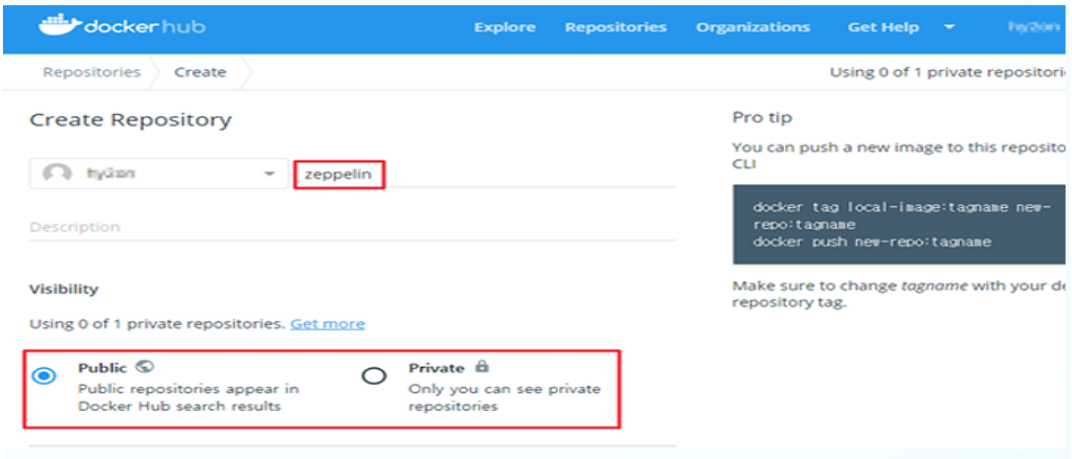
DevOps for SpringBoot

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

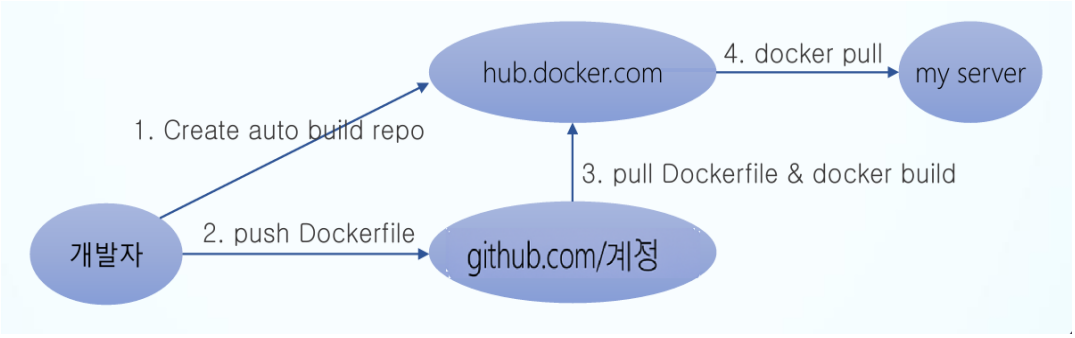
도커허브와 레지스트리

- Docker Hub 는 **공개 이미지 저장소**로, 별도의 Registry 서버 구축 없이 이미지를 저장하고 배포할 수 있다.
- **개인 사용자**는 오직 **1 개의 Private 저장소**만 무료로 운영 가능.
- Public 저장소는 누구나 접근 가능하며, Private 저장소는 권한이 있는 사용자만 접근 가능.

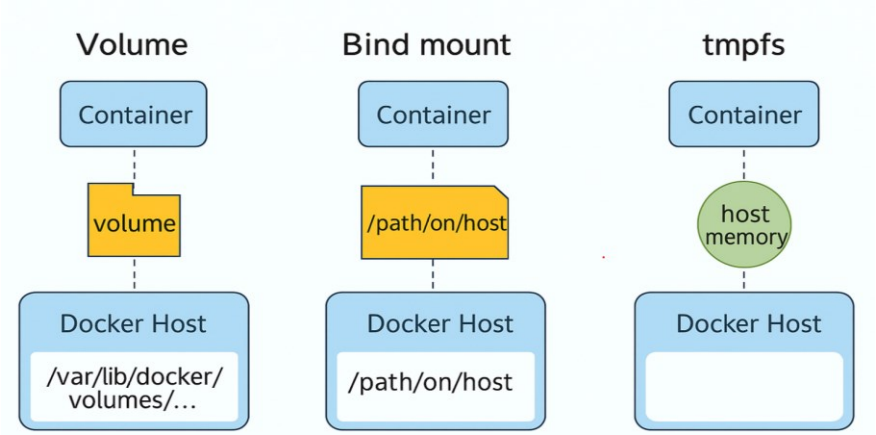


Docker Registry 란?

- Docker Registry 는 도커 이미지 저장소를 포함하는 호스팅 서비스이며, Registry API 를 통해 도커 이미지의 저장과 관리를 가능하게 한다.
- 대표적인 예로 Docker Hub 가 있으며, 이는 가장 널리 사용되는 공식 레지스트리이다.



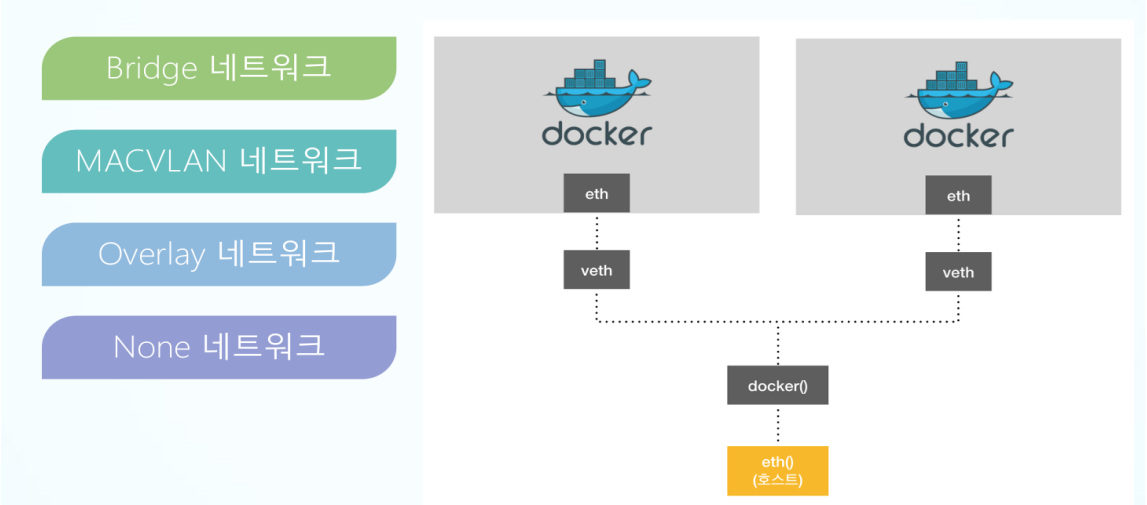
Docker 볼륨과 데이터 관리



구분	설명
1. Volume	- Docker 가 관리하는 특별한 디렉토리에 데이터 저장 (/var/lib/docker/volumes/...) - CLI 또는 Docker Compose 로 쉽게 생성 및 관리 - 백업, 복사, 공유가 용이함
2. Bind Mounts	- 호스트의 특정 디렉토리(예: /home/user/data)를 컨테이너에 연결 - 코드 개발 시 로컬 변경사항을 즉시 반영할 때 유용 - 호스트 의존도가 높고, 관리 어려움
3. tmpfs	- 데이터가 호스트의 메모리에 저장됨 (디스크 X) - 휘발성 데이터 처리에 유리 - 시스템 재부팅 시 데이터 소멸

Docker 네트워크 구축

https://docs.docker.com/network/



드라이버 이름	주요 특징	설명
host	호스트 네트워크 공유	- 컨테이너가 호스트와 네트워크 스택을 공유함 - 별도 네임스페이스 분리가 없음 - 컨테이너가 호스트의 IP 및 포트를 그대로 사용함 docker run --net=host httpd web01
bridge	기본값, 격리된 네트워크	- Docker 기본 네트워크- 컨테이너 간 통신 가능 (같은 브리지 내) ,- p 옵션으로 외부 액세스 가능, NAT 기반 docker network inspect bridge
overlay	다중 호스트 클러스터	- Swarm 환경에서 사용, VXLAN 기술을 통해 컨테이너 간 통신, 다중 호스트 간의 오버레이 네트워크 구성 가능
macvlan	물리 네트워크와 직접 연결	- 컨테이너에 고유의 MAC/IP 주소 부여 - 컨테이너가 물리 네트워크의 일부처럼 동작 - VLAN 트렁킹도 가능, 고급 네트워크 설정에 사용
none	네트워크 미구성 상태	- 네트워크 네임스페이스만 생성됨, 기본 인터페이스 없음 - 완전 격리된 네트워크 (직접 설정 필요)
docker run -d --name br-test nginx # bridge 네트워크로 실행 (기본) docker run -d --network host nginx # host 네트워크로 실행 docker run -d --network none nginx # none 네트워크로 실행		

```
docker network ls  # 1. 컨테이너 네트워크 확인
docker network inspect bridge # 2.  특정  네트워크 확인

user01@Dominica:~$ docker ps -a --format "{{.Names}}"  -> 이름확인

#표로 확인
docker ps -a --format "table {{.Names}}₩t{{.Image}}₩t{{.Status}}"


docker exec -it <container_name> ip addr  #  3. 실행 중 컨테이너의 IP 주소 확인
ip link | grep veth         # 4. veth 인터페이스 확인 (host 기준, Linux)
```

```
docker ps -a --format "{{.Names}}"
docker exec -it mylab01 ip addr
docker ps -a --filter "name=mylab01"
docker start mylab01
docker exec -it mylab01 ip addr
```

https://docs.docker.com/engine/reference/commandline/ps/

Docker Compose?

- **멀티 컨테이너 애플리케이션을 정의하고 실행**하기 위한 도구이다.
- 하나의 YAML 파일에 여러 컨테이너 환경을 정의하고, docker-compose 명령어로 통합적으로 관리할 수 있다.

기본 사용 절차

1. **Dockerfile 정의**: 앱 환경을 구성할 베이스 이미지 및 설정 작성
2. **docker-compose.yml 작성**: 컨테이너 정의, 포트, 볼륨, 네트워크 등 설정
3. **docker-compose 명령 실행**: 애플리케이션 통합 실행 및 관리

Dockerfile

https://docs.docker.com/reference/dockerfile/

명령어	설명	예시	참고 사항
FROM	베이스 이미지 지정 (필수, 가장 먼저 나와야 함)	FROM ubuntu:20.04	Dockerfile 은 반드시 FROM 으로 시작해야 함
RUN	이미지 빌드 시 실행할 명령어	RUN apt-get update	빌드 시 1 회 실행됨. CMD 와 혼동 주의
CMD	컨테이너 시작 시 실행할 기본 명령 (여러 개 중 하나만 사용됨)	CMD ["python3", "app.py"]	docker run 시 별도 명령이 없을 경우 실행됨
WORKDIR	이후 명령에서 사용할 작업 디렉토리 설정	WORKDIR /app	RUN, COPY, CMD 등의 기준 경로가 됨
COPY	호스트에서 컨테이너로 파일/디렉토리 복사	COPY . /app	ADD 보다 단순하며 일반적으로 권장됨
ADD	COPY 와 유사하지만 URL 다운로드, 압축 해제 기능 포함	ADD https://... /file	특별한 경우 아니면 COPY 사용 권장
ENV	환경 변수 설정	ENV PORT=8080	docker run -e 로 override 가능
EXPOSE	컨테이너가 사용하는 포트를 명시	EXPOSE 80	실제 포트 매핑은 docker run -p 필요

LABEL	이미지에 메타데이터(key-value) 추가	LABEL maintainer="user@ex.com"	이미지 정보 명시 용도 (작성자, 버전 등)
USER	명령 실행 시 사용할 사용자 설정	USER appuser	RUN, CMD 등 이후 명령에 영향
VOLUME	컨테이너 외부와 공유할 볼륨 지정	VOLUME /data	영속적 데이터 저장에 유용
STOPSIGNAL	컨테이너 종료 시 사용할 signal 설정	STOPSIGNAL SIGTERM	graceful shutdown 처리 용도
ONBUILD	이 이미지를 기반으로 한 Dockerfile 빌드 시 실행될 명령 지정	ONBUILD COPY . /app/src	상속 구조에서 자식 이미지 빌드시 자동 실행됨

[실 습 01] Docker file 을 실습해보자

Q1) Docker Hub 에서 공개된 이미지 Lab01 이미지 다운로드를 다운로드 받아 확인한다.

```
user01@Dominica:~$ docker pull finish07sds/lab01
```

```
user01@Dominica:~$ docker image ls
```

Q2) 컨테이너 실행하고 파일 내용 확인한다.

```
user01@Dominica:~$ docker run --name lab01-test finish07sds/lab01
Welcome to Docker Lab01 - Work Environment
```

Q3) docker run 으로 임시 컨테이너를 띄워서 내부 파일 확인한다.

컨테이너를 생성해서 **일시적으로 쉘에 접속**하는 명령으로 --rm 옵션은 접속 종료 시 컨테이너를 자동 삭제한다.

```
user01@Dominica:~$ docker run --rm -it finish07sds/lab01 /bin/sh
/ # ls /work
hello.txt
/ # cat /work/hello.txt
Welcome to Docker Lab01 - Work Environment
/ #
```

Q4) 컨테이너 상태 확인 후 삭제한다 -> 종료 된 컨테이너 목록을 확인하고 삭제

```
user01@Dominica:~$ docker  ps  -a
```

```
user01@Dominica:~$ docker rm lab01-test
```

Q5) 다른 이름으로 컨테이너 실행 후 파일 존재 확인한다.

```
user01@Dominica:~$ docker run --name mylab01 -it finish07sds/lab01 /bin/sh
user01@Dominica:~$ docker run --name mylab01 -it finish07sds/lab01 /bin/sh
/ # ls /work
hello.txt
/ # cat /work/hello.txt
Welcome to Docker Lab01 - Work Environment
/ # exit
```

Q6) 컨테이너의 현재 상태를 복사 후 이미지 생성해보자,

```
user01@Dominica:~$ docker commit mylab01 yourdockerid/lab01-copy
user01@Dominica:~$ docker commit mylab01 finish07sds/lab01-copy
sha256:c57db3a0e9246c95ab033fd47375285648bb29ded828a71aff858e5680df85b1
user01@Dominica:~$ docker images
REPOSITORY                      TAG       IMAGE ID       CREATED          SIZE
finish07sds/lab01-copy          latest    c57db3a0e924   13 seconds ago   8.31MB
```

Q7) 복사된 나의 계정 이미지에 한 줄 추가해서 변경해 보자.

```
user01@Dominica:~$ docker commit mylab01 yourdockerid/lab01-copy

#컨테이너 안에서
echo "This is my lab version" >> /work/hello.txt
exit
docker commit editlab yourdockerid/lab01-v2
```

```
user01@Dominica:~$ docker run --name editlab -it finish07sds/lab01 /bin/sh
/ # echo "Edited by student" >> /work/hello.txt
/ # exit
user01@Dominica:~$ docker commit editlab finish07sds/lab01-copy
sha256:024b74a8d4f46dc45e9632249fec93fda4072172c8c7a79a65b23c78b08a3e76
```

Q8) 이미지 목록 확인 과 docker run -it 본인계정/lab01-copy /bin/sh 확인

```
user01@Dominica:~$ docker images
REPOSITORY                      TAG       IMAGE ID       CREATED              SIZE
finish07sds/lab01-copy          latest    024b74a8d4f4   About a minute ago   8.31MB
```

```
/ # tree /work
/work
└── hello.txt

0 directories, 1 files
/ # cat /work/hello.txt
Welcome to Docker Lab01 - Work Environment
Edited by student
```

Q9) Docker Hub 에 업로드해보자.

```
user01@Dominica:~$ docker login
Authenticating with existing credentials... [Username: finish07sds]

i Info → To login with a different account, run 'docker logout' followed by 'docker login'


Login Succeeded
user01@Dominica:~$ docker push finish07sds/lab01-copy
Using default tag: latest
The push refers to repository [docker.io/finish07sds/lab01-copy]
c84eb24bab88: Pushed
fb6d656c01c2: Pushed
fd2758d7a50e: Pushed
latest: digest: sha256:182a67301f65ec2887f6888dda91586a06e2b13116edb20b4a9d875ea40c526a size: 942
```

Docker 컨테이너 CRUD 명령어 (finish07sds/lab01 기준)

단계	목적	명령어 예시	설명
C	생성	docker create --name lab01-container finish07sds/lab01	컨테이너를 생성만 함 (실행 안 함)
	생성+실행	docker run --name lab01-container finish07sds/lab01	컨테이너 생성과 동시에 실행
	임시 실행	docker run --rm -it finish07sds/lab01 /bin/sh	종료 시 자동 삭제되는 임시 컨테이너
R	전체 조회	docker ps -a	모든 컨테이너 목록 확인
	실행 중 조회	docker ps	실행 중인 컨테이너만 확인
	상세 정보	docker inspect lab01-container	컨테이너의 상세 정보 확인
	로그 확인	docker logs lab01-container	표준 출력 로그 확인
U	내부 수정	docker exec -it lab01-container /bin/sh	컨테이너 내부 쉘로 접속하여 수동 변경
	이미지로 저장	docker commit lab01-container finish07sds/lab01-copy	현재 컨테이너 상태를 새 이미지로 저장
D	중지	docker stop lab01-container	실행 중인 컨테이너 중지
	삭제	docker rm lab01-container	컨테이너 완전 삭제
	강제 삭제	docker rm -f lab01-container	실행 중이어도 강제로 삭제
	전체 정리	docker container prune	중지된 모든 컨테이너 일괄 삭제

docker-compose.yml 작성

https://docs.docker.com/compose/

```
version: "3.8"

services:
  lab01:
    container_name: lab01-container
    image: finish07sds/lab01
    restart: unless-stopped
    stdin_open: true
    tty: true
```

항목	설명
version	Compose 파일 버전 (3.8 은 Docker 최신 버전 호환)
services	실행할 컨테이너 목록 (lab01 이라는 이름으로 정의됨)
container_name	생성할 컨테이너 이름 (기본 랜덤 이름 대신 직접 지정)
image	사용할 이미지 이름 (finish07sds/lab01)
restart	재시작 정책: unless-stopped (수동으로 중지할 때까지 재시작)
stdin_open: true + tty: true	컨테이너에 쉘 접속이 가능하도록 설정 (옵션)

```
# docker-compose.yml 이 있는 디렉토리에서 실행
docker-compose up -d

docker-compose down      # 컨테이너 중지 및 삭제
docker-compose down --rmi all  # 이미지까지 삭제
```

[실습 2] docker-compose.yml 실행

1 단계: SpringLab06 설정파일 수정 -> ip 확인 후

application.yml

```yaml
spring:
  datasource:
    url: jdbc:mysql://172.23.84.149:3306/spring_lab06?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=Asia/S
    username: root
    password: admin1234
    driver-class-name: com.mysql.cj.jdbc.Driver
```

```
docker inspect mysql-container | grep IPAddress
```

2 단계 프로젝트 구조 예시 (SpringLab06 + Docker 설정)

```
SpringLab06/
├──── Dockerfile
├──── docker-compose.yml
├──── target/
|     └──── springlab06-0.0.1-SNAPSHOT.jar
└──── src/
```

Dockerfile :	SpringLab06 애플리케이션을 이미지로 빌드할 때 사용하는 **도커 이미지 설정 파일**
docker-compose.yml:	여러 개의 서비스(SPRING + MYSQL 등)를 정의하고 실행하기 위한 **YAML 형식의 설정 파일**

3 단계 _pom.xml ,도커파일 수정

```xml
<build>
  <finalName>springlab06-0.0.1-SNAPSHOT</finalName>
```

Dockerfile

```
FROM openjdk:21
VOLUME /tmp
ARG JAR_FILE=target/springlab06-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

4 단계 _docker-compose.yml

```
version: '3.8'

services:
  mysql:
    image: mysql:8
    container_name: mysql-container
    environment:
      MYSQL_ROOT_PASSWORD: admin1234
      MYSQL_DATABASE: spring_lab06
    ports:
      - "3306:3306"
    volumes:
      - mysql-data:/var/lib/mysql
    networks:
      - spring-net

  springlab06:
    build: .
    container_name: springlab06-app
    ports:
      - "8080:8080"
    depends_on:
      - mysql
    networks:
      - spring-net
   environment:
     SPRING_DATASOURCE_URL: jdbc:mysql://mysql-
container:3306/spring_lab06?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=Asia/Seoul
     SPRING_DATASOURCE_USERNAME: root
     SPRING_DATASOURCE_PASSWORD: admin1234

volumes:
  mysql-data:

networks:
  spring-net:
```

5 단계 install maven

```
user01@Dominica:/$ sudo apt install maven
```

6 단계 : 마운트 이동하고 Maven Wrapper 설정

```
cd /mnt/d/myWork/MySpringBoot/SpringBootLab06

mvn clean package

 cp -r /mnt/d/myWork/MySpringBoot/SpringBootLab06 ~/SpringBootLab06

 cd ~/SpringBootLab06

mvn -N io.takari:maven:wrapper
```

7 단계 : docker-compose 설치

```
sudo apt update
sudo apt install docker-compose
```

```
user01@Dominica:~/SpringBootLab06$ docker compose version
Docker Compose version v2.36.2
```

8 단계 : java 설치

```
sudo apt update
sudo apt install openjdk-21-jdk

sudo update-alternatives --config java  # JAVA_HOME 확인
```

버전확인 , 전역패스 : nano ~/.bashrc -> 파일 맨 아래에 내용 추가

```
export JAVA_HOME=/usr/lib/jvm/java-21-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
```

적용하기

```
user01@Dominica:~/SpringBootLab06$ source ~/.bashrc
```

```
user01@Dominica:~/SpringBootLab06$ javac -version
javac 21.0.7
```

9 단계 : 실행

```
user01@Dominica:~/SpringBootLab06$ sudo ./mvnw clean package
```

```
figuration com.sec01.Sec01Application for test class com.sec01.controller.StudentCourseControllerTest

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.5.0)

2025-06-22T18:12:00.262+09:00  INFO 4219 --- [           main] c.s.c.StudentCourseControllerTest        : Starting Stude
ntCourseControllerTest using Java 21.0.7 with PID 4219 (started by root in /home/user01/SpringBootLab06)
2025-06-22T18:12:00.273+09:00  INFO 4219 --- [           main] c.s.c.StudentCourseControllerTest        : No active prof
ile set, falling back to 1 default profile: "default"
2025-06-22T18:12:01.210+09:00  INFO 4219 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping
Spring Data JPA repositories in DEFAULT mode.
2025-06-22T18:12:01.273+09:00  INFO 4219 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Sprin
g Data repository scanning in 52 ms. Found 2 JPA repository interfaces.
2025-06-22T18:12:01.797+09:00  INFO 4219 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Pro
cessing PersistenceUnitInfo [name: default]
2025-06-22T18:12:01.841+09:00  INFO 4219 --- [           main] org.hibernate.Version                    : HHH000412: Hib
ernate ORM core version 6.6.15.Final
2025-06-22T18:12:01.877+09:00  INFO 4219 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Sec
ond-level cache disabled
2025-06-22T18:12:02.193+09:00  INFO 4219 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWea
ver setup: ignoring JPA class transformer
2025-06-22T18:12:02.229+09:00  INFO 4219 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 -
 Starting...
```

```
user01@Dominica:~/SpringBootLab06$ java -jar target/SpringBootLab06-0.0.1-SNAPSHOT.jar
```

주의!! -> springlab06-0.0.1-SNAPSHOT.jar 파일이 **없기 때문에 실행 안 되는 상황확인후 생성 ./mvnw clean package -DskipTests**

```
user01@Dominica:~/SpringBootLab06$ ls -al target/
total 36
drwxr-xr-x 9 root   root   4096 Jun 22 18:20 .
drwxrwxr-x 8 user01 user01 4096 Jun 22 18:20 ..
drwxr-xr-x 4 root   root   4096 Jun 22 18:20 classes
drwxr-xr-x 3 root   root   4096 Jun 22 18:20 generated-sources
drwxr-xr-x 3 root   root   4096 Jun 22 18:20 generated-test-sources
drwxr-xr-x 3 root   root   4096 Jun 22 18:20 maven-status
drwxr-xr-x 2 root   root   4096 Jun 22 18:23 surefire
drwxr-xr-x 2 root   root   4096 Jun 22 18:23 surefire-reports
drwxr-xr-x 3 root   root   4096 Jun 22 18:20 test-classes
user01@Dominica:~/SpringBootLab06$ sudo ./mvnw clean package -DskipTests
```

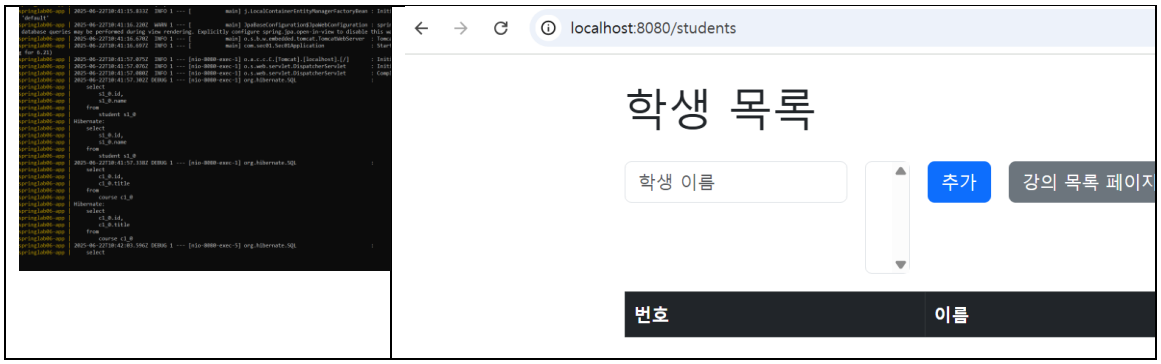
```
user01@Dominica:~/SpringBootLab06$ ls -al target/
total 63068
drwxr-xr-x 8 root   root       4096 Jun 22 18:25 .
drwxrwxr-x 8 user01 user01     4096 Jun 22 18:25 ..
-rw-r--r-- 1 root   root   64525523 Jun 22 18:25 SpringBootLab06-0.0.1-SNAPSHOT.jar
-rw-r--r-- 1 root   root      20416 Jun 22 18:25 SpringBootLab06-0.0.1-SNAPSHOT.jar.original
```

10 단계 : 컨테이너 실행-

```
user01@Dominica:~/SpringBootLab06$ docker rm -f mysql-container
mysql-container
```

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

두개의 컨테이너가 실행됨을 확인



재실행 순서

```
# 1. 전체 컨테이너 종료 및 정리
docker-compose down -v --remove-orphans

# 2. 빌드
./mvnw clean package -DskipTests

# 3. 재시작
docker-compose up --build
```