

Docker

Docker

❖ Docker

- ✓ 도커(Docker)는 Container형 가상화 기술을 구현하기 위한 상주 애플리케이션과 이 애플리케이션을 조작하기 위한 명령행 도구로 구성되는 프로젝트

❖ Docker의 장점

- ✓ 애플리케이션 배포에 특화돼 있기 때문에 애플리케이션 개발 및 운영을 Container 중심으로 할 수 있음
- ✓ 가상화 소프트웨어보다 더 가볍게 동작
- ✓ 도커는 이식성이 뛰어나서 로컬 머신의 도커 환경에서 실행하던 Container를 다른 서버에 있는 도커 환경에 배포하거나 반대로 다른 서버의 도커 환경에서 동작하던 Container를 로컬로 가져올 수 있는데 개발 환경 과 운영 환경을 거의 동등하게 재현할 수 있음
- ✓ Container 간의 연동이나 클라우드 플랫폼 지원 등 여러 면에서 장점이 있음

❖ Docker를 추천하지 않는 경우

- ✓ 도커 Container의 내부는 리눅스 계열 운영 체제와 같은 구성을 취하는 것이 많은데 Container는 운영체제의 동작을 완전히 재현하지는 못하기 때문에 좀 더 엄밀한 리눅스 계열 운영 체제의 동작이 요구되는 가상 환경을 구축해야 한다면 기존 방법대로 VMWare나 VirtualBox 같은 가상화 소프트웨어를 사용하는 것이 나음
- ✓ FreeBSD 같은 비 리눅스 환경이 필요한 경우에도 도커가 적합하지 않음

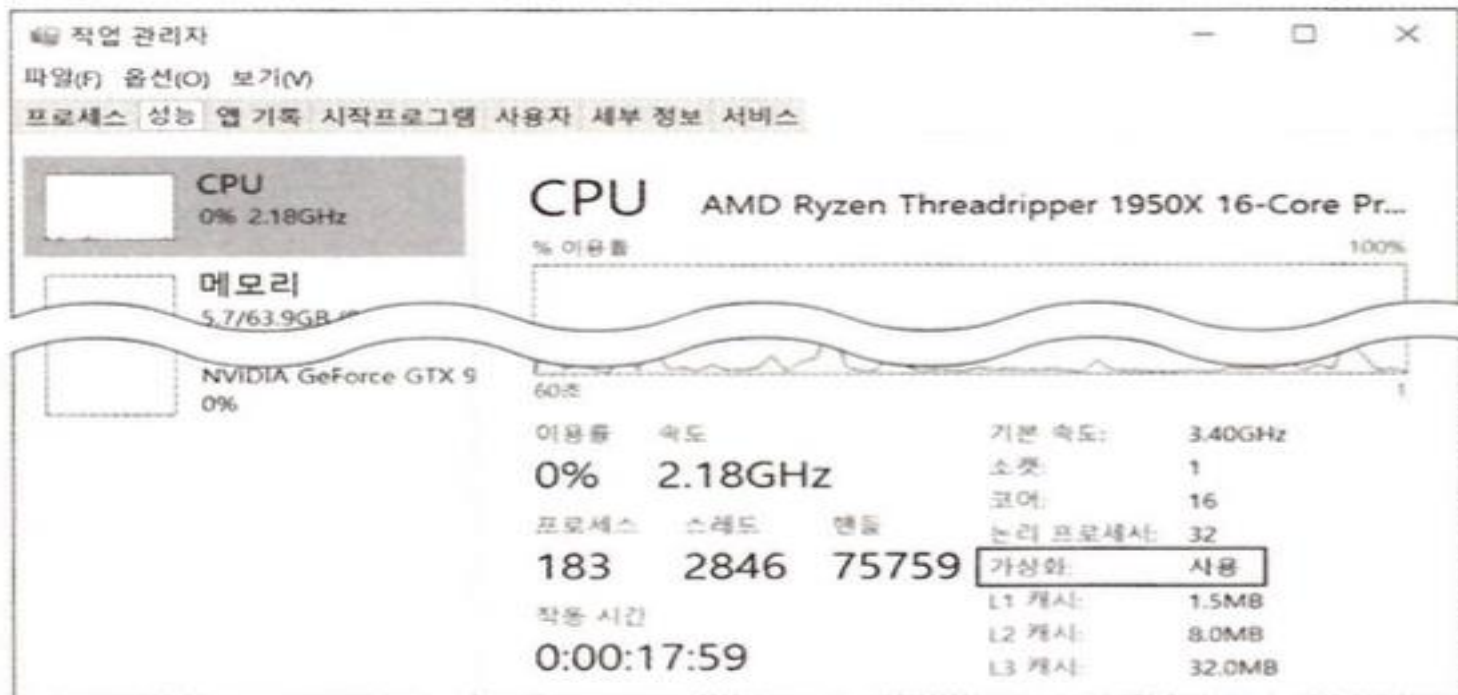
Docker

❖ Docker 설치

✓ 윈도우 용 도커를 사용하려면 Hyper-V 기능이 필요

□ 윈도우의 작업 관리자 창에서 성능 탭을 선택해서 가상화 항목이 사용 으로 돼 있는지 확인

❑ 사용하지 않음으로 돼 있다면 컴퓨터의 바이오스(UEFI) 설정에서 가상화 기능을 활성화해야 하는데 인텔 CPU를 사용하는 컴퓨터라면 Intel(R)Virtualization Technology, AMD CPU를 사용하는 컴퓨터라면 SVM 항목을 활성화하면 됨



Docker

❖ Docker 설치

- ✓ Docker Hub 사이트에 접속해서 회원 가입:

<https://hub.docker.com/signup?next=%2Feditions%2Fcommunity%2Fdocker-ce-desktop-mac%3Fref%3Dlogin>

- ✓ 다운로드

- ☐ Windows: <https://hub.docker.com/editions/community/docker-ce-desktop-windows>

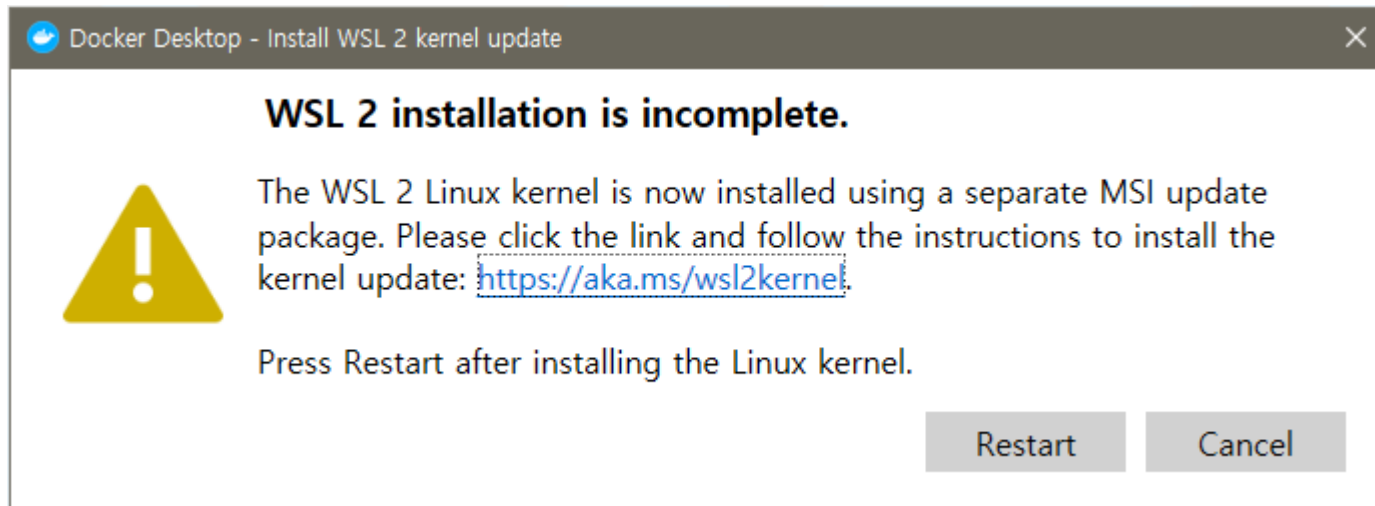
- ☐ Mac: <https://hub.docker.com/editions/community/docker-ce-desktop-mac>

- ☐ Linux에서 설치

- `sudo apt update -y`
- `sudo apt install -y apt-transport-https ca-certificates curl software-properties-common`
- `curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`
- `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"`
- `sudo apt update -y`
- `sudo apt install -y docker-ce`
- `docker version`

Docker

- ❖ Docker 설치
 - ✓ Windows 에서 실행이 되지 않는 경우



Docker

❖ Docker 설치

✓ Windows 에서 실행이 되지 않는 경우

- 파워셸을 관리자 권한으로 실행

- 리눅스 서브시스템 활성화 명령어 입력

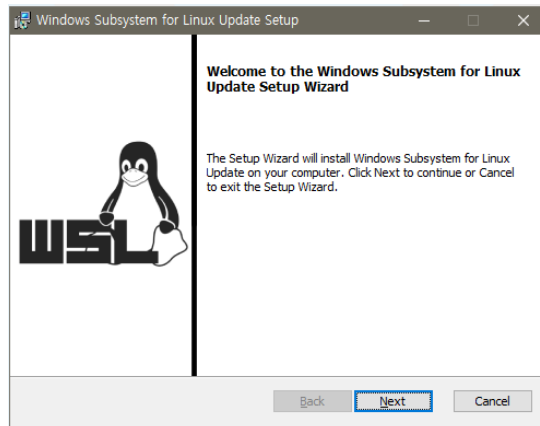
```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

- 가상 머신 플랫폼 기능 활성화 명령어 입력

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

- x64 머신용 최신 WSL2 Linux 커널 업데이트 패키지 다운로드, 설치

wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

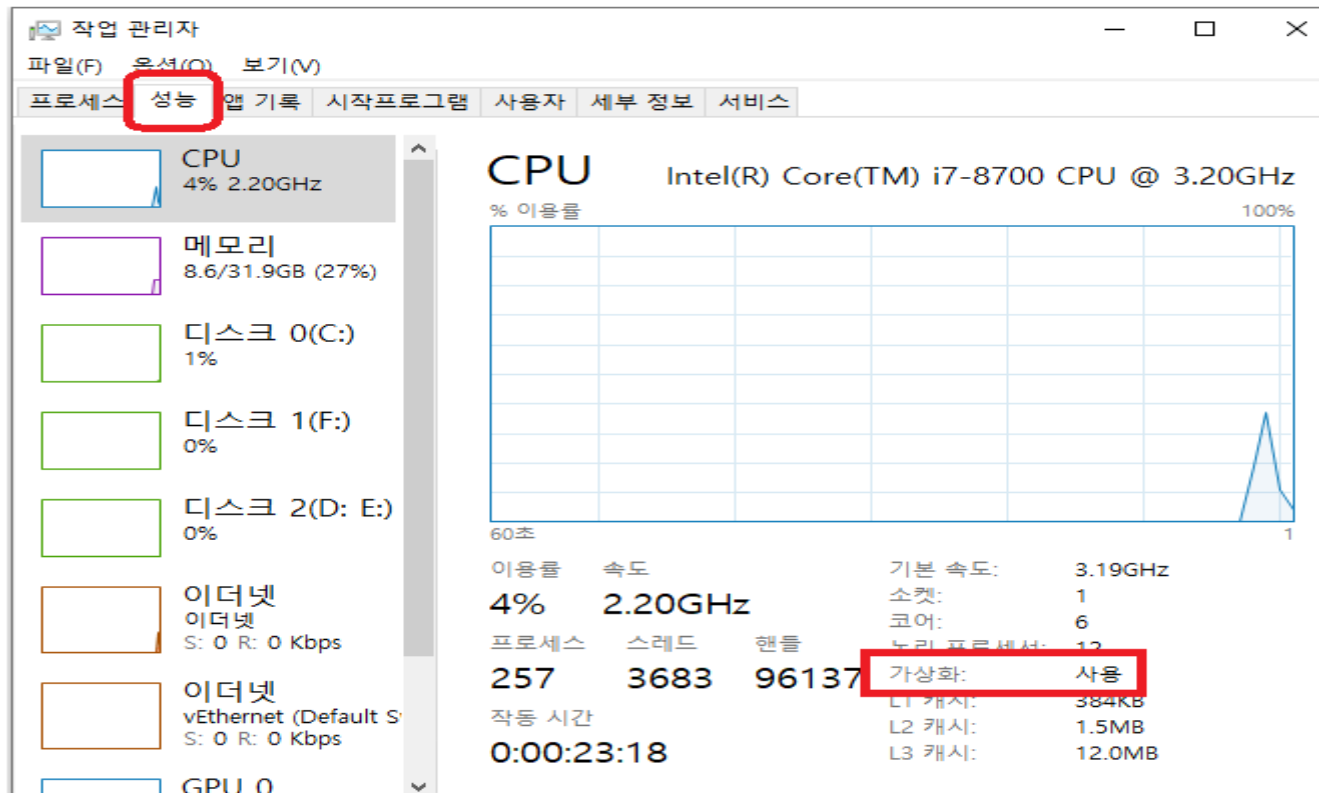


Docker

❖ Docker 설치

✓ Windows 에서 실행이 되지 않는 경우

- 작업 관리자 - CPU 가상화 기능 활성화

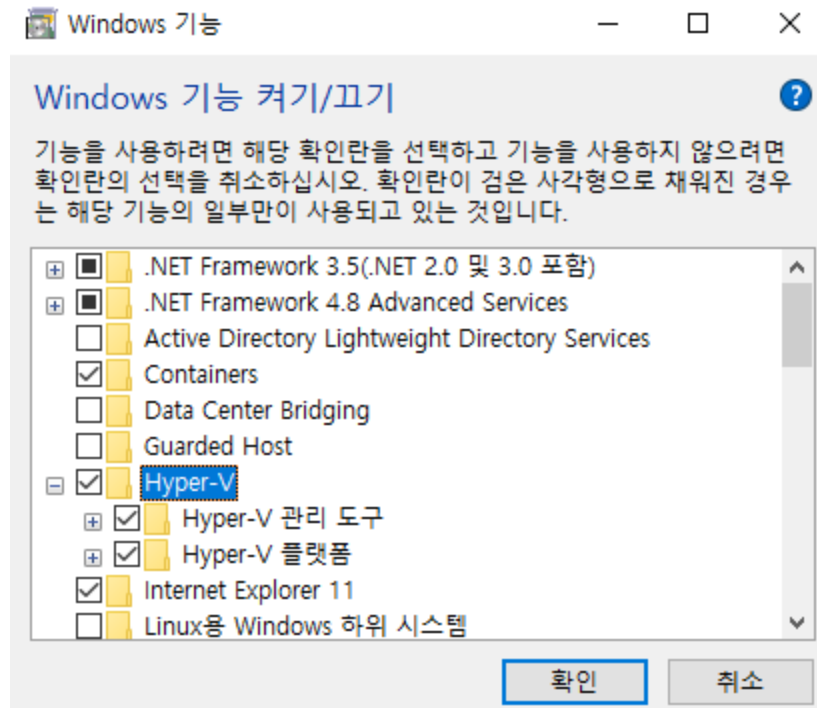


Docker

❖ Docker 설치

✓ Windows 에서 실행이 되지 않는 경우

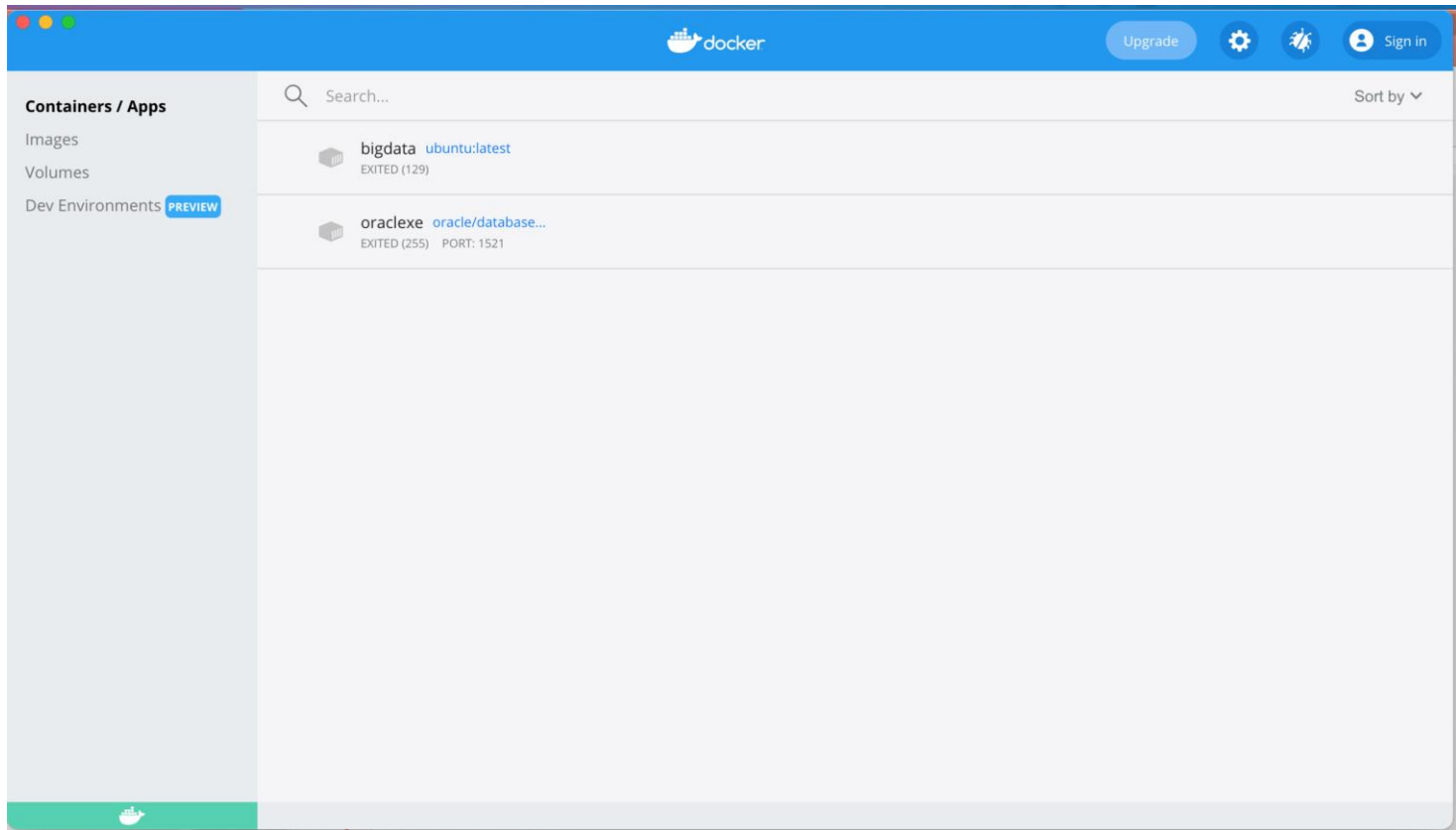
- Hyper-V 옵션 켜기



Docker

❖ Docker 기본 설정

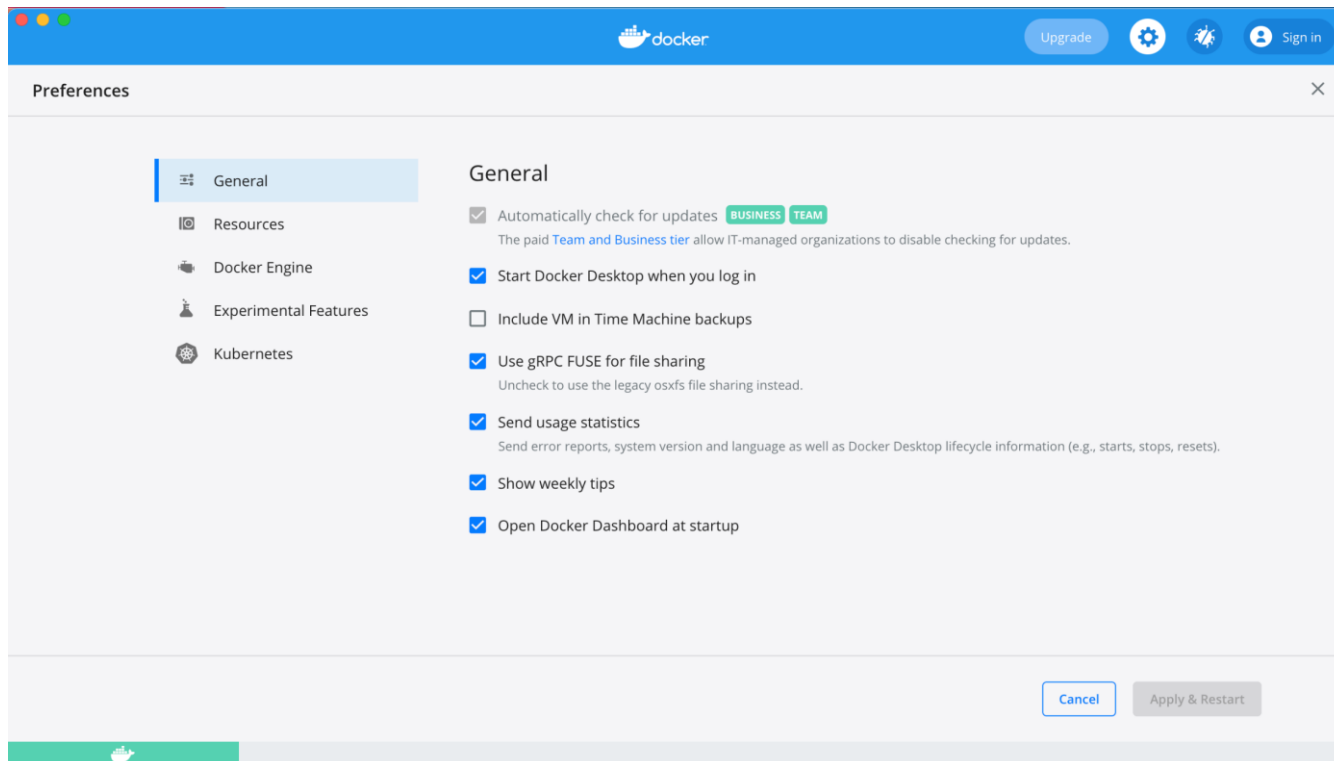
✓ Docker Dashboard – 설치된 image 와 Container 확인 가능



Docker

❖ Docker 기본 설정

✓ Docker Preference – 자동 실행 및 자동 업데이트 및 각종 환경 설정 가능



Docker

❖ Docker 기본 설정

✓ Docker Preference

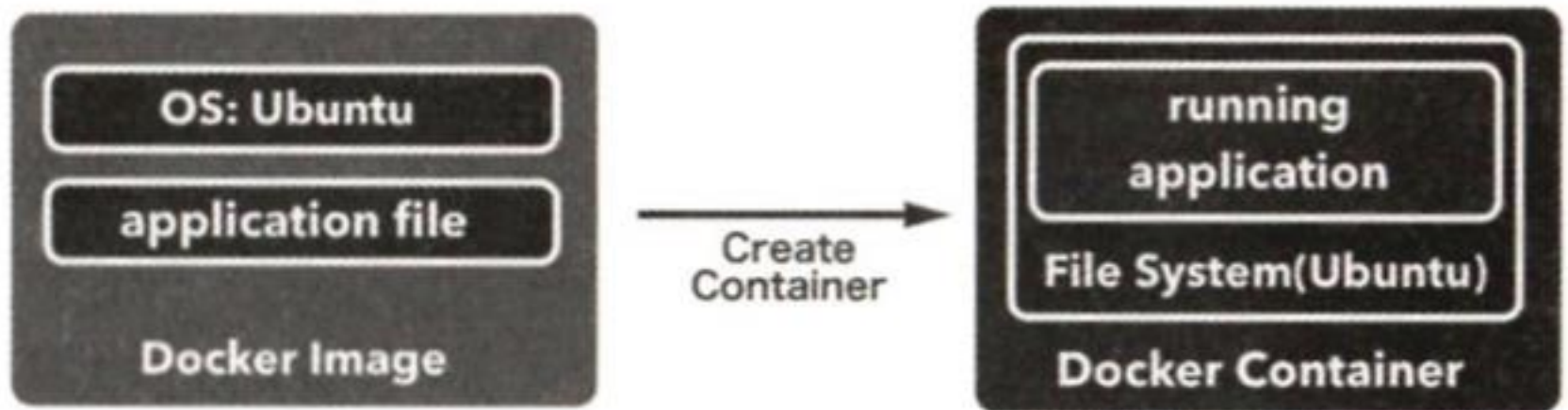
- ☐ General: 자동 로그인 및 자동 업데이트 설정
- ☐ Resources: CPU, RAM, Disk Size 설정
- ☐ Docker Engine: JSON 포맷으로 도커를 설정할 수 있는데 윈도우용/macOS용 도커의 설정 화면에 나오지 않는 사항을 변경하려면 이곳에서 JSON 문자열을 수정
- ☐ Experimental Features: 현재 실험하고 있는 기능
- ☐ Kubernetes: 쿠버네티스 사용 설정



Docker Container Deployment

- ❖ Container로 애플리케이션 실행
 - ✓ Image 와 Container

개념	역할
Image	도커 Container를 구성하는 파일 시스템과 실행할 애플리케이션 설정을 하나로 합친 것으로. Container를 생성하는 템플릿 역할을 한다. 도커 image
Container	도커 image를 기반으로 생성되며. 파일 시스템과 애플리케이션이 구체화돼 실행되는 상태



Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 와 Container

- ❑ 도커 image 하나로 여러 개의 Container를 생성할 수 있음
- ❑ 도커 image는 우분투 파일 시스템을 실행하는 애플리케이션 파일을 담고 있음
- ❑ Container가 생성될 때 image로부터 이를 구체화하고 Container 안의 우분투 파일 시스템 상에서 애플리케이션이 실행됨
- ❑ Container로 애플리케이션을 실행하려면 Container 형태로 구체화될 템플릿 역할을 하는 image를 먼저 만들어야 함



Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 다운로드 와 Container 생성 및 실행 과정

❑ giyodocker/echo:latest image 다운로드

```
docker image pull giyodocker/echo:latest
```

❑ 다운로드 받은 image 실행

```
docker container run -t -p 9000:8080 giyodocker/echo:latest
```

❑ 다른 터미널을 실행시켜 확인

```
curl http://localhost:9000/
```

A screenshot of a terminal window titled 'adam - zsh - 80x24'. The window shows the output of a 'curl' command. The text in the terminal is: 'Last login: Tue Sep 28 07:23:26 on ttys000', '[(base) adam@Adams-MacBook-Pro ~ % curl http://localhost:9000/]', 'Hello Docker!!', and '[(base) adam@Adams-MacBook-Pro ~ %]'. The terminal has a light pink title bar and standard macOS window controls (red, yellow, green buttons) on the left.

```
adam - zsh - 80x24
Last login: Tue Sep 28 07:23:26 on ttys000
[(base) adam@Adams-MacBook-Pro ~ % curl http://localhost:9000/
Hello Docker!!
[(base) adam@Adams-MacBook-Pro ~ % ]
```

❑ Container 정지

Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 다운로드 와 Container 생성 및 실행 과정

❑ 실행 중인 Container 확인

```
docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	NAMES	CREATED	STATUS	PORTS
76c4c6fa2498	gihyodocker/echo:latest	"go run /echo/main.go"	bold_dijkstra	About a minute ago	Up About a minute	0.0.0.0:9000->8080/tcp, :::9000->8080/tcp
5de6a649c945	gihyodocker/echo:latest	"go run /echo/main.go"	objective_shamir	7 minutes ago	Exited (2) 3 minutes ago	
6fca9c58bc30	ubuntu:latest	"bash"	bigdata	2 weeks ago	Exited (129) 2 weeks ago	
927f5d6b2294	oracle/database:18.4.0-xe	"/bin/sh -c 'exec \$0...'"	oraclexe	6 months ago	Exited (255) 3 weeks ago	0.0.0.0:1521->1521/tcp, :::1521->1521/tcp, 0.0.0.0:5500->5500/tcp, :::5500->5500/tcp

Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 다운로드 와 Container 생성 및 실행 과정

□ 실행 중인 Container 중지

- 특정 Container 중지: `docker stop ContainerID 또는 Name`
- 모든 Container 중지: `docker stop $(docker ps -a -q)`

□ Container 삭제

- 특정 Container 삭제: `docker rm ContainerID 또는 Name`
- 모든 Container 삭제: `docker rm $(docker ps -a -q)`



Docker Container Deployment

- ❖ Container로 애플리케이션 실행
 - ✓ Image 생성 과 Container 생성 및 실행

❑ 애플리케이션 생성 – main.go

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

func main() {
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        log.Println("received request")
        fmt.Fprintf(w, "Hello Docker!!")
    })

    log.Println("start server")
    server := &http.Server{Addr: ":8080"}
    if err := server.ListenAndServe(); err != nil {
        log.Println(err)
    }
}
```

Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 생성 과 Container 생성 및 실행

- ❑ Docker image를 생성하기 위한 설정 파일 생성 – Dockerfile을 main.go 파일과 동일한 디렉토리에 작성

```
FROM golang:1.9
```

```
RUN mkdir /echo
```

```
COPY main.go /echo
```

```
CMD ["go", "run", "/echo/main.go"]
```

- Dockerfile에 전용 도메인 언어로 image의 구성을 정의하는데 여기 사용된 FROM이나 RUN 같은 키워드를 인스트럭션(명령)이라고 함



Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 생성 과 Container 생성 및 실행

□ 인스트럭션

○ FROM 인스트럭션

- FROM 인스트럭션은 도커 image의 바탕이 될 베이스 image를 지정
- Dockerfile로 image를 빌드할 때 먼저 FROM 인스트럭션에 지정된 image를 내려받음
- FROM에서 받아오는 도커 image는 도커 허브(Docker Hub)라는 레지스트리에 공개된 것으로 도커는 FROM에서 지정한 image를 기본적으로 도커 허브 레지스트리에서 참조
- main.go를 실행하려면 Go 언어의 런타임이 설치된 image가 있어야 하는데 이 런타임이 설치된 golang image를 사용하는데 1.9 라고 된 부분은 태그라고 하는데 각 image의 버전 등을 구별하는 식별자
- 각 도커 image는 고유의 해시값을 갖는데 이 해시만으로는 필요한 image가 무엇인지 특정하기가 어렵기 때문에 특정 버전에 태그를 붙여두면 사람이 그 내용을 쉽게 파악할 수 있어서 대부분 golang 언어 image처럼 언어 버전 등을 따서 태그를 붙이는 경우가 일반적

Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 생성 과 Container 생성 및 실행

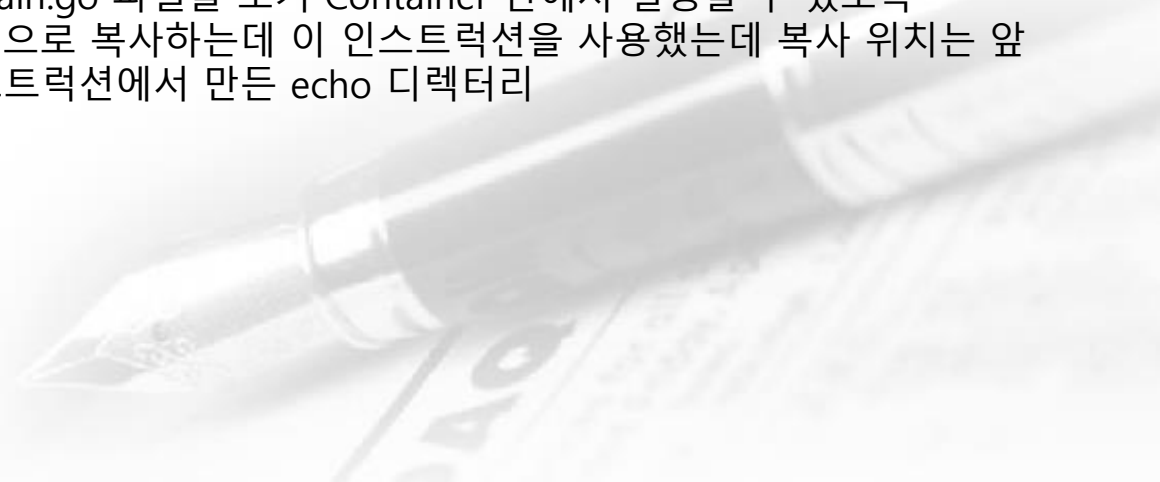
□ 인스트럭션

○ RUN 인스트럭션

- RUN 인스트럭션은 도커 image를 실행할 때 Container 안에서 실행할 명령을 정의하는 인스트럭션
- 인자로 도커 Container 안에서 실행할 명령을 그대로 기술하는데 여기서는 main.go 애플리케이션을 배치하기 위한 /echo 디렉터리를 mkdir 명령으로 생성

○ COPY 인스트럭션

- COPY 인스트럭션은 도커가 동작 중인 호스트 머신의 파일이나 디렉터리를 도커 Container 안으로 복사하는 인스트럭션으로 이 예제에서는 호스트에서 작성한 main.go 파일을 도커 Container 안에서 실행할 수 있도록 Container 안으로 복사하는데 이 인스트럭션을 사용했는데 복사 위치는 앞의 RUN 인스트럭션에서 만든 echo 디렉터리



Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 생성 과 Container 생성 및 실행

□ 인스트럭션

○ CMD 인스트럭션

- CMD 인스트럭션은 도커 Container를 실행할 때 Container 안에서 실행할 프로세스를 지정하는 것으로 RUN 인스트럭션은 image를 빌드할 때 실행되고 CMD 인스트럭션은 Container를 시작할 때 한 번 실행되며 RUN은 애플리케이션 업데이트 및 배치에 CMD는 애플리케이션 자체를 실행하는 명령으로 셸 스크립트로 치면 다음과 같은 실행 명령 역할

```
$ go run /echo/main.go
```

- 이 명령을 CMD 인스트럭션에 기술하면 다음과 같이 명령을 공백으로 나눈 배열로 나타냄

```
CMD ["go", "run", "/echo/main.go"]
```



Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 생성 과 Container 생성 및 실행

□ 도커 image 빌드

- main.go 파일과 Dockerfile 작성이 끝났으면 docker image build 명령으로 도커 image를 빌드
- docker image build 명령은 도커 image를 빌드하기 위한 명령
- docker image build 명령의 기본 문법
docker image build -t image명[:태그명] Dockerfile의_경로
 - -t 옵션으로 image명을 지정하는데 태그명도 지정할 수 있으며 생략 시에는 latest 태그가 붙음
 - -t 옵션과 image명은 반드시 지정해야 한다고 생각하는 편이 좋은데 -t 옵션 없이도 빌드 자체는 가능하지만 image명 없이는 해시값만으로 image를 구별해야 하므로 사용하기가 상당히 번거로움
- 이 예제에서는 example/echo라는 image명을 사용했는데 이때 앞에서 참조했던 golang image에는 없었던 /가 추가되었는데 / 앞에 오는 example은 네임스페이스로 image명에 이렇게 사용자 네임스페이스를 추가할 수 있음
- image명의 충돌을 피하기 위해 되도록 네임스페이스를 붙일 것을 추천
- 현재 작업 디렉터리에 Dockerfile이 있다면 마지막 인자를 .(현재 작업 디렉터리)로 설정
\$ docker image build -t example/echo:latest .

Docker Container Deployment

- ❖ Container로 애플리케이션 실행
 - ✓ Image 생성 과 Container 생성 및 실행
 - 도커 image 확인
 - docker image ls

```
[ (base) adam@Adams-MacBook-Pro ~ % docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
example/echo	latest	9497612b263e	36 minutes ago	750MB
ubuntu	latest	fb52e22af1b0	3 weeks ago	72.8MB
oracle/database	18.4.0-xe	95026988ae53	7 months ago	5.89GB
golang	1.10	6fd1f7edb6ab	2 years ago	760MB
gihyodocker/echo	latest	3dbbae6eb30d	3 years ago	733MB



Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 생성 과 Container 생성 및 실행

□ 도커 Container 실행

`docker container run example/echo:latest`

- `docker container run` 명령으로 echo Container를 실행
- 이 Container는 계속 포어그라운드에서 동작
- Container를 종료하고 싶다면 터미널에 `Ctrl + C`(SIGINT 전송)를 입력

`docker container run -d example/echo:latest`

- 백그라운드 Container로 실행



Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 생성 과 Container 생성 및 실행

□ 포트 포워딩

- 도커 Container는 가상 환경이지만 외부에서 봤을 때 독립된 하나의 머신처럼 다룰 수 있다는 특징이 있음
- echo 애플리케이션은 8080 포트를 리스닝하고 있지만 이 포트는 Container 포트라고 해서 Container 안에 한정된 포트
- curl을 Container 안에서 실행하면 올바른 응답을 받을 수 있겠지만 Container 밖에서는 Container 포트를 바로 사용할 수 없기 때문에 Connection refused라는 메시지가 출력됨
- HTTP 요청을 받는 애플리케이션을 사용하려면 Container 밖에서 온 요청을 Container 안에 있는 애플리케이션에 전달해줘야 하는데 그 역할을 담당하는 것이 바로 도커의 포트 포워딩
- 포트 포워딩이란 호스트 머신의 포트를 Container 포트와 연결해 Container 밖에서 온 통신을 Container 포트에 전달
- 이 기능 덕분에 Container 포트를 Container 외부에서도 이용할 수 있음
- docker container run 명령에서 -p 옵션을 붙이면 포트 포워딩을 지정할 수 있는데 -p 옵션값은 호스트_포트:Container_포트 형식으로 기술하면 됨

Docker Container Deployment

❖ Container로 애플리케이션 실행

✓ Image 생성 과 Container 생성 및 실행

□ 포트 포워딩

- 포트 포워딩을 이용한 main.go 실행

```
docker container run -d -p 9000:8080 example/echo:latest
```

- 외부에서 확인

```
curl http://localhost:9000
```

- 외부 포트를 생략한 경우에는 docker container ls 명령으로 외부 포트를 확인해서 사용



Docker Container Deployment

- ❖ 도커 image 관련 명령어
 - ✓ 도커 명령어 도움말
 - 기본 명령어
 - docker help
 - 상세 명령어
 - docker 명령어 -help



Docker Container Deployment

❖ 도커 image 관련 명령어

✓ docker image build - image 빌드

❑ docker image build는 Dockerfile에 기술된 구성을 따라 도커 image를 생성하는 명령

`docker image build -t image명[:태그명] Dockerfile의_경로`

○ docker image build 명령에는 옵션이 몇 가지 있음

○ -t 옵션은 image명과 태그명을 붙이는 것으로 실제 사용에서 거의 필수적으로 사용되는데 Dockerfile 경로는 말 그대로 Dockerfile이 위치한 디렉터리 경로를 기재하면 되고 docker image build 명령에는 반드시 Dockerfile이 필요하므로 그 경로에 Dockerfile이 없다면 명령을 실행할 수 없음

`docker image build -t example/echo:latest .`

○ docker image build 명령은 기본으로 Dockerfile이라는 이름으로 된 Dockerfile을 찾는데 그 외 파일명으로 된 Dockerfile을 사용하려면 -f 옵션을 사용해야 하는데 예를 들어 Dockerfile-test라는 이름으로 된 Dockerfile을 사용하고자 하는 경우

`docker image build -f Dockerfile-test -t example/echo:latest .`

○ --pull 옵션을 이용하면 image를 베이스 image를 기반으로 생성하는데 이 옵션이 없으면 이전에 빌드한 데이터가 있으면 그 image를 이용하고 변경된 부분만을 반영해 빌드를 시도

`docker image build --pull=true -t example/echo:latest .`

Docker Container Deployment

❖ 도커 image 관련 명령어

✓ docker search - image 검색

- ❑ 도커 허브는 도커 image 레지스트리로 마치 깃허브처럼 사용자나 조직 이름으로 리포지토리를 만들 수 있고 이 리포지토리를 사용해 도커 image를 관리
- ❑ 도커 허브에는 모든 image의 기반이 되는 운영체제(CentOS, 우분투 등) 리포지토리, 언어 런타임이나 유명 미들웨어 image 등이 관리되는 수많은 리포지토리가 있기 때문에 모든 도커 image를 직접 만드는 대신 다른 사람이나 조직에서 만들어 둔 image를 사용할 수 있음
- ❑ 도커 허브를 활용할 때 빼놓을 수 없는 것이 docker search 명령인데 docker search 명령을 사용하면 도커 허브에 등록된 리포지토리를 검색할 수 있음

docker search [options] 검색_키워드

❑ mysql image를 5개로 제한해서 검색

docker search --limit 5 mysql

- mysql과 관련된 리포지토리 목록을 볼 수 있는데 검색 결과 첫 번째에 나오는 mysql 리포지토리는 리포지토리 이름에 네임스페이스가 생략돼 있는데 이 리포지토리가 mysql 공식 리포지토리 이기 때문
- 공식 리포지토리의 네임스페이스는 일률적으로 library 이므로 이 리포지토리의 정확한 이름도 library/mysql이 되는데 공식 리포지토리의 네임스페이스는 생략할 수 있음
- 검색 결과는 STARS 순으로 출력되는데 도커 허브에 등록된 리포지토리에도 깃허브처럼 스타 수가 매겨지는데 스타 수는 도커 image를 평가하는 주요 지표 중 하나.

Docker Container Deployment

❖ 도커 image 관련 명령어

✓ docker image pull - image 내려받기

docker image pull [options] 리포지토리명[:태그명]

❑ jenkins image 다운로드

docker image pull jenkins/jenkins:its



Docker Container Deployment

- ❖ 도커 image 관련 명령어
 - ✓ docker image ls - image 목록 보기
 - docker image ls [options] [리포지토리[:태그]]



Docker Container Deployment

❖ 도커 image 관련 명령어

✓ docker image tag – image에 태그 붙이기

docker image tag 기반image명[:태그] 새image명[:태그]

- example/echo의 latest image에 0.1.0 태그를 부여

docker image tag example/echo:latest example/echo:0.1.0

docker image ls



Docker Container Deployment

❖ 도커 image 관련 명령어

✓ docker image push – image를 외부에 공개

docker image push [options] 리포지토리명[:태그]

❑ 리포지토리 이름은 자신의 docker ID

❑ image 이름 변경

docker image tag example/echo:latest ggangpae1/echo:latest

❑ image 확인

docker image ls

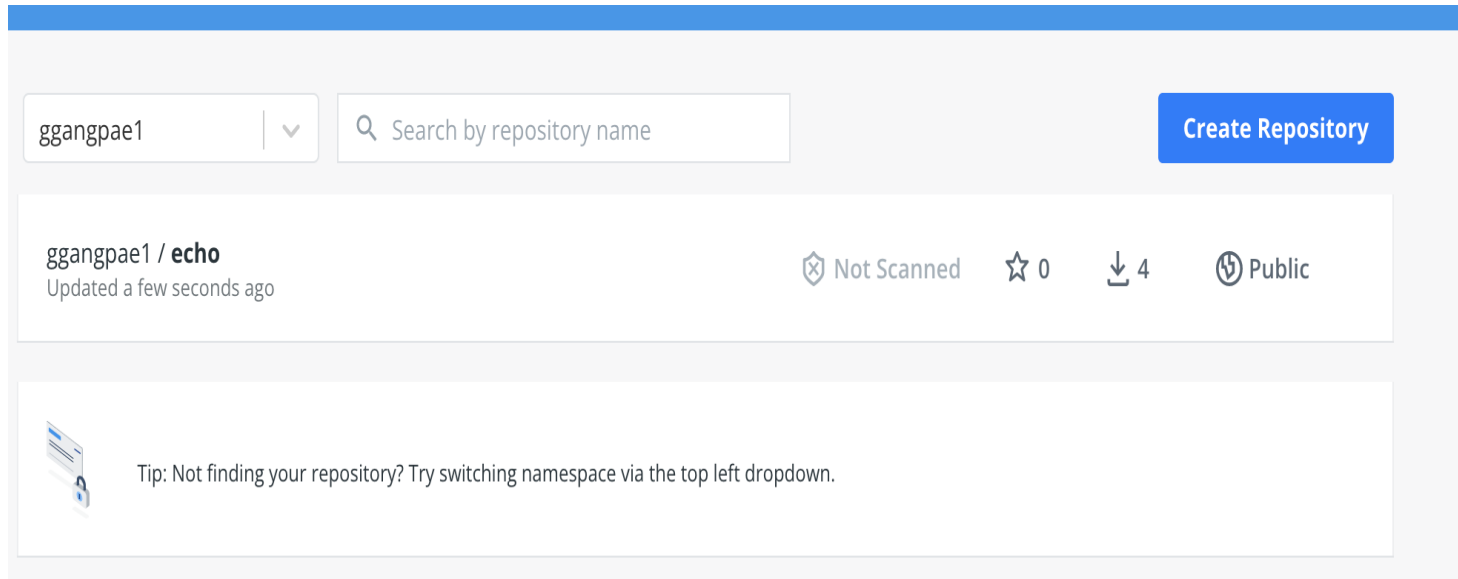
❑ 업로드

docker image push ggangpae1/echo:latest



Docker Container Deployment

- ❖ 도커 image 관련 명령어
 - ✓ docker image push – image를 외부에 공개
 - ❑ hub.docker.com 에서 확인



Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container는 가상 환경으로 파일 시스템과 애플리케이션이 함께 담겨 있는 박스

✓ 도커 Container의 생명 주기

□ 실행 중, 정지, 파기의 3가지 상태

□ 실행 중 상태

- docker container run 명령의 인자로 지정된 도커 image를 기반으로 Container가 생성되면 이 image를 생성했던 Dockerfile에 포함된 CMD 및 ENTRYPOINT 인스트럭션에 정의된 애플리케이션이 실행되며 이 애플리케이션이 실행 중인 상태가 Container의 실행 중 상태가 됨
- HTTP 요청을 받는 서버 애플리케이션이면 오류로 인해 종료되지 않는 한 실행 중 상태가 지속되므로 실행 기간이 길며 이에 비해 명령이 바로 실행되고 끝나는 명령행 도구 등의 Container는 실행 중 상태가 길게 유지되지 않음
- 실행이 끝나면 정지 상태가 됨

□ 정지 상태

- 실행 중 상태에 있는 Container를 사용자가 명시적으로 정지하거나 Container에서 실행된 애플리케이션이 정상/오류 여부를 막론하고 종료된 경우에는 Container가 자동으로 정지 상태가 됨
- Container를 정지시키면 가상 환경으로서는 더 이상 동작하지 않지만 디스크에 Container가 종료되던 시점의 상태가 저장돼 남기때문에 정지시킨 Container를 다시 실행할 수 있음

Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container의 생명 주기

□ 파기 상태

- 정지 상태의 Container는 명시적으로 파기하지 않는 이상 디스크에 그대로 남아 있음
- Container를 자주 생성하고 정지해야 하는 상황에서는 디스크를 차지하는 용량이 점점 늘어나므로 불필요한 Container를 완전히 삭제하는 것이 바람직
- 같은 image로 새로운 Container를 생성했다고 해도 각 Container가 실행된 시각 등이 서로 다르고 애플리케이션의 처리 결과도 이에 따라 달라질 수 있기 때문에 완전히 같은 Container를 새로 생성할 수는 없음



Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container의 생성 및 실행

`docker container run [options] image명[:태그] [명령] [명령인자...]`

`docker container run [options] imageID [명령] [명령인자...]`

❑ `example/echo:latest` image를 기반으로 Container를 백그라운드에서 실행

`docker container run -d -p 9000:8080 example/echo:latest`

`curl http://localhost:9000/`

❑ `docker container run` 명령에 명령 인자를 전달하면 Dockerfile에서 정의했던 CMD 인스트럭션을 수행 할 수 있음

`docker image pull alpine:3.7`

`docker container run -it alpine:3.7 uname -a`

❑ `docker create --name 컨테이너이름 이미지이름` 으로 컨테이너이름 생성 가능

❑ `docker run` 컨테이너이름으로 컨테이너 시작 가능

Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container의 생성 및 실행

□ 도커 Container에 이름 붙이기

- Container 정지 등 Container를 다루는 명령을 실행할 때는 Container ID 등으로 Container를 특정해줘야 하지만 Container ID와 자동 부여된 Container 이름 모두 Container가 실행되고 나야 알 수 있는데 개발 업무 중에는 같은 docker 명령을 시행착오로 반복 실행하게 되는 경우가 잦으므로 매번 docker container ls 명령으로 Container ID 및 Container 이름을 확인하기가 번거롭기 때문에 Container에 이름을 붙이는 기능을 제공
- docker container run 명령에 -name 옵션을 사용하면 Container에 원하는 이름을 붙일 수 있음
- 이름이 있으면 도커 명령을 사용할 때도 알기 쉬운 이름으로 Container를 특정할 수 있기 때문에 개발 업무에서도 매우 편리

```
docker container run --name [Container명] [image명]:[태그]
```

```
docker container run -t -d --name ggangpae1-echo example/echo:latest
```



Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container의 생성 및 실행

□ docker run 명령의 옵션

- -i 옵션은 Container를 실행할 때 Container 쪽 표준 입력과의 연결을 그대로 유지해주는데 Container 쪽 셸에 들어가서 명령을 실행할 수 있음
- -t 옵션은 유사 터미널 기능을 활성화 하는 옵션인데 -i 옵션을 사용하지 않으면 유사 터미널을 실행해도 여기에 입력할 수가 없으므로 -i 와 -t 옵션을 같이 사용하거나 이들 옵션을 합쳐 축약한 -it 옵션을 사용
- --rm 옵션은 Container를 종료할 때 Container를 파기하도록 하는 옵션으로 1번 실행한 후에 더 이상 유지할 필요가 없는 명령행 도구 Container를 실행할 때 유용
- -v 옵션은 호스트와 Container 간에 디렉터리나 파일을 공유하기 위해 사용하는 옵션



Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container의 목록 보기

`docker container ls [options]`

❑ docker container 명령의 옵션

- `-q` 옵션을 이용하면 Container ID 만 출력
- `--filter "필터명=값"` 을 이용하면 필터 조건을 만족하는 Container 만 출력
- `-a` 옵션을 이용하면 이미 종료된 Container를 포함한 모든 Container 목록을 출력



Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container 중지

`docker container stop ContainerID 또는 Container이름`

✓ 도커 Container 재시작

`docker container restart ContainerID 또는 Container이름`

✓ 도커 Container 삭제

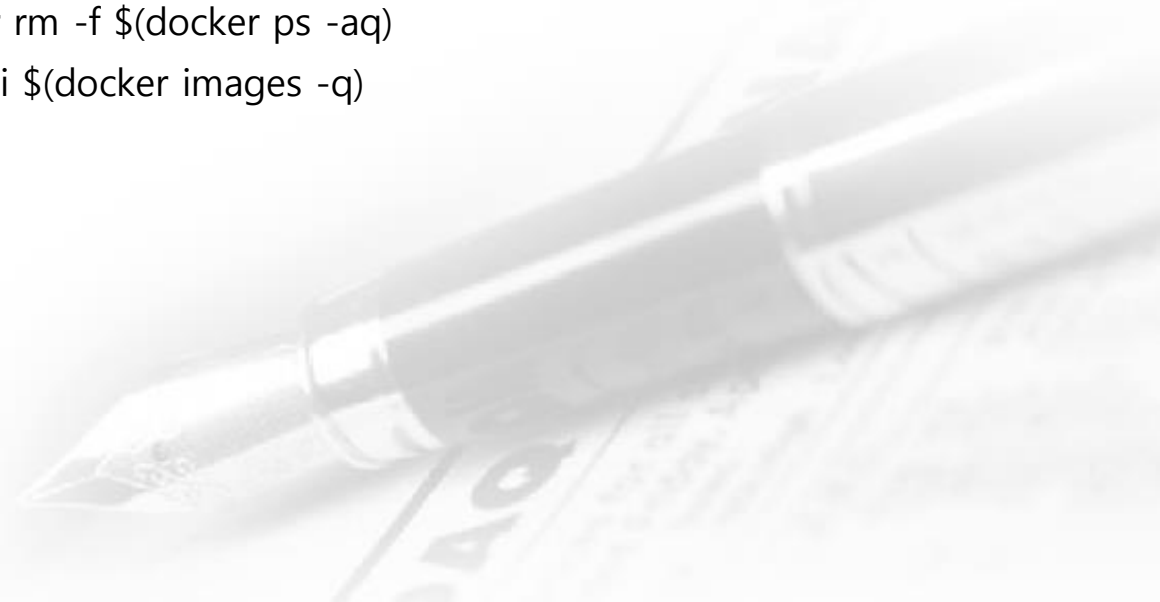
`docker container rm ContainerID 또는 Container이름`

❑ 현재 실행 중인 Container를 삭제할 때는 rm 다음에 -f 옵션을 추가

❑ container 대신에 image를 사용하고 image 이름을 대입하면 image를 삭제함

❑ 모든 Container 삭제: `docker rm -f $(docker ps -aq)`

❑ 모든 image 삭제: `docker rmi $(docker images -q)`



Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container 표준 출력 연결

`docker container logs [Option] ContainerID 또는 Container이름`

❑ `-f` 옵션을 이용해서 새로 출력되는 표준 출력 내용을 계속 보여줌

✓ 도커 Container에서 명령 수행

`docker container exec [Option] ContainerID 또는 Container이름 실행할명령`

❑ `docker container exec -it echo sh` 을 이용하면 ssh에 로그인한 것 처럼 사용



Docker Container Deployment

❖ 도커 Container 명령

✓ 도커 Container 파일 복사

- ❑ 호스트 운영체제의 파일을 Container에 복사: `docker container cp [Option] ContainerID 또는 Container이름:원본파일 대상파일`
- ❑ Container의 파일을 호스트 운영체제에게 복사: `docker container cp [Option] 호스트원본파일 ContainerID 또는 Container이름:대상파일`



Docker Container Deployment

- ❖ 도커 운영 과 관리를 위한 명령
 - ✓ 실행 중인 Container를 제외한 모든 Container 파기
docker container prune
 - ✓ 태그가 붙지 않은 모든 image 파기
docker image prune
 - ✓ 사용하지 않는 도커 image 및 Container, 볼륨, 네트워크 등 모든 도커 리소스를 일괄적으로 삭제
docker system prune
 - ✓ 사용 현황 확인
docker container stats



Docker Container Deployment

❖ docker-compose

- ✓ yaml 포맷으로 기술된 설정 파일을 이용해서 여러 Container 간의 실행이나 관계를 설정할 수 있음
- ✓ `docker container run -d -p 9000:8080 example/echo:latest` 명령을 yaml 파일을 이용해서 수행
 - ❑ 디렉토리에 `docker-compose.yml` 파일을 생성하고 작성

```
version: "3"
```

```
services:
```

```
  echo:
```

```
    image: example/echo:latest
```

```
    ports:
```

```
      - 9000:8080
```

- ❑ 아래 명령을 파일이 저장된 디렉토리로 이동해서 실행

```
docker-compose up -d
```

- ❑ 중지

```
docker-compose down
```



Docker Container Deployment

❖ docker-compose

✓ Dockerfile을 yaml 파일을 이용해서 수행

❑ 디렉토리에 docker-compose.yml 파일을 생성하고 작성

```
version: "3"
```

```
services:
```

```
  echo:
```

```
    build: .
```

```
    ports:
```

```
      - 9000:8080
```

❑ 아래 명령을 파일이 저장된 디렉토리로 이동해서 실행

```
docker-compose up -d --build
```



Docker Container Deployment

❖ docker-compose

✓ 여러 개의 Container를 yaml 파일을 이용해서 수행

❑ 디렉토리에 docker-compose.yml 파일을 생성하고 작성

```
version: "3"
services:
  master:
    container_name: master
    image: jenkinsci/jenkins:2.142-slim
    ports:
      - 8080:8080
    volumes:
      - ./jenkins_home:/var/jenkins_home
    links:
      - slave01

  slave01:
    container_name: slave01
    image: jenkinsci/ssh-slave
    environment:
      - JENKINS_SLAVE_SSH_PUBKEY=ssh-rsa AAAAB3NzaC1yc2
```

❑ 아래 명령을 파일이 저장된 디렉토리로 이동해서 실행

```
docker-compose up -d --build
```

Jenkins

- ❖ 젠킨스(Jenkins)는 소프트웨어 개발 시 지속적 통합(continuous integration) 서비스를 제공하는 툴
- ❖ 다수의 개발자들이 하나의 프로그램을 개발할 때 버전 충돌을 방지하기 위해 각자 작업한 내용을 공유 영역에 있는 Git등의 저장소에 빈번히 업로드함으로써 지속적 통합이 가능하도록 해 줌
- ❖ 개발자로 일을 하고자 할 때는 중요한 Tool



Ubuntu Linux

❖ 리눅스 설치

✓ Ubuntu image 검색 : `docker search --limit 25 ubuntu`

```
(base) adam@Adams-MacBook-Pro docker % docker search --limit 25 ubuntu
```

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
ubuntu	Ubuntu is a Debian-based Linux operating sys...	12861	[OK]	
dorowu/ubuntu-desktop-lxde-vnc	Docker image to provide HTML5 VNC interface ...	571		[OK]
websphere-liberty	WebSphere Liberty multi-architecture images ...	280	[OK]	
rastasheep/ubuntu-ssh	Dockerized SSH service, built on top of offi...	255		[OK]
consol/ubuntu-xfce-vnc	Ubuntu container with "headless" VNC session...	241		[OK]
ubuntu-upstart	DEPRECATED, as is Upstart (find other proces...	113	[OK]	
ansible/ubuntu14.04-ansible	Ubuntu 14.04 LTS with ansible	98		[OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mysql-5	ubuntu-16-nginx-php-phpmyadmin-mysql-5	50		[OK]
open-liberty	Open Liberty multi-architecture images based...	48	[OK]	
ubuntu-debootstrap	DEPRECATED; use "ubuntu" instead	44	[OK]	
i386/ubuntu	Ubuntu is a Debian-based Linux operating sys...	25		
nuagebec/ubuntu	Simple always updated Ubuntu docker images w...	24		[OK]
1and1internet/ubuntu-16-apache-php-5.6	ubuntu-16-apache-php-5.6	14		[OK]
1and1internet/ubuntu-16-apache-php-7.0	ubuntu-16-apache-php-7.0	13		[OK]
1and1internet/ubuntu-16-nginx-php-phpmyadmin-mariadb-10	ubuntu-16-nginx-php-phpmyadmin-mariadb-10	11		[OK]
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4	ubuntu-16-nginx-php-5.6-wordpress-4	9		[OK]
1and1internet/ubuntu-16-nginx-php-5.6	ubuntu-16-nginx-php-5.6	8		[OK]
1and1internet/ubuntu-16-apache-php-7.1	ubuntu-16-apache-php-7.1	7		[OK]
1and1internet/ubuntu-16-nginx-php-7.1	ubuntu-16-nginx-php-7.1	5		[OK]
1and1internet/ubuntu-16-nginx-php-7.0	ubuntu-16-nginx-php-7.0	4		[OK]
1and1internet/ubuntu-16-sshd	ubuntu-16-sshd	1		[OK]
smartentry/ubuntu	ubuntu with smartentry	1		[OK]
1and1internet/ubuntu-16-php-7.1	ubuntu-16-php-7.1	1		[OK]
1and1internet/ubuntu-16-rspec	ubuntu-16-rspec	0		[OK]
ossobv/ubuntu	Custom ubuntu image from scratch (based on o...	0		

Ubuntu Linux

❖ 리눅스 설치

✓ Ubuntu image 다운로드 : docker pull ubuntu

```
(base) adam@Adams-MacBook-Pro docker % docker pull ubuntu
```

```
Using default tag: latest
```


```
latest: Pulling from library/ubuntu
```

```
35807b77a593: Pull complete
```

```
Digest: sha256:9d6a8699fb5c9c39cf08a0871bd6219f0400981c570894cd8cbea30d3424a31f
```

```
Status: Downloaded newer image for ubuntu:latest
```

```
docker.io/library/ubuntu:latest
```



Ubuntu Linux

❖ 리눅스 설치

✓ Ubuntu 컨테이너 생성 : `docker create -it --name ubuntu_server ubuntu`

```
[(base) adam@Adams-MacBook-Pro docker % docker create -it --name ubuntu_server ubuntu
09634fa86dfb2c9af132e900269d6fc97b5a0276b48e5caff6d563612c5cb9e4
```

```
[(base) adam@Adams-MacBook-Pro docker % docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
09634fa86dfb	ubuntu	"bash"	23 seconds ago	Created		ubuntu_server

```
[[base] adam@Adams-MacBook-Pro docker % docker exec -it ubuntu_server bash
```



Ubuntu Linux

❖ 리눅스 설치

✓ Ubuntu 컨테이너 실행 : docker start ubuntu_server

```
[(base) adam@Adams-MacBook-Pro docker % docker start ubuntu_server  
ubuntu_server
```

.....



Ubuntu Linux

❖ 리눅스 설치

✓ Ubuntu 컨테이너 접속 : `docker attach ubuntu_server`

```
(base) adam@Adams-MacBook-Pro docker % docker attach ubuntu_server  
root@09634fa86dfb:/#
```



Ubuntu Linux

❖ 리눅스 설치

✓ Ubuntu 명령 수행 : cat etc/issue

Ubuntu 20.04.3 LTS \backslash n \backslash l

✓ Ubuntu 명령 수행 : apt-get update

✓ Ubuntu 명령 수행 : apt-get upgrade

✓ Ubuntu 명령 수행 : apt-get install vim



Oracle 설치

❖ Mac - Oracle 11g Express Edition 설치

✓ 오라클 11g 이미지 확인

> docker search oracle-xe-11g

✓ 이미지 다운로드 및 설치

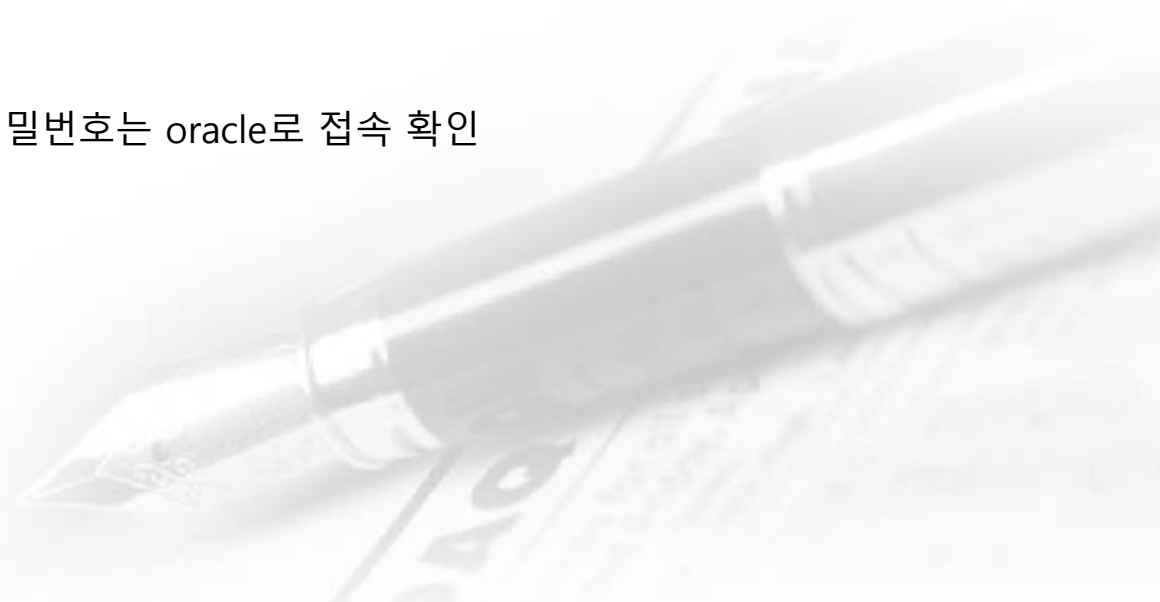
> docker pull jaspeen/oracle-xe-11g

✓ 도커 이미지 실행

> docker run -name oracle11g -d -p 8080:8080 -p 1521:1521 jaspeen/oracle-xe-11g

✓ 설치 확인

> sid는 xe 계정은 system 비밀번호는 oracle로 접속 확인



Oracle 설치

❖ Mac - Oracle 18c Express Edition 설치

- ✓ 오라클 18c Express Edition 리눅스 버전 다운로드

<https://www.oracle.com/database/technologies/xe-downloads.html>

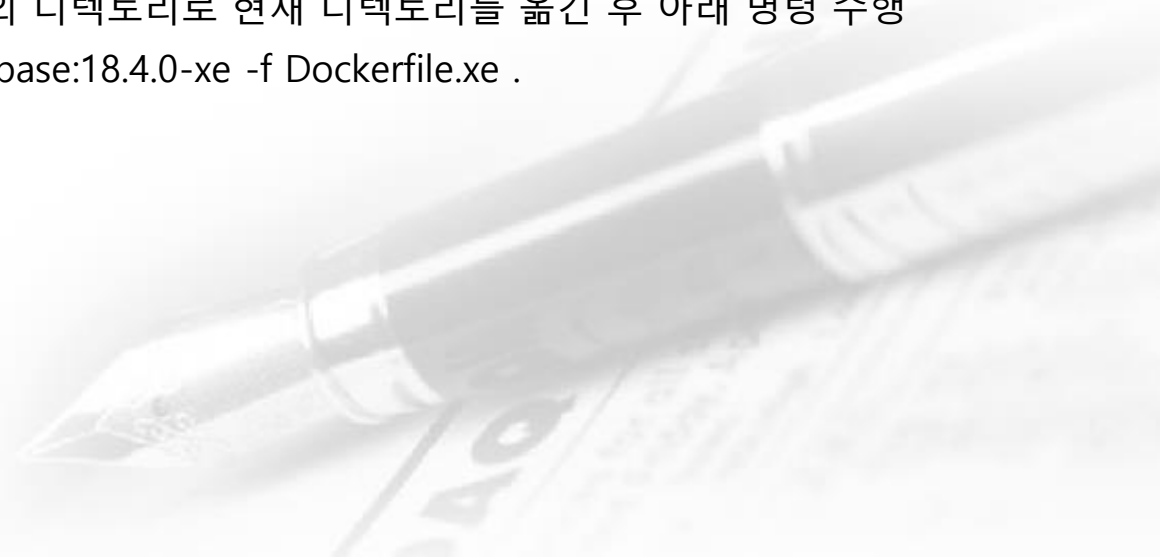
- ✓ 오라클의 git 에 접속해서 아래 디렉토리의 파일들만 특정 디렉토리에 복사

<https://github.com/oracle/docker-images/tree/master/OracleDatabase/SingleInstance/dockerfiles/18.4.0>

- ✓ Download 받은 rpm 파일을 위에서 생성한 디렉토리에 복사

- ✓ 이미지 생성 – 터미널에서 위의 디렉토리로 현재 디렉토리를 옮긴 후 아래 명령 수행
docker build -t oracle/database:18.4.0-xe -f Dockerfile.xe .

- ✓ 이미지 확인
docker images



Oracle 설치

❖ Mac - Oracle 18c Express Edition 실행

<https://github.com/oracle/docker-images/tree/master/OracleDatabase/SingleInstance#running-oracle-database-18c-express-edition-in-a-docker-container>

```
docker run --name <container name>  
-p <host port>:1521 -p <host port>:5500  
-e ORACLE_PWD=<your database passwords>  
-e ORACLE_CHARACTERSET=<your character set>  
-v [<host mount point>:]/opt/oracle/oradata  
oracle/database:18.4.0-xe
```

✓ 샘플

```
docker run --name myoracle  
-p 1521:1521 -p 5500:5500  
-e ORACLE_PWD=wnddkd  
-v $PWD/mount/data:/opt/oracle/oradata  
oracle/database:18.4.0-xe
```



Oracle 설치

❖ Mac - Oracle 18c Express Edition 실행

```
oracleimage — com.docker.cli ◀ docker run --name myoracle -p 1521:1521 -p 5...  
The Oracle base remains unchanged with value /opt/oracle  
#####  
DATABASE IS READY TO USE!  
#####  
The following output is now a tail of the alert.log:  
Pluggable database XEPDB1 opened read write  
Completed: alter pluggable database XEPDB1 open  
2020-10-02T03:47:15.385772+00:00  
XEPDB1(3):CREATE SMALLFILE TABLESPACE "USERS" LOGGING DATAFILE '/opt/oracle/or  
adata/XE/XEPDB1/users01.dbf' SIZE 5M REUSE AUTOEXTEND ON NEXT 1280K MAXSIZE UNL  
IMITED EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO  
XEPDB1(3):Completed: CREATE SMALLFILE TABLESPACE "USERS" LOGGING DATAFILE '/op  
t/oracle/oradata/XE/XEPDB1/users01.dbf' SIZE 5M REUSE AUTOEXTEND ON NEXT 1280K  
MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO  
XEPDB1(3):ALTER DATABASE DEFAULT TABLESPACE "USERS"  
XEPDB1(3):Completed: ALTER DATABASE DEFAULT TABLESPACE "USERS"  
2020-10-02T03:47:17.593899+00:00  
ALTER PLUGGABLE DATABASE XEPDB1 SAVE STATE  
Completed: ALTER PLUGGABLE DATABASE XEPDB1 SAVE STATE  
  
2020-10-02T04:06:24.823352+00:00  
XEPDB1(3):Resize operation completed for file# 10, old size 368640K, new size 37  
8880K
```

Oracle 설치

- ❖ Oracle 18c Express Edition 접속
 - ✓ 접속 위치는 localhost
 - ✓ 포트번호는 1521
 - ✓ SID는 XE
 - ✓ 계정은 system
 - ✓ 비밀번호는 wnddkd
- ❖ Docker에서 sqlplus를 이용해서 접속
 - ✓ `docker exec -it myoracle sqlplus sys/wnddkd@//localhost:1521/XE as sysdba`



MySQL 설치

❖ MySQL 8.0 설치

- ✓ 이미지 확인: `docker search mysql`
- ✓ 최신 버전 다운로드: `docker pull mysql`
- ✓ 컨테이너 생성 및 실행: `docker run --name 컨테이너이름 -e MYSQL_ROOT_PASSWORD=root비밀번호 -d -p 3306:3306 mysql:latest`
- ✓ 컨테이너 접속: `docker exec -it 컨테이너이름 bash`



MySQL 설치

❖ MySQL 8.0 명령어

- ✓ 루트 접속: `mysql -u root -p` 명령 실행 후 root 비밀번호 입력
- ✓ 데이터베이스 생성: `create database user00;`
- ✓ 사용자 생성 및 비밀번호 설정: `create user 'user00'@'%' identified by 'user00 ' ;`
- ✓ 권한 부여: `GRANT all privileges on *.* TO 'user00'@'%' ;`
- ✓ 외부 접속 허용:
 - `ALTER USER 'user00'@'%' IDENTIFIED WITH mysql_native_password BY 'user00';`
 - `FLUSH privileges;`
 - 데이터베이스 이름 대신에 *을 대입하면 모든 데이터베이스 사용 가능
 - %대신에 ip를 기재하면 특정 ip에서만 접속이 허용되며 localhost로 지정하면 현재 컴퓨터에서만 접속이 가능

