

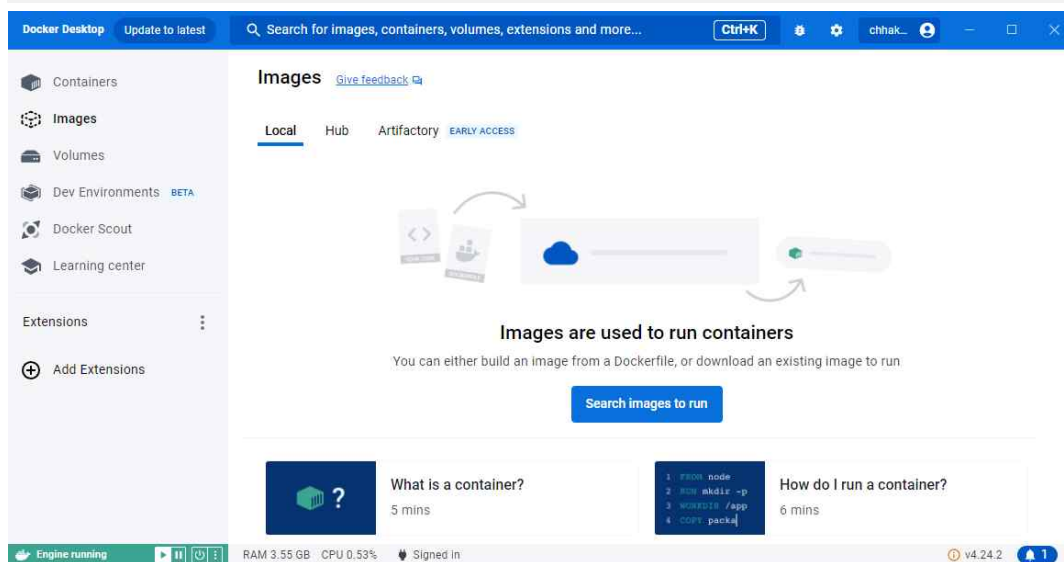
1. Docker 설치 및 기본 명령어

1) Docker 설치

실습1-1 Windows Docker 설치

- 1) <https://www.docker.com/> 이동 후 회원가입
- 2) Docker Desktop installer.exe 다운로드/설치
- 3) Docker Desktop 실행
 - Docker Desktop 실행할 때 'WSL Kernel version too low' 발생하면 PowerShell 실행

PS C:\Users\Wjava> wsl --update 입력



실습1-2 Amazon Linux(EC2) Docker 설치

```
# 패키지 업데이트
$ sudo yum update

# Docker 설치
$ sudo yum install -y docker

# Docker 실행
$ sudo systemctl start docker

# Docker 재실행 등록
$ sudo chkconfig docker on

# Docker 버전 확인
$ docker -v
```

2) Docker 기본 명령어

명령어	설명
\$ docker version \$ docker -v	- 상세버전 - 요약버전
\$ docker search <이미지>	- docker 이미지 검색 - docker hub 확인
\$ docker pull <이미지>:<태그>	- docker 이미지 다운로드 - <태그> 생략시 latest 다운로드
\$ docker inspect <이미지>	- docker 이미지 세부정보 확인
\$ docker create <이미지> \$ docker create --name <컨테이너명> <이미지>	- 이미지로 컨테이너 생성(컨테이너명 랜덤) - 이미지로 컨테이너 생성(컨테이너명 지정)
\$ docker start <컨테이너명 or 해쉬값>	- 컨테이너 실행
\$ docker run <이미지> \$ docker run \ -i \ -t \ -d \ --rm \ --name <컨테이너명> \ -p <외부포트>:<내부포트> \ -v <외부볼륨>:<내부볼륨> \ <이미지>:<태그>	- 컨테이너 생성(create)+실행(start) -i : interactive(표준 입력 활성화) -t : terminal(터미널 모드 실행) -d : detach(컨테이너 백그라운드 실행) -p : port(호스트-컨테이너 포트 연결) -v : volume(호스트-컨테이너 디렉토리 연결) --rm : 프로세스 종료 후 컨테이너 자동 삭제 --name : 컨테이너 이름 지정
\$ docker ps \$ docker ps -a	- 현재 실행 중인 컨테이너 프로세스 확인 - 종료된 컨테이너 프로세스 포함 확인
\$ docker pause <컨테이너명or해쉬값> \$ docker unpause <컨테이너명or해쉬값>	- 컨테이너 일시 중지 - 컨테이너 재개
\$ docker stop <컨테이너명or해쉬값> \$ docker kill <컨테이너명or해쉬값>	- 컨테이너 종료 - 컨테이너 강제 종료
\$ docker container restart <컨테이너명or해쉬값>	- 종료(Exited)된 컨테이너 다시 실행
\$ docker exec -it <컨테이너명or해쉬값> <명령어>	- 실행 중인 컨테이너 명령어 실행
\$ docker rmi <이미지명or해쉬값> \$ docker rmi -f <이미지명or해쉬값> \$ docker rm <컨테이너명or해쉬값> \$ docker rm -f <컨테이너명or해쉬값>	- 이미지 삭제 - 종료된 컨테이너 삭제
\$ docker container prune \$ docker rm \$(docker ps -a -q)	- 종료된 모든 컨테이너 삭제 - 모든 컨테이너 강제 종료 후 모두 삭제

실습2-1 Docker 이미지 검색 및 컨테이너 실행

```
# 이미지 검색
$ docker search hello-world

# 이미지 내려받기
$ docker pull hello-world

# 이미지 확인
$ docker images

# 이미지 세부정보 확인
$ docker inspect

# 이미지로 컨테이너 생성
$ docker create --name hello-world-container hello-world

# 컨테이너 실행
$ docker start hello-world

# 컨테이너 실행
$ docker run hello-world

# 컨테이너 프로세스 확인
$ docker ps
$ docker ps -a

# 컨테이너 다시 실행
$ docker container restart hello-world

# 컨테이너 삭제
$ docker rm hello-world
$ docker container prune
```

실습2-2 Docker Apache Webserver 실습

```
# 이미지 내려받기
$ docker pull httpd

# 이미지로 컨테이너 생성/실행
$ docker run -d --name apache-container -p 8282:80 httpd

# 브라우저 확인
http://localhost:8282
```

실습2-3 Docker MySQL 실습

```
# 이미지 내려받기
$ docker pull mysql

# 이미지로 컨테이너 생성/실행(한줄로 입력 또는 PowerShell 세로쓰기는 backtick)
$ docker run --name mysql-container \
  -e MYSQL_ROOT_PASSWORD=1234 \
  -v C:\mysql\data:/var/lib/mysql \
  -d -p 3307:3306 \
  mysql

# 컨테이너 bash 실행
$ docker exec -it mysql-container bash

# MySQL 접속/데이터베이스 생성/사용자 생성
bash-4.4# mysql -u root -p
Enter password:

mysql> show databases;
mysql> create database 'DB명';
mysql> create user '계정명'@'%' identified by '1234';
mysql> grant all privileges on DB명.* to '계정명'@'%;
mysql> flush privileges;

# MySQL 종료 후 Hei안뻘 접속
mysql> exit
```

실습2-4 Docker Spring Application(JAR) 실습

```
# 이미지 내려받기
$ docker pull openjdk:17

# 이미지로 컨테이너 생성/실행(한줄로 입력 또는 PowerShell 세로쓰기는 backtick)
$ docker run --name spring-hello-container \
  -p 8080:8080 \
  -v C:\Users\chhak\Desktop\Workspace\hello\build\libs:/home/app \
  openjdk:17 \
  java -jar /home/app/hello-0.0.1-SNAPSHOT.jar

# 컨테이너 프로세스 실행
$ docker ps
```

3) Docker Commit과 Push

명령어	설명
\$ docker commit <컨테이너> <계정>/<이미지>:<태그>	<ul style="list-style-type: none"> - 컨테이너를 새로운 이미지로 저장 - 도커허브 계정명 반드시 포함 - 태그는 보통 버전 표기
\$ docker push <계정>/<이미지>:<태그>	<ul style="list-style-type: none"> - Docker 이미지 Registry 등록 - Docker Hub 계정명 반드시 포함

실습3-1 Docker Commit 실습

```
# apache 컨테이너 실행
$ docker start apache-container

# 컨테이너 bash 환경 접속
$ docker exec -it apache-container bash

# 컨테이너 bash 업데이트
root@f5829c461298:~# apt-get update

# 컨테이너 bash vim 설치
root@f5829c461298:~# apt-get install vim

# apache 홈 디렉터리 이동
root@f5829c461298:~# cd /usr/local/apache2/htdocs/

# index.html 수정/저장/브라우저 확인
root@f5829c461298:~# vi index.html
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>hello</title>
</head>
<body>
  <h1>Hello Docker Apache WebServer</h1>
</body>
</html>

# 컨테이너 bash 종료
root@f5829c461298:~# exit

# 컨테이너 커밋(반드시 도커허브 계정명 포함)
$ docker commit apache-container 계정명/my-apache:0.1
```

실습3-2 Docker Push 실습

```
# 도커허브(https://hub.docker.com) 로그인
$ docker login
Authenticating with existing credentials...
Login Succeeded

# 커밋한 이미지 도커허브 올리기, 도커허브(https://hub.docker.com)에서 확인
$ docker push 계정명/my-apache:0.1

# 도커허브(https://hub.docker.com)에서 내려받기
$ docker pull 계정명/my-apache:0.1

# 컨테이너 실행
$ docker run -d -p 8383:80 계정명/my-apache:0.1

# 브라우저 확인
http://localhost:8383
```

2. Dockerfile

주요 지시어	설명
FROM	베이스 이미지 지정
ARG	Dockerfile 변수 지정
RUN	베이스 이미지 위에서 명령어 실행
WORKDIR	컨테이너 내부 작업 디렉토리 설정
COPY	파일 복사
VOLUME	볼륨 마운트(디렉터리) 설정
EXPOSE	외부 포트 설정
CMD	컨테이너 실행 명령어 정의, 일반 인자로 실행 명령 대체 가능
ENTRYPOINT	컨테이너 실행 명령어 정의, 일반 인자로 실행 명령 대체 불가능

실습1-1 Apache Dockerfile 실습

실습 프로젝트 생성

practice1

├─ Dockerfile

└─ <> index.html

Workspace > Docker > practice1 > index.html 생성 후 아래 작성

```
<html>
  <h1>Hello, Docker with Apache!</h1>
</html>
```

Workspace > Docker > practice1 > Dockerfile 생성 후 아래 작성

```
FROM httpd
COPY ./index.html /usr/local/apache2/htdocs
```

빌드 > 실행 > 확인

```
# Dockerfile 빌드(이미지 생성, 마지막 .(점)은 현재 경로 의미)
$ docker build -t practice1-hello-apache:0.0.1 .

# 컨테이너 실행
$ docker run -d --name practice1-container -p 8282:80 practice1-hello-apache:0.0.1

# 브라우저 확인
http://localhost:8282
```

-t : tag, 태그는 이미지의 버전 정보(생략하면 기본은 latest)

실습1-2 Java Dockerfile 실습

☞ 실습 프로젝트 생성

practice2

├─ 📄 Dockerfile

└─ 📄 Hello.java

☞ Workspace > Docker > practice2 > Hello.java 생성 후 아래 작성

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, Docker with Java!");  
    }  
}
```

☞ Workspace > Docker > practice2 > Dockerfile 생성 후 아래 작성

```
# 1) 베이스 이미지 선택 (OpenJDK 17 사용)  
FROM openjdk:17  
  
# 2) 작업 디렉토리 설정 (컨테이너 내부 작업 디렉토리 생성, 이후 명령어는 /app에서 실행)  
WORKDIR /app  
  
# 3) 소스 코드 복사 (/컨테이너/app/Hello.java 복사)  
COPY Hello.java .  
  
# 4) 자바 코드 컴파일 (/컨테이너/app/Hello.class 파일 생성)  
RUN javac Hello.java  
  
# 5) 실행 명령 (exec 형식, 명령어와 인자를 JSON 배열로 표현)  
CMD ["java", "Hello"]
```

☞ 빌드 > 실행 > 확인

```
# Dockerfile 빌드(이미지 생성, 마지막 .(점)은 현재 경로 의미)  
docker build -t practice2-hello-java-test .  
  
# 컨테이너 실행  
docker run --rm practice2-hello-java-test  
  
# 결과 출력  
Hello, Docker with Java!
```

--rm : remove, 실행된 컨테이너가 종료되면 자동으로 컨테이너를 삭제

실습1-3 SpringBoot Jar Dockerfile 실습

☞ 실습 프로젝트 생성

practice3

├─ 🐳 Dockerfile

└─ 📄 hello.jar

✓ Spring Boot hello 프로젝트 jar 파일 복사

☞ Workspace > Docker > practice3 > Dockerfile 생성 후 아래 작성

```
# 1) OpenJDK 17 이미지를 사용하여 JDK 설치
FROM openjdk:17

# 2) 빌드된 JAR 파일을 컨테이너 내로 복사할 디렉토리 설정
WORKDIR /app

# 3) 로컬 시스템에서 JAR 파일을 컨테이너로 복사
COPY ./*.jar app.jar

# 4) Spring Boot 애플리케이션을 실행할 명령 설정
ENTRYPOINT ["java", "-jar", "app.jar"]

# 5) 애플리케이션이 사용할 포트 설정
EXPOSE 8080
```

☞ 빌드 > 실행 > 확인

```
# Dockerfile 빌드(이미지 생성, 마지막 .(점)은 현재 경로 의미)
$ docker build -t practice3-spring-hello-app .

# 컨테이너 실행
$ docker run -p 8080:8080 practice3-spring-hello-app

# 브라우저 확인
http://localhost:8080/hello
```

3. Docker Compose

- docker-compose.yml 파일 기본 구조

기본 구조	주요 속성 설명
<pre>version: "3.8" services: [서비스명]: container_name: [컨테이너명] image: mysql ports: - 3306:3306 volumes: - ./db/mysql/data:/var/lib/mysql environment: MYSQL_ROOT_PASSWORD: 1234 MYSQL_DATABASE: chhak networks: - [네트워크명] [서비스명]: container_name: [컨테이너이름] build: . ports: - 80:8080 depends_on: - [서비스명] networks: - [네트워크명] networks: [네트워크명]: driver: bridge</pre>	<pre>version - docker-compose 버전 규격 - 홈페이지 참고 services - 컨테이너 서비스 설정 - 서비스명 직접 네이밍 image - 도커 이미지 설정 ports - 호스트:컨테이너 포트 설정 volumes - 호스트:컨테이너 공유 디렉터리 설정 environment - 서비스 환경변수 설정 build - Dockerfile 경로 설정 - 현재 디렉터리 . depends_on - 의존 서비스 설정 networks - 서비스간 통신 네트워크 설정 - bridge는 기본 도커 네트워크 유형</pre>

- docker compose 기본 명령어

명령어	설명
\$ docker-compose -v	- 도커 컴포즈 버전 확인
\$ docker-compose config	- 도커 컴포즈 구성 파일 확인
\$ docker-compose up \$ docker-compose up -d	- 도커 컴포즈 실행 - 도커 컴포즈 백그라운드 실행
\$ docker-compose stop	- 컨테이너 중지
\$ docker-compose down	- 컨테이너 중지 및 관련 리소스 제거

실습1-1 MySQL 도커 컴포즈 실습

실습 프로젝트 생성

```
practice4
├── 📁 docker-compose.yml
```

Workspace > Docker > practice4 > docker-compose.yml

```
version: "3.8"
services:
  mysql_service:
    image: mysql
    container_name: mysql_container
    ports:
      - 3307:3306
    environment:
      MYSQL_ROOT_PASSWORD: <비밀번호>
      MYSQL_DATABASE: <DB명>
      MYSQL_USER: <계정명>
      MYSQL_PASSWORD: <비밀번호>
    volumes:
      - ./db/mysql/data:/var/lib/mysql
```

실행

```
# 경로 이동
$ cd ../Workspace/Docker/practice4

# 도커 컴포즈 실행
$ docker-compose up

종료 ctrl + c
```

실습1-2 Spring Boot App과 MySQL 도커 컴포즈 실습

실습 프로젝트 생성

```
practice5
├── 📁 Dockerfile
├── 📁 docker-compose.yml
└── 📄 ch05.jar
```

📁 Workspace > Docker > practice5 > Dockerfile 작성

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY ./*.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

📁 Workspace > Docker > practice5 > docker-compose.yml 작성

```
version: "3.8"
services:
  spring_app_service:
    build: .
    container_name: spring_app_container
    restart: always
    ports:
      - 8080:8080
    depends_on:
      - mysql_service

  mysql_service:
    image: mysql
    container_name: mysql_container
    restart: always
    ports:
      - 3307:3306
    environment:
      MYSQL_ROOT_PASSWORD: <비밀번호>
      MYSQL_DATABASE: <DB명>
      MYSQL_USER: <계정명>
      MYSQL_PASSWORD: <비밀번호>
    volumes:
      - ./db/mysql/data:/var/lib/mysql
```

📁 실행

```
# 경로 이동
$ cd ../Workspace/Docker/practice5

# 도커 컴포즈 실행
$ docker-compose up -d

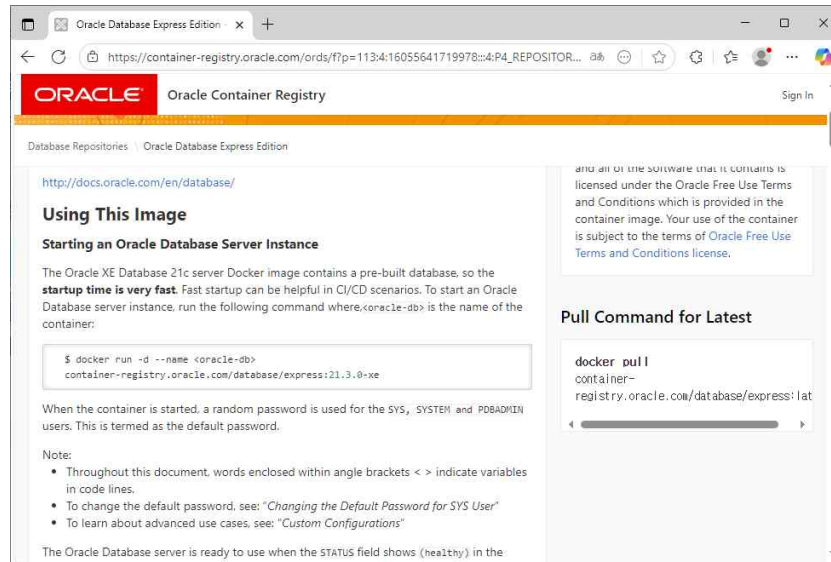
# 도커 컴포즈 프로세스 확인
$ docker-compose ps
```

4. Docker 기반 데이터베이스

1) Oracle XE(Express) 실습

실습1-1 Oracle Container Registry 접속 및 Oracle XE Docker 이미지 다운로드

☞ Oracle 컨테이너 이미지 저장소 접속(<https://container-registry.oracle.com>)



- Oracle XE(Express) Docker 이미지 다운로드 명령어 복사

☞ Oracle XE(Express) Docker 이미지 다운로드

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

# Oracle(XE) Docker 이미지 다운로드 (약 3~5분 정도 시간 걸림)
C:> docker pull container-registry.oracle.com/database/express:latest
latest: Pulling from database/express
753e0fae7e64: Downloading [==>] 32.51MB/696.1MB
2318f8572025: Downloading [=====>] 13.63MB/52.54MB
c6250726c822: Download complete
33ac5ea7f7dd: Downloading [=>] 30.41MB/3.003GB
...
# 다운로드 완료 후 이미지 목록 조회
C:> docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
container-registry.oracle... latest       dcf137aab02d     2 years ago     15.2GB
```

☞ Docker desktop 확인

	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	container-registry.oracle.com/database/express	latest	dcf137aab02d	2 years ago	15.21 GB	

실습1-2 Oracle XE 설치 및 일반 사용자 생성

Windows PowerShell

Copyright (C) Microsoft Corporation. All rights reserved.

Oracle 컨테이너 실행(Oracle XE 설치, 윈도우 PowerShell에서는 `(backtick)`으로 줄바꿈)

```
C:> docker run -d \
>> -p 1522:1521 \
>> -e ORACLE_PWD=1234 \
>> --name oracle-xe-container \
>> --restart=always \
>> container-registry.oracle.com/database/express:latest
```

Oracle 컨테이너 sqlplus 실행

```
C:> docker exec -it oracle-xe-container sqlplus
```

Oracle 최고관리자 로그인

Enter user-name: system

Enter password: 1234

sqlplus 접속 후 사용자 및 버전 확인

SQL> SHOW USER;

SQL> SELECT * FROM V\$VERSION;

일반 사용자 생성을 위한 PDB 전환, XEPDB1(Oracle XE의 기본 PDB)

SQL> ALTER SESSION SET CONTAINER = XEPDB1;

Session altered.

일반 사용자 생성

SQL> CREATE USER <계정명> IDENTIFIED BY <비밀번호>;

User created.

권한 설정







SQL> GRANT CONNECT, RESOURCE, UNLIMITED TABLESPACE TO <계정명>;

Grant succeeded.

종료

SQL> exit

Docker desktop 확인

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU...	Last started	Actions
<input type="checkbox"/>	oracle-xe-container	a34417dacef1	database/express:latest	1522:1521 	0.72%	46 minutes ago	    

실습1-3 Oracle XE 기본 실습

```
# 일반 사용자 로그인, 계정명 뒤에 반드시 생성했던 계정의 PDB 지정(여기서는 XEPDB1)
C:> docker exec -it oracle-xe-container sqlplus
Enter user-name: <계정명>@XEPDB1
Enter password: <비밀번호>

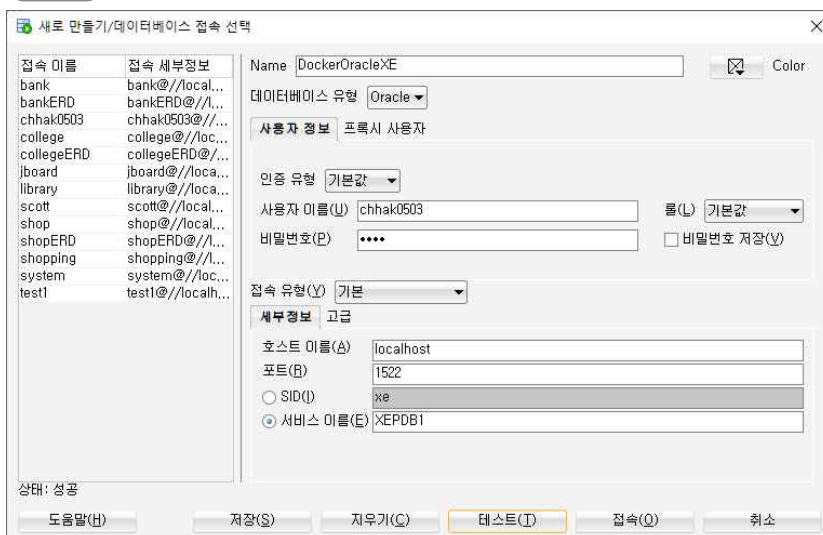
# 테이블 생성
SQL> CREATE TABLE USER1 (
  2  USID      VARCHAR2(20) PRIMARY KEY,
  3  NAME      VARCHAR2(20),
  4  AGE       NUMBER(2)
  5  );
Table created.

# 데이터 삽입
SQL> INSERT INTO USER1 VALUES ('a101', 'James', 21);

# 데이터 조회
SQL> SELECT * FROM USER1;
USID      NAME      AGE
-----
a101      James      21

# 종료(로그아웃)
SQL> exit
```

실습1-4 SQL Developer 접속



- 접속이름 : oracle-xe-container
- 호스트명 : localhost
- 포트번호 : 1522
- 서비스명 : XEPDB1

2) Redis 실습

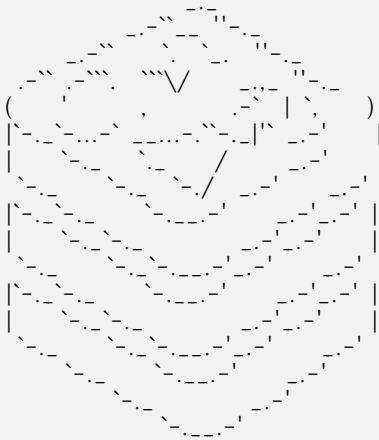
실습2-1 Docker Redis 설치

```
#Redis 이미지 pull
$ docker pull redis

# Redis 컨테이너 실행
$ docker run -d -p 6379:6379 --name=redis-container redis

#Redis 컨테이너 bash 실행
$ docker exec -it redis-container bash

#버전확인
root@265bf85ca76a:/data# redis-server
```



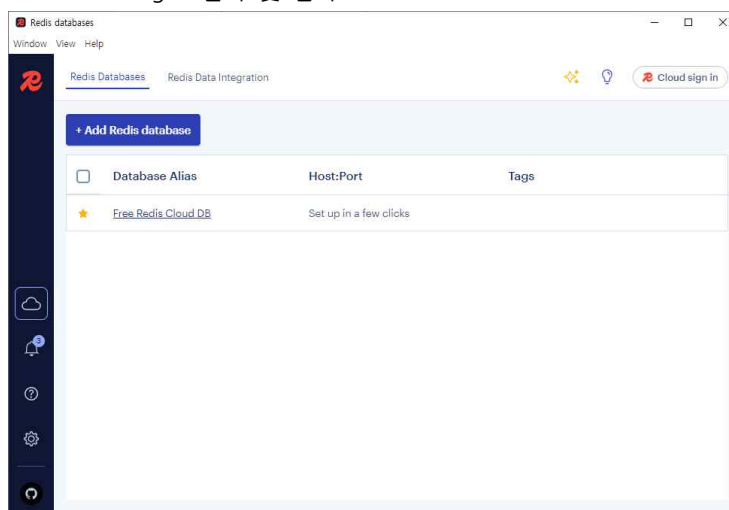
Redis 7.2.3 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 28

<https://redis.io>

```
#Redis-cli 실행
root@802ce898c0c6:/# redis-cli
```

Redis Insight 설치 및 접속



Redis 기본 데이터 구조와 명령어

데이터 구조	명령어	성능	설명
String	set	O(1)	key-value 데이터 저장
	get	O(1)	key 데이터 추출
	append	O(1)	key 문자열 데이터 연결
	incr	O(1)	숫자 증가
	decr	O(1)	숫자 감소
List	lpush	O(1)	left push
	lpop	O(1)	left pop
	rpush	O(1)	right push
	rpop	O(1)	right pop
	lrange	O(S+N)	리스트 데이터를 start(0)부터 end(-1)까지 추출
	lindex	O(N)	리스트 인덱스 값 추출
	llen	O(1)	리스트 길이 추출
Set	sadd	O(N)	set의 key에 member(데이터) 추가
	srem	O(N)	set의 key에 member(데이터) 삭제
	smembers	O(N)	set의 key에 모든 member(데이터) 조회
Sorted Set	zadd	O(log(N))	SortedSet score(가중치)와 값을 저장
	zrange	O(log(N)+M)	SortedSet score(가중치) 기준 오름차순 조회
	zrevrange	O(log(N)+M)	SortedSet score(가중치) 기준 내림차순 조회
Hash	hset	O(1)	해시의 지정된 필드와 값 저장
	hget	O(1)	해시의 지정된 필드의 값 조회
	hgetall	O(N)	해시의 모든 필드와 값 조회

실습2-2 Redis String 명령어 실습

```
127.0.0.1:6379> set my-key 'hello world'
OK
127.0.0.1:6379> get my-key
"hello world"

127.0.0.1:6379> set name 'kim'
OK
127.0.0.1:6379> set name 'Lee'
OK
127.0.0.1:6379> get name
"Lee"

127.0.0.1:6379> set user:a101:name '김유신'
OK
127.0.0.1:6379> set user:a101:age 23
OK
127.0.0.1:6379> set user:a102:name '김춘추'
OK
127.0.0.1:6379> get user:a102:age 21
OK
127.0.0.1:6379> get user:a102:name
"김춘추"

127.0.0.1:6379> set subject Java
OK
127.0.0.1:6379> append subject Jsp
OK
127.0.0.1:6379> append subject "Spring Boot"
OK
127.0.0.1:6379> get subject
"JavaJspSpring Boot"

127.0.0.1:6379> set num 0
OK
127.0.0.1:6379> incr num
(integer) 1
127.0.0.1:6379> incr num
(integer) 2
127.0.0.1:6379> decr num
(integer) 1
127.0.0.1:6379> get num
"1"
```

실습2-3 Redis List 명령어 실습

```
127.0.0.1:6379> lpush my-list1 java
(integer) 1
127.0.0.1:6379> lpush my-list1 jsp
(integer) 2
127.0.0.1:6379> lpush my-list1 spring
(integer) 3
127.0.0.1:6379> lpush my-list1 'spring boot'
(integer) 4
127.0.0.1:6379> lpush my-list1 mysql mongodb redis
(integer) 7
127.0.0.1:6379> lrange my-list1 0 -1
1) "redis"
2) "mongodb"
3) "mysql"
4) "spring boot"
5) "spring"
6) "jsp"
7) "java"

127.0.0.1:6379> lindex my-list1 0
"redis"
127.0.0.1:6379> lindex my-list1 3
"spring boot"

127.0.0.1:6379> rpush my-list2 5 6 7 8 9
(integer) 5
127.0.0.1:6379> lpush my-list2 1 2 3 4
(integer) 9
127.0.0.1:6379> llen my-list2
(integer) 7
127.0.0.1:6379> lrange my-list2 0 -1
1) "4"
2) "3"
3) "2"
4) "1"
5) "5"
6) "6"
7) "7"
8) "8"
9) "9"

127.0.0.1:6379> lpop my-list2
"4"
127.0.0.1:6379> rpop my-list2
"9"
```

실습2-4 Redis Set 명령어 실습

```
127.0.0.1:6379> sadd animal dog
(integer) 1
127.0.0.1:6379> sadd animal cat
(integer) 1
127.0.0.1:6379> sadd animal cow
(integer) 1
127.0.0.1:6379> sadd animal dog
(integer) 0
127.0.0.1:6379> smembers animal
1) "dog"
2) "cat"
3) "cow"
127.0.0.1:6379> srem animal cow
(integer) 1
127.0.0.1:6379> smembers animal
1) "dog"
2) "cat"
```

실습2-5 Redis SortedSet 명령어 실습

```
127.0.0.1:6379> zadd my-sort-set 1 tiger
(integer) 1
127.0.0.1:6379> zadd my-sort-set 2 lion
(integer) 1
127.0.0.1:6379> zadd my-sort-set 5 elephant
(integer) 1
127.0.0.1:6379> zadd my-sort-set 3 eagle
(integer) 1
127.0.0.1:6379> zrange my-sort-set 0 -1
1) "tiger"
2) "lion"
3) "eagle"
4) "elephant"
127.0.0.1:6379> zrange my-sort-set 0 -1 withscores
1) "tiger"
2) "1"
3) "lion"
4) "2"
5) "eagle"
6) "3"
7) "elephant"
8) "5"
127.0.0.1:6379> zrevrange my-sort-set 0 -1 withscores
...
```

실습2-6 Redis Hash 명령어 실습

```
127.0.0.1:6379> hset my-user a101 'kim yu sin'
(integer) 1
127.0.0.1:6379> hset my-user a102 'kim chun chu'
(integer) 1
127.0.0.1:6379> hset my-user a103 'jang bo go'
(integer) 1
127.0.0.1:6379> hset my-user a104 'gang gam chan'
(integer) 1
127.0.0.1:6379> hset my-user a105 'lee sun sin'
(integer) 1
127.0.0.1:6379> hget my-user a103
"jang bo go"
127.0.0.1:6379> hget my-user a105
"lee sun sin"
127.0.0.1:6379> hgetall my-hset1
1) "a101"
2) "kim yu sin"
3) "a102"
4) "kim chun chu"
5) "a103"
6) "jang bo go"
7) "a104"
8) "gang gam chan"
9) "a105"
10) "lee sun sin"

127.0.0.1:6379> hset user1 a101 '{name:"김유신", age:23, addr:"김해"}'
(integer) 1
127.0.0.1:6379> hset user1 a102 '{name:"김춘추", age:21, addr:"경주"}'
(integer) 1
127.0.0.1:6379> hset user1 a103 '{name:"장보고", age:33, addr:"완도"}'
(integer) 1
127.0.0.1:6379> hset user1 a104 '{name:"강감찬", age:43, addr:"서울"}'
(integer) 1
127.0.0.1:6379> hset user1 a105 '{name:"이순신", age:53, addr:"부산"}'
(integer) 1
127.0.0.1:6379> hgetall user1
...

```

Redis 키 관리

명령어	시간복잡도	설명
<code>keys *</code>	$O(N)$	전체 key 확인
<code>flushall</code>	$O(N)$	전체 key 삭제
<code>keys *검색어*</code>	$O(N)$	key 검색
<code>del</code>	$O(1)$	key 삭제
<code>exists</code>	$O(1)$	key 존재 여부 확인
<code>expire</code>	$O(1)$	key 만료시간 설정(초 단위)
<code>ttl</code>	$O(1)$	key 잔여시간 조회(초 단위, Time To Live)
<code>persist</code>	$O(1)$	key 만료시간 해제

실습2-7 Redis 키 관리 실습

```

127.0.0.1:6379> keys *
1) "sort:key"
2) "my-sort-set"
127.0.0.1:6379> keys an*
1) "animal"
127.0.0.1:6379> keys ?u*
1) "subject"
2) "num"
127.0.0.1:6379> del num
(integer) 1
127.0.0.1:6379> flushall
OK
127.0.0.1:6379> set my:key1 'test value1'
OK
127.0.0.1:6379> ttl my:key1
(integer) -1
127.0.0.1:6379> expire my:key1 10
(integer) 1
127.0.0.1:6379> ttl my:key1
(integer) 7
127.0.0.1:6379> exists my:key1
(nil)
127.0.0.1:6379> set my:key2 'test value2'
OK
127.0.0.1:6379> expire my:key2 60
(integer) 1
127.0.0.1:6379> persist my:key2
(integer) 1
127.0.0.1:6379> ttl my:key2
(integer) -1

```