



Docker+CI/CD - 1주차



본 강의에서는 AWS 클라우드를 사용한 실습을 진행합니다.

실습을 진행할 때는 Free Tier(무료 요금제)를 사용하여 요금이 발생하지 않지만,

강의 실습 이후에는 아래의 자료를 참조하여 추가 요금이 발생하지 않도록 해주세요!



[\[스파르타코딩클럽\] AWS 사용/해지 가이드_2024](#)

[목차]

00. 1주차 오늘 배울 것



목표

01. CI/CD란 무엇인가?

02. Docker 기초

왜 Docker인가

Docker 및 Docker Compose 설치 (Windows용)

Docker 및 Docker Compose 설치 (MacOS용)

Container 실행테스트

03. Docker Image 관리

docker image 이해와 구조 확인

04. Docker Container와 Container 를 다루는 CLI

1주차 끝



모든 토글을 열고 닫는 단축키

Windows :

`Ctrl + alt + t`

Mac :

`⌘ + ⌥ + t`

00. 1주차 오늘 배울 것

✓ CI/CD와 Docker에 대한 기초적인 내용을 학습해요.

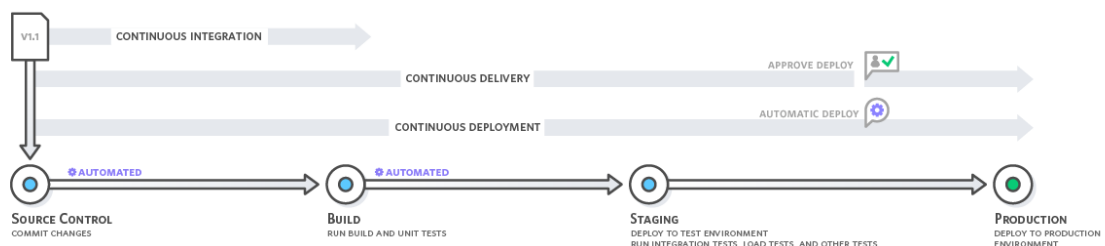
✓ 목표

- CI/CD 사용 이유를 이해합니다.
- Docker 사용 이유를 이해합니다.
- 윈도우11 이나 MacOS에서 도커 실행 환경을 구성합니다.
- 간단한 Container 서비스 구현 실습을 통해 앱을 실행합니다.

01. CI/CD란 무엇인가?

✓ CI/CD가 무엇인지 알아보아요

▼ CI/CD의 정의

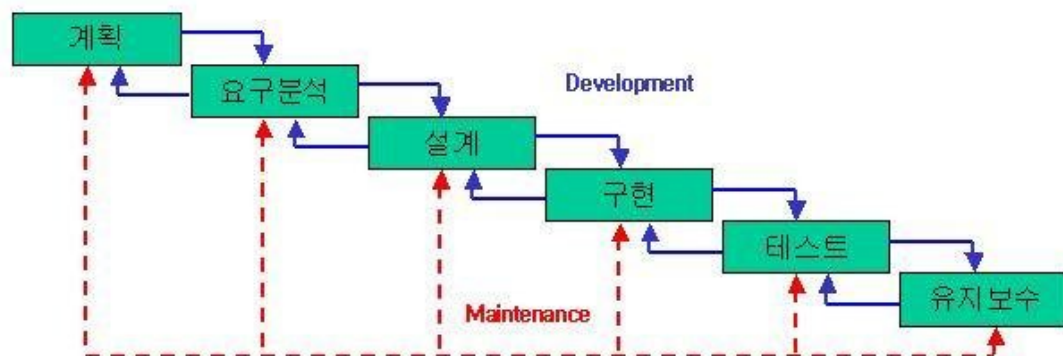


- Continuous Integration/Continuous Deployment(Delivery)의 약자로, 지속적인 통합과 지속적인 제공을 의미
- 기본 개념
 - 지속적인 통합(Continuous Integration)
 - 지속적인 서비스 제공 (Continuous Delivery)

- 지속적인 배포(Continuous Deployment)
- 지속적인 통합(Continuous Integration) : 코드의 지속적인 통합
 - 자동화된 빌드와 자동화된 테스트를 제공
 - 안정적인 코드를 빠르게 제공할 수 있는 밑거름
- 지속적인 서비스 제공(Continuous Delivery)
- 지속적인 배포(Continuous Deployment)
 - 배포를 자동화하여 배포 시간을 단축하고 코드 결과물을 빠르게 지속적으로 제공
- 단계
 - **코드 작성**: 개발자들은 소스 코드를 작성하고 저장소(repository)에 업로드
 - **빌드**: 저장소에서 최신 소스 코드를 가져와 빌드를 수행. 빌드는 소스 코드를 컴파일하고, 라이브러리를 추가하고, 필요한 파일을 생성하는 과정.
 - **테스트**: 빌드된 결과물을 대상으로 테스트를 수행. 테스트는 기능이 정상적으로 작동하는지 확인하고, 버그를 발견하고 수정하는 과정.
 - **배포**: 테스트를 통과한 결과물을 배포. 배포는 서버에 업로드하거나, 사용자에게 제공하는 과정

▼ 과거의 배포

- 소프트웨어 개발주기(SDLC, Software Development Life Cycle) 중 폭포수 개발 방식

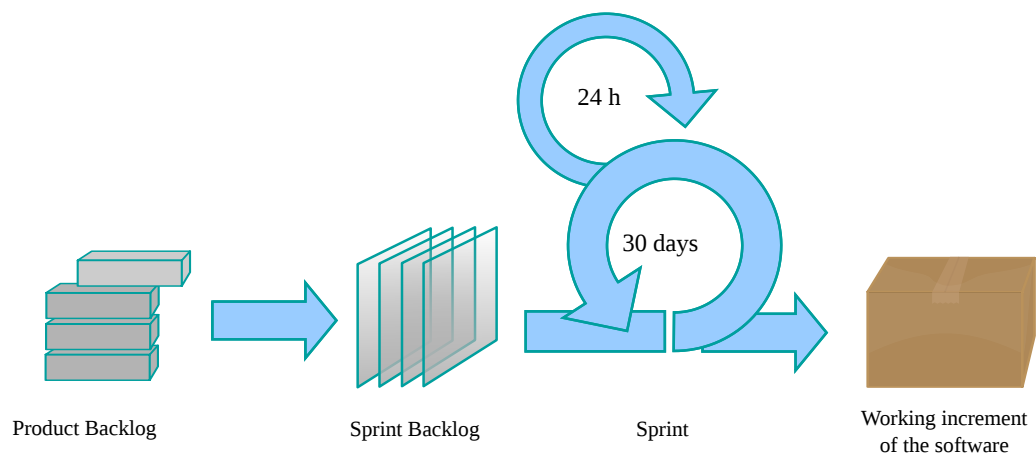


- 오랜 시간 동안 구현하고 테스트하여 가끔 배포
 - 이 과정에서 한 대의 서버에 배포하기 위해 다음과 같은 과정을 n개 서버에 반복

- 클러스터에서 서버 1 분리
 - 서버 1에서 톰캣 종료
 - 기존 앱 버전(WAR 파일) 제거
 - 새 앱 버전(WAR 파일) 복사
 - ssh 접속 또는 scp 복사
 - 구성 파일의 속성 업데이트
 - 톰캣 시작
 - 클러스터에 서버 1 다시 연결
- 출시 직후 진행된 프로덕션 테스트에서 아무도 발견하지 못한 버그가 있는 경우에는 어떻게 해야 할까요?
 - 되돌리거나(롤백) 빠르게 수정하고 테스트하여 다시 추가 배포하거나 하는 큰 비용

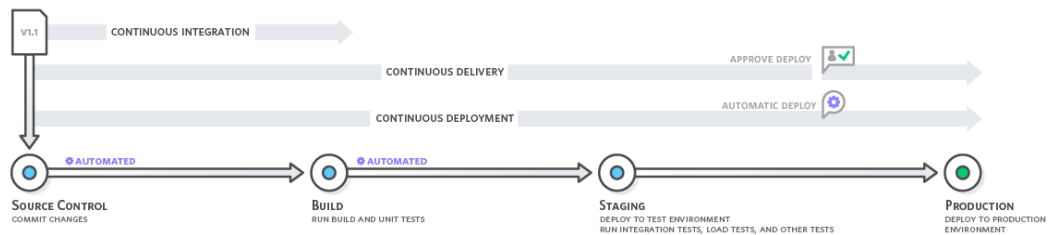
▼ 현대적인 개발 과정

- 스크럼으로 대표되는 애자일 개발

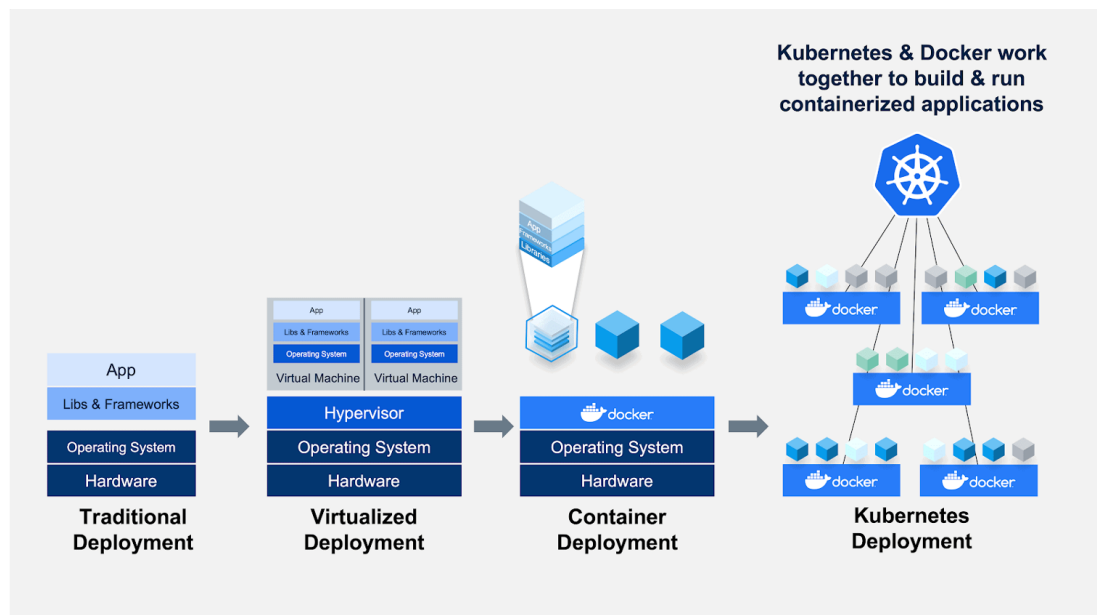


- 특정 주기마다 개발, 테스트 및 프로덕션에 통합된 기능을 출시
- 이 상황에서 테스트 및 기능 출시에 오랜 기능이 걸리고 “손배포” 를 통해 실패 위험성을 안고 있다면?
 - 빠르게 배포하는 것이 사실상 불가능

- Docker를 통해 서버를 표준화하고 같은 환경에서 테스트 및 배포 테스트를 진행하고 이 과정을 자동화
 - 테스트로 검증된 자동화 배포를 사용하여 실패 확률 저하
- 자동화된 과정으로 지속적으로 코드를 통합하여 지속적으로 자동 배포



- 컨테이너와 빌드/테스트 도구의 발전에 따라 Docker가 테스트 뿐만 아니라 실제 배포도 담당



<https://k21academy.com/docker-kubernetes/docker-and-kubernetes/>

02. Docker 기초

왜 Docker인가

✓ Docker를 사용하는 이유를 알아보아요.

▼ 도커 사용 이유

- 애플리케이션 개발과 배포가 편해져요.
 - Docker Container 내부에서 여러 소프트웨어를 설치해도 호스트 OS에는 영향이 없어요.
 - CI/CD에서 지속적인 통합(Continuous Integration) 과정의 테스트에서 Docker를 활용해요.
 - 어떤 서버에 올리더라도 같은 환경으로 구성된 컨테이너로 동작하기 때문에 표준화된 배포를 구성할 수 있어요.
- 여러 애플리케이션의 독립성과 확장성이 높아져요.
- Docker가 가상화에서 사실상 표준의 위치에요.



지금은 이해하기 어려운 내용이지만, 이 과정을 모두 듣고 나서 이 부분을 다시 살펴보시면 잘 이해하실 수 있을 거예요.

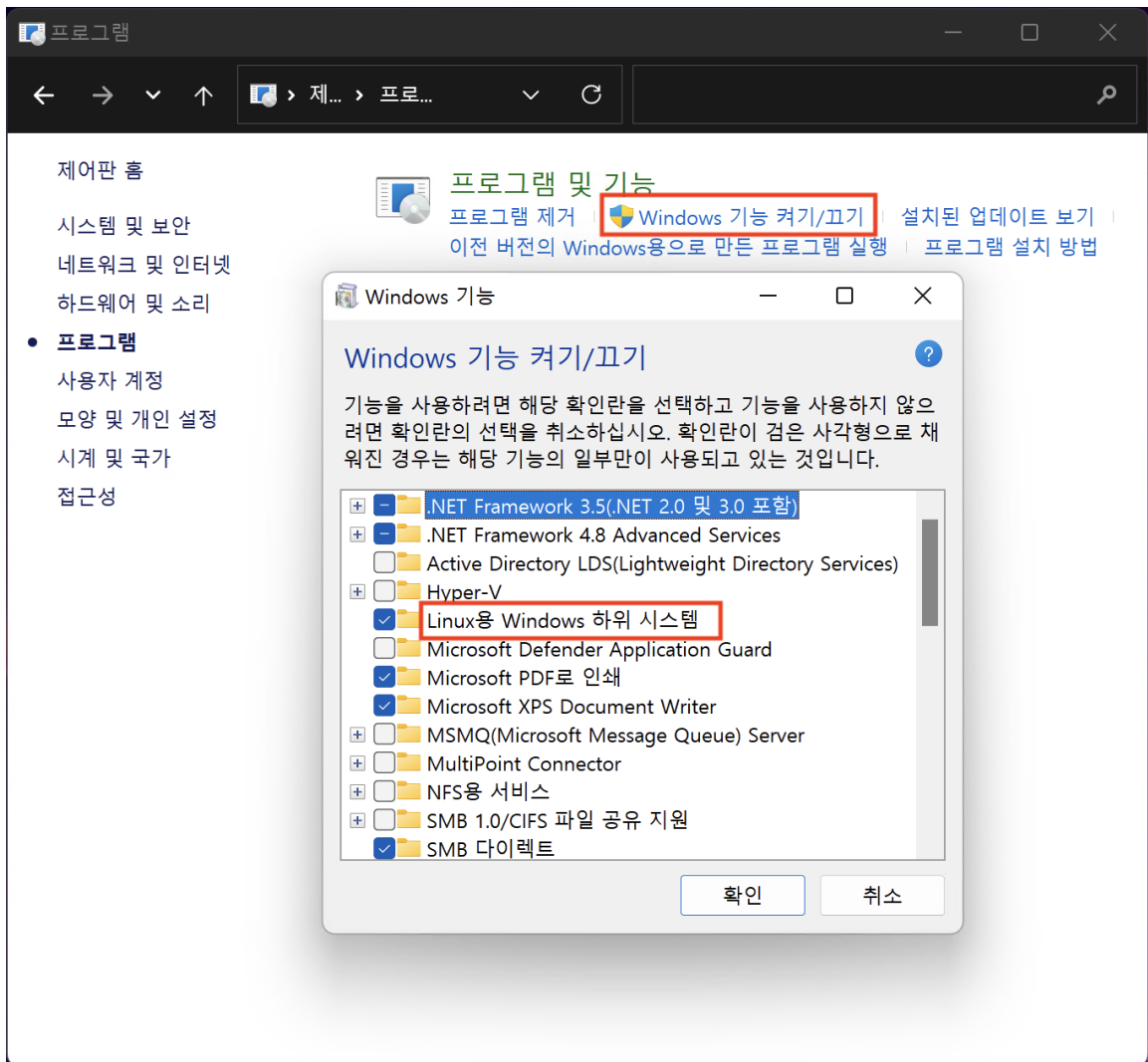
Docker 및 Docker Compose 설치 (Windows용)



Docker를 학습하기 위한 기본 설정을 준비해요.

▼ WSL2 설치를 위한 사전 준비

시작 버튼 → 제어판 → 프로그램 및 기능 → Windows 기능 켜기/끄기 → Linux용 Windows 하위 시스템



▼ WSL2 설치

WSL 설치

wsl --install 명령을 사용하여 Linux용 Windows 하위 시스템을 설치합니다. Ubuntu, Debian, SUSE, Kali, Fedora, Penguin, Alpine 등 원하는 Linux 배포판에서 실행되는

 <https://learn.microsoft.com/ko-kr/windows/wsl/install>

 Microsoft Learn

▼ wsl 설치

```
wsl --install
```

```
dism.exe /online /enable-feature /featurename:VirtualMa
```

```
wsl --update
```

▼ Ubuntu 22.04 설치

Ubuntu 22.04.2 LTS - Microsoft Store 공식 앱

Install a complete Ubuntu terminal environment in minutes with Windows Subsystem for Linux (WSL).
Develop cross-platform applications, improve your data

 <https://apps.microsoft.com/detail/9PN20MSR04DW?gl=KR&hl=ko-kr>



```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: user
New password:
Retype new password:
passwd: password updated successfully
작업을 완료했습니다.
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.90.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This message is shown once a day. To disable it please create the
/home/user/.hushlogin file.
user@jeff-x1-gen11:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:   Ubuntu 22.04.2 LTS
Release:      22.04
Codename:     jammy
user@jeff-x1-gen11:~$
```

▼ Ubuntu 22.04 에 최신 버전의 docker 설치

```
# docker engine gpg 키 등록
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# apt source 에 docker 관련 추가
echo \
    "deb [arch="$(dpkg --print-architecture)" signed-by=/etc
```



```

"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# docker engine 설치
sudo apt-get install -y docker-ce docker-ce-cli containerd
docker-buildx-plugin docker-compose-plugin docker-compose

# docker 그룹에 현재 계정을 등록하여 sudo 없이 docker 명령을 사용하
sudo usermod -aG docker user
sudo service docker restart

# 새로운 터미널을 열고 확인
docker version

```

▼ Docker 및 Docker Compose 버전 확인


```

# docker version 확인
docker --version

# docker compose version 확인
docker-compose --version


```

▼ Windows 에 Docker Desktop 설치



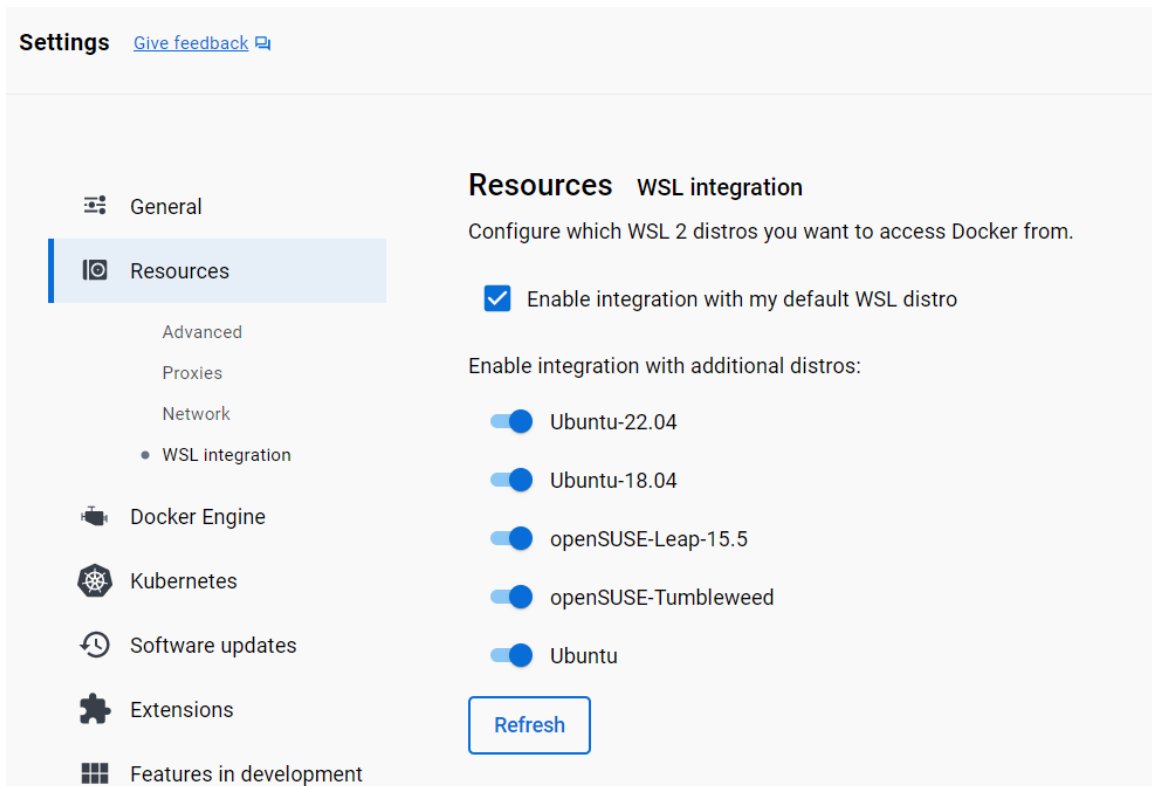
Install Docker Desktop on Windows

Get started with Docker for Windows. This guide covers system requirements, where to download, and instructions on how to install and update.

 [https://docs.docker.com/desktop/install/windows-in
stall/](https://docs.docker.com/desktop/install/windows-install/)

▼ Windows Docker Desktop 설정

우측 상단 톱니바퀴 → 왼쪽 Resources → WSL Integration → Apply&restart



Docker 및 Docker Compose 설치 (MacOS용)

✓ Docker를 학습하기 위한 기본 설정을 준비해요.

▼ homebrew를 이용해 Docker, Docker Compose 설치

- <https://brew.sh/ko>
 - 커맨드 복사
 - 터미널에서 복사한 커맨드 실행
 - 설치 후 `brew doctor` 실행

```
# 사전에 homebrew 설치 필수
```

```
# docker for mac 설치  
brew install docker docker-compose
```


```
# docker version 확인
```

```
docker --version
```

```
# docker compose version 확인  
docker-compose --version
```

패키지 관리자 (Apple Silicon/M1)

패키지 관리자 설정 방법을 알아봅니다.

 <https://subicura.com/mac/dev/apple-silicon.html#command-line-tools>

>_


**A
Beginner's Guide
to macOS**
by subicura



▼ Docker 공식 사이트에서 Docker Desktop 다운로드 및 설치

Install Docker Desktop on Mac

Install Docker for Mac to get started. This guide covers system requirements, where to download, and instructions on how to install and update.

 <https://docs.docker.com/desktop/install/mac-install/>



Container 실행테스트

✓ 예제를 통해 Docker 가 잘 실행되는지 확인해요.

▼ Docker 엔진과 구성 확인

```
docker info
```

✓ Docker 엔진이 잘 설치되었는지 확인해요.

▼ Container 실행 테스트

```
# nginx 이미지 다운받기
docker image pull nginx:1.25.3-alpine

docker images

docker image history nginx:1.25.3-alpine

docker run -d -p 8001:80 --name webserver01 nginx:1.25.3-a

docker ps | grep webserver01

docker port webserver01

curl localhost:8001
```

```
cackyyhk@jeff-x1-gen11:~$ docker image pull nginx:1.25.3-alpine
1.25.3-alpine: Pulling from library/nginx
96526aa774ef: Already exists
740091335c74: Pull complete
da9c2e764c5b: Pull complete
ade17ad21ef4: Pull complete
4e6f462c8a69: Pull complete
1324d9977cd2: Pull complete
1b9b96da2c74: Pull complete
153aef7ca07f: Pull complete
Digest: sha256:db353d0f0c479c91bd15e01fc68ed0f33d9c4c52f3415e63332c3d0bf7a4bb77
Status: Downloaded newer image for nginx:1.25.3-alpine
docker.io/library/nginx:1.25.3-alpine
cackyyhk@jeff-x1-gen11:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
nginx                1.25.3-alpine      b135667c9898       2 weeks ago        47.7MB
cackyyhk@jeff-x1-gen11:~$ docker image history nginx:1.25.3-alpine
IMAGE               CREATED             CREATED BY          SIZE      COMMENT
b135667c9898       2 weeks ago       /bin/sh -c set -x   && apkArch="$(cat /etc... 30.7MB
<missing>          2 weeks ago       /bin/sh -c #(nop)  ENV NJS_VERSION=0.8.2  0B
<missing>          2 weeks ago       /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon... 0B
<missing>          2 weeks ago       /bin/sh -c #(nop)  STOPSIGNAL SIGQUIT        0B
<missing>          2 weeks ago       /bin/sh -c #(nop)  EXPOSE 80                    0B
<missing>          2 weeks ago       /bin/sh -c #(nop)  ENTRYPOINT ["/docker-entr... 0B
<missing>          2 weeks ago       /bin/sh -c #(nop)  COPY file:9e3b2b63db9f8fc7... 4.62kB
<missing>          2 weeks ago       /bin/sh -c #(nop)  COPY file:57846632acc8975... 3.02kB
<missing>          2 weeks ago       /bin/sh -c #(nop)  COPY file:3b1b9915b7dd898a... 298B
<missing>          2 weeks ago       /bin/sh -c #(nop)  COPY file:caec368f5a54f70a... 2.12kB
<missing>          2 weeks ago       /bin/sh -c #(nop)  COPY file:01e75c6dd0ce317d... 1.62kB
<missing>          2 weeks ago       /bin/sh -c set -x   && addgroup -g 101 -S ... 9.61MB
<missing>          2 weeks ago       /bin/sh -c #(nop)  ENV PKG_RELEASE=1          0B
<missing>          2 weeks ago       /bin/sh -c #(nop)  ENV NGINX_VERSION=1.25.3   0B
<missing>          3 weeks ago       /bin/sh -c #(nop)  LABEL maintainer=NGINX Do... 0B
<missing>          6 weeks ago       /bin/sh -c #(nop)  CMD ["/bin/sh"]           0B
<missing>          6 weeks ago       /bin/sh -c #(nop)  ADD file:756183bba9c7f4593... 7.34MB
cackyyhk@jeff-x1-gen11:~$
```

```

user@jeff-x1-gen11:~$ docker run -d -p 8001:80 --name webserver01 nginx:1.25.3-alpine
82124d423142c5b24f59a7a752ab7edb5ceca240c1749845bd820a2cf7aed8aa
user@jeff-x1-gen11:~$ docker ps | grep webserver01
82124d423142 nginx:1.25.3-alpine "/docker-entrypoint..." 7 seconds ago Up 6 seconds 0.0.0.0:8001->80/tcp webs
erver01
user@jeff-x1-gen11:~$ docker port webserver01
80/tcp -> 0.0.0.0:8001
user@jeff-x1-gen11:~$ curl localhost:8001
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
user@jeff-x1-gen11:~$

```

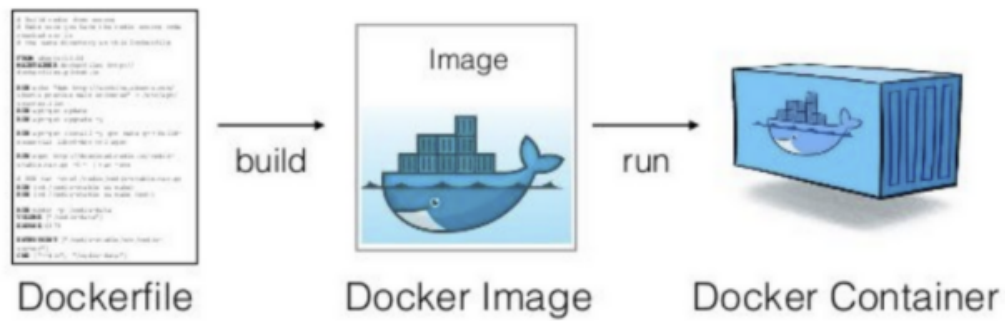
03. Docker Image 관리

docker image 이해와 구조 확인

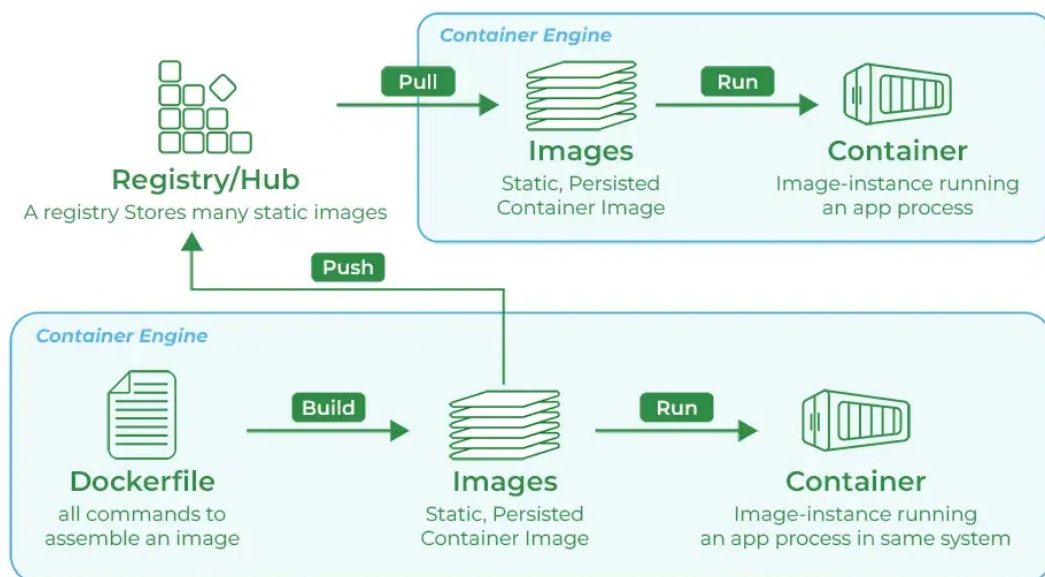
✓ Docker Image에 대해서 알아봅니다.

▼ Docker 이미지 이해

- Docker Container 서비스를 위한 이미지는 Container 런타임에 필요한 바이너리, 라이브러리 및 설정 값 등을 포함하고, 변경되는 상태값을 보유하지 않고 (stateless) 변하지 않음(Immutable, Read-Only)
- **상태 저장 없음(Stateless):** 애플리케이션과 관련된 모든 파일과 라이브러리를 포함하고 있기 때문에, 다른 환경에서도 동일한 애플리케이션을 실행 가능
- **불변성(Immutable):** 이미지가 한번 생성되면 변경할 수 없는 것을 의미
- 도커 이미지는 필요한 파일만 포함하고 있기 때문에, 용량이 작으며, 이미지를 변경할 필요가 있을 경우에는 새로운 이미지를 생성 필요



▼ **docker pull** : Docker 이미지 내려받기



- hub.docker.com 에서 이미지를 제공받거나 해당 사이트로 이미지를 제공
- Private Registry 서버를 통해 이미지를 제공받거나 제공 가능
- 예제

```
# docker [image] pull [options] name:[tag]

# 최초에는 docker.io가 default registry로 설정됨.
docker pull debian[:latest]
docker pull library/debian:10
docker pull docker.io/library/debian:10
```

```
docker pull index.docker.io/library/debian:10
docker pull nginx:latest

# private registry 나 클라우드 저장소의 이미지를 받는 경우
docker pull 192.168.0.101:5000/debian:10 # 현재는 실제로 동작
docker pull gcr.io/google-samples/hello-app:1.0
```

▼ `docker image inspect` : Docker 이미지 구조 확인

```
docker image inspect nginx:latest
[
  {
    "Id": "sha256:593aee2afb642798b83a85306d2625fd7f08",
    "RepoTags": [
      "nginx:latest"
    ],
    "RepoDigests": [
      "nginx@sha256:add4792d930c25dd2abf2ef9ea79de57"
    ],
    "Parent": "",
    "Comment": "",
    "Created": "2023-10-25T01:21:47.343274012Z",
    "Container": "1e4063a23e5d6d56cbf5478ff7227b8c6940",
    "ContainerConfig": {
      "Hostname": "1e4063a23e5d",
      "Domainname": "",
      "User": "",
      "AttachStdin": false,
      ...
    }
  }
]
```

▼ `docker image inspect` : 생성된 이미지의 내부 구조 정보를 json 형태로 제공

```
docker image inspect --format="{{.Os}}" nginx:latest
linux
docker image inspect --format="{{.RepoTags}}" nginx:latest
[nginx:latest]
docker image inspect --format="{{.ContainerConfig.ExposedPorts}}" nginx:latest
map[80/tcp:{}]
```

```
docker image inspect --format="{{.RepoTags}} {{.Os}}" nginx:latest
[nginx:latest] linux
```

▼ `docker image history` : `Dockerfile` 에 대한 정보

- 여러 개의 계층 구조로 구성

```
docker image history nginx:latest
```

IMAGE	CREATED	CREATED BY
593aee2afb64	4 days ago	/bin/sh -c #(nop) CMD ["nginx"]
<missing>	4 days ago	/bin/sh -c #(nop) STOPSIGNAL
<missing>	4 days ago	/bin/sh -c #(nop) EXPOSE 80
<missing>	4 days ago	/bin/sh -c #(nop) ENTRYPOINT
<missing>	4 days ago	/bin/sh -c #(nop) COPY file:9
<missing>	4 days ago	/bin/sh -c #(nop) COPY file:5
<missing>	4 days ago	/bin/sh -c #(nop) COPY file:3
<missing>	4 days ago	/bin/sh -c #(nop) COPY file:c
<missing>	4 days ago	/bin/sh -c #(nop) COPY file:0
<missing>	4 days ago	/bin/sh -c set -x && grou
<missing>	4 days ago	/bin/sh -c #(nop) ENV PKG_RE
<missing>	4 days ago	/bin/sh -c #(nop) ENV NJS_VE
<missing>	4 days ago	/bin/sh -c #(nop) ENV NGINX_
<missing>	2 weeks ago	/bin/sh -c #(nop) LABEL main
<missing>	2 weeks ago	/bin/sh -c #(nop) CMD ["bash
<missing>	2 weeks ago	/bin/sh -c #(nop) ADD file:55

▼ `docker pull` : 이미지가 계층 구조인 것을 확인

- 다운로드된 계층들 정보는 전용 경로에 저장

```
docker pull nginx:latest
latest: Pulling from library/nginx
a378f10b3218: Pull complete
5b5e4b85559a: Pull complete
508092f60780: Pull complete
59c24706ed13: Pull complete
1a8747e4a8f8: Pull complete
```



```
ad85f053b4ed: Pull complete
3000e3c97745: Pull complete
Digest: sha256:add4792d930c25dd2abf2ef9ea79de578097a1c175a
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

▼ Docker login/logout : hub.docker.com에서 회원가입 후 실행

- 예제

```
docker login
Log in with your Docker ID or email address to push and pull images.
You can log in with your password or a Personal Access Token.

Username: {자신의 계정}
Password: {자신의 암호}
Login Succeeded

$ docker logout
Removing login credentials for https://index.docker.io/v1/
```

- 명령을 통해 등록 후 관리되는 Access Token 리스트
 - • <https://hub.docker.com/settings/security>



User Joined November 25, 2014

General

Security

Default Privacy

Notifications

Convert Account

Deactivate Account

Access Tokens

New Access Token

Tokens marked **AUTO-GENERATED** are created on your behalf by Docker Desktop for the CLI to use for authentication. You can have a maximum of 5 auto-generated tokens associated with your account. [Learn more](#)

	Description	Source	Scope	Last Used	Created	
<input type="checkbox"/>	Generated by Docker De...	AUTO-GENERATED	Read, Write, Delete	Oct 29, 2023 16:22:18	Oct 28, 2023 15:23:31	⋮
<input type="checkbox"/>	Generated by Docker De...	AUTO-GENERATED	Read, Write, Delete	Oct 18, 2023 17:37:50	Oct 16, 2023 09:19:21	⋮
<input type="checkbox"/>	Generated by Docker De...	AUTO-GENERATED	Read, Write, Delete	Oct 08, 2023 10:03:29	Oct 04, 2023 09:43:14	⋮
<input type="checkbox"/>	Generated by Docker De...	AUTO-GENERATED	Read, Write, Delete	Never	Oct 15, 2023 22:29:13	⋮
<input type="checkbox"/>	Generated by Docker De...	AUTO-GENERATED	Read, Write, Delete	Never	Oct 11, 2023 15:14:36	⋮

Two-Factor Authentication

Two-factor authentication is not enabled yet.
Two-factor authentication adds an extra layer of security to your account by requiring more than just a password to sign in. [Learn more](#)

Enable Two-Factor Authentication

▼ Docker Desktop 확인

Containers

Images

Volumes

Dev Environments BETA

Docker Scout

Learning center

Extensions

Add Extensions

Images

Local Hub Artifactory EARLY ACCESS

Try the NGINX extension to edit a running NGINX configuration. [View details](#)

186.35 MB / 450.96 MB in use 3 images Last refresh: 9 minutes ago

Search

	Name	Tag	Status	Created	Size	Actions
<input type="checkbox"/>	node-test 5c98ff41fa7	1.0	In use	6 days ago	186.35 MB	▶ ⋮ 🗑
<input type="checkbox"/>	nginx 593aee2afb64	latest	Unused	10 days ago	186.77 MB	▶ ⋮ 🗑
<input type="checkbox"/>	ubuntu e4c58958181a	22.04	Unused	1 month ago	77.82 MB	▶ ⋮ 🗑

04. Docker Container와 Container 를 다루는 CLI



Docker Image와 Docker Container 사이의 관계에 대해 알아봅니다.

▼ Docker Image와 Docker Container의 관계

- Image: 컨테이너에 대한 OS, Application, Library 등등의 정보를 담고 있음
- Container: Image를 실행한 상태. 1개의 Image로 부터 N개의 Container를 생성할 수 있는 1:N의 관계.
 - Image는 내가 만들고 싶은 붕어빵의 정보를 갖고 있는 틀이라면, Container는 구워낸 붕어빵



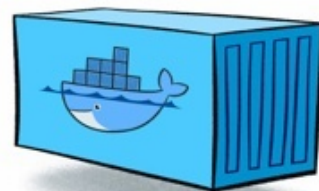
Java Class



Java Instance



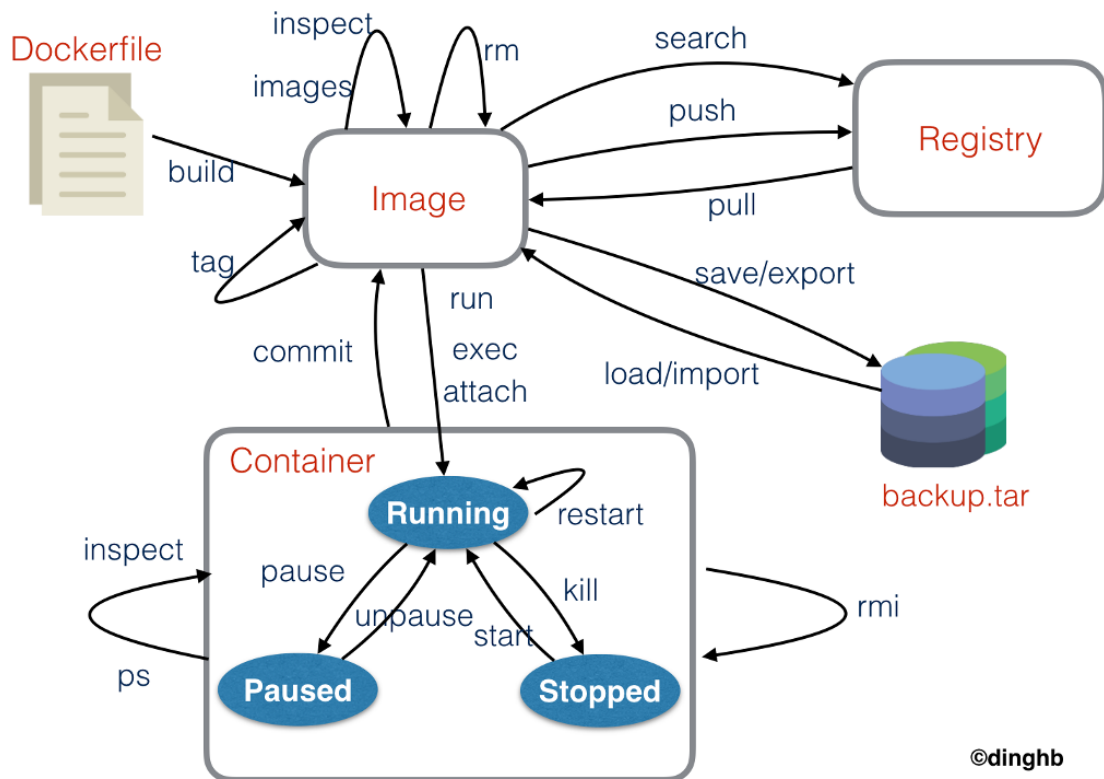
Docker Image



Docker container

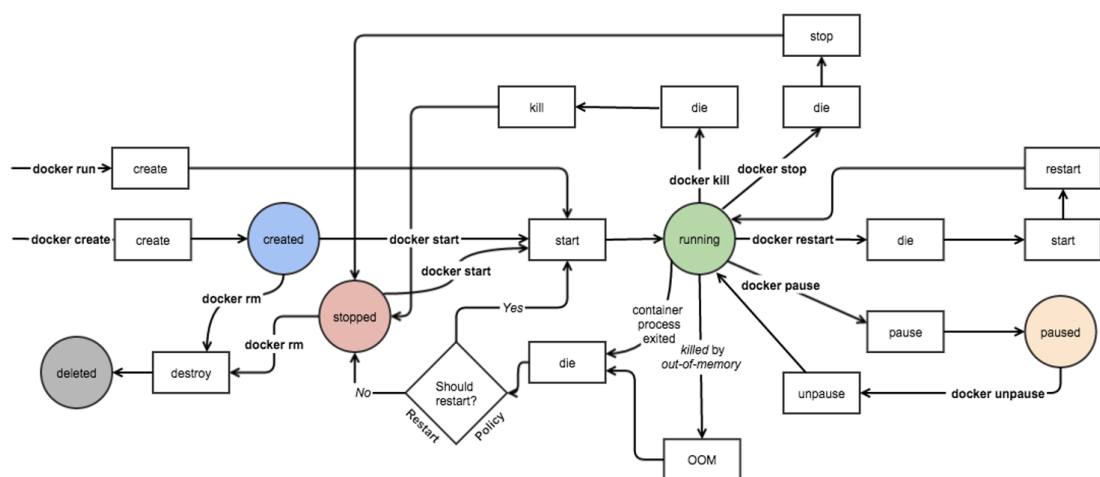
✓ **Docker Container에 대해서 알아보고 Container를 다루는 명령어들을 알아봅시다.**

▼ Docker 이미지/Container 관련 명령어



<https://segmentfault.com/a/1190000005802339>

▼ Docker 생애 주기(Lifecycle)



<https://docker-saigon.github.io/post/Docker-Internals/>

▼ Docker Container 수동 생성

```

docker pull ubuntu:22.04
docker images

# docker create 은 실제 실행하지 않고 컨테이너 생성만
docker create -ti --name ubuntu2204test ubuntu:22.04
docker ps -a
CONTAINER ID   IMAGE          COMMAND                  CREATED
2ccc1b2a1144   ubuntu:22.04   "/bin/bash"             4 seconds ago

docker start ubuntu2204test
Ubuntu2204test
docker attach ubuntu2204test

# docker run 은 create/start/attach 를 순차적으로 한 번에 실행하
docker run -ti --name=ubuntu2204test2 ubuntu:22.04 /bin/ba
root@57a1a1c759b6:/#

```

▼ Docker Container 는 프로세스

```

docker run -ti --name=ubuntu2204test3 ubuntu:22.04 /bin/ba
root@1cd125b32870:/#

# 터미널을 한 개 더 열고
ps -ef | grep ubuntu2204test3
user    9710   7637   0 17:17 pts/4        00:00:00 docker run -t
user    9921   9377   0 17:17 pts/5        00:00:00 grep --color=

```

▼ Container 명령 테스트

- 샘플 파일

[container-sample.zip](#)

- 컨테이너 명령 테스트

```

cd ~
mkdir nodejsapp
cd nodejsapp
vi app.js # 테스트용 nodejs 앱
vi Dockerfile # 새로운 도커 이미지를 위한 Dockerfile
docker buildx build -t node-test:1.0 . # 1.0 태그를 추가하여
docker images | grep node-test # 빌드 완료한 이미지 보기
docker image history node-test:1.0 # 1.0으로 태그 추가한 이미지
docker run -itd -p 6060:6060 --name=node-test -h node-test
docker ps | grep node-test
curl http://localhost:6060

```

▼ docker run 자주 사용하는 옵션

- `-d` : detached mode; 백그라운드 모드
- `-p` : 호스트와 컨테이너의 포트를 연결(포워딩)
- `-v` : 호스트와 컨테이너의 디렉토리를 연결(마운트)
- `-e` : 컨테이너 내에서 사용할 환경변수 설정
- `-name` : 컨테이너 이름 설정
- `-rm` : 프로세스 종료 시 컨테이너 자동 삭제
- `-ti` : `-i` 와 `-t` 를 동시에 사용한 것으로 터미널 입력을 위한 옵션

▼ 실행 중인 Container에 대한 정보

```

# 컨테이너에서 실행 중인 프로세스 조회
docker top node-test

```

UID	PID	PPID
root	2398	2378
root	2421	2398

```

# 컨테이너에 매핑된 포트 조회
docker port node-test
8080/tcp -> 0.0.0.0:8080

# 컨테이너 리소스 통계 출력 (1회)
docker stats node-test --no-stream

```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT
14c475f7ac09	node-test	0.01%	9.035MiB / 15.45GiB

```
# 컨테이너 리소스 통계 출력 (스트림)
docker stats node-test
```

▼ docker logs

```
# 표준 출력(stdout), 표준에러(stderr) 출력
docker logs node-test
```

```
...
```

```
# 로그를 계속 출력
docker logs -f node-test
```

```
...
```

```
...
```

```
# 출력된 로그는 파일로 관리되기 때문에 HostOS 의 disk 를 사용
docker info | grep -i log
```

▼ docker [container] inspect

```
# 컨테이너 내부 확인
docker inspect node-test
[
  {
    "Id": "2ccc1b2a114495e2d0b67f84e55c9c21e79c6bffff63",
    "Created": "2023-10-29T08:04:36.295616146Z",
    "Path": "/bin/bash",
    "Args": [],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
```

```

        "Pid": 1814,
        "ExitCode": 0,
        "Error": "",
        "StartedAt": "2023-10-29T08:05:15.974255879Z",
        "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image": "sha256:e4c58958181a5925816faa528ce959e48",
    "ResolvConfPath": "/var/lib/docker/containers/2ccc1b",
    "HostnamePath": "/var/lib/docker/containers/2ccc1b"
}

```

▼ `docker stop` | `start` | `pause` | `unpause`

```

# 터미널1, 도커 상태 확인
docker stats

# 터미널2, 도커 프로세스 이벤트 확인
docker events

# 터미널3, docker start
docker stop node-test
docker ps -a
docker start node-test

#
docker pause node-test
docker unpause node-test
docker ps -a

```

▼ `docker exit code`

- 0
 - Docker Process가 수행해야 할 모든 Command 또는 Shell을 실행하고 정상 종료
- 255
 - Docker Image에 정의된 EntryPoint 또는 CMD가 수행이 완료되었을 경우 발생

- 125
 - Docker run 명령어의 실패로 실제 docker process가 기동되지 않음
- 126
 - Docker Container 내부에서 Command를 실행하지 못할 경우 발생
- 127
 - Docker Container 내부에서 Command를 발견하지 못하였을 경우 발생
- 137
 - kill -9로 인해 종료 됨
- 141
 - 잘못된 메모리 참조하여 종료 됨
- 143
 - Linux Signal로 정상 종료 됨
- 147
 - 터미널에서 입력된 정지 시그널로 종료 됨
- 149
 - 자식 프로세스가 종료 되어 종료 됨



Docker Container를 정리하는 방법을 알아봅니다.



`docker container prune`

- 실행 중이 아닌 모든 컨테이너를 삭제

```
# 중지된 컨테이너를 포함하여 모든 컨테이너 리스트
docker container ls -a

#
docker container prune
```

▼ `docker image prune`

- 태그가 붙지 않은(dangling) 모든 이미지 삭제

```
#
docker image prune

# 남아 있는 이미지 리스트 확인 - 실행 중인 컨테이너의 이미지 등
docker image ls
```

▼ `docker system prune`

- 사용하지 않는 도커 이미지, 컨테이너, 볼륨, 네트워크 등 모든 도커 리소스를 일괄적으로 삭제

```
docker system prune
```

1주차 끝

👉 전체 강의자료 보기

📖 Docker+CI/CD

다음 주차 👉

📖 Docker+CI/CD - 2주차

Copyright © TeamSparta All rights reserved.