LAB-04

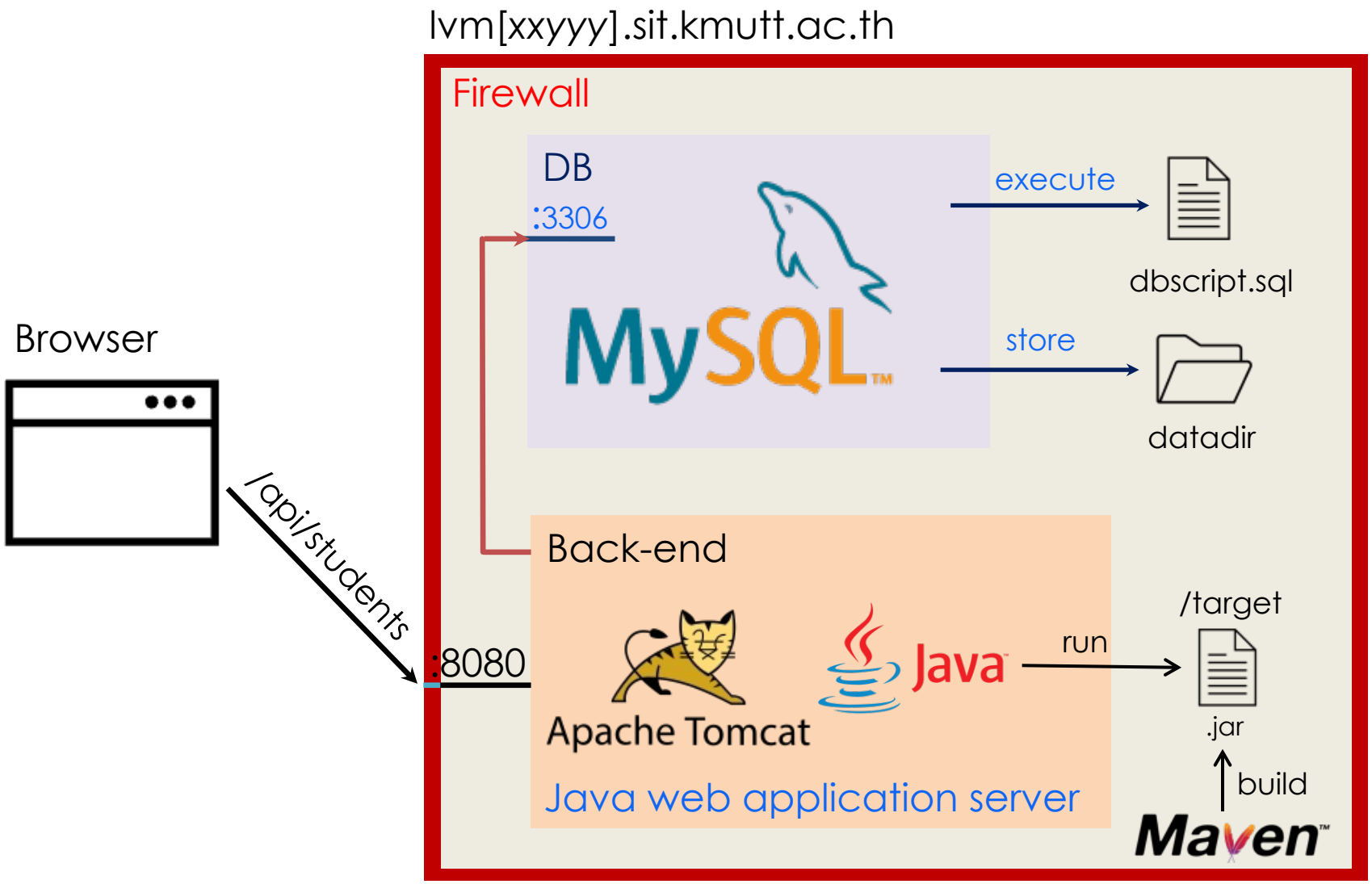
Run DB Container

Olarn Rojanapornpun

Outline

- Run DB container with mysql/mysql-server
- Integrate DB container with API container
- Build DB image
- Use Docker Volume with DB container
- Connecting containers to user-defined network

Architecture



https://github.com/olarnr/int210-studentproj.git

mysql/mysql-server


```
mkdir 209lab4 && cd 209lab4
git clone https://github.com/olarnr/int210-studentproj.git studentproj
cd studentproj/studentdb

# run a mysql-server container, passing user/password/database environment variables,
# bind mount scripts during initialization.
# note the script will run every file in docker-entrypoint-initdb.d in alphabetical order
docker run --name dbserver -d \
--env MYSQL_USER=dev --env MYSQL_PASSWORD=x_eRT2vv4 --env MYSQL_DATABASE=school \
--mount type=bind,src=./scripts/,target=/docker-entrypoint-initdb.d/ \
mysql/mysql-server

# the root user password is generated and can be viewed in logs
docker logs -f dbserver

# the user password is stored in the container!
docker exec dbserver printenv

# check that school database has base records
docker exec -i dbserver mysql -u dev --password=x_eRT2vv4 < selectstudent.sql
```

https://hub.docker.com/r/mysql/mysql-server

mysql/mysql-server


```
# may use variables exported in current shell
cat db-env.sh
export MYSQL_USER=dev
export MYSQL_PASSWORD=x_eRT2vv4
export MYSQL_DATABASE=school
source db-env.sh

# remove previous dbserver container
docker rm -f dbserver
docker run --name dbserver -d \
--env MYSQL_USER --env MYSQL_PASSWORD --env MYSQL_DATABASE \
--mount type=bind,src=./scripts/,target=/docker-entrypoint-initdb.d/ \
mysql/mysql-server

# make sure that the env are set in the container
docker exec dbserver printenv

# check that school database has base records
docker exec -i dbserver mysql -u dev --password=$MYSQL_PASSWORD < selectstudent.sql
```

mysql/mysql-server


```
# may store variables in a file
cat env.list
MYSQL_USER=dev
MYSQL_PASSWORD=x_eRT2vv4
MYSQL_DATABASE=school


# remove previous dbserver container
docker rm -f dbserver

docker run --name dbserver -d \
--env-file env.list \
--mount type=bind,src=./scripts/,target=/docker-entrypoint-initdb.d/ \
mysql/mysql-server

# check that the env are set in the container
docker exec dbserver printenv

# check that school database has base records
# note that we use MYSQL_PASSWORD that was exported previously
docker exec -i dbserver mysql -u dev --password=$MYSQL_PASSWORD < selectstudent.sql
```

Integrate with API


```
cd ~/209lab4/studentproj/studentapi

# set mysql_url variable to dbserver container's IP address
# note that mysql_url variable is configured in application.properties
export mysql_url=$(docker inspect --format='{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' dbserver)

# test that we can access dbserver from $mysql_url
telnet $mysql_url 3306

# build jar package
./mvnw clean package

cat Dockerfile
FROM openjdk:17-jdk-alpine
COPY target/*.jar /api.jar
ENTRYPOINT ["java","-jar","/api.jar"]

# build studentapi image
docker build -t studentproj/studentapi .

# note that mysql_url must be passed to container
docker run --name studentapi -d --env mysql_url -p 8080:8080 studentproj/studentapi

curl localhost:8080/api/students
```

Build dbserver Image

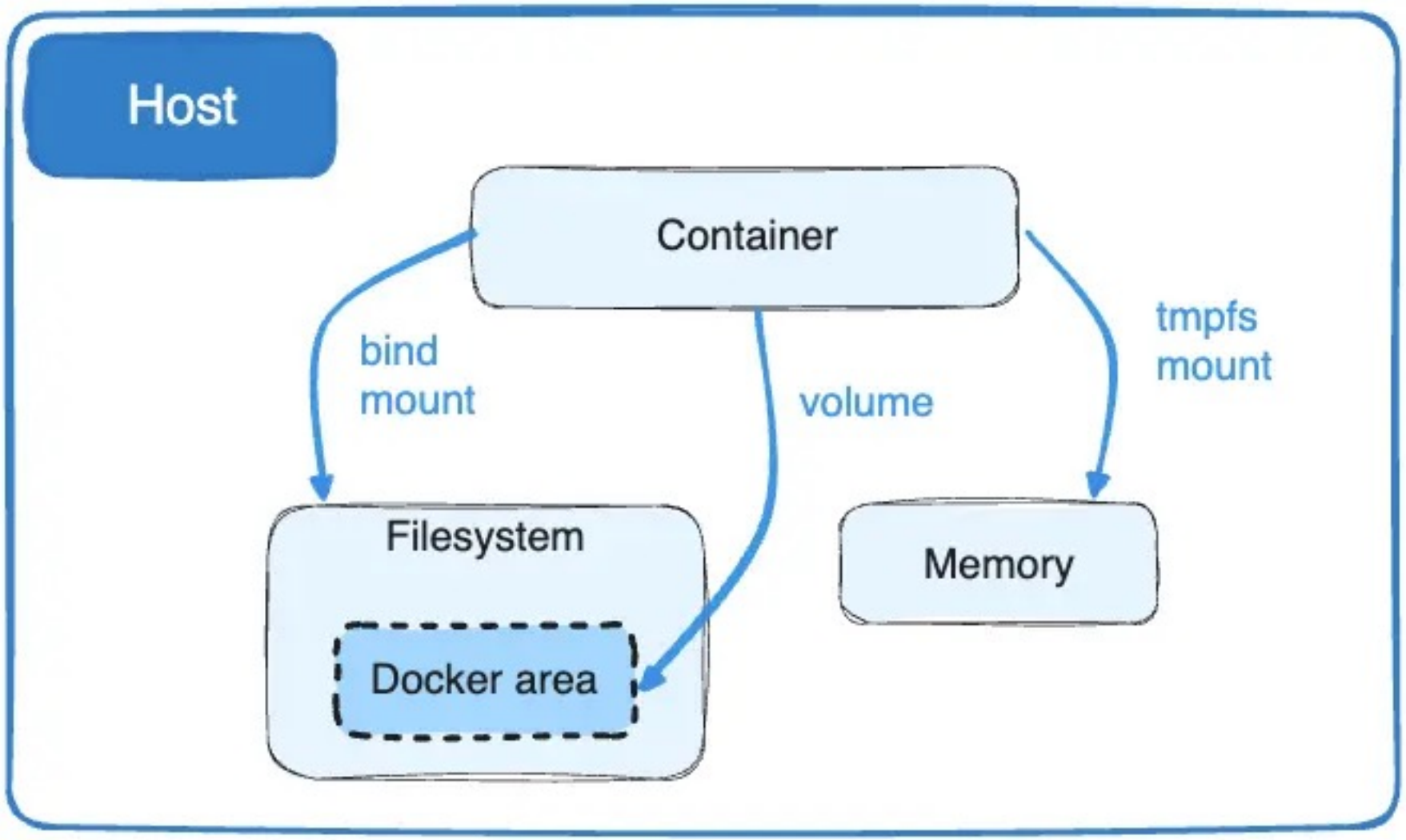

```
cd ~/209lab4/studentproj/studentdb
cat Dockerfile
FROM mysql/mysql-server

ENV MYSQL_RANDOM_ROOT_PASSWORD=true
ENV MYSQL_USER=dev
ENV MYSQL_PASSWORD=x_eRT2vv4
ENV MYSQL_DATABASE=school

COPY scripts/ /docker-entrypoint-initdb.d/

# build the image
docker build -t studentproj/studentdb .

# remove previous dbserver container
docker rm -f dbserver
docker run --name dbserver -d studentproj/studentdb

# test that we can still access dbserver from $mysql_url
telnet $mysql_url 3306

# if the IP address is changed, need to set the new IP address and restart API container
export mysql_url=$(docker inspect --format='{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' dbserver)
docker restart studentapi
curl localhost:8080/api/students
```

Docker Volumes

- Volumes are the preferred mechanism for persisting data generated by and used by Docker containers
- While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker.
- Volumes have several advantages over bind mounts:
  - Volumes are easier to back up or migrate than bind mounts
  - You can manage volumes using Docker CLI commands or the Docker API
  - Volumes work on both Linux and Windows containers
  - Volumes can be more safely shared among multiple containers
  - Volume drivers let you store volumes on remote hosts or cloud providers, encrypt the contents of volumes, or add other functionality
  - New volumes can have their content pre-populated by a container
  - Volumes on Docker Desktop have much higher performance than bind mounts from Mac and Windows hosts.

https://docs.docker.com/storage/volumes/

Docker Volumes



https://docs.docker.com/storage/volumes/#create-and-manage-volumes

Docker Volumes


```
# create a volume
docker volume create my-vol

# list volumes
docker volume ls

# Inspect a volume
docker volume inspect my-vol
[
    {
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
        "Name": "my-vol",
        "Options": {},
        "Scope": "local"
    }
]

# remove a volume
docker volume rm my-vol
```

https://docs.docker.com/storage/volumes/

Mounting a volume


```
cd studentproj/studentdb

# remove previous dbserver container
docker rm -f dbserver

# if a volume does not exist, Docker creates an empty volume
docker run --name dbserver -d \
--mount src=studentdb-vol,target=/var/lib/mysql studentproj/studentdb

# inspect dbserver to verify that Docker created the volume and it mounted correctly.
# Look for the Mounts section:
docker inspect dbserver
```


```
"Mounts": [
            {
                "Type": "volume",
                "Name": "studentdb-vol",
                "Source": "/var/lib/docker/volumes/studentdb-vol/_data",
                "Destination": "/var/lib/mysql",
                "Driver": "local",
                "Mode": "z",
                "RW": true,
                "Propagation": ""
            }
        ],
```


https://docs.docker.com/storage/volumes/

Mounting a volume


```
# remove previous dbserver container
docker rm -f dbserver

# if a volume does exist, Docker uses the existing volume
docker run --name dbserver -d \
--mount src=studentdb-vol,target=/var/lib/mysql studentproj/studentdb

# quickly check that school database has base records
docker exec -i dbserver mysql -u dev --password=x_eRT2vv4 < selectstudent.sql

# if the database exists, mysql does not initialize the database!
# need to remove the volume manually, if you want to initialize the database again
# note that the container is ready much faster (1.12s vs 19.63s)
docker logs -f dbserver
```

Docker Network

- Newly-started containers connect to the default bridge network (also called bridge) unless otherwise specified
- Containers on the default bridge network can only access each other by IP addresses
- On a user-defined bridge network, containers can resolve each other by name or alias

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

student-net Network


```
# view online help
docker network --help

# create studentproj-net network
docker network create studentproj-net

# view list of networks
docker network ls

# view studentproj-net details (containers in studentproj-net)
docker network inspect studentproj-net

# add dbserver to studentproj-net
docker network connect studentproj-net dbserver

# confirm that dbserver is in ‘bridge’ and ‘studentproj-net’
docker network inspect studentproj-net
docker inspect dbserver
```

Docker Network


```
# remove dbserver from ‘bridge’ network
docker network connect studentproj-net dbserver

# remove previous studentapi, unset mysql_url,
# and make sure that dbserver is not specify in /etc/hosts
docker rm -f studentapi
unset mysql_url
nslookup dbserver

cd ~/209lab4/studentproj/studentapi

# run api container inside studentproj-net
docker run -d --name studentapi --network studentproj-net \
-p 8080:8080 studentproj/studentapi

# test the api
curl localhost:8080/api/students
```