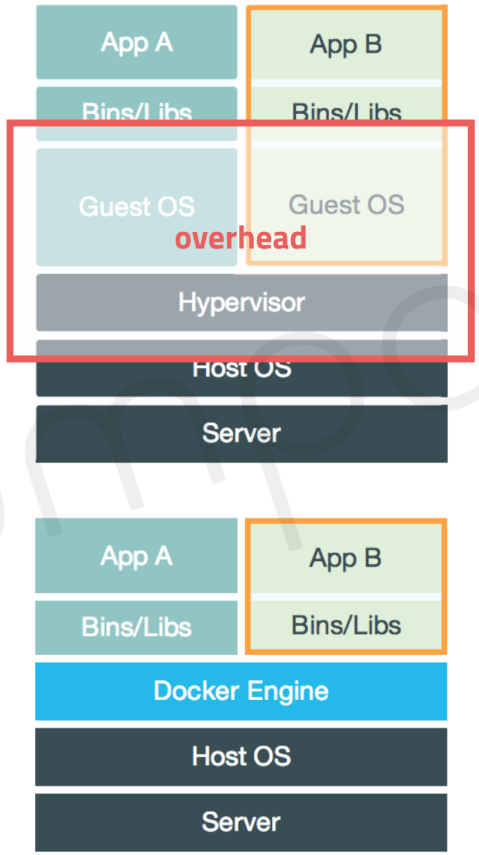
https



Docker와 컨테이너 가상화 기술

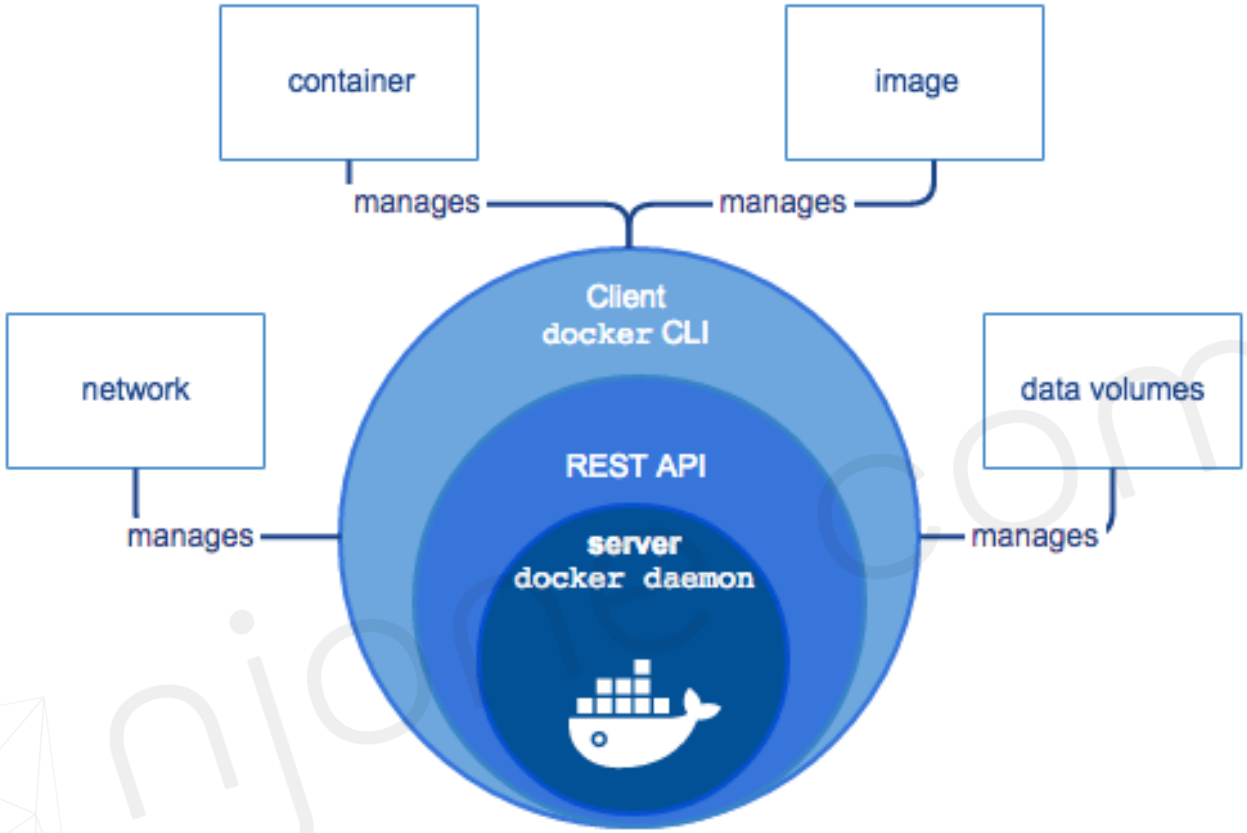
Docker Architecture

- **기존 가상화 방식 ➔ OS를 가상화**
  - VMWare, VirtualBox (Host OS 위에 Guest OS 전체를 가상화)
  - 무겁고 느림
- **CPU의 가상화 기술 이용 방식 ➔ Kernel-based Virtual Machine**
  - 전체 OS를 가상화 하지 않음, 호스트 형식에 비해 속도 향상
  - OpenStack, AWS 등의 클라우드 서비스
  - 추가적인 OS는 여전히 필요, 성능 문제
- **프로세스 격리 ➔ 리눅스 컨테이너**
  - CPU나 메모리는 프로세스에 필요한 만큼만 추가로 사용
  - 성능 손실 거의 없음
  - 컨테이너들 사이는 서로 영향을 주지 않음
  - 컨테이너 생성 속도 빠름 (1-2초 내)



Docker Architecture

- **Docker Engine**

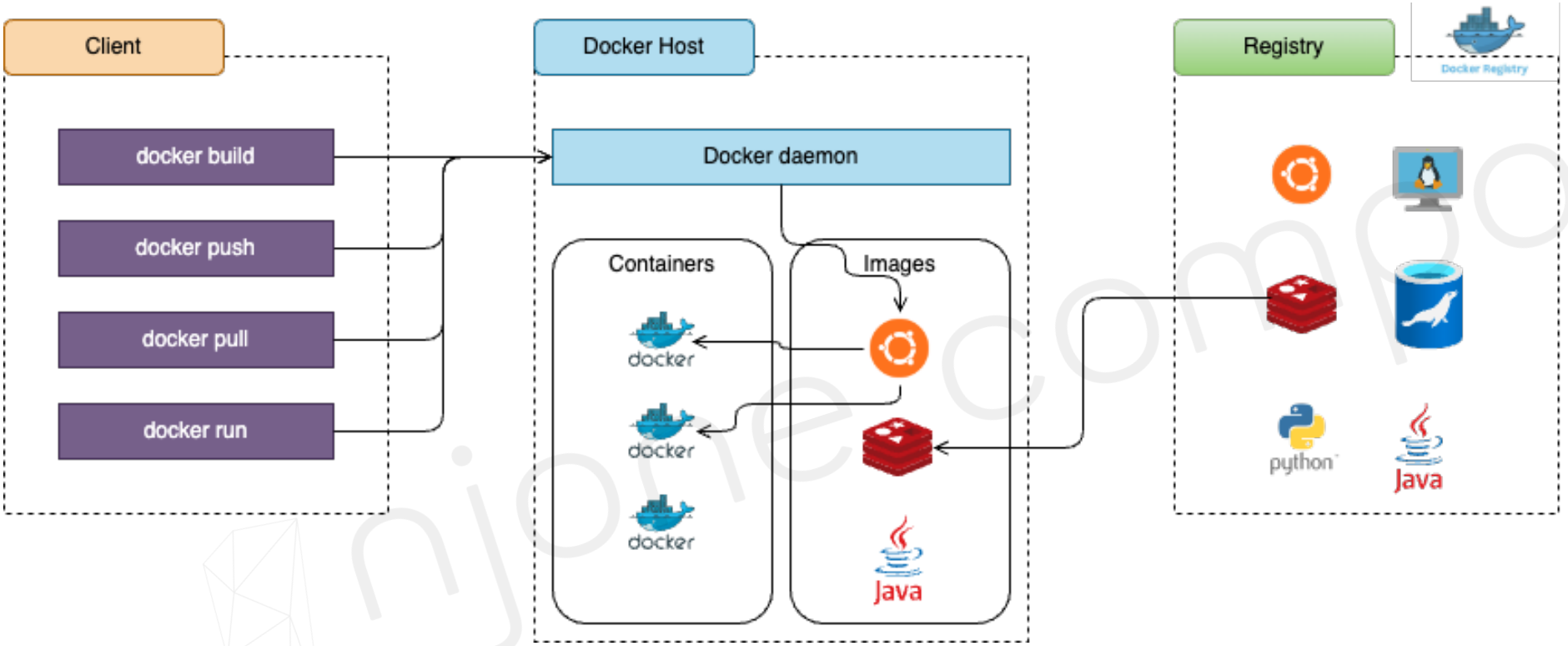


Docker와 컨테이너 가상화 기술

Docker vs Container

- **컨테이너 실행에 필요한 파일과 설정 값 등을 포함 ➔ 상태값 X, Immutable**

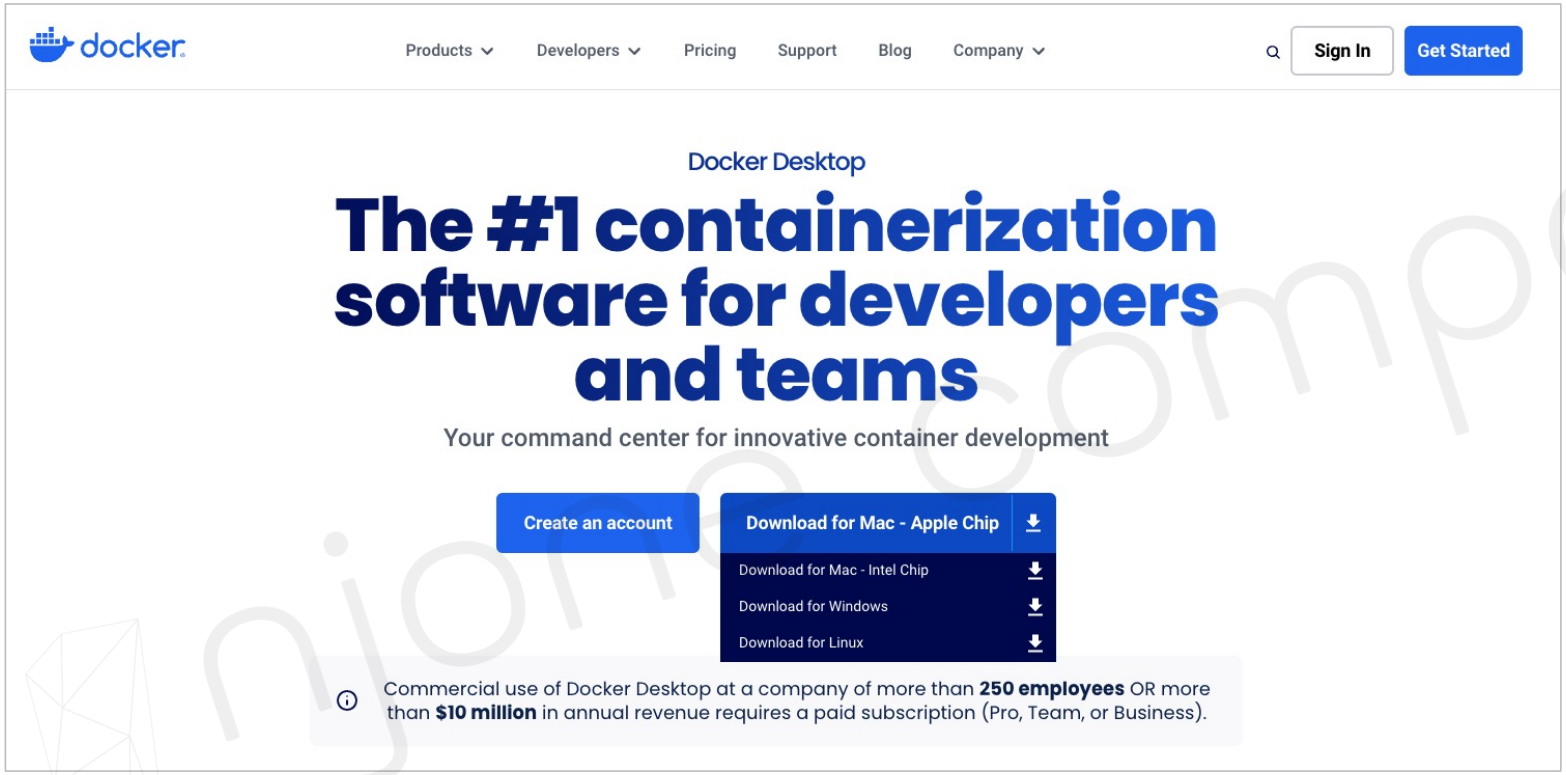
  실체화 → Container



Docker 실습 환경 구성

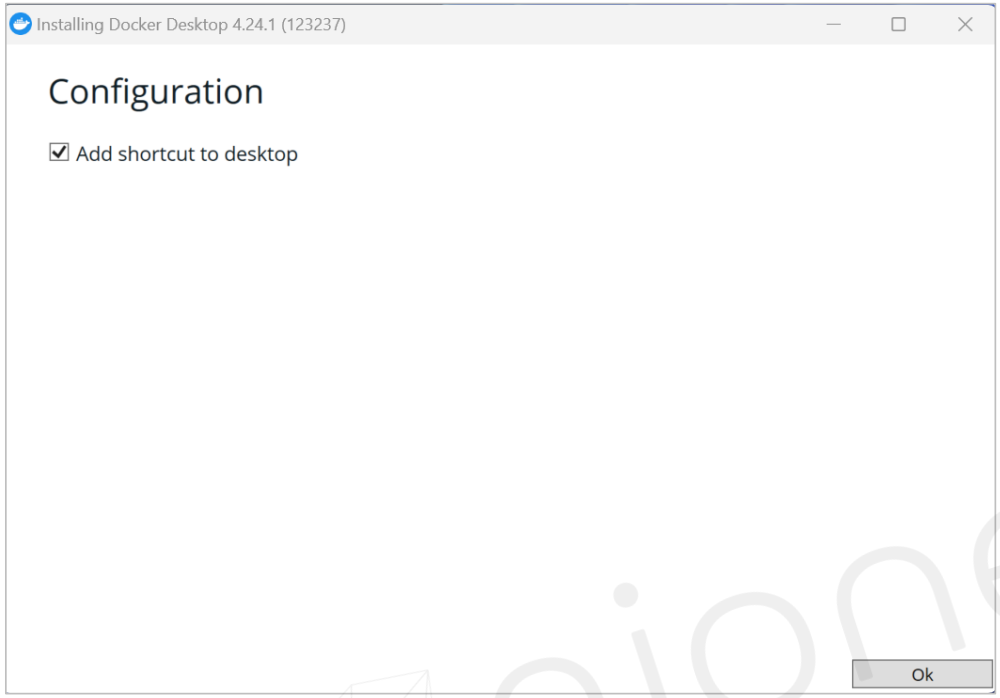
- **Docker for Mac/Docker for Windows → Docker Desktop**

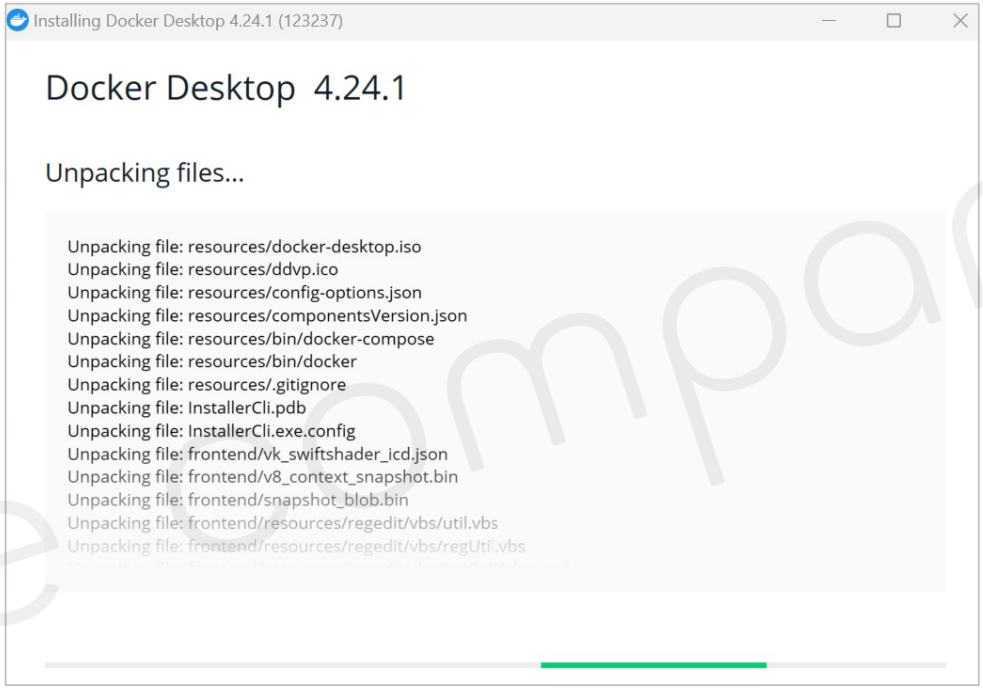
  https://www.docker.com/products/docker-desktop



Docker 실습 환경 구성

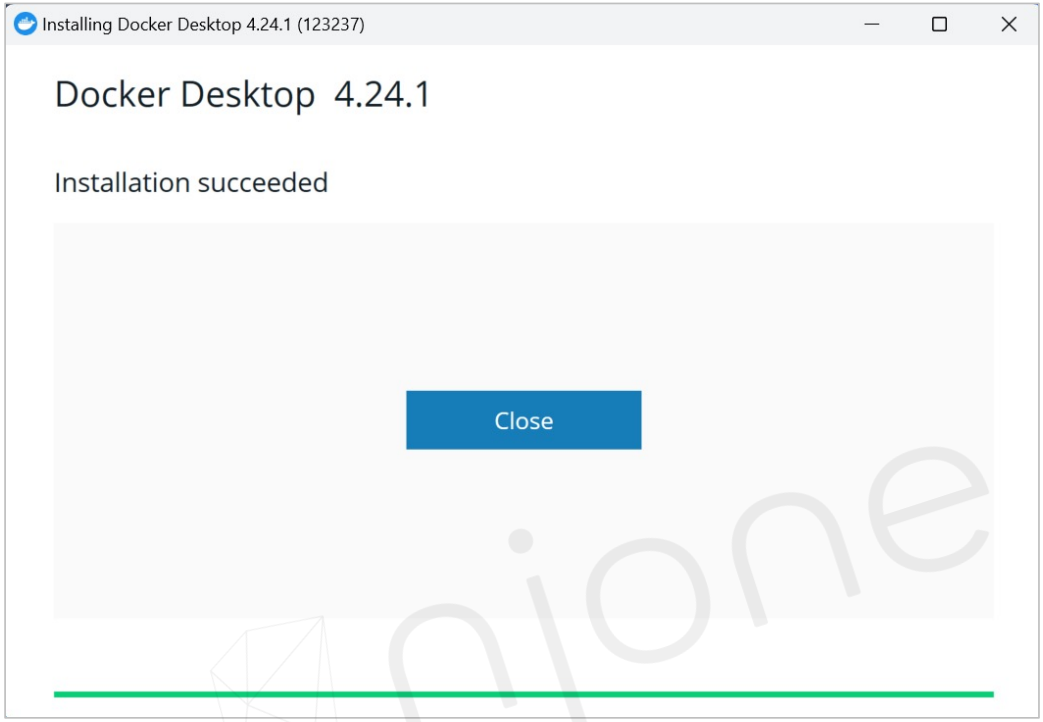
- Docker for Windows → Docker Desktop





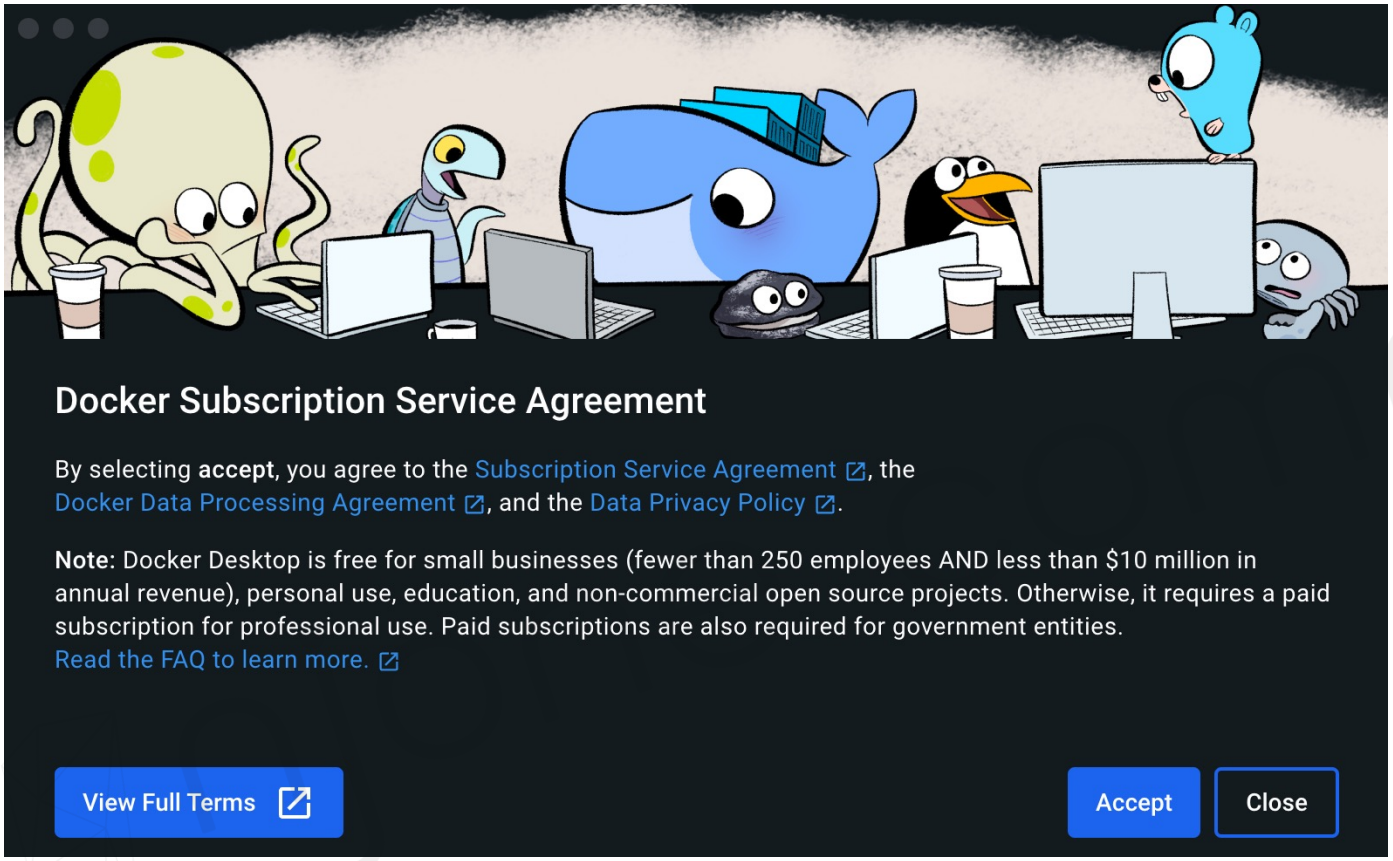
Docker 실습 환경 구성

- Docker for Windows → Docker Desktop



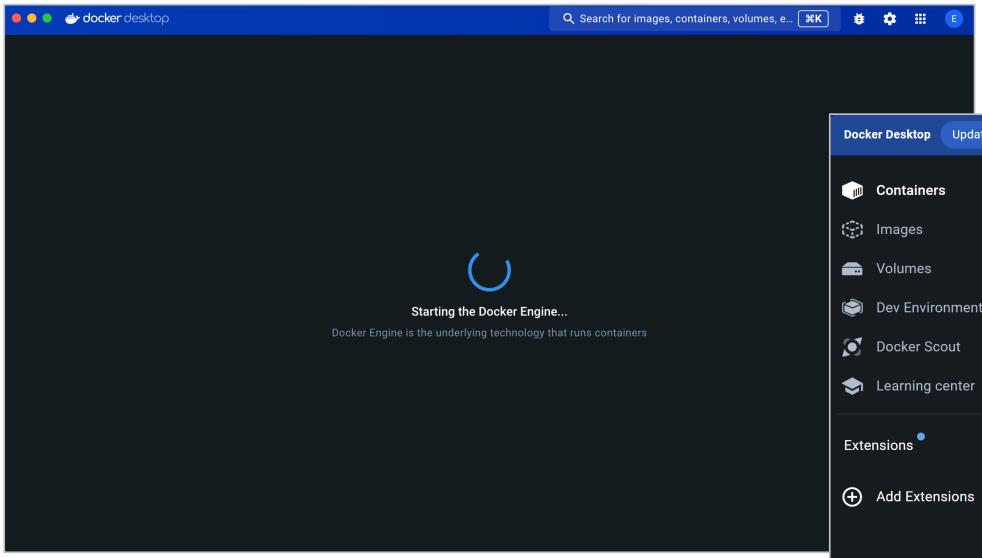
Docker 실습 환경 구성

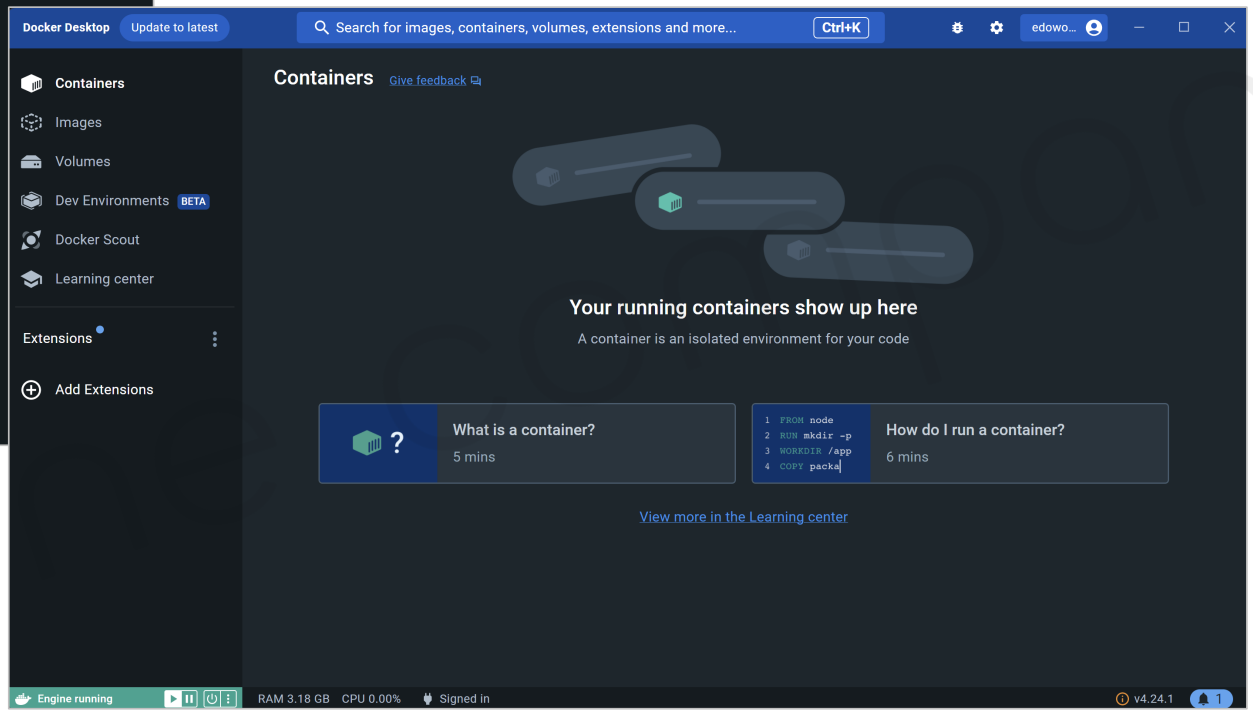
- Docker for Mac/Docker for Windows → Docker Desktop



Docker 실습 환경 구성

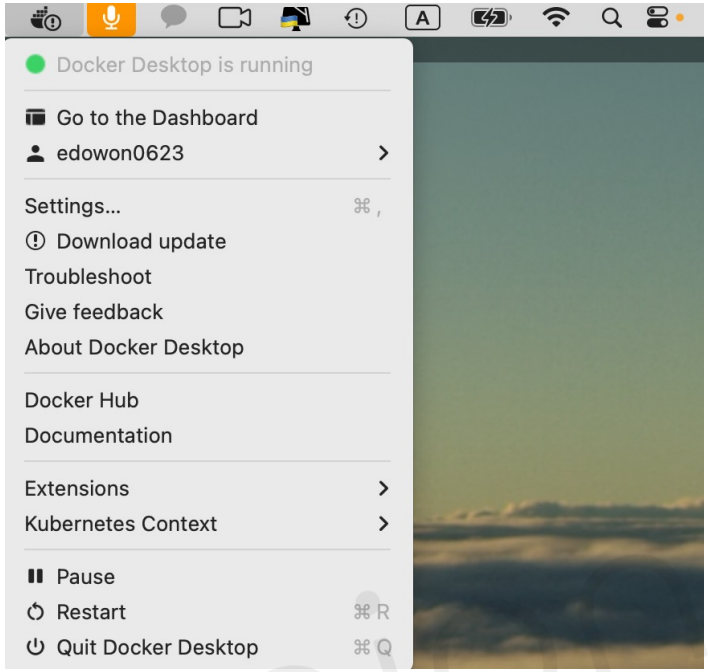
- Docker for Mac/Docker for Windows → Docker Desktop



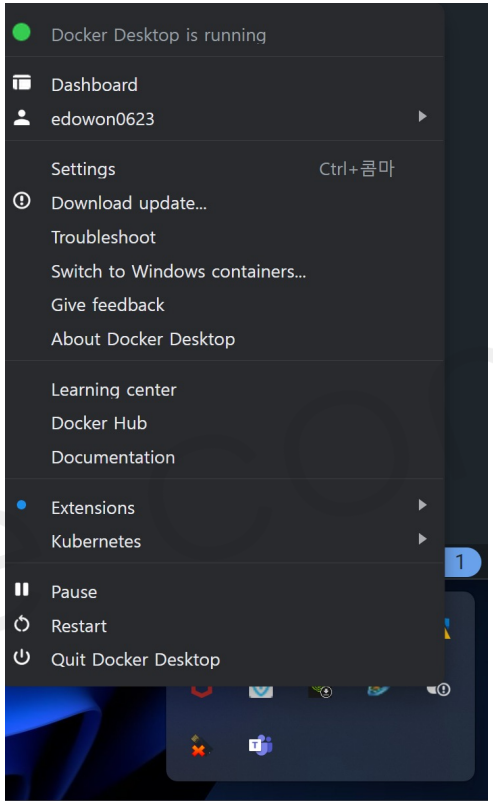


Docker 실습 환경 구성

- **Docker for Mac/Docker for Windows → Docker Desktop**



Docker for Mac



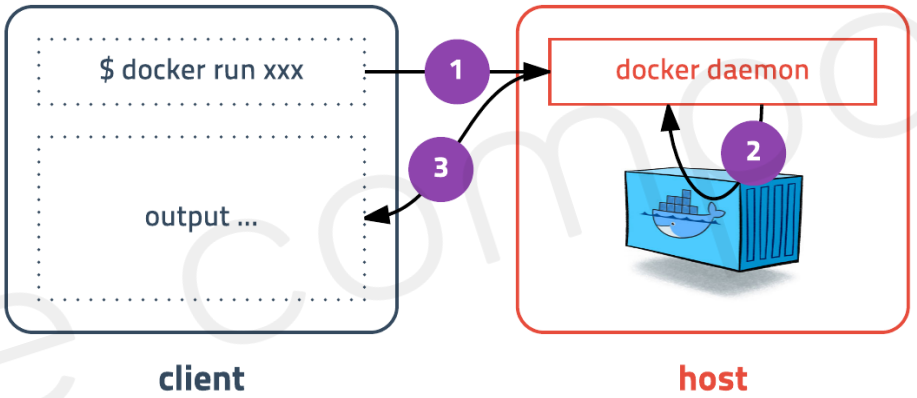
Docker for Windows

Docker 실습 환경 구성

- **설치 확인**

  | $ docker version

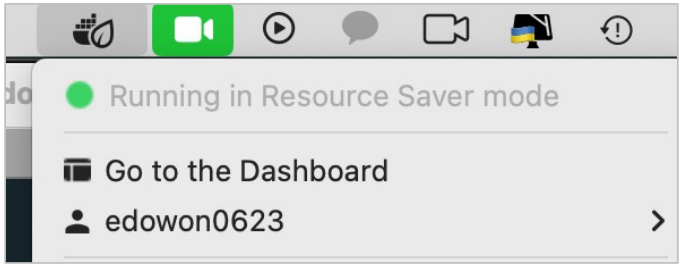
```
bcadmin@hlf03:~$ docker version
Client:
 Cloud integration: v1.0.35+desktop.13
 Version:          26.0.0
 API version:      1.45
 Go version:       go1.21.8
 Git commit:       2ae903e
 Built:            Wed Mar 20 15:14:46 2024
 OS/Arch:          darwin/arm64
 Context:          desktop-linux

Server: Docker Desktop 4.29.0 (145265)
 Engine:
  Version:         26.0.0
  API version:     1.45 (minimum version 1.24)
  Go version:      go1.21.8
  Git commit:      8b79278
  Built:           Wed Mar 20 15:18:02 2024
  OS/Arch:         linux/arm64
  Experimental:    false
```

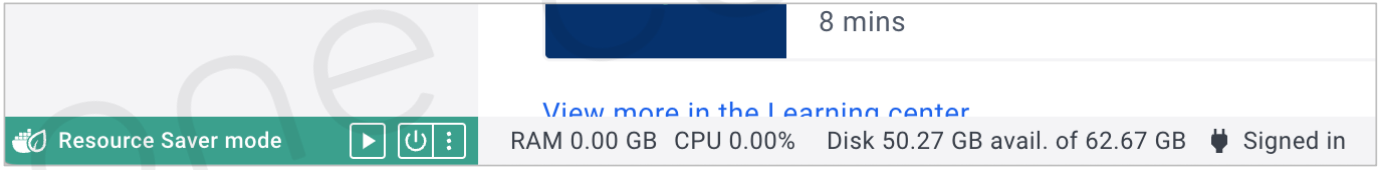


Resource Saver mode

- **Resource Saver**

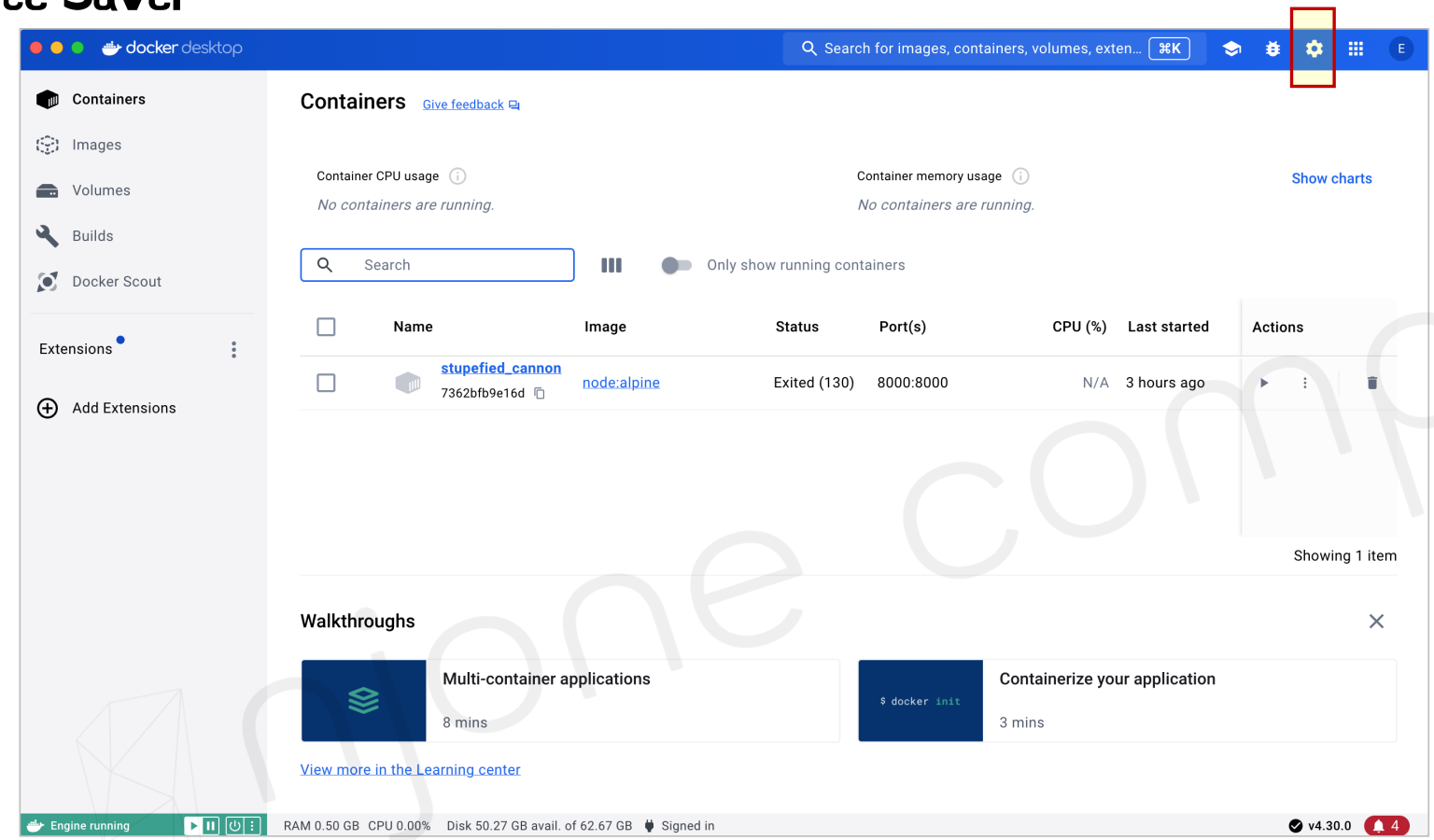
  - https://docs.docker.com/desktop/use-desktop/resource-saver/
  - Docker Desktop version 4.24 부터 적용 → 사용 중인 컨테이너가 없을 때 Docker를 Idle 상태로 운영





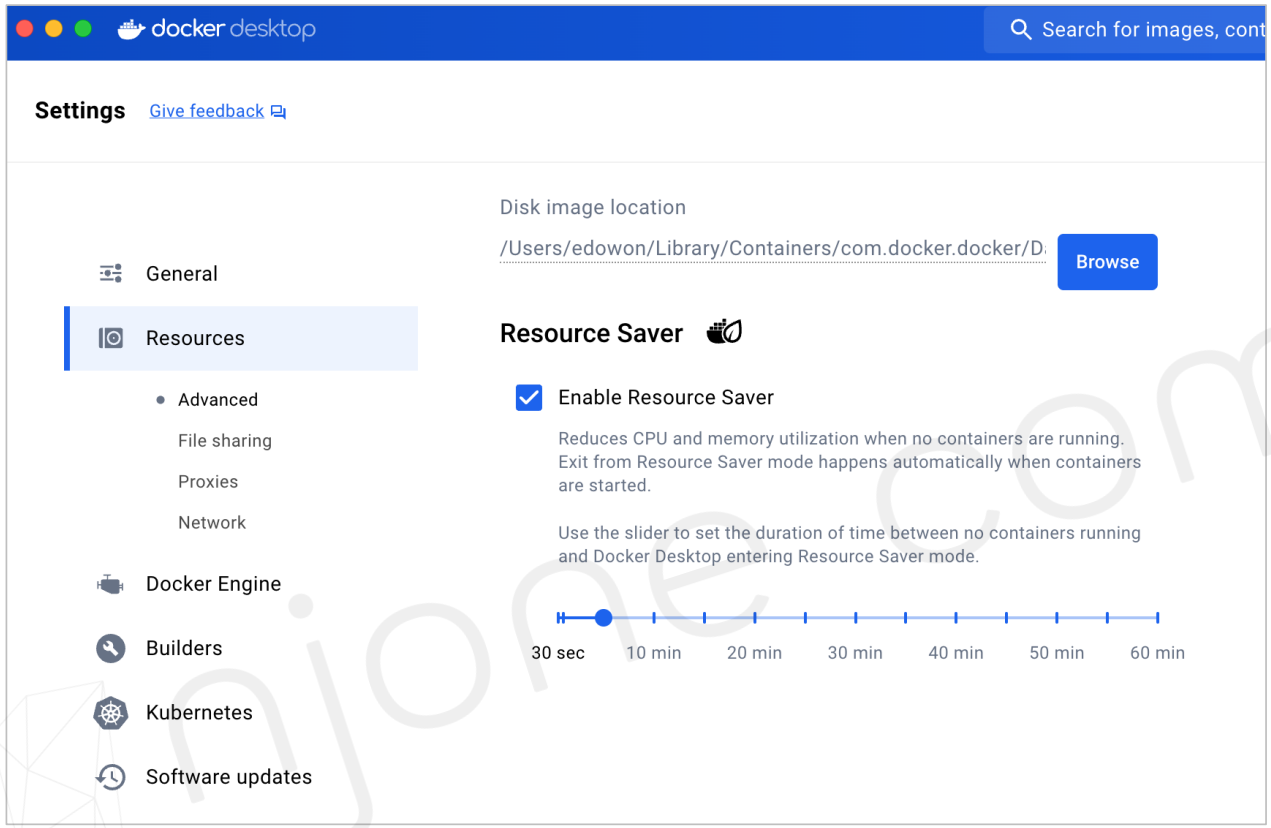
Resource Saver mode

- **Resource Saver**



Resource Saver mode

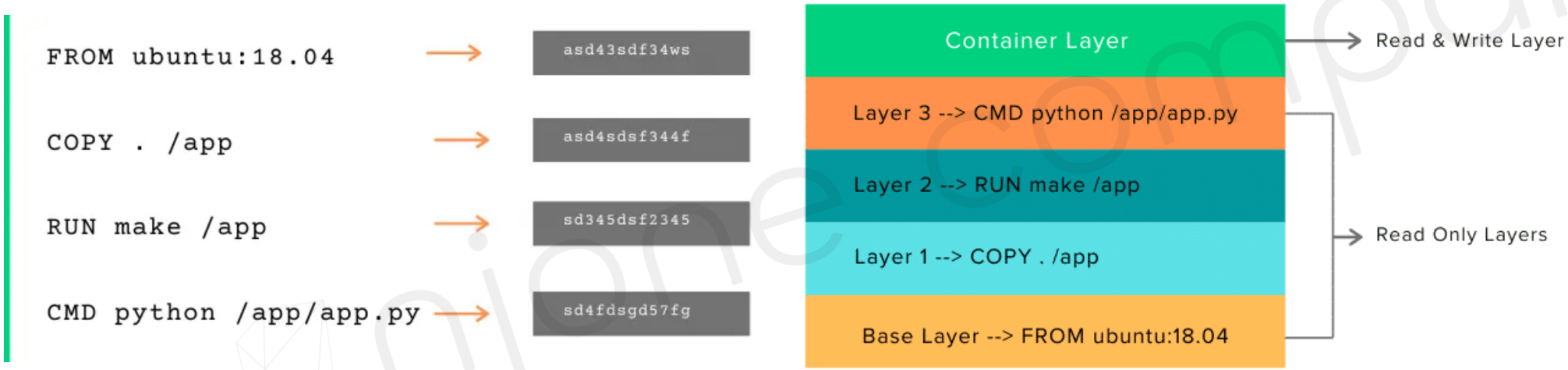
- **Resource Saver**



Docker Image

Docker Image 구조

- **Docker Hub에 등록 or Docker Registry 저장소를 직접 만들어 관리**
  - 공개된 도커 이미지는 820만개 이상, 다운로드 수는 100억회 이상
- **Layer 저장 방식**
  - 유니온 파일 시스템을 이용 → 여러 개의 Layer를 하나의 파일시스템으로 사용 가능



Docker Image

Dockerfile

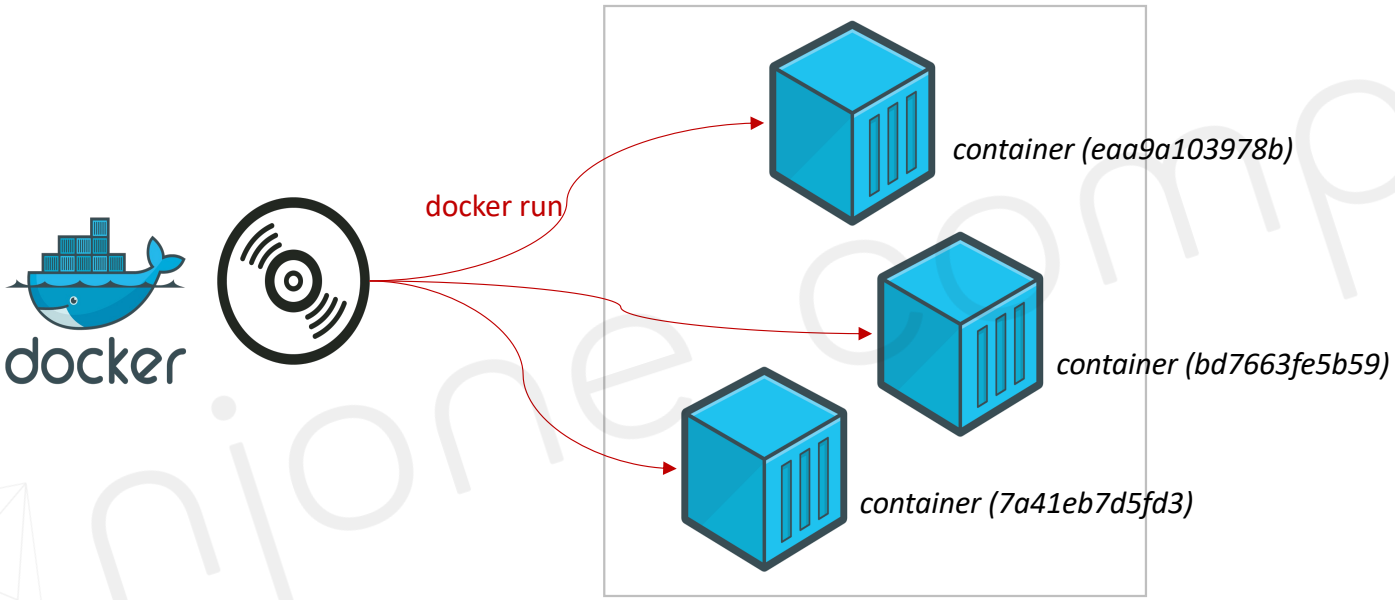
- **Docker Image를 생성하기 위한 스크립트 파일**
  - 자체 DSL(Domain-Specific language) 언어 사용 → 이미지 생성과정 기술
  - 서버에 프로그램을 설치하는 과정을 Dockerfile로 기록, 관리
  - 소스와 함께 버전 관리가 되며, 누구나 수정 가능

```
FROM node:alpine as builder
WORKDIR /app
COPY ./package.json .
RUN npm install
COPY . .
RUN npm run build

FROM nginx
COPY --from=builder /app/build /usr/share/nginx/html
```

Docker Container

Docker Container 이해

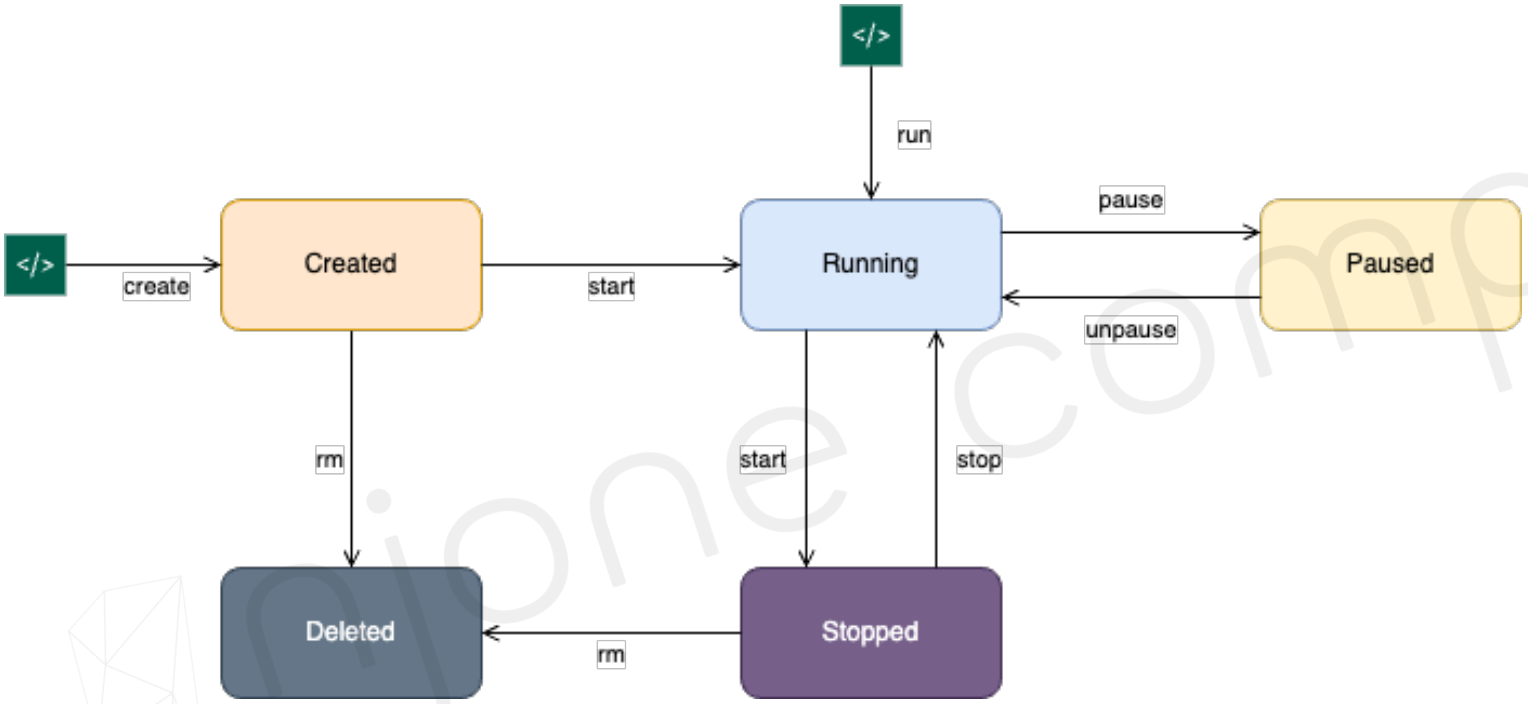
- **도커 이미지를 실행한 상태 → 컨테이너 (=인스턴스)**
  - 격리 된 시스템 자원 및 네트워크를 사용할 수 있는 독립적인 실행 단위
  - 읽기 전용 상태인 이미지에 변경 된 사항을 저장할 수 있는 컨테이너 계층에 저장



Docker Container 이해

- **Lifecycle**

  Docker Container의 실행과 중지에 따른 상태 변화



Docker Container 실행

▪ Docker Container 명령어 ①

$ docker run ***[OPTIONS]*** IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]

옵션	설명
-d, --detach	Detached mode, Background mode
-p, --publish	호스트와 컨테이너의 포트를 연결 (포워딩)
-v, --volume	호스트와 컨테이너의 디렉토리를 연결(마운트)
-e, --env	컨테이너 내에서 사용할 환경변수 설정
--name	컨테이너 이름 설정
--rm	프로세스 종료시 컨테이너 자동 제거
-it	-i와 –t를 동시에 사용한 것으로 터미널 입력을 위한 옵션
-link	컨테이너 연결 [컨테이너명:별칭]

Docker Container 실행

▪ Docker Container 명령어 ①

실습) ubuntu 16.04 container 생성하고 컨테이너 내부 접속

```
$ docker run ubuntu:16.04
```

```
[admin@centos7 ~]$ docker run ubuntu:16.04
Unable to find image 'ubuntu:16.04' locally
16.04: Pulling from library/ubuntu
35b42117c431: Pull complete
ad9c569a8d98: Pull complete
293b44f45162: Pull complete
0c175077525d: Pull complete
Digest: sha256:a4d8e674ee993e5ec88823391de828a5
Status: Downloaded newer image for ubuntu:16.04
```

Docker Container 실행

▪ Docker Container 명령어 ①

컨테이너는 프로세스이기 때문에 실행 중인 프로세스가 없으면 컨테이너는 종료 됨

```
$ docker run --rm -it ubuntu:16.04 /bin/bash
```

```
[admin@centos7 ~]$ docker run --rm -it ubuntu:16.04 /bin/bash
root@b0eac8e53eaa:/# cat /etc/issue
Ubuntu 16.04.6 LTS \n \l

root@b0eac8e53eaa:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc
```

Docker Container 실행

▪ Docker Container 명령어 ①

실습) MySQL 5.7 버전을 도커로 실행 https://hub.docker.com/_/mysql

```
$ docker run -d -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD=true --name mysql mysql:5.7
```

```
bcadmin@hlf03:~$ docker run -d -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD=true --name mysql mysql:5.7
Unable to find image 'mysql:5.7' locally
5.7: Pulling from library/mysql
f17d81b4b692: Pull complete
c691115e6ae9: Pull complete
41544cb19235: Pull complete
254d04f5f66d: Pull complete
4fe240edfdc9: Pull complete
0cd4fcc94b67: Pull complete
8df36ec4b34a: Pull complete
b8edeb9ec9e2: Pull complete
2b5adb9b92bf: Pull complete
5358eb71259b: Pull complete
e8d149f0c48f: Pull complete
Digest: sha256:42bab37eda993e417c5e7d751f1008b653c3fd85ad6aa416a519f161
Status: Downloaded newer image for mysql:5.7
842ff7eb6799b714050615ce505c1f96625343c0589f5d3124e2ec50c852b0f7
```

```
$ docker exec -it mysql bash
```

```
bcadmin@hlf03:~$ docker exec -it mysql bash
root@842ff7eb6799:/# mysql -h127.0.0.1 -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.24 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql>
```

Docker Container 실행

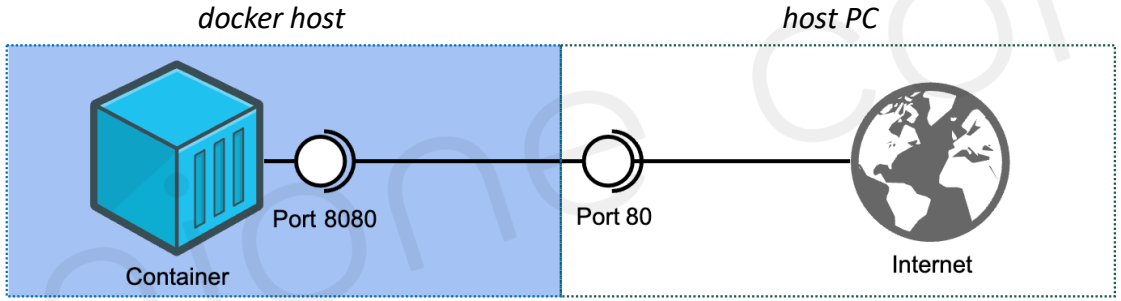
- **Docker Container 명령어 ②**

```
$ docker container ls [OPTIONS]  ( = docker ps )
$ docker container stop [OPTIONS] CONTAINER [CONTAINER ...]
$ docker container rm [OPTIONS] CONTAINER [CONTAINER ...]
$ docker container logs ${CONTAINER_ID}
        ex) docker logs -f $[CONTAINER_ID}
$ docker container exec [OPTIONS] CONTAINER COMMAND [ARG...]
        ex) docker exec -it mysql /bin/bash
$ docker container inspect ${CONTAINER_ID}
$ docker image ls [OPTIONS] [REPOSITORY[:TAG]]
$ docker image rm [OPTIONS] IMAGE [IMAGE ...]
$ docker image pull [OPTIONS] NAME[:TAG|@DIGEST]
        ex) docker pull ubuntu:16.04
$ docker system prune
```

Docker Container 실행

- **Port Mapping**
  - 도커 컨테이너에서 사용하고자 하는 Port를 자유롭게 설정 가능
  - 호스트 시스템에서 도커 컨테이너 Port를 사용하기 위해서는 Port Mapping 필요
  - $ docker run -p *host_port:container_port* <IMAGE NAME>

```
$ docker run -p 80:8080 <IMAGE NAME>
```

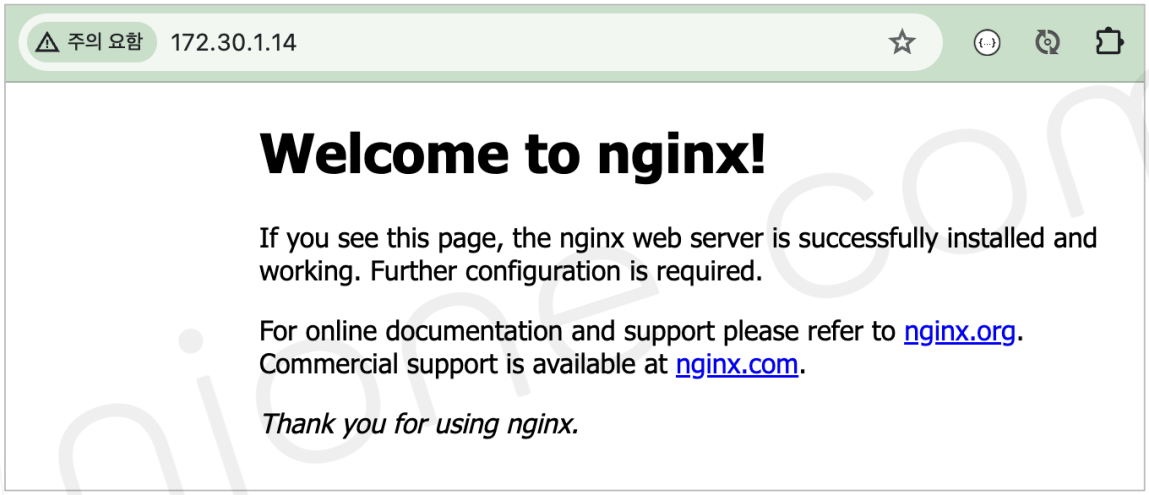


Docker Container 실행

- **Port Mapping**

실습) 다음과 같은 명령어로 nginx 서버를 기동하였을 때, host PC에서의 접속 방법은?

```
$ docker run -p 80:80 nginx
```



Docker Container 실행

- **Detached Mode**

  컨테이너를 백그라운드 모드를 실행

```
$ docker run -d -p 80:80 nginx
```

```
docker run -d -p 80:80 nginx
2fc5d862942c9898b63dd914161fbdab90a1f7018dff0c025f113a47f5e5724a
(base)  edowon  ~
```

- **Container List**

  컨테이너를 백그라운드 모드를 실행

```
$ docker ps
```

```
docker ps
CONTAINER ID   IMAGE
2fc5d862942c   nginx
```

Docker Container 실행

- **Remove Container**

  컨테이너 종료와 함께 컨테이너 삭제

```
$ docker run --rm  -d -p 80:80 nginx
```

```
docker stop 64fe72
64fe72
(base)  edowon  ~
docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED         STATUS         PORTS
```

```
docker run --rm -d -p 80:80 nginx
64fe724dff0efa5f05e1195132baee36cbf625ece1c20c2a78b7feb2609c6e3f
(base)  edowon  ~
docker ps -a
CONTAINER ID   IMAGE                     COMMAND                  CREATED          STATUS          PORTS
64fe724dff0e   nginx                     "/docker-entrypoint.…"   10 seconds ago   Up 9 seconds    0.0.0.0:80->80/tcp
```

Docker Container 실행

- **Docker Container에 명령어 전달 → 실행**

  실행 된 도커 컨테이너 내부에 명령어 실행

```
$ docker exec <CONTAINER_ID> <COMMAND>
```

```
$ docker exec 6sdkjh ls
```

```
$ docker exec -it 6sdkjh bash
```

Docker로 데이터베이스 실행

- **실습) MySQL 5.7 버전을 도커로 실행** https://hub.docker.com/_/mysql

```
$ docker run -d -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD=true --name mysql mysql:5.7
```

- **실습) MySQL Client로 접속 테스트**
  - MySQL Workbench https://dev.mysql.com/downloads/workbench/
  - HeidiSQL https://www.heidisql.com/
  - DataGrip https://www.jetbrains.com/datagrip/

Docker로 Node.js 실행

- 실습) 아래 Github의 Repository를 Clone

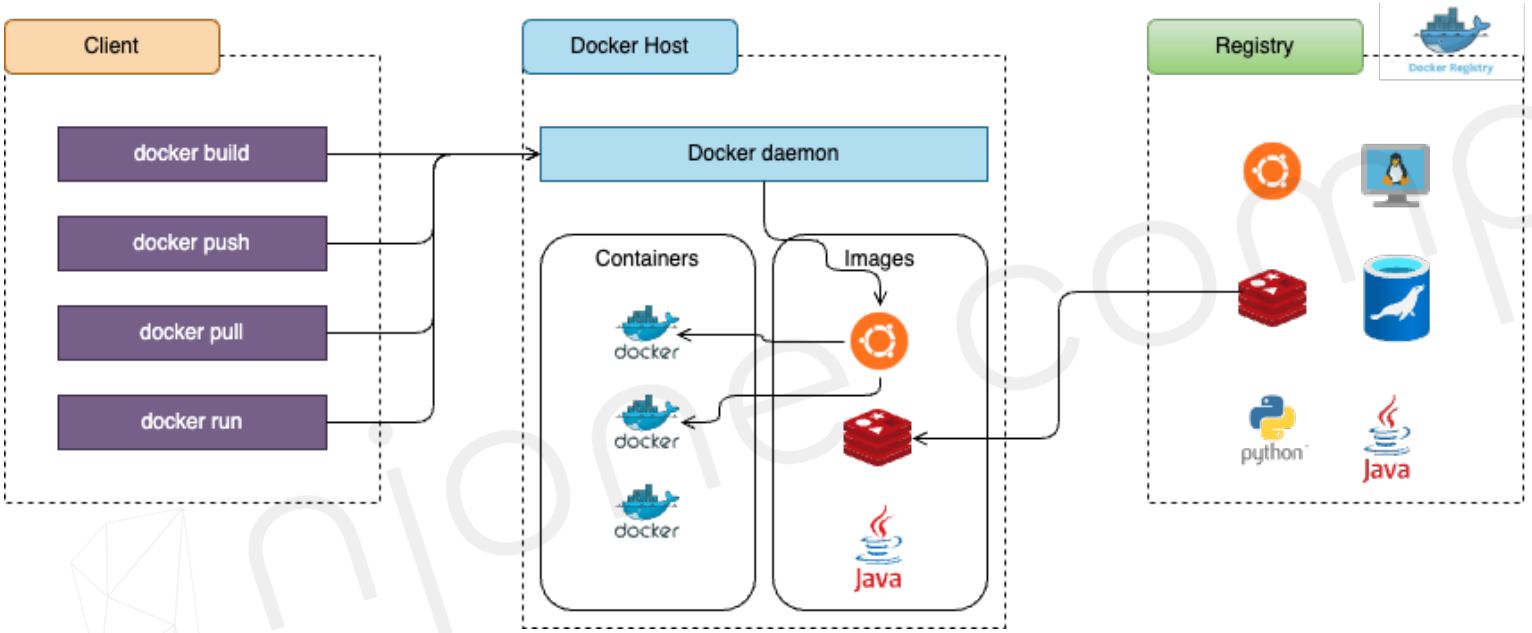
  https://github.com/joneconsulting/devops-docker

```
$ cd devops-docker/my-nodejs
$ docker run -it -v ./docker_nodejs:/app -p 8000:8000 node:alpine sh
/ # cd app
/ # node app.js
```

Docker Image 작성

Docker Image (= Docker 실행 환경)

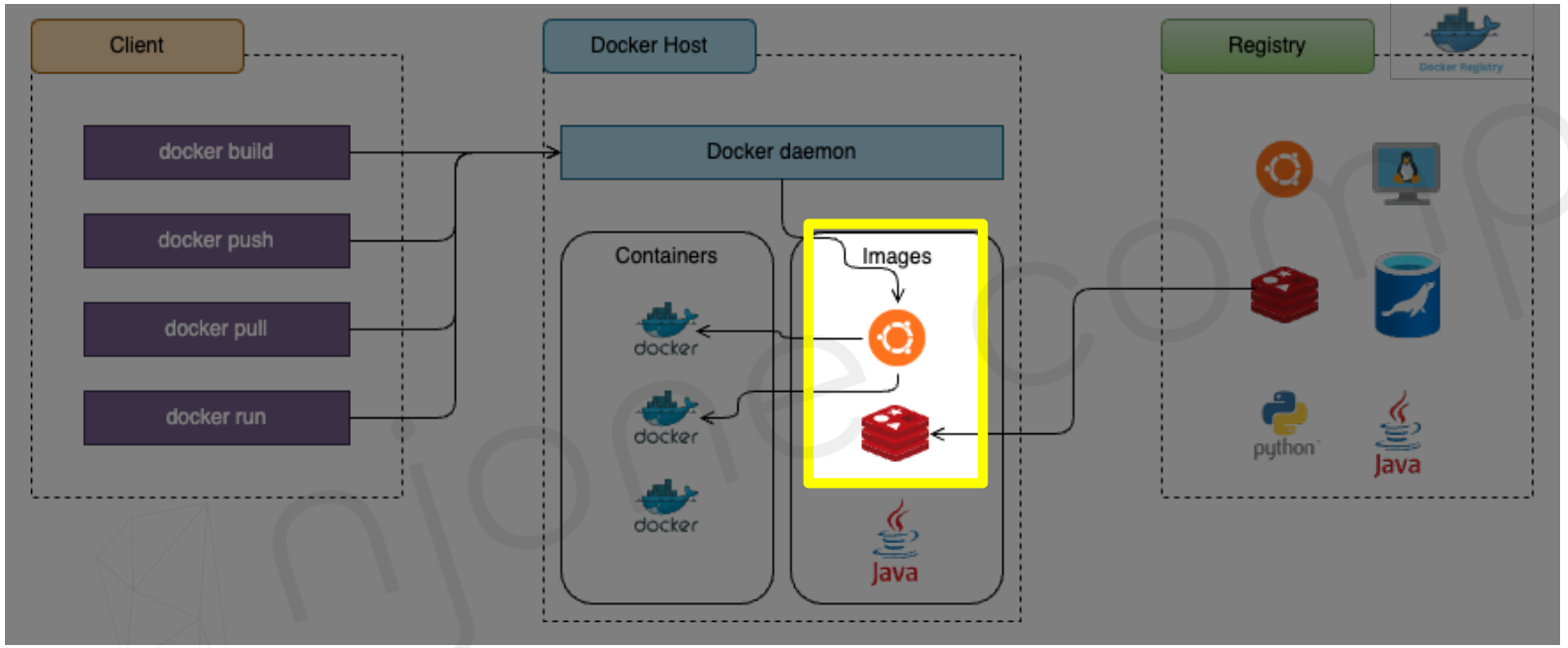
- 컨테이너를 만드는 데에 필요한 읽기 전용 상태의 템플릿
- 컨테이너 실행에 필요한 파일과 설정 값 등을 포함하고 있지만, 상태 값 X
  - 실체화 → Container



Docker Image 작성

Docker Image (= Docker 실행 환경)

- 컨테이너를 만드는 데에 필요한 읽기 전용 상태의 템플릿
- 컨테이너 실행에 필요한 파일과 설정 값 등을 포함하고 있지만, 상태 값 X
  - 실체화 → Container



Docker Image 작성

Docker Image 생성 방법

- **Dockerfile: 이미지 빌드용 DSL(Domain Specific Language)**

```
$ touch Dockerfile
$ vi Dockerfile
```

```
FROM ubuntu:16.04
```

```
$ docker build --tag first-image:0.1 -f Dockerfile .
```

```
$ docker image ls
```

Docker Image 생성 방법

- **Dockerfile 작성 명령어**

명령어	설명
FROM	Base Image 지정 명령어
RUN	특정 Layer 생성
COPY	이미지 파일 생성 시 호스트 파일 복사
ADD	이미지 파일 생성 시 호스트 파일 복사 (tar, url 가능)
WORKDIR	이미지 파일 생성 시 명령어가 실행 될 작업 디렉토리
ENTRYPOINT	컨테이너가 실행 될 때 가장 먼저 실행될 프로그램 지정 (컨테이너 실행 시 명령어 Overwrite 불가능)
CMD	컨테이너가 실행 될 때 가장 먼저 실행될 프로그램 지정 (컨테이너 실행 시 명령어 Overwrite 가능)
ENV	컨테이너 내의 환경변수 설정
EXPOSE	컨테이너의 특정 포트를 외부에 오픈

가상화 시스템 구축

Docker 환경에서 웹 서비스 구축하기 – Node.js

- https://github.com/joneconsulting/devops-docker/tree/section2/my-nodejs

```
{
    "name": "example",
    "version": "1.0.0",
    "main": "app.js",
    "scripts": {
        "build": "babel app.js -d dist",
        "serve": "node app.js"
    },
    "dependencies": {
        "express": "^4.17.1"
    },
    "devDependencies": {
        "@babel/cli": "7.18.10",
        "@babel/core": "7.19.1",
        "@babel/node": "7.19.1",
        "@babel/runtime": "7.12.5",
        "@babel/preset-env": "7.19.1"
    }
}
```

package.json

```
var express = require('express');
var app = express();

app.get('/', (req, res) => {
  res.status(200).json({
    message: `Hello, ${req.query.name}!`
  });
});

app.listen(8000, () => {
  console.log(`Example app for CI/CD listening on port 8000`)
})
```

app.js

가상화 시스템 구축

Docker 환경에서 웹 서비스 구축하기 – Node.js

- **Docker Image 작성하기**

Step 1)

```
1 FROM alpine
2
3 RUN npm install
4
5 CMD ["npm", "start"]
```

Step 2)

```
1 FROM node:apline
2
3 RUN npm install
4
5 CMD ["npm", "start"]
```

Step 3)

```
1 FROM node:apline
2
3 COPY ./package.json ./package.json
4 COPY ./index.js ./index.js
5
6 RUN npm install
7
8 CMD ["npm", "start"]
```

Step 4)

```
1  FROM node:apline
2
3  WORKDIR /home/node
4
5  COPY ./package.json ./package.json
6  COPY ./index.js ./index.js
7
8  RUN npm install
9
10 CMD ["npm", "start"]
```

Docker Registry

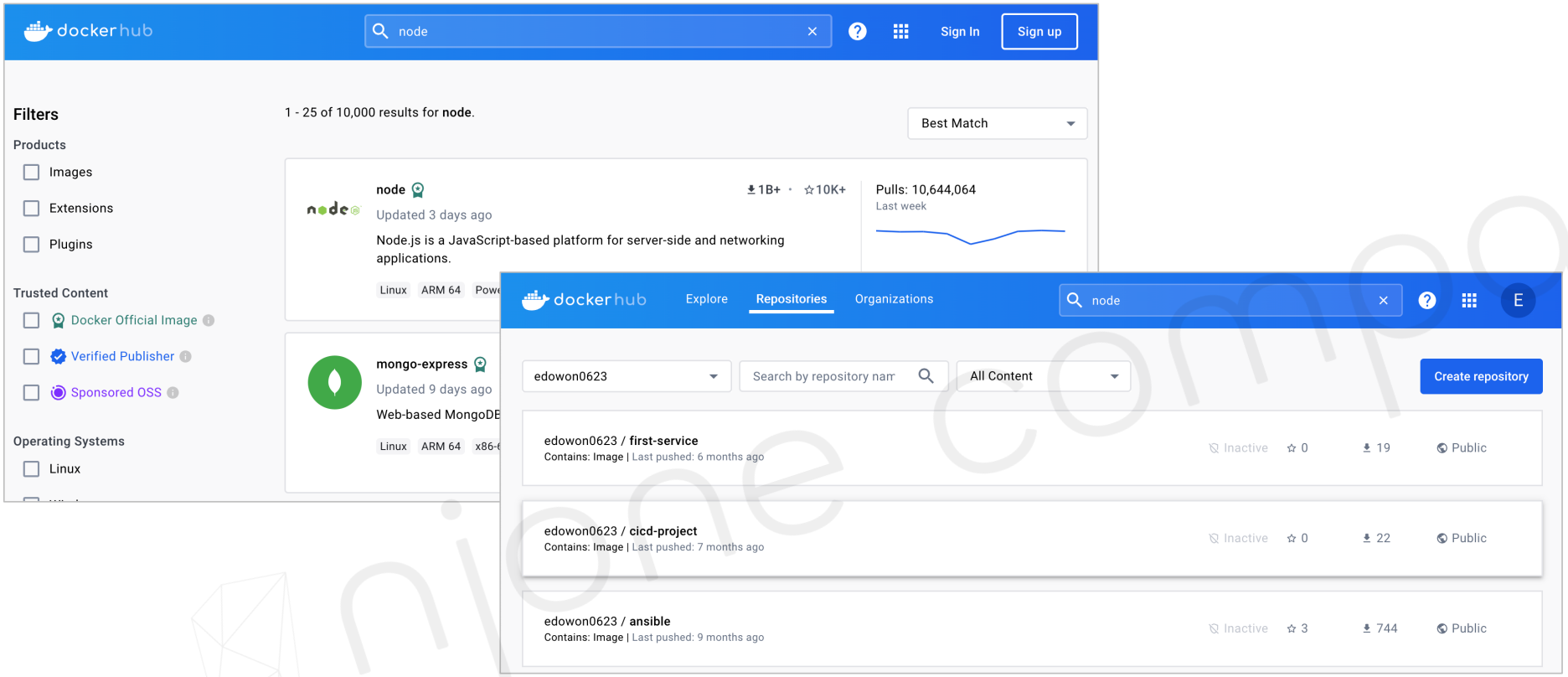
Docker Image 저장소란?

- **Docker를 통해 생성하는 Image들을 저장해주는 저장소 (Repository)**
  - Public Registry (Docker Hub 사이트)
  - Private Registry (기업망 내부 / 자신의 Local PC)
- **Docker Image들의 위치 제어 및 CI/CD를 위한 자동화 Pipeline 구축 가능**

Docker Registry

Docker Image 저장소 사용

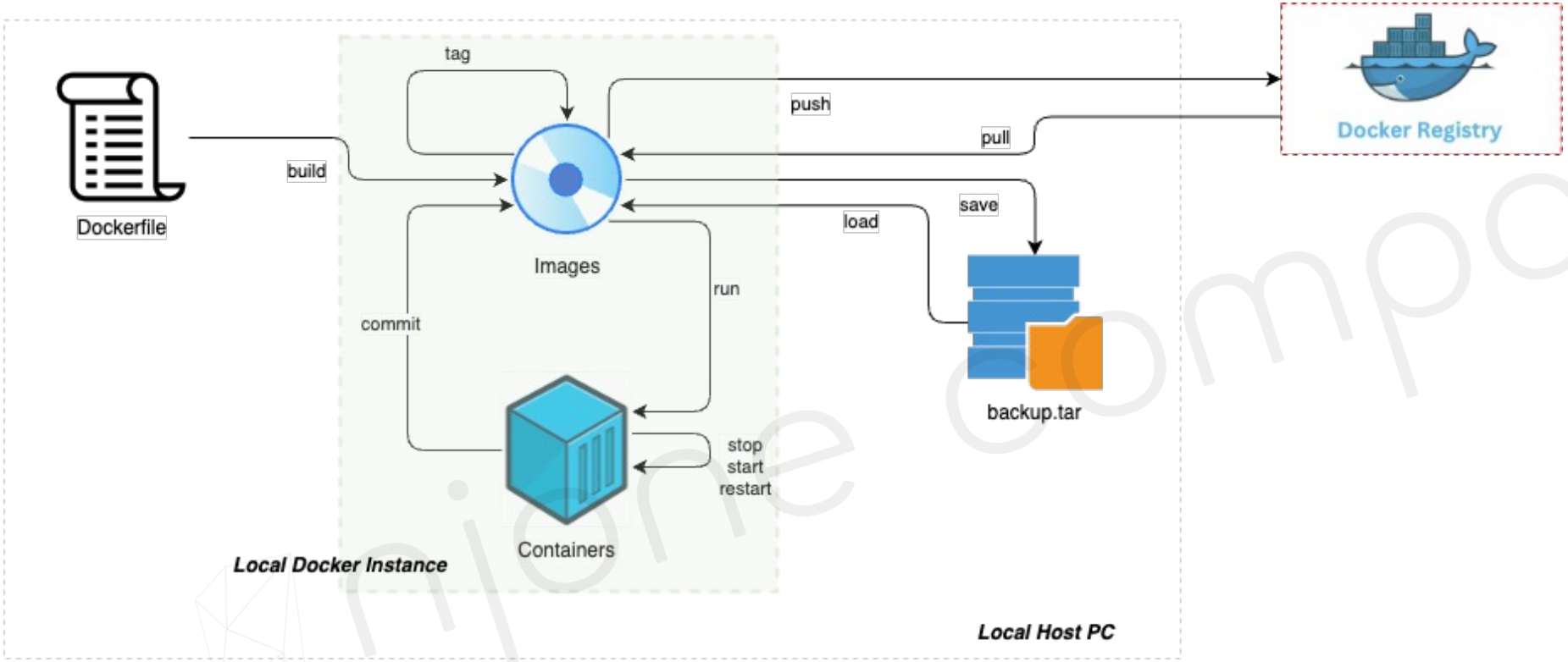
- https://hub.docker.com



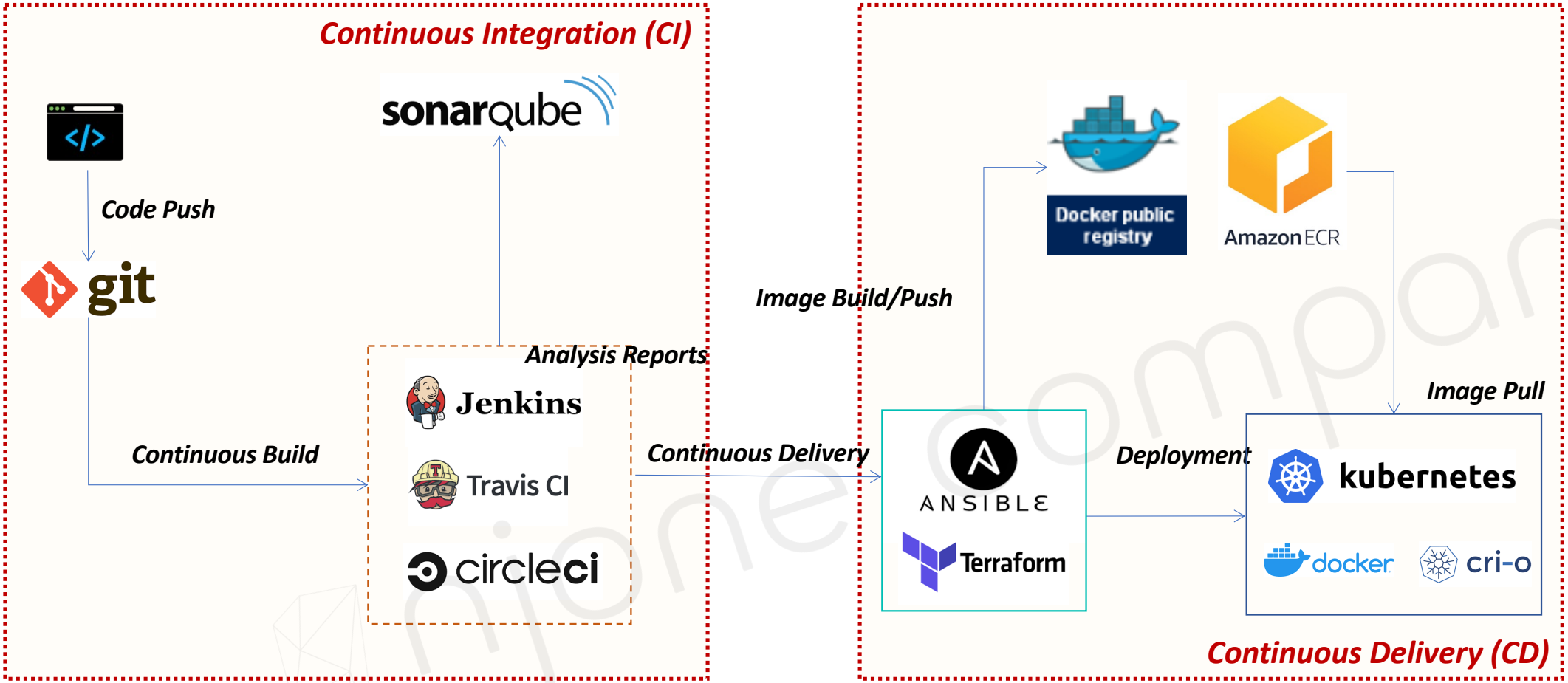
Docker Registry

Docker Image 저장소 사용

- Docker Image의 생성과 사용 Flow



CI/CD 자동화 파이프라인



Local Registry 사용

- **Registry Container 사용**

  https://hub.docker.com/search?q=registry

```
$ docker run -d -p 5000:5000 --restart always --name registry registry:2
```

```
$ docker pull ubuntu
```

```
$ docker tag ubuntu localhost:5000/ubuntu
```

```
$ docker push localhost:5000/ubuntu
```

Docker Registry

Cloud Registry 사용

- **Docker Hub 사이트**

  https://hub.docker.com