

도커 볼륨을 활용해 데이터 유실 방지하기

1. Docker Volume(도커 볼륨)

✓ 컨테이너가 가진 문제점

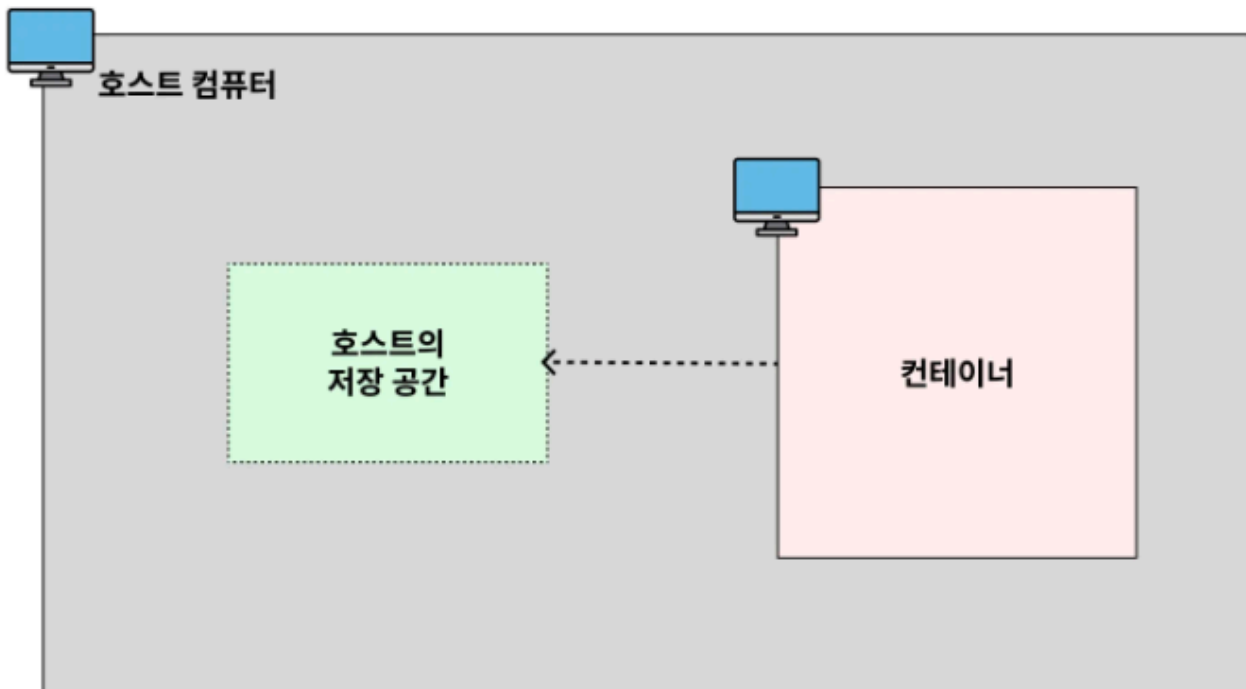
Docker를 활용하면 특정 프로그램을 컨테이너로 띄울 수 있다. 이 프로그램에 기능이 추가되면 새로운 이미지를 만들어서 컨테이너를 실행시켜야 한다. 이 때, Docker는 기존 컨테이너에서 변경된 부분을 수정하지 않고, 새로운 컨테이너를 만들어서 통째로 갈아끼우는 방식으로 교체를 한다. 이게 효율적이라고 생각했던 것이다.

이런 특징 때문에 기존 컨테이너를 새로운 컨테이너로 교체하면, 기존 컨테이너 내부에 있던 데이터도 같이 삭제된다. 만약 이 컨테이너가 MySQL을 실행시키는 컨테이너였다면 MySQL에 저장된 데이터도 같이 삭제 돼버린다.

따라서 컨테이너 내부에 저장된 데이터가 삭제되면 안 되는 경우에는 **볼륨(Volume)**이라는 개념을 활용해야 한다.

✓ Docker Volume(도커 볼륨)이란?

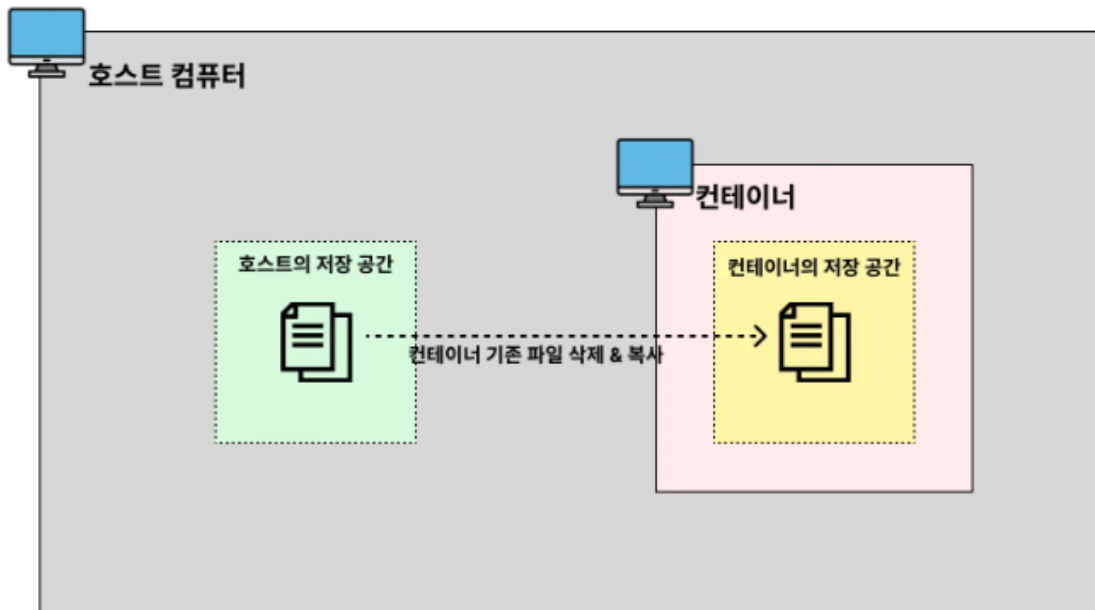
도커의 볼륨(Volume)이란 도커 컨테이너에서 데이터를 영속적으로 저장하기 위한 방법이다. 볼륨(Volume)은 컨테이너 자체의 저장 공간을 사용하지 않고, 호스트 자체의 저장 공간을 공유해서 사용하는 형태이다.



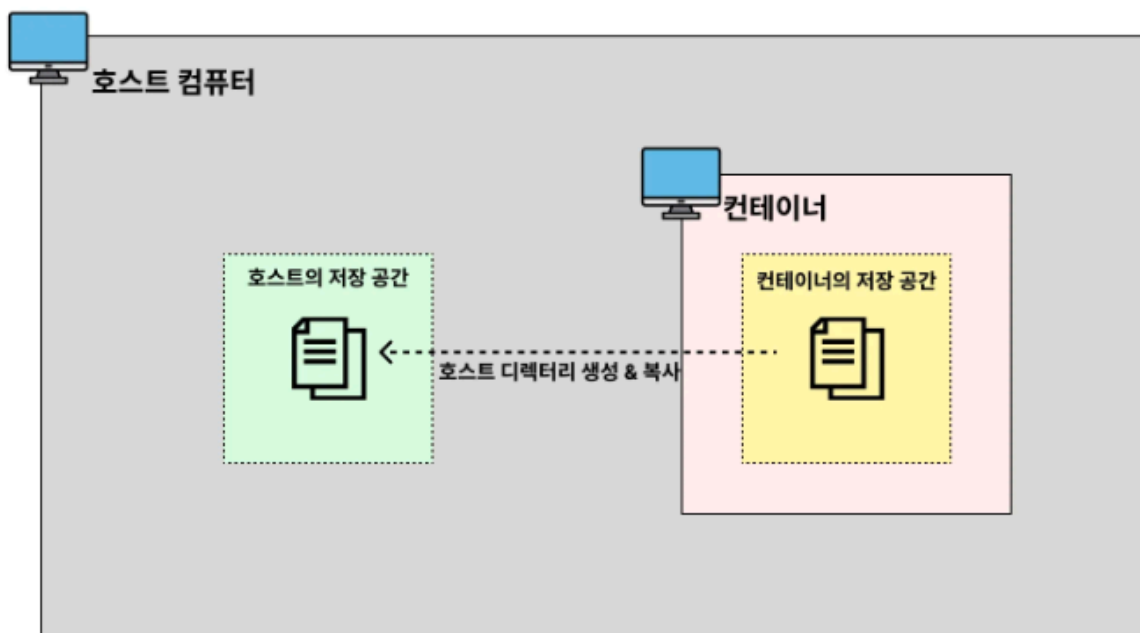
✓ 볼륨(Volume)을 사용하는 명령어

```
$ docker run -v [호스트의 디렉토리 절대경로]:[컨테이너의 디렉토리 절대경로]  
[이미지명]:[태그명]
```

- **[호스트의 디렉토리 절대 경로]**에 디렉토리가 이미 존재할 경우, 호스트의 디렉터리가 컨테이너의 디렉터리를 덮어씌운다.



- **[호스트의 디렉토리 절대 경로]**에 디렉토리가 존재하지 않을 경우, 호스트의 디렉터리 절대 경로에 디렉터리를 새로 만들고 컨테이너의 디렉토리에 있는 파일들을 호스트의 디렉터리로 복사해온다.



[실습] Docker로 MySQL 실행시켜보기 - 1

✓ Docker로 MySQL 실행시켜보기

1. MySQL 이미지를 바탕으로 컨테이너 실행시키기

[mysql - Official Image | Docker Hub](#)

```
$ docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306 -d mysql
```

- 참고) `docker pull` 과정은 생략해도 상관없다. 왜냐하면 `docker run mysql`로 실행시켰을 때, 로컬에 이미지가 없으면 Dockerhub으로부터 MySQL 이미지를 알아서 다운받아서 실행시키기 때문이다.
- `-e MYSQL_ROOT_PASSWORD=password123` : `-e` 옵션은 컨테이너의 환경 변수를 설정하는 옵션이다.
- Dockerhub의 MySQL 공식 문서를 보면 환경 변수로 `MYSQL_ROOT_PASSWORD`를 정해주어야만 정상적으로 컨테이너가 실행된다고 적혀져있다.
- 아래의 명령어로 컨테이너로 들어가서 환경 변수를 직접 눈으로 확인해보자.

```
$ docker exec -it [MySQL 컨테이너 ID] bash  
  
$ echo $MYSQL_ROOT_PASSWORD  
  
# MYSQL_ROOT_PASSWORD라는 환경변수 값 출력  
  
$ export # 설정되어 있는 모든 환경변수 출력
```

```
apple@appleui-MacBookPro ~ % docker exec -it 7488 bash
bash-5.1# echo $MYSQL_ROOT_PASSWORD
password123
bash-5.1# export
declare -x GOSU_VERSION="1.17"
declare -x HOME="/root"
declare -x HOSTNAME="748803928e45"
declare -x MYSQL_MAJOR="innovation"
declare -x MYSQL_ROOT_PASSWORD="password123"
declare -x MYSQL_SHELL_VERSION="9.1.0-1.el9"
declare -x MYSQL_VERSION="9.1.0-1.el9"
declare -x OLDPWD
declare -x PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
declare -x PWD="/"
declare -x SHLVL="1"
declare -x TERM="xterm"
```

2. 컨테이너가 잘 실행되고 있는 지 체크

```
$ docker ps
```

```
apple@appleui-MacBookPro ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS
748803928e45   mysql    "docker-entrypoint.s..." 2 minutes ago Up 2 minutes   0.0.0.0:3306->3306/tcp, 33060/tcp
```

3. 컨테이너 실행시킬 때 에러 없이 잘 실행됐는 지 로그 체크

```
$ docker logs [컨테이너 ID 또는 컨테이너명]
```

```
apple@appleui-MacBookPro ~ % docker logs 7488
2024-11-26 07:55:08+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 9.1.0-1.el9
2024-11-26 07:55:09+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2024-11-26 07:55:09+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 9.1.0-1.el9
2024-11-26 07:55:09+00:00 [Note] [Entrypoint]: Initializing database files
2024-11-26T07:55:09.495289Z 0 [System] [MY-015017] [Server] MySQL Server Initialization - sta
2024-11-26T07:55:09.497067Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 9.1.0) i
of server in progress as process 81
2024-11-26T07:55:09.505504Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started
2024-11-26T07:55:09.947940Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
2024-11-26T07:55:11.285077Z 6 [Warning] [MY-010453] [Server] root@localhost is created with a
sword ! Please consider switching off the --initialize-insecure option.
2024-11-26T07:55:14.160036Z 0 [System] [MY-015018] [Server] MySQL Server Initialization - end
2024-11-26 07:55:14+00:00 [Note] [Entrypoint]: Database files initialized
2024-11-26 07:55:14+00:00 [Note] [Entrypoint]: Starting temporary server
2024-11-26T07:55:14.230327Z 0 [System] [MY-015015] [Server] MySQL Server - start.
2024-11-26T07:55:14.546314Z 0 [System] [MY-010116] [Server] /usr/sbin/mysqld (mysqld 9.1.0) s
process 122
2024-11-26T07:55:14.576120Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started
```

4. DBeaver에도 연결시켜보기

The screenshot shows the 'Connection "localhost" configuration' window in DBeaver. The 'Main' tab is selected, displaying the following settings:

- Connect by:** Host (selected), URL
- URL:** jdbc:mysql://localhost:3306/
- Server Host:** localhost, **Port:** 3306
- Database:** (empty)
- Authentication (Database Native):**
 - Username:** root
 - Password:** (masked with dots), ☒ Save password
- Advanced:**
 - Server Time Zone:** Auto-detect
 - Local Client:** (empty)

At the bottom, there is a 'Test Connection ...' button (highlighted with a red box), 'Cancel', and 'OK' buttons.

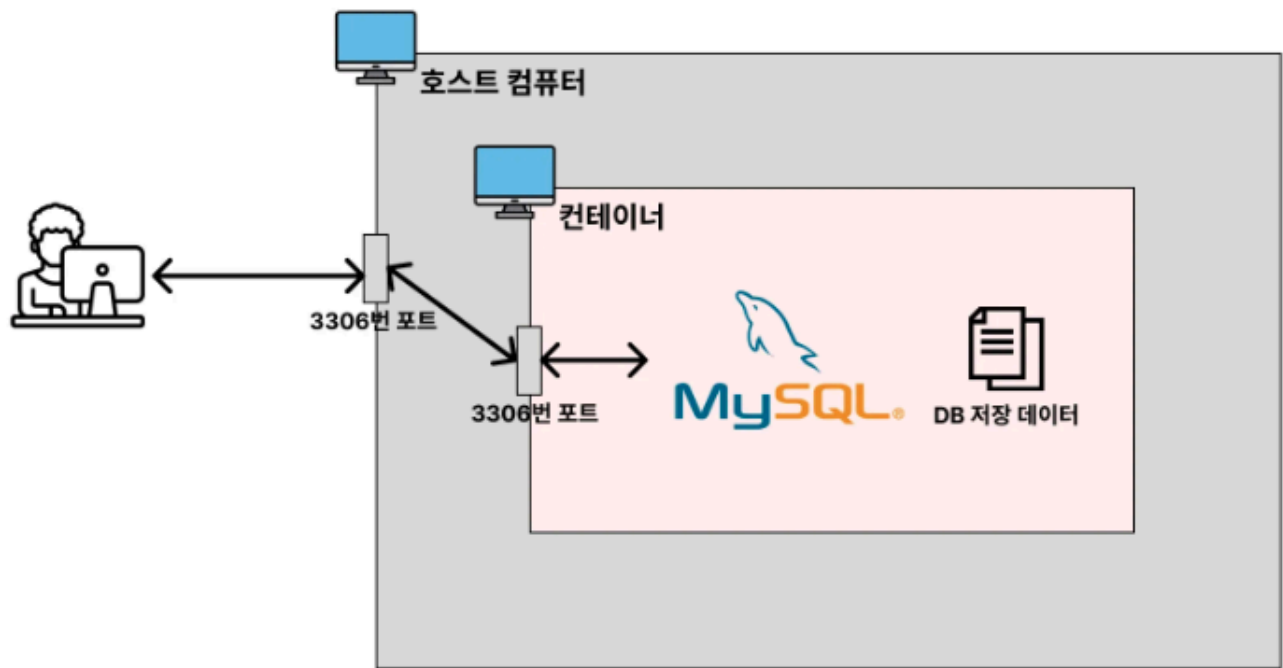
public key retrieval is not allowed 오류 해결 방법

The screenshot shows the 'Connection "localhost" configuration' window in DBeaver, with the 'Driver properties' tab selected. A table lists various driver properties and their values. The 'allowPublicKeyRetrieval' property is highlighted with a red box and labeled 'true로 변경' (change to true).

Name	Value
KeyManagerFactoryProvider	
allowLoadLocalInfile	false
allowLoadLocalInfileInPath	
allowMultiQueries	false
allowNanAndInf	false
allowPublicKeyRetrieval	TRUE
allowReplicaDownConnections	false
allowSourceDownConnections	false
allowUrlInLocalInfile	false
alwaysSendSetIsolation	true
authenticationFidoCallbackHandler	
authenticationPlugins	
authenticationWebAuthnCallbackHandler	
autoClosePstmtStreams	false
autoGenerateTestcaseScript	false
autoReconnect	false
autoReconnectForPools	false
autoSlowLog	true
blobSendChunkSize	1048576
blobZeroStream	false

At the bottom, there is a 'Test Connection ...' button, 'Cancel', and 'OK' buttons.

✓ 그림으로 이해하기



[실습] Docker로 MySQL 실행시켜보기 - 2

✓ MySQL 컨테이너에 직접 접속해보기

1. MySQL 컨테이너에 접속

```
$ docker exec -it [MySQL 컨테이너 ID] bash
```

```
apple@appleui-MacBookPro ~ % docker exec -it 7488 bash  
bash-5.1#
```

2. 컨테이너에서 MySQL에 접근하기

```
$ mysql -u root -p
```

```
bash-5.1# mysql -u root -p  
Enter password:  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 16  
Server version: 9.1.0 MySQL Community Server - GPL  
  
Copyright (c) 2000, 2024, 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>
```

3. MySQL 접근에 성공했다면 데이터베이스 조회해보기

```
mysql> show databases;
```

```
mysql> show database;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
L server version for the right syntax to use near 'database' at line 1
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.01 sec)
```

4. 데이터베이스 만들기

```
mysql> create database mydb;

mysql> show databases;
```

```
mysql> create database mydb;
Query OK, 1 row affected (0.01 sec)

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

5. 컨테이너 종료 후 다시 생성해보기

```
# 컨테이너 종료

$ docker stop [MySQL 컨테이너 ID]

$ docker rm [MySQL 컨테이너 ID]

# 컨테이너 생성
```




```
$ docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306 -d mysql
```

```
$ docker exec -it [MySQL 컨테이너 ID] bash
```

```
$ mysql -u root -p
```

```
mysql> show databases; # 아까 생성한 데이터베이스가 없어진 걸 확인할 수 있다.
```

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

 위 방식은 볼륨(**Volume**)을 활용하지 않고 **MySQL** 컨테이너를 띄웠다. 그래서 **MySQL** 컨테이너를 삭제함과 동시에 **MySQL** 내부에 저장되어 있던 데이터도 함께 삭제되어 없어졌다. 이를 방지하기 위해 볼륨(**Volume**)을 활용해 **MySQL** 컨테이너를 띄우는 방식에 대해 알아볼 것이다.

[실습] Docker로 MySQL 실행시켜보기 - 3

✓ 볼륨(Volume)을 활용해 MySQL 컨테이너 띄우기

1. MySQL 컨테이너 띄우기

```
$ cd c:DevData/  
  
$ mkdir docker-mysql # MySQL 데이터를 저장하고 싶은 폴더 만들기  
  
# docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306 -v {호스트의  
절대경로}/mysql_data:/var/lib/mysql -d mysql  
  
$ docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306 -v  
/Users/apple/DevData/docker-mysql/mysql_data:/var/lib/mysql -d mysql
```

- **pwd** 명령어로 볼륨으로 사용하고자 하는 경로를 확인한 뒤 입력해주자.

윈도우의 경우 아래와 같이 경로가 작성될 수 있다.

```
# 예시  
  
$ docker run -e MYSQL_ROOT_PASSWORD=password123 -p  
3306:3306 -v C:/DevData/docker-mysql:/var/lib/mysql -d mysql
```

```
apple@appleui-MacBookPro docker-mysql % pwd  
/Users/apple/DevData/Docker/docker-mysql  
apple@appleui-MacBookPro docker-mysql % docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306 -v /Users/apple/DevData/Docker/docker-mysql/mysql_data:/var/lib/mysql -d mysql  
bcee892394cb81b9706c3e6ea0cdce409122a3587b7d6a2e33f81dea432c7e8a  
apple@appleui-MacBookPro docker-mysql % docker ps  
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS  
bcee892394cb   mysql     "docker-entrypoint.s..." 32 seconds ago Up 30 seconds  0.0.0.0:3306->3306/tcp, 33060/tcp  
upbeat_carver
```

- 주의) **mysql_data** 디렉토리를 미리 만들어 놓으면 안 된다. 그래야 처음 이미지를 실행시킬 때 **mysql** 내부에 있는 **/var/lib/mysql** 파일들을 호스트 컴퓨터로 공유받을 수 있다. **mysql_data** 디렉토리를 미리 만들어놓을 경우, 기존 컨테이너의 **/var/lib/mysql** 파일들을 전부 삭제한 뒤에 **mysql_data**로 덮어씌워 버린다.
- DB에 관련된 데이터가 저장되는 곳이 **/var/lib/mysql**인지는 **Dockerhub MySQL**의 공식 문서에 나와있다.

Caveats

Where to Store Data

Important note: There are several ways to store data used by applications that run in Docker containers. We encourage users of the `mysql` images to familiarize themselves with the options available, including:

- Let Docker manage the storage of your database data [by writing the database files to disk on the host system using its own internal volume management](#). This is the default and is easy and fairly transparent to the user. The downside is that the files may be hard to locate for tools and applications that run directly on the host system, i.e. outside containers.
- Create a data directory on the host system (outside the container) and [mount this to a directory visible from inside the container](#). This places the database files in a known location on the host system, and makes it easy for tools and applications on the host system to access the files. The downside is that the user needs to make sure that the directory exists, and that e.g. directory permissions and other security mechanisms on the host system are set up correctly.

The Docker documentation is a good starting point for understanding the different storage options and variations, and there are multiple blogs and forum postings that discuss and give advice in this area. We will simply show the basic procedure here for the latter option above:

1. Create a data directory on a suitable volume on your host system, e.g. `/my/own/datadir`.
2. Start your `mysql` container like this:

```
$ docker run --name some-mysql -v /my/own/datadir:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:tag
```

The `-v /my/own/datadir:/var/lib/mysql` part of the command mounts the `/my/own/datadir` directory from the underlying host system as `/var/lib/mysql` inside the container, where MySQL by default will write its data files.

2. MySQL 컨테이너에 접속해서 데이터베이스 만들기

```
$ docker exec -it [MySQL 컨테이너 ID] bash
```

```
$ mysql -u root -p
```

```
mysql> show databases;
```

```
mysql> create database mydb;
```

```
mysql> show databases;
```

```
docker-mysql — com.docker.cli • docker exec -it 4ee bash — 106x27
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.01 sec)

mysql> create database mydb;
Query OK, 1 row affected (0.00 sec)

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

3. 컨테이너 종료 후 다시 생성해보기

```
# 컨테이너 종료

$ docker stop [MySQL 컨테이너 ID]

$ docker rm [MySQL 컨테이너 ID]

# 컨테이너 생성

$ docker run -e MYSQL_ROOT_PASSWORD=password123 -p 3306:3306 -v
/Users/jaeseong/Documents/Develop/docker-mysql/mysql_data:/var/lib/mysql -d mysql

$ docker exec -it [MySQL 컨테이너 ID] bash

$ mysql -u root -p

mysql> show databases; # 아까 생성한 데이터베이스가 그대로 존재하는 걸 확인할 수
있다.
```

```

bash-5.1# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 9.1.0 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

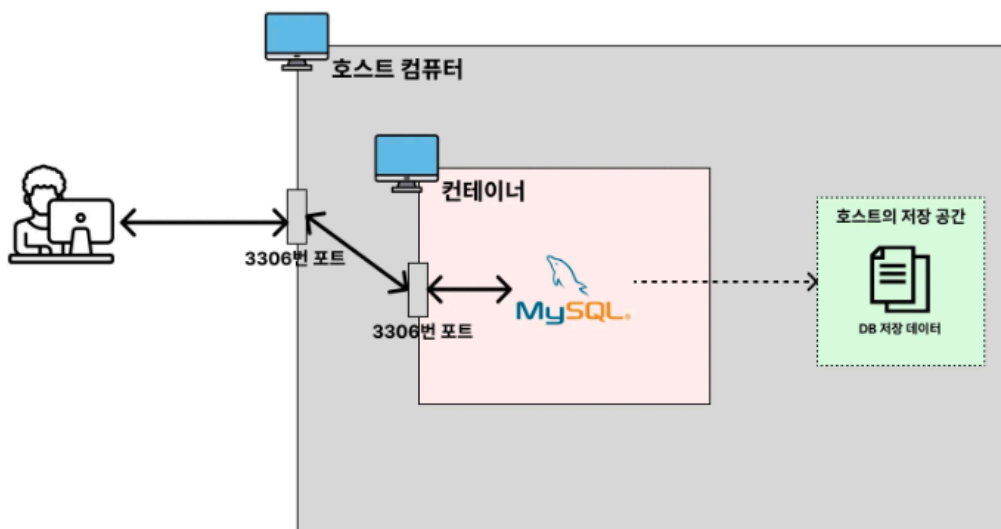
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mydb      |
| mysql     |
| performance_schema |
| sys       |
+-----+
5 rows in set (0.01 sec)

```

✓ 그림으로 이해하기



✓ MySQL 컨테이너 삭제하고 다시 띄워보기

```

# 컨테이너 종료

$ docker stop [MySQL 컨테이너 ID]

$ docker rm [MySQL 컨테이너 ID]

# 비밀번호 바꿔서 컨테이너 생성

```

```
$ docker run -e MYSQL_ROOT_PASSWORD=**pwd1234** -p 3306:3306 -v  
/Users/jaeseong/Documents/Develop/docker-mysql/mysql_data:/var/lib/mysql -d mysql  
  
$ docker exec -it [MySQL 컨테이너 ID] bash  
  
$ mysql -u root -p # 접속이 안 됨...
```

분명 **MYSQL_ROOT_PASSWORD** 값을 바꿔서 새로 컨테이너를 띄웠는데
비밀번호는 바뀌지 않은걸까? 이 부분 때문에 많은 분들이 헤맨다.

그 이유는 **Volume**으로 설정해둔 폴더에 이미 비밀번호 정보가 저장되어버렸기
때문이다.

[실습] Docker로 PostgreSQL 실행시켜보기

✓ Docker로 PostgreSQL 실행시켜보기

1. PostgreSQL 이미지를 바탕으로 컨테이너 실행시키기

[postgres - Official Image | Docker Hub](#)

```
$ cd /Users/DevData  
  
$ mkdir docker-postgresql  
  
$ docker run -e POSTGRES_PASSWORD=password123 -p 5432:5432 -v  
/Users/jaeseong/Documents/Develop/docker-postgresql/postgresql_data:/var/lib/postgresql/data  
-d postgres
```

2. 컨테이너가 잘 실행되고 있는 지 체크

```
$ docker ps
```

3. 컨테이너 실행시킬 때 에러 없이 잘 실행됐는 지 로그 체크

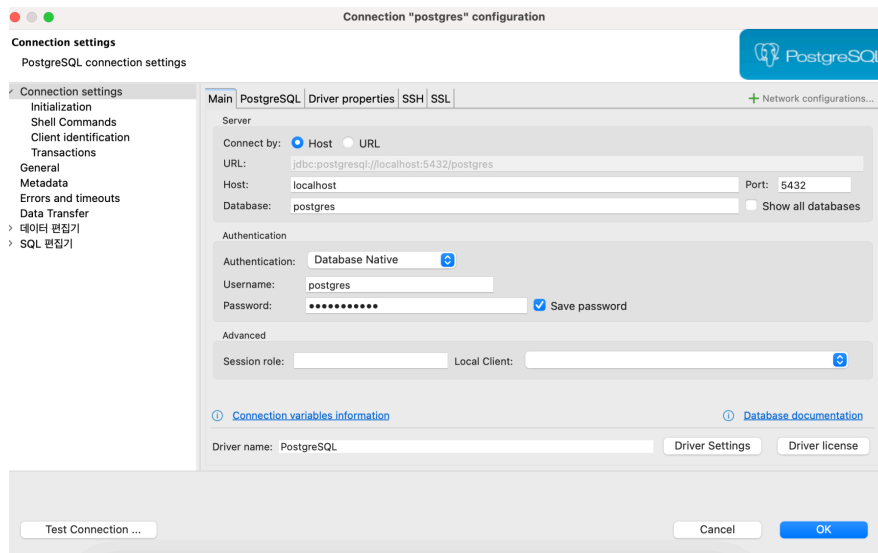
```
$ docker logs [컨테이너 ID 또는 컨테이너명]
```

4. PostgreSQL 컨테이너에 접속

```
$ docker exec -it [컨테이너 ID 또는 컨테이너명] bash
```

```
접속> psql postgres -U postgres
```

5. PostgreSQL 연결해보기(DBBeaver에서 실행하기)



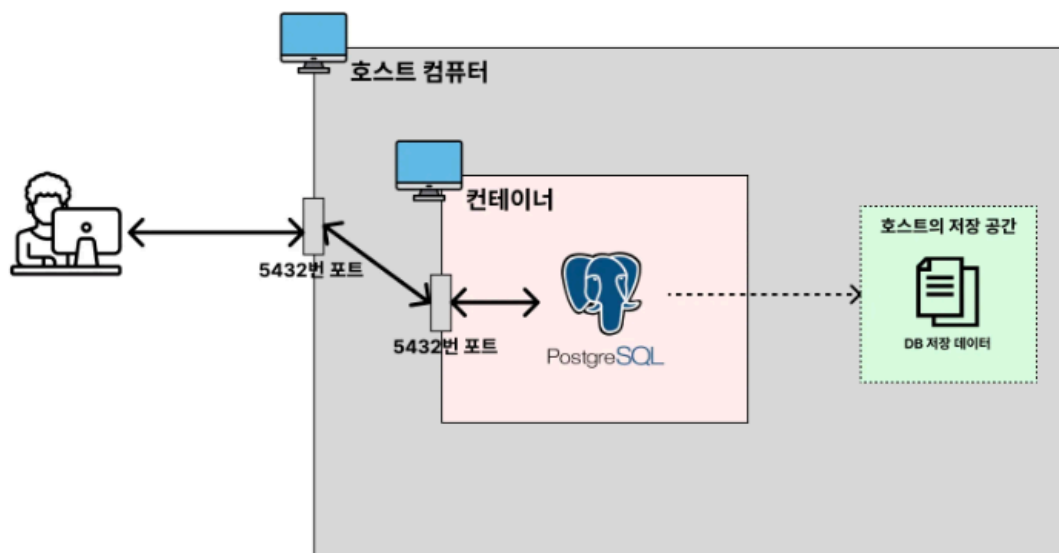
6. PostgreSQL 데이터가 볼륨에 잘 저장되고 있는 지 확인해보기

```
$ cd /Users/jaeseong/Documents/Develop/docker-postgresql/postgresql_data
```

```
$ ls
```

```
apple@appleui-MacBookPro DevData % cd docker-postgresql/postgresql_data
apple@appleui-MacBookPro postgresql_data % ls
PG_VERSION          pg_multixact         pg_tblspc
base                 pg_notify            pg_twophase
global               pg_replslot          pg_wal
pg_commit_ts         pg_serial             pg_xact
pg_dynshmem          pg_snapshots          postgresql.auto.conf
pg_hba.conf          pg_stat               postgresql.conf
pg_ident.conf        pg_stat_tmp           postmaster.opts
pg_logical            pg_subtrans           postmaster.pid
```

✓ 그림으로 이해하기



[실습] Docker로 MongoDB 실행시켜보기

✓ Docker로 MongoDB 실행시켜보기

1. MongoDB 이미지를 바탕으로 컨테이너 실행시키기

[mongo - Official Image | Docker Hub](#)

```
$ cd /Users/jaeseong/Documents/Develop  
  
$ mkdir docker-mongodb  
  
$ docker run -e MONGO_INITDB_ROOT_USERNAME=root -e  
MONGO_INITDB_ROOT_PASSWORD=password123 -p 27017:27017 -v  
/Users/apple/DevData/Docker/docker-mongodb/mongodb_data:/data/db -d mongo
```

2. 컨테이너가 잘 실행되고 있는 지 체크

```
$ docker ps
```

3. 컨테이너 실행시킬 때 에러 없이 잘 실행됐는 지 로그 체크

```
$ docker logs [컨테이너 ID 또는 컨테이너명]
```

4. MongoDB 컨테이너에 접속

```
$ docker exec -it [컨테이너 ID 또는 컨테이너명] bash
```

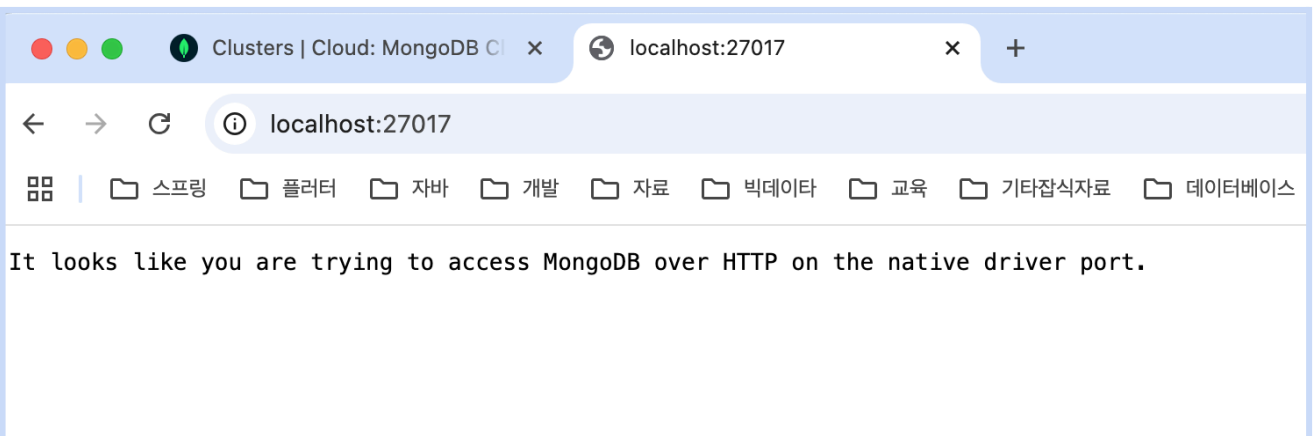
5. 컨테이너에서 MongoDB에 접근하기

```
$ mongosh -u root -p password123
```

```
root@f9b69247af5b:/# mongosh -u root -p password123
Current Mongosh Log ID: 6746ac8eccb9fcf3fbc1c18b
Connecting to:      mongodb://<credentials>@127.0.0.1:27017/?directConnection=true&se
rverSelectionTimeoutMS=2000&appName=mongosh+2.3.3
Using MongoDB:      8.0.3
Using Mongosh:      2.3.3
```

For mongosh info see: <https://www.mongodb.com/docs/mongodb-shell/>

```
-----
The server generated these startup warnings when booting
2024-11-27T04:59:32.352+00:00: For customers running the current memory allocator, we
suggest changing the contents of the following sysfsFile
2024-11-27T04:59:32.353+00:00: For customers running the current memory allocator, we
suggest changing the contents of the following sysfsFile
2024-11-27T04:59:32.353+00:00: We suggest setting the contents of sysfsFile to 0.
2024-11-27T04:59:32.353+00:00: Your system has glibc support for rseq built in, which
is not yet supported by tcmalloc-google and has critical performance implications. Please
set the environment variable GLIBC_TUNABLES=glibc.pthread.rseq=0
2024-11-27T04:59:32.353+00:00: vm.max_map_count is too low
2024-11-27T04:59:32.355+00:00: We suggest setting swappiness to 0 or 1, as swapping ca
n cause performance problems.
-----
```



http://localhost:27017 에 접속했을 때 It looks like you are trying to access MongoDB over HTTP on the native driver port. 라는 문구가 뜨면 MongoDB 서버에 잘 연결된 것이다.

6. MongoDB 인스턴스가 실행 중인지 확인하려면 Hello 명령을 실행합니다.

```
db.runCommand(
  {
    hello: 1
  }
)
```

결과화면

```
{
  isWritablePrimary: true,
  topologyVersion: {
    processId: ObjectId('6746a730fc399faa388fcab5'),
    counter: Long('0')
  },
  maxBsonObjectSize: 16777216,
  maxMessageSizeBytes: 48000000,
  maxWriteBatchSize: 100000,
  localTime: ISODate('2024-11-27T06:22:07.892Z'),
  logicalSessionTimeoutMinutes: 30,
  connectionId: 9,
  minWireVersion: 0,
  maxWireVersion: 25,
  readOnly: false,
  ok: 1
}
```

7. MongoDB 데이터가 볼륨에 잘 저장되고 있는 지 확인해보기

```
$cd /Users/apple/DevData/Docker/docker-mongodb/mongodb_data
```

```
$ ls
```

✅ 그림으로 이해하기

