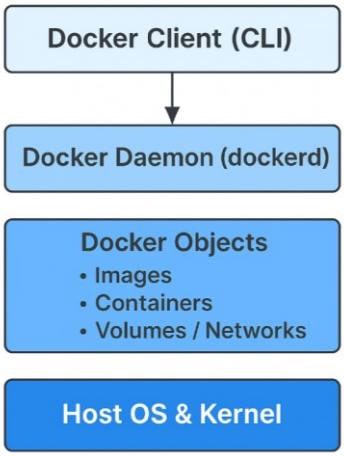
DevOps for SpringBoot

단계	주제	내용
[1]	리눅스 기초 I (명령어와 파일 시스템)	- 리눅스 디렉토리 구조 이해- 기본 명령어 (ls, cd, mkdir, rm, cp, mv, cat, chmod 등)- 사용자 및 퍼미션 이해- 실습: 파일 조작, 사용자 추가, 권한 변경
[2]	리눅스 기초 II (WSL2 네트워크 실습)	- 네트워크 명령어: ping, curl, ss, netstat, ip- 패키지 설치: apt update, apt install- 프로세스/포트 확인: ps aux, top, ss -tuln- 실습: 네트워크 점검, 패키지 관리, 실행 중인 프로세스 추적
[3]	Docker & Spring Boot 컨테이너화	- Docker 기본 구조 및 명령어 실습- Dockerfile 작성- Spring Boot 컨테이너화- Docker Compose 로 DB 연동 실습
[4]	Jenkins 설치 및 CI 구성	- Docker 로 Jenkins 설치- Jenkins 초기 설정 및 Git 연동- Jenkins Job 생성 및 빌드 자동화- Jenkinsfile 작성: Build + Test
[5]	Kubernetes 설치 및 수동 배포	- Minikube 설치 및 kubectl 설정- Kubernetes 개념 (Pod, Deployment, Service)- Spring Boot 앱 수동 배포 (YAML 작성)- 실습: 서비스 접속 및 로깅 확인
[6]	Jenkins → Kubernetes 자동 배포 (CD)	- Jenkins 에 kubectl 연동 (kubeconfig)- Jenkinsfile 수정: Build → Deploy 자동화- 실습: Git Push → Jenkins → K8s 자동배포 구성
[7]	Ingress-Nginx + 경로 기반 라우팅	- Ingress Controller 설치 (Minikube Addon)- 도메인/경로 기반 서비스 분리- 실습: /api, /admin, /user 등으로 분기 라우팅 구성
[8]	Prometheus + Grafana 모니터링	- Prometheus 설치 및 Spring Boot 와 연동- actuator, micrometer 설정- Grafana 설치 및 대시보드 구성- 실습: JVM 메모리, 요청 수, 응답 속도 시각화
[9]	전체 통합 배포 흐름 구성	- Git → Jenkins → Docker → K8s → Ingress → Grafana- 장애 복구 및 배포 실패 대응 실습- 실습: 실제 시나리오 기반 전체 배포 테스트

1. Docker 란?

Docker 는 소프트웨어를 컨테이너라는 단위로 패키징하고 실행하는 플랫폼이다.

- 애플리케이션과 그 실행에 필요한 모든 환경(라이브러리, 설정 등)을 하나로 묶은 실행 단위(컨테이너)를 만들 수 있다.
- 컨테이너는 운영체제와 무관하게 어디서든 똑같이 실행된다.
- Docker 는 개발부터 배포까지 환경을 통째로 묶어 실행할 수 있게 해주는 도구이다.
- 도커는 사용자 명령을 도커 클라이언트 → 도커 데몬을 통해 전달하고, 리눅스 커널 기능을 이용하여 이미지에서 컨테이너를 실행하는 **경량 가상화 플랫폼**이다.



1. Docker Client (도커 클라이언트)

- 명령어 입력하는 인터페이스 (docker run, docker build 등)
- 내부적으로는 REST API 요청을 도커 데몬에게 전달

2. Docker Daemon (dockerd)

- 모든 컨테이너 생성, 실행, 삭제를 관리하는 **중앙 엔진**
- 클라이언트 명령을 받아 Docker API 처리
- 리눅스 커널 기능(cgroups, namespace 등)과 직접 통신

3. Docker Objects (객체들)

- Images : 실행 가능한 애플리케이션 패키지 (OS + App + 설정 포함)
- Containers : 이미지를 기반으로 실행된 **격리된 프로세스**
- Volumes : 데이터 저장소 (컨테이너가 종료되어도 데이터 유지)
- Networks : 컨테이너 간 네트워크를 연결

4. Host OS

- 도커는 **리눅스 커널 기능**을 활용해서 격리 환경(Container)을 제공
- Windows, macOS 에서는 가상 머신(VM)을 통해 리눅스 환경을 제공한 뒤 동작

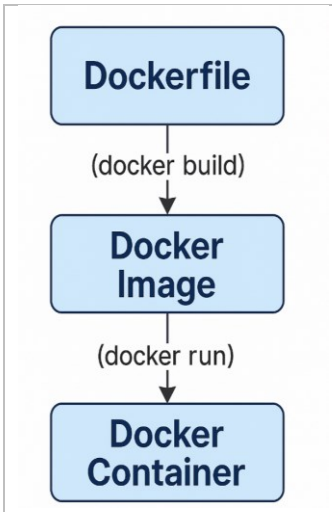
2. 왜 Docker 가 필요한가?

전통적인 방식	Docker 방식
서버마다 환경이 달라 오류 발생	동일한 환경이 컨테이너에 포함됨
개발-PC 와 운영서버 환경 차이 존재	어디서든 똑같이 실행됨
설정, 설치, 배포가 번거로움	한 번에 패키징하고 간편히 실행 가능

3. Docker 의 핵심 구성요소

- 이미지 (Image): 컨테이너를 만들기 위한 **템플릿,** 애플리케이션 + 설정 + 종속 라이브러리들이 포함됨
- 컨테이너 (Container) : 이미지를 **실행한 실제 인스턴스,** 격리된 환경에서 독립적으로 실행됨
- Docker file: 이미지를 만들기 위한 **설계도,** 설치할 패키지, 복사할 파일, 실행할 명령어 등을 정의
- Docker Engine: 컨테이너를 생성하고 관리하는 **Docker 의 핵심 실행 엔진**

4. Docker 의 동작 흐름



사용자가 운영 체제 선택, 소프트웨어 설치, 코드 복사, 환경 변수 설정 등 이미지 생성에 필요한 모든 단계를 명시

애플리케이션과 그 실행에 필요한 모든 것(코드, 런타임, 시스템 도구, 라이브러리, 설정 등)을 포함하는 읽기 전용 템플릿

Docker 이미지의 실행 가능한 인스턴스, 호스트 시스템의 다른 프로세스와 격리되어 작동

.5. Docker 의 장점

장 점	설 명
일관된 실행 환경	어디서나 동일하게 동작 (OS 상관 없음)
빠른 배포	이미지를 빌드하고 바로 실행 가능
가벼운 성능	기존 VM 보다 가볍고 빠름
버전 관리	이미지 태그를 이용한 버전 관리 가능
격리성 보장	컨테이너별 독립 실행 가능

6. Docker vs 가상머신(VM)

항목	Docker	Virtual Machine
실행 단위	컨테이너	가상 머신
OS 포함 여부	Host OS 공유	별도 Guest OS 포함
속도	매우 빠름	상대적으로 느림
용량	수 MB~GB	수 GB 이상
리소스 사용	효율적	비교적 무거움

https://hub.docker.com 란?

Docker Hub 는 전 세계 개발자들이 만든 Docker 이미지를 **검색, 다운로드, 공유, 배포**할 수 있는 **클라우드 기반 레지스트리 서비스**이다.

기능	설명
이미지 검색 (docker search)	MySQL, Nginx, Ubuntu 등 다양한 이미지 검색 가능
이미지 다운로드 (docker pull)	원하는 이미지를 내 PC 에 내려받기
이미지 업로드 (docker push)	만든 이미지를 Docker Hub 에 업로드하여 공유
자동 빌드	GitHub 연동 → 코드 커밋 시 자동으로 이미지 빌드
팀 & 조직 관리	private repository, 권한 설정 가능

7. Docker 로 할 수 있는 일

- 웹 서버, API 서버 실행
- Spring Boot, Node.js 등 컨테이너 배포
- DB, Redis, Kafka 등 개발용 환경 실행
- CI/CD 파이프라인 구성
- 마이크로서비스 아키텍처 운영

8. 대표적인 Docker 명령어 요약

명령어	설명
`docker --version`	도커 버전 확인
`docker pull 이미지`	이미지 다운로드
`docker build`	Dockerfile 기반 이미지 생성
`docker run`	이미지 실행하여 컨테이너 생성
`docker ps`	실행 중인 컨테이너 목록
`docker stop`	컨테이너 중지
`docker rm`	컨테이너 삭제
`docker images`	다운로드된 이미지 목록

[Ubuntu + WSL2] Docker 설치

1 단계: 시스템 업데이트

```
sudo apt update && sudo apt upgrade -y
```

2 단계: 필요한 패키지 설치

```
user01@Dominica:~$ sudo apt install -y \
>     ca-certificates \
>     curl \
>     gnupg \
>     lsb-release
```

패키지	용도
ca-certificates	HTTPS 인증서 검증용 루트 인증서들. Docker 저장소 접근 시 필수
curl	URL 에서 데이터 다운로드 (예: Docker GPG 키 받기용)
gnupg	GPG 키 관리 도구. Docker 저장소의 서명을 검증하는 데 사용
lsb-release	Ubuntu 배포판 정보를 얻기 위한 유틸리티 (lsb_release -cs 등에서 사용)

3 단계: Docker 공식 GPG 키 등록

curl 로 Docker 의 GPG 서명 키를 받아서, gpg --dearmor 로 리눅스에서 사용할 수 있는 .gpg 형식으로 변환하고 저장

sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | ₩

sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

```
user01@Dominica:~$ sudo mkdir -p /etc/apt/keyrings

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
```

4 단계: Docker 저장소 추가

현재 시스템 아키텍처(amd64 또는 arm64)와 배포판 이름(focal, jammy, noble 등)에 맞는 Docker 저장소를 /etc/apt/sources.list.d 에 등록

```
echo ₩
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] ₩
  https://download.docker.com/linux/ubuntu jammy stable" | ₩
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
user01@Dominica:~$ echo \
>   "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \
https:>   https://download.docker.com/linux/ubuntu jammy stable" | \
sudo te>   sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5 단계: 패키지 목록 업데이트 -> sudo apt update

Ubuntu 24.04 "noble" 버전용 저장소들이 잘 연결된 상태확인{시간소요}
https://download.docker.com 관련 항목이 표시되면 정상 등록

```
user01@Dominica:~$ sudo apt update
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Hit:2 http://archive.ubuntu.com/ubuntu noble-updates InRelease
Hit:3 http://archive.ubuntu.com/ubuntu noble-backports InRelease
Hit:4 https://download.docker.com/linux/ubuntu jammy InRelease
Hit:5 http://security.ubuntu.com/ubuntu noble-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
```

6 단계: Docker 설치

```
sudo apt install -y ₩
    docker-ce ₩
    docker-ce-cli ₩
    containerd.io ₩
    docker-buildx-plugin ₩
    docker-compose-plugin
```

```
user01@Dominica:~$ sudo apt install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

패키지	역할
docker-ce	Docker 엔진 (Community Edition)
docker-ce-cli	도커 CLI (명령줄 인터페이스)
containerd.io	컨테이너 런타임 백엔드
docker-buildx-plugin	고급 이미지 빌드 지원 (멀티 플랫폼 등)
docker-compose-plugin	Compose V2 지원 (docker compose 명령 사용 가능)

7 단계 : 설치 후 확인 -> hello-world 컨테이너 테스트

```
docker --version
sudo docker run hello-world
```

```
user01@Dominica:~$ docker --version
Docker version 28.2.2, build e6534b4
user01@Dominica:~$
user01@Dominica:~$ sudo docker run hello-world
```

아래 메시지가 출력되면 **Docker 설치 완료 및 정상 작동** 상태

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

sudo 없이 docker 명령 사용

```
user01@Dominica:~$ sudo usermod -aG docker $USER
```

8. 도커 설치 후 꼭 알아야 할 주요 파일과 디렉토리

경로	설명
/var/run/docker.sock	도커 CLI ↔ 데몬 통신용 소켓 파일
/etc/docker/	도커의 설정 파일 위치 (예: daemon.json)
/var/lib/docker/	실제 컨테이너/이미지/볼륨 데이터 저장소
/usr/bin/docker	docker 명령 실행파일 (CLI)
/etc/systemd/system/docker.service 또는 docker.service	도커 시작/중지 제어 파일 (systemd)
~/.docker/	사용자의 Docker CLI 환경 설정 디렉토리
/etc/apt/sources.list.d/docker.list	apt 저장소에 등록된 Docker repo 파일

```
user02@Dominica:~$ ls -l /var/run/docker.sock
srw-rw---- 1 root docker 0 Jun 19 17:05 /var/run/docker.sock
```

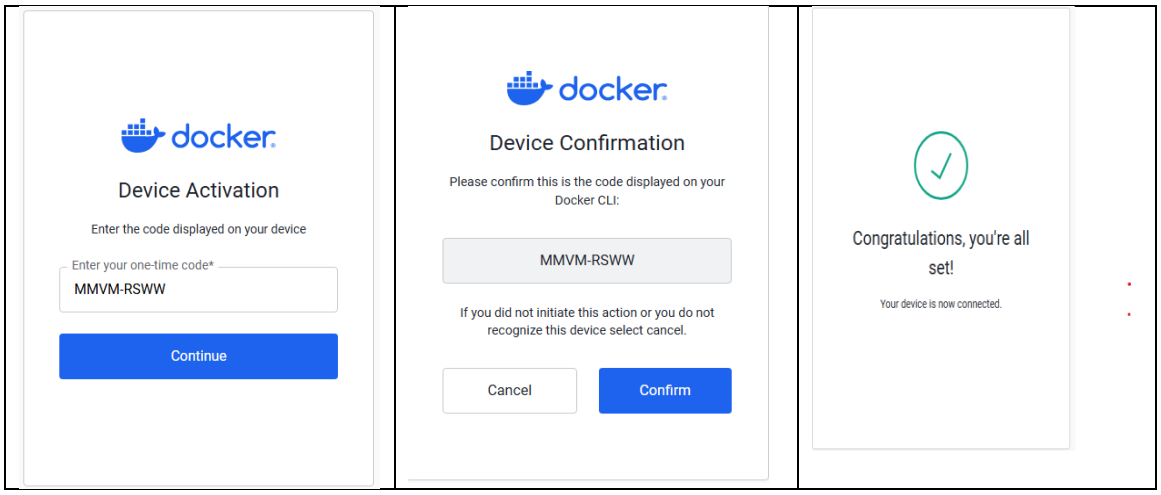
9. 웹 기반 로그인 : docker login 하면 터미널 대기 시작

```
user01@Dominica:~$ docker login

USING WEB-BASED LOGIN

i Info → To sign in with credentials on the command line, use 'docker login -u <usern
ame>'


Your one-time device confirmation code is: MMVM-RSWW
Press ENTER to open your browser or submit your device code here: https://login.docke
r.com/activate

Waiting for authentication in the browser…
```

https://login.docker.com/activate 로그인 -> MMVM-RSWW 입력

사용자 전용 context 설정, 인증서 설정 / Docker Hub ID 기억

Username 필드에 원하는 사용자 이름을 입력 -> Sign up



터미널로 돌아가면 자동으로 로그인 완료 메시지와

/home/user01/.docker/config.json 파일에 Docker Hub 로그인 정보가 저장된 상태로 출력

```
WARNING! Your credentials are stored unencrypted in '/home/user01/.docker/config.json'.
Configure a credential helper to remove this warning. See
https://docs.docker.com/go/credential-store/

Login Succeeded
```

저장 확인 / https://hub.docker.com 에서 Docker Id 확인

```
user01@Dominica:~$ cat ~/.docker/config.json
{
        "auths": {
                "https://index.docker.io/v1/": {
                        "auth": "ZmluaXNoMDdzZHM6ZGNrcl9wYXRfTWQ0NGdXQVdjTU15V2RmeFVMcVpFRnNQdjhz"
                },
                "https://index.docker.io/v1/access-token": {
```

```
user01@Dominica:~$ docker info | grep Username
```

이미지 다운로드

```
user01@Dominica:~$ docker pull ubuntu
```

나만의 이미지 업로드 준비

1 단계: 로컬 이미지 태그 다시 지정

Docker 에 기본으로 있는 hello-world 이미지는 **공식 이미지**이므로,
push 하려면 먼저 **내 Docker Hub 이름을 포함한 새 태그**로 다시 태그 설정

docker tag hello-world ID/hello-world:latest

```
user01@Dominica:~$ docker tag hello-world finish07sds/hello-world:latest
```

2 단계: 푸시 시도

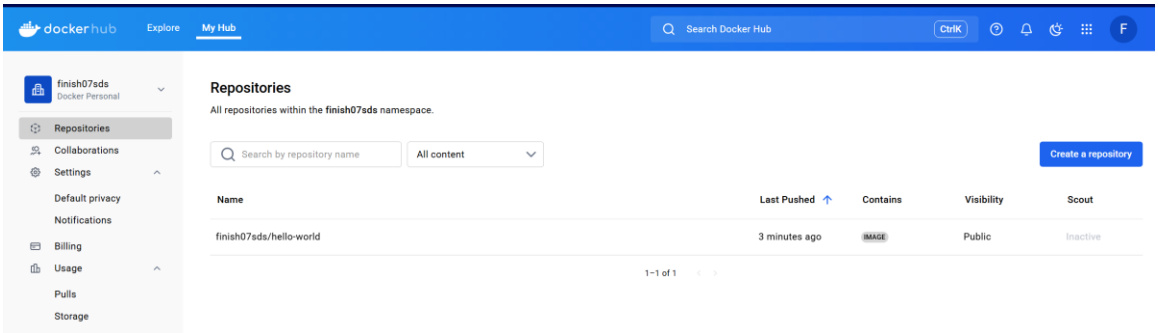
docker push ID/hello-world:latest

```
user01@Dominica:~$ docker push finish07sds/hello-world:latest
The push refers to repository [docker.io/finish07sds/hello-world]
63a41026379f: Pushed
latest: digest: sha256:7565f2c7034d87673c5ddc3b1b8e97f8da794c31d9aa73ed26afffa1c8194889 size: 524
```

3 단계 :확인

웹에서 확인 https://hub.docker.com/repositories

CLI 에서 확인 : docker search hello-world



[실습 01] 나만의 Dockerfile 을 작성과 이미지 실행

단 계	명령어
디렉토리 생성	mkdir my-docker-test && cd my-docker-test
Dockerfile 작성	nano Dockerfile
이미지 빌드	docker build -t my-first-docker-image .
이미지 실행	docker run my-first-docker-image

```
user01@Dominica:~$ mkdir my-docker-test
my-docker-test
user01@Dominica:~$ cd my-docker-test
user01@Dominica:~/my-docker-test$
```

nano Dockerfile

```
# 1. 베이스 이미지 선택 (가볍고 빠른 Alpine 리눅스 사용)
FROM alpine

# 2. 컨테이너 시작 시 실행할 명령
CMD ["echo", "Hello Docker! This is my custom image."]
```

이미지 빌드 : docker build -t my-first-docker-image .

- -t → 이미지 이름 태그 , . → 현재 디렉토리의 Dockerfile 을 기반으로 빌드

```
user01@Dominica:~/my-docker-test$ docker build -t my-first-docker-image .
[+] Building 3.8s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 229B
```

이미지 실행 : docker run my-first-docker-image

```
user01@Dominica:~/my-docker-test$ docker run my-first-docker-image
Hello Docker! This is my custom image.
```

이미지 확인 : docker images

```
user01@Dominica:~/my-docker-test$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
my-first-docker-image     latest    807028f5b60b   2 weeks ago     8.31MB
ubuntu                    latest    bf16bdcff9c9   2 weeks ago     78.1MB
```

[실습 02] 나만의 Dockerfile 확장 실습 문제

Alpine 리눅스를 베이스로 한 Docker 이미지에서 실행 시 {현재 날짜와 사용자 이름을 출력}하도록 Dockerfile 을 작성해보자

1 단계: 디렉토리 및 Dockerfile 생성

my-echo-docker 디렉토리 생성 , 내부에 Dockerfile 생성

2 단계: 아래 명령어가 실행되도록 Dockerfile 작성

```
FROM alpine
CMD ["sh", "-c", "echo Hello, $(whoami)! Today is $(date)."]
```

3 단계: 이미지 이름은 echo-today 로 지정하여 빌드

docker build -t echo-today .

```
user01@Dominica:~/my-echo-docker$ docker build -t echo-today .
```

4 단계: 이미지 실행 시 아래와 같은 출력이 나오도록 만들 것

```
user01@Dominica:~/my-echo-docker$ docker run echo-today
Hello, root! Today is Thu Jun 19 09:19:16 UTC 2025.
```

```
user01@Dominica:~/my-echo-docker$ docker images
REPOSITORY               TAG       IMAGE ID       CREATED         SIZE
my-first-docker-image    latest    807028f5b60b   2 weeks ago     8.31MB
echo-today               latest    b5fc14f6499a   2 weeks ago     8.31MB
```

[실습 03] 하나의 디렉토리에서 여러 Dockerfile 사용하기

- 동일한 디렉토리 내에 2 개의 Dockerfile(Dockerfile.dev, Dockerfile.prod)을 만들어 사용해 본다.

- docker build -f 명령으로 원하는 Dockerfile 을 선택하여 실행한다.

1 단계: my-docker-app 디렉토리를 만들고 이동하시오.

```
user01@Dominica:~$ mkdir my-docker-app
user01@Dominica:~$ cd my-docker-app
```

2 단계 :아래 내용을 각각의 Dockerfile 에 작성하시오

```
user01@Dominica: ~/my-docker-app
  GNU nano 7.2                                          Dockerfile.dev *
FROM alpine
CMD ["sh", "-c", "echo '[DEV] Hello developer! This is the dev container.'"]
```

```
user01@Dominica: ~/my-docker-app
  GNU nano 7.2                                          Dockerfile.prod
FROM alpine
CMD ["sh", "-c", "echo '[PROD] Hello user! This is the production container.'"]
```

3 단계 :두 개의 이미지를 각각 이름을 다르게 빌드 하시오

```
user01@Dominica:~/my-docker-app$ docker build -f Dockerfile.dev -t myapp:dev .
user01@Dominica:~/my-docker-app$ docker build -f Dockerfile.prod -t myapp:prod .
```

4 단계 : 이미지를 실행하여 결과를 확인하시오

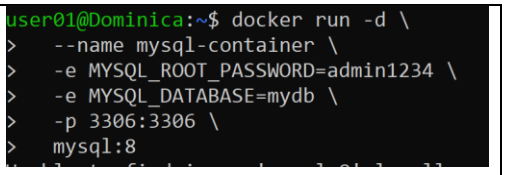
```
user01@Dominica:~/my-docker-app$ docker run myapp:dev
[DEV] Hello developer! This is the dev container.
user01@Dominica:~/my-docker-app$ docker run myapp:prod
[PROD] Hello user! This is the production container.
```

[실습 : Docker 에서 MySQL 설치]

[조건] Docker 에 MySQL 을 설치하고 Windows 에 있는 Spring Boot 앱이 Docker 의 MySQL 에 접속하도록 설정해보자.	

1 단계: Docker 에서 MySQL 컨테이너 실행

옵션	설명
--name mysql-container	컨테이너 이름
-e MYSQL_ROOT_PASSWORD=root1234	루트 비밀번호 설정
-e MYSQL_DATABASE=mydb	자동 생성될 DB 이름
-p 3306:3306	호스트:컨테이너 포트 매핑

docker run -d ₩ --name mysql-container ₩ -e MYSQL_ROOT_PASSWORD=root1234 ₩ -e MYSQL_DATABASE=mydb ₩ -p 3306:3306 ₩ mysql:8	

2 단계: 실행 확인 _ 컨테이너 리스트 보기

```
user01@Dominica:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
PORTS                                                  NAMES
5d68f265ff1a   mysql:8   "docker-entrypoint.s…"   28 minutes ago   Up 28 minutes
0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp, 33060/tcp   mysql-container
```

지금 MySQL 은 어디에 설치되어 있을까?

=> **Docker 컨테이너 안에서 실행 중** Ubuntu(WSL2)의 파일시스템이나 Windows 로컬에 설치된 것이 아님

Windows
└── WSL2 (Ubuntu)
└── Docker
└── MySQL 컨테이너
└── /var/lib/mysql ← DB 저장 경로 (컨테이너 내부)

3 단계: 컨테이너 내부 접속 (선택)

docker exec -it mysql-container mysql -uroot -p 비번

```
user01@Dominica:~$ docker exec -it mysql-container mysql -uroot -padmin1234
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.4.5 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

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>
```

4 단계 : MySQL 데이터 저장 경로 확인

docker inspect mysql-container | grep /var/lib/mysql

```
user01@Dominica:~$ docker inspect mysql-container | grep /var/lib/mysql
                "Destination": "/var/lib/mysql",
                "/var/lib/mysql": {}
```

→ /var/lib/mysql 은 컨테이너 안의 경로이며, 실제 데이터는 해당 컨테이너 안에 저장

? 데이터는 어디에 저장될까?

- 기본 설정에서는 **컨테이너 내부에 저장**
- 컨테이너 삭제 시 함께 사라짐
- **지속적인 데이터 저장이 필요하다면 volume 설정 필요**

  -v mysql-data:/var/lib/mysql

기존 컨테이너 제거 후 MySQL 을 volume 에 연결해서 실행

```
docker run -d ₩
  --name mysql-container ₩
  -e MYSQL_ROOT_PASSWORD=admin1234 ₩
  -e MYSQL_DATABASE=mydb ₩
  -p 3306:3306 ₩
  -v mysql-data:/var/lib/mysql ₩
  mysql:8
```

```
user01@Dominica:~$ docker run -d \
  --name>   --name mysql-container \
>   -e MYSQL_ROOT_PASSWORD=admin1234 \
>   -e MYSQL_DATABASE=mydb \
>   -p 3306:3306 \
>   -v mysql-data:/var/lib/mysql \
>   mysql:8
4bb8d71faa9c4f9fa62537df1edb8749f716e21100e1c1f1e92647cab9778d19
```

볼륨확인

```
user01@Dominica:~$ docker volume ls
DRIVER    VOLUME NAME
local     933570409af0d0b3fc3ab7c4ee5ad9e51b20783cffad4742dbc3769559e0c549
local     mysql-data
```

5 단계 : Docker Hub 에서 이미지 확인

```
user01@Dominica:~$ docker images
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
my-first-docker-image     latest    807028f5b60b   2 weeks ago     8.31MB
echo-today                latest    b5fc14f6499a   2 weeks ago     8.31MB
myapp                     prod      08e74f9f7c1b   2 weeks ago     8.31MB
myapp                     dev       580d4babd2e5   2 weeks ago     8.31MB
ubuntu                    latest    bf16bdcff9c9   3 weeks ago     78.1MB
mysql                     8         8fbbf951246a   2 months ago    777MB
```

로컬에 컨테이너가 실행되고 있는가?

```
user01@Dominica:~$ docker ps -a
CONTAINER ID   IMAGE                  COMMAND                  CREATED             STATUS
               PORTS                                                    NAMES
4bb8d71faa9c   mysql:8                "docker-entrypoint.s…"   9 minutes ago       Up 9 minutes
               0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp, 33060/tcp   mysql-container
```

Docker Hub 사이트에서 내 이미지 업로드 여부

Docker Hub **"내 계정"의 저장소 목록에는 나타나지 않는다**

직접 만든 이미지를 푸시해야만 **Docker Hub 에 등록된다.**

```
docker build → docker tag → docker push
```

6 단계: Windows MySQL 중지 후 spring-boot 프로젝트 실행

```
C:₩WINDOWS₩system32>net stop mysql80
```

스크립트실행

```
docker exec -i mysql-container ₩
  mysql -uroot -padmin1234 mydb <
/mnt/d/myWork/MySpringBoot/SpringBootLab06/notice/spring_lab06.sql
```

```
user01@Dominica:~$ docker exec -i mysql-container \
>   mysql -uroot -padmin1234 mydb < /mnt/d/myWork/MySpringBoot/SpringBootLab06/notice/spring_lab06.s
ql
```

docker exec -it mysql-container mysql -uroot -padmin1234 mydb

```
user01@Dominica:~$ docker exec -it mysql-container mysql -uroot -padmin1234 mydb
```

확인!!

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb               |
| mysql              |
| performance_schema |
| spring_lab06       |
| sys                |
+--------------------+
6 rows in set (0.00 sec)
```

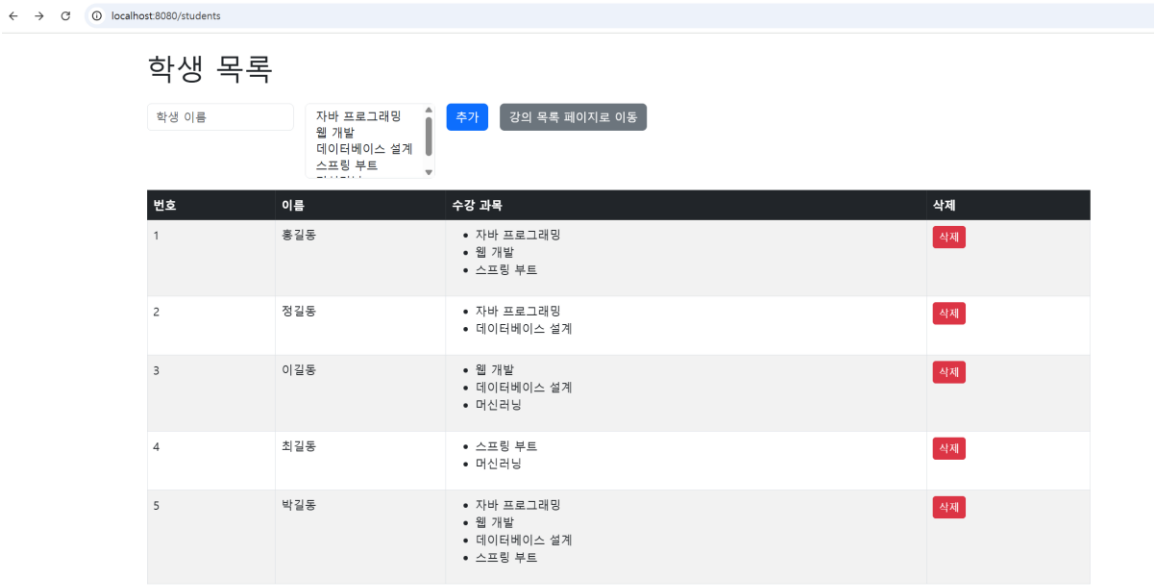
포트확인

```
user01@Dominica:~$ docker inspect mysql-container | grep '"HostPort"'
                        "HostPort": "3306"
                        "HostPort": "3306"
                        "HostPort": "3306"
```

Spring Boot 접속 설정 _SpringLab06 설정 파일 변경 후 실행

```
user01@Dominica:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:6f:4a:77 brd ff:ff:ff:ff:ff:ff
    inet 172.23.84.149/20 brd 172.23.95.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fe6f:4a77/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether da:62:6d:c6:b4:8c brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::d862:6dff:fec6:b48c/64 scope link
       valid_lft forever preferred_lft forever
6: veth4194df0@if2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master docker0 state UP group default
    link/ether 16:2b:7c:47:1e:2d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::142b:7cff:fe47:1e2d/64 scope link
       valid_lft forever preferred_lft forever
```

```
spring:
  datasource:
    url: jdbc:mysql://172.23.84.149:3306//spring_lab06
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: root
    password: admin1234
```



한글 세팅 깨지면 case 1:

```
spring:
  datasource:
    url:
jdbc:mysql://172.23.84.149:3306/spring_lab06?useSSL=false&chara
cterEncoding=UTF-8&useUnicode=true&serverTimezone=Asia/Seoul
    username:
    password:
    driver-class-name: com.mysql.cj.jdbc.Driver
```

한글 세팅 깨지면 case 2:

클라이언트 설치 후 서버 접속

```
sudo apt update
sudo apt install mysql-client
```

```
user01@Dominica:~$ mysql -h 172.23.84.149 -P 3306 -u root -p --default-character-set=utf8mb4
```