

# 와플스튜디오 Spring Seminar

세미나장: 정원식

2023.11.22.(수) 19:00

Week4

# Table of Contents

- Week3 과제 리뷰
- 배포
  - 스프링 profile
  - swagger를 통한 문서화
  - jar 배포와 docker 배포
- Week4 과제

# @Transactional은 스레드 기반으로 동작한다.

@Transactional 현재 스레드에서 실행

```
override fun create(playlistId: Long, userId: Long, at: LocalDateTime): Future<Boolean> {
    return executors.submit<Boolean> { 타 스레드에서 실행
        if (shouldIgnore(playlistId = playlistId, userId = userId, at = at)) {
            return@submit false
        }

        playlistViewRepository.save(
            PlaylistViewEntity(
                playlistId = playlistId,
                userId = userId,
                createdAt = at
            )
        ) ^submit

        playlistRepository.incrementViewCnt(playlistId)

        true ^submit
    }
}
```

타 스레드에서 실행되는  
코드에는 트랜잭션이 적용되지  
않는다.

# @Transacational은 스레드 기반으로 동작한다.

```
override fun create(playlistId: Long, userId: Long, at: LocalDateTime): Future<Boolean> {
    return executors.submit<Boolean> {
        txTemplate.execute { 트랜잭션을 타 스레드에서 열어줌.
            if (shouldIgnore(playlistId = playlistId, userId = userId, at = at)) {
                return@execute false
            }

            playlistViewRepository.save(
                PlaylistViewEntity(
                    playlistId = playlistId,
                    userId = userId,
                    createdAt = at
                )
            )

            playlistRepository.incrementViewCnt(playlistId)

            true ^execute
        }
    }
}
```

# Future로 래핑한다고 비동기로 실행되는 것이 아니다.

```
override operator fun invoke(slowResponse: SlowResponse): Future<Boolean> {
    val log = "[API-RESPONSE] ${slowResponse.method} ${slowResponse.path}, took ${slowResponse.duration}ms, PFCJeong"
    logger.warn(log)

    val slackResponse : ResponseEntity<SlackResponse> = restTemplate.postForEntity<SlackResponse>(
        url: "/api/chat.postMessage",
        HttpEntity(
            mapOf(
                "text" to log,
                "channel" to "#spring-assignment-channel"
            )
        )
    )

    return CompletableFuture.completedFuture( value: slackResponse.body?.ok ?: false)
}
```

클라이언트 요청을 처리하는 스레드가, 슬랙  
통신도 처리하기 때문에 그만큼 응답이 느려진다.

# Future로 래핑한다고 비동기로 실행되는 것이 아니다.

```
override operator fun invoke(slowResponse: SlowResponse): Future<Boolean> {
    return threads.submit<Boolean> { 슬랙 통신은 타 스레드에 위임하면, 바로
        val log =
            "[API-RESPONSE] ${slowResponse.method} ${slowResponse.path}, took ${slowResponse.duration}ms, PFCJeong"
        logger.warn(log)

        val slackResponse : ResponseEntity<SlackResponse> = restTemplate.postForEntity<SlackResponse>(
            url: "/api/chat.postMessage",
            HttpEntity(
                mapOf(
                    "text" to log,
                    "channel" to "#spring-assignment-channel"
                )
            )
        )

        slackResponse.body?.ok ?: false ^submit
    }
}
```

# Job을 순차적으로 get한 경우, 병렬처리되지 않는다.

```
override fun insertAlbums(albumInfos: List<BatchAlbumInfo>) {  
    albumInfos  
        .forEach { albumInfo ->  
            val job : Future<*> = threads.submit { 타 스레드에서 실행  
                txTemplate.executeWithoutResult { insertAlbum(albumInfo) }  
            }  
  
            job.get() 현재 스레드에서 타 스레드에 위임한 작업이  
                    완료되기를 기다림.  
        }  
}
```

Job A 위임 → Job A 완료까지 대기 → Job B 위임 → Job B 완료까지 대기 → Job C…

# Job을 순차적으로 get한 경우, 병렬처리되지 않는다.

```
override fun insertAlbums(albumInfos: List<BatchAlbumInfo>) {  
    val jobs : List<Future<*>!> = albumInfos  
    .map { albumInfo ->  
        threads.submit {  
            txTemplate.executeWithoutResult { insertAlbum(albumInfo) }  
        }  
    }  
  
    jobs.forEach { it.get() } // 일단 모든 작업을 위임하고나서 작업들이 완료될 때까지 대기  
}
```

Job A 위임 → Job B 위임 → Job B → Job C 위임 → Job A,B,C 모두 완료될 때 까지 대기

# HttpServletRequest.attribute, ThreadLocal

```
override fun preHandle(request: HttpServletRequest, response: HttpServletResponse, handler: Any): Boolean {  
    logRequest(Request(method = request.method, path = request.requestURI))  
  
    request.setAttribute(name: "request_at", System.currentTimeMillis())  
  
    return super.preHandle(request, response, handler)  
}
```

# HttpServletRequest.attribute, ThreadLocal

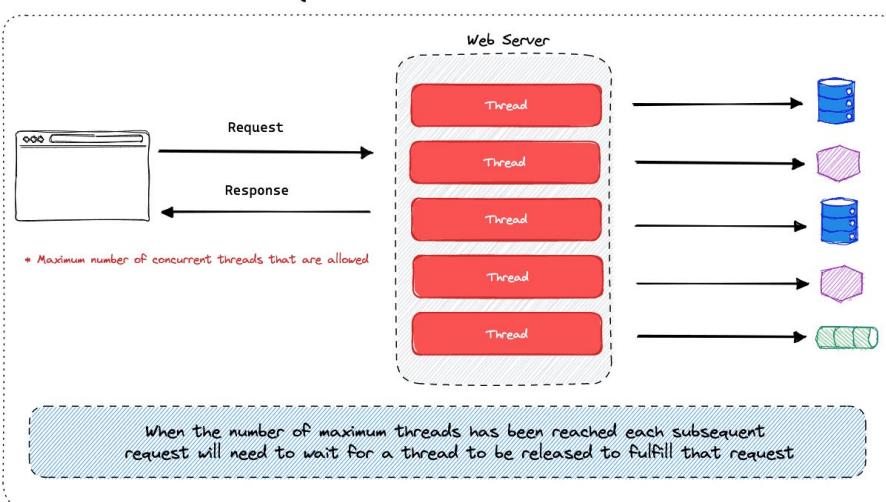
```
private val requestStartAt: ThreadLocal<Long> = ThreadLocal()

override fun preHandle(request: HttpServletRequest, response: HttpServletResponse, handler: Any): Boolean {
    logRequest(Request(method = request.method, path = request.requestURI))

    requestStartAt.set(System.currentTimeMillis())

    return super.preHandle(request, response, handler)
}
```

## Thread Per Request



스프링 MVC에서는 한 개의 스레드가 한 개의 클라이언트 요청을 처리한다.

# 영속성 컨텍스트는 트랜잭션 안에서만 동작한다.

```
/*
 * 주의: @Transactional 사용하지 않기. (데이터 롤백, 레이지 로드 등의 이슈를 피할 수 있도록 구현)
 */
@SpringBootTest
class CustomPlaylistServiceTest @Autowired constructor(
    private val customPlaylistService: CustomPlaylistService,
    private val customPlaylistRepository: CustomPlaylistRepository,
) {
```

```
@Test
fun `커스텀 플레이리스트에 곡 추가시, songCnt가 올라가고 연결 테이블의 row가 생성된다`() {
    val created: CustomPlaylistBrief = customPlaylistService.create(userId = 1L)
```

# 영속성 컨텍스트는 트랜잭션 안에서만 동작한다.

@Test @Transactional이 적용되어 있지 않음.

```
fun `커스텀 플레이리스트에 곡 추가시, songCnt가 올라가고 연결 테이블의 row가 생성된다`() {
    val created :CustomPlaylistBrief = customPlaylistService.create(userId = 1L)

    customPlaylistService.addSong(
        userId = 1L,
        customPlaylistId = created.id,
        songId = 1L
    )

    // songCnt 컬럼 체크
    val customPlaylistEntity :CustomPlaylistEntity = customPlaylistRepository.findById(created.id).get()

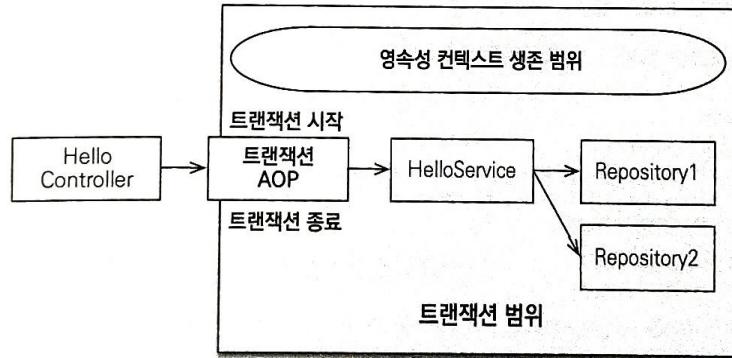
    assertThat(customPlaylistEntity.songCnt).isEqualTo(1)

    // customPlaylistSong 연결 테이블 체크
    val customPlaylist :CustomPlaylist = customPlaylistService.get(userId = 1L, customPlaylistId = created.id)

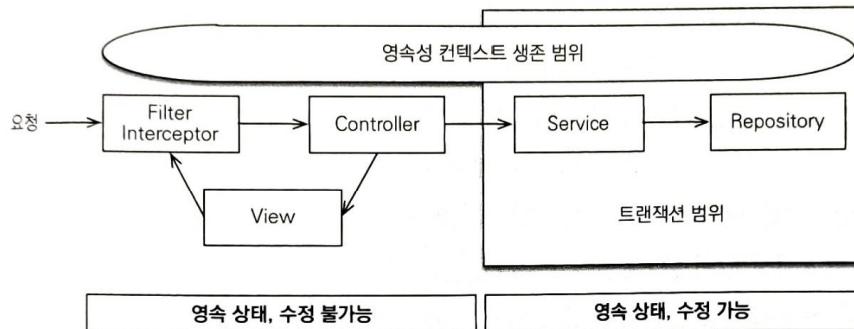
    assertThat(customPlaylist.songs.size).isEqualTo(1)
    assertThat(customPlaylist.songs.first().id).isEqualTo(1L)
}
```

따라서, 커스텀 플레이리스트 조회 시, lazy load를 사용할 수 없다.

# 영속성 컨텍스트는 트랜잭션 안에서만 동작한다.



영속성 컨텍스트는 기본적으로  
트랜잭션 안에서만 동작



OSIV 옵션을 활성화시키면, 트랜잭션  
범위 바깥에서도 동작

그림 13.7 스프링 OSIV – 비즈니스 계층 트랜잭션

# 영속성 컨텍스트는 트랜잭션 안에서만 동작한다.

```
@Query(  
    """  
        SELECT ps FROM custom_playlists ps  
        JOIN FETCH ps.songs pss  
        WHERE ps.id = :id AND ps.userId = :userId  
    """  
)  
fun findByIdWithSongs(id: Long, userId: Long): CustomPlaylistEntity
```

lazy load를 사용할 수 없으므로, Fetch Join으로 해결.

# Spring 프로필

- 같은 코드로 동작하지만, 상황에 따라 다른 속성 값을 적용해주고 싶을 때 사용
    - ex) dev 서버와 prod 서버가 사용하는 데이터베이스 서버가 다를 때
  - 기본적인 profile은 default (local, dev, prod 등등 다양하게 사용)

# Spring 프로필

## 요구 사항

- 로컬에서는 h2 데이터베이스(어플리케이션이 종료되면 데이터 증발)를 사용하고, 배포 환경에서는 mysql 데이터베이스를 사용하고 싶다.
- 로컬에서는 응답이 느릴 때 보내는 슬랙 알람을 비활성화시키고 싶다.
- 로컬 환경에서는 local, 배포 환경에서는 prod라는 프로필명을 사용하고 싶다.

# Spring 프로필

## application.yaml 설정

```
server:
  port: 8080

spring:
  profiles:
    active: local
  config:
    import: classpath:application-db.yaml

cache:
  ttl: 10s
  size: 100
```

spring.profiles.active에 원하는 프로필명을 입력.

yaml도 성격에 맞게 분리하여 관리할 수 있다.

# Spring 프로필

## application-db.yaml 설정

The screenshot shows a Java IDE interface with a project tree on the left and a code editor on the right.

**Project Tree:**

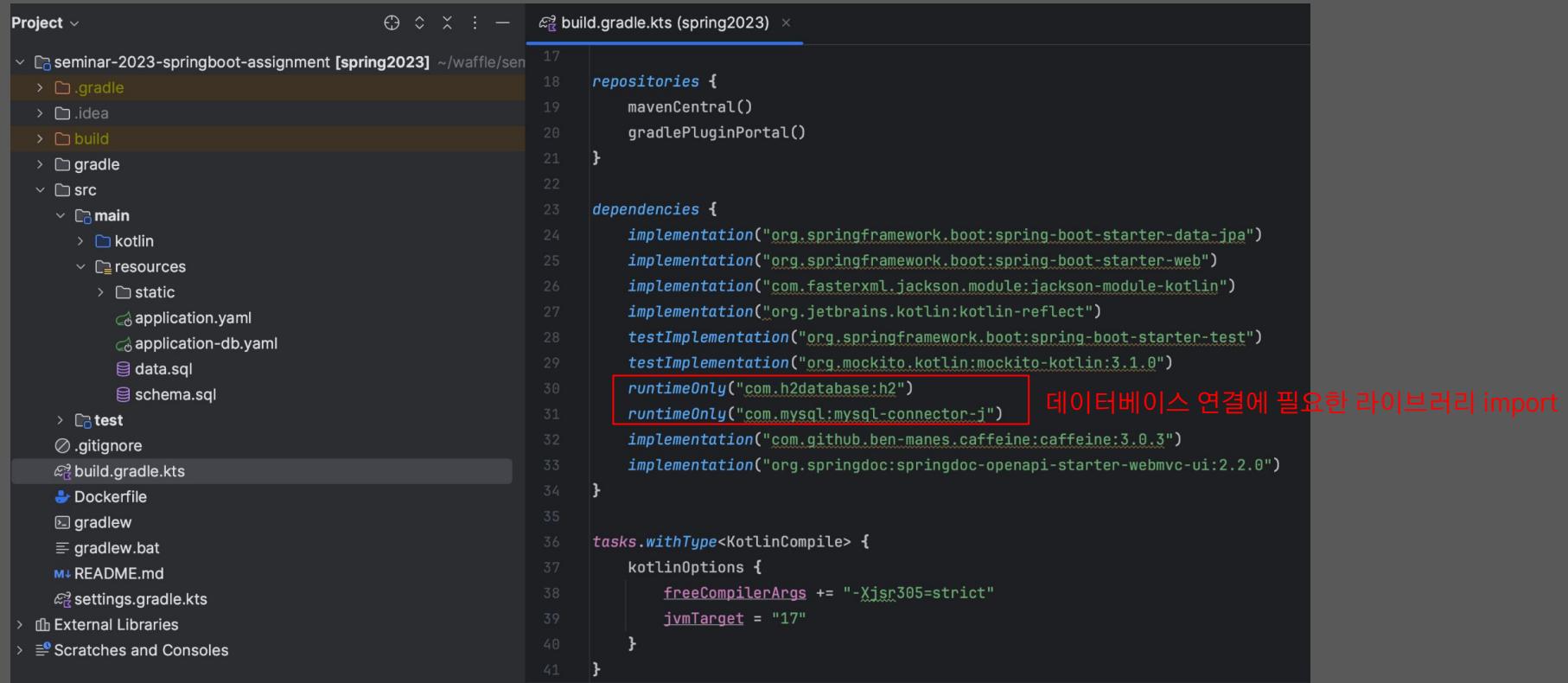
- seminar-2023-springboot-assignment [spring2023] (~/waffle/seminar)
- .gradle
- .idea
- build
- gradle
- src
  - main
    - kotlin
    - resources
      - static
      - application.yaml
      - application-db.yaml
      - data.sql
      - schema.sql
  - test
- .gitignore
- build.gradle.kts
- Dockerfile
- gradlew
- gradlew.bat
- README.md
- settings.gradle.kts
- External Libraries

**Code Editor (application-db.yaml):**

```
spring.config.activate.on-profile: local
  프로필이 local일 때 아래 설정이 적용
  spring:
    h2:
      console:
        enabled: true
        path: /h2-console
    datasource:
      url: jdbc:h2:mem:testdb;MODE=MySQL
      driver-class-name: org.h2.Driver
      username: sa
      password: 1234
    jpa:
      defer-datasource-initialization: true
  ---
spring.config.activate.on-profile: prod
  프로필이 prod일 때 아래 설정이 적용
  spring:
    datasource:
      url: jdbc:mysql://localhost:3306/spring_seminar
      driver-class-name: com.mysql.jdbc.Driver
      username: root
```

# Spring 프로필

## build.gradle.kts



The screenshot shows the IntelliJ IDEA interface with the build.gradle.kts file open in the main editor window. The left sidebar displays the project structure for 'seminar-2023-springboot-assignment [spring2023]'. The build.gradle.kts file contains Groovy-like code defining repositories, dependencies, and tasks. A red box highlights the 'runtimeOnly' imports for H2 and MySQL connectors. A red annotation in Korean is overlaid on the right side of the code, pointing to the MySQL connector import line.

```
repositories {
    mavenCentral()
    gradlePluginPortal()
}

dependencies {
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")
    implementation("org.springframework.boot:spring-boot-starter-web")
    implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
    implementation("org.jetbrains.kotlin:kotlin-reflect")
    testImplementation("org.springframework.boot:spring-boot-starter-test")
    testImplementation("org.mockito.kotlin:mockito-kotlin:3.1.0")
    runtimeOnly("com.h2database:h2")
    runtimeOnly("com.mysql:mysql-connector-j")
    implementation("com.github.ben-manes.caffeine:caffeine:3.0.3")
    implementation("org.springdoc:springdoc-openapi-starter-webmvc-ui:2.2.0")
}

tasks.withType<KotlinCompile> {
    kotlinOptions {
        freeCompilerArgs += "-Xjsr305=strict"
        jvmTarget = "17"
    }
}
```

데이터베이스 연결에 필요한 라이브러리 import

# Spring 프로필

## 프로필별 Bean 생성

```
@Profile("!prod") 프로필이 prod가 아닐 때 빈 생성
```

```
@Component  
class AlertSlowResponseNoOpImpl : AlertSlowResponse {  
  
    override fun invoke(slowResponse: SlowResponse): Future<Boolean> {  
        return CompletableFuture.completedFuture(true) // do nothing  
    }  
}
```

```
@Profile("prod") 프로필이 prod일 때 빈 생성
```

```
@Component  
class AlertSlowResponseImpl(  
    restTemplateBuilder: RestTemplateBuilder,  
) : AlertSlowResponse {  
  
    private val logger = LoggerFactory.getLogger(javaClass)  
    private val threads = Executors.newFixedThreadPool(4)  
    private val restTemplate = restTemplateBuilder  
        .rootUri("https://slack.com")  
        .defaultHeader(name = "Authorization", values = "Bearer xoxb-5766809406786-6098325284464-Jzfs2Dx0fD7DzCpccZhG6EfG")  
        .build()  
  
    override operator fun invoke(slowResponse: SlowResponse): Future<Boolean> {  
        return threads.submit<Boolean> {  
            val log =  
                "[API-RESPONSE] ${slowResponse.method} ${slowResponse.path}, took ${slowResponse.duration}ms, PFCJeong"
```

```
@Component
```

```
class LogInterceptor(
```

```
    private val logRequest: LogRequest,  
    private val logSlowResponse: AlertSlowResponse,
```

```
) : HandlerInterceptor {
```

prod라면, AlertSlowResponseImpl이 주입  
local이라면, AlertSlowResponseNoOpImpl이 주입

# Swagger를 통한 문서화

```
dependencies {  
    implementation("org.springframework.boot:spring-boot-starter-data-jpa")  
    implementation("org.springframework.boot:spring-boot-starter-web")  
    implementation("com.fasterxml.jackson.module:jackson-module-kotlin")  
    implementation("org.jetbrains.kotlin:kotlin-reflect")  
    testImplementation("org.springframework.boot:spring-boot-starter-test")  
    testImplementation("org.mockito.kotlin:mockito-kotlin:3.1.0")  
    runtimeOnly("com.h2database:h2")  
    runtimeOnly("com.mysql:mysql-connector-j")  
    implementation("com.github.ben-manes.caffeine:caffeine:3.0.3")  
    implementation("org.springdoc:springdoc-openapi-starter-webmvc-ui:2.2.0")  
}
```

# Swagger를 통한 문서화

<http://localhost:8080/swagger-ui.html>

The screenshot shows the Swagger UI interface for an OpenAPI definition (v0 OAS 3.0) at the URL <http://localhost:8080/v3/api-docs>. The top navigation bar includes the Swagger logo, a search bar with the path `/v3/api-docs`, and an **Explore** button.

**Servers**:  
http://localhost:8080 - Generated server url

**user-controller**

- POST** /api/v1/signup
- POST** /api/v1/signin
- GET** /api/v1/users/me

**playlist-controller**

- POST** /api/v1/playlists/{id}/likes
- DELETE** /api/v1/playlists/{id}/likes
- GET** /api/v2/playlists/{id}
- GET** /api/v1/playlists/{id}
- GET** /api/v1/playlist-groups

# Swagger를 통한 문서화

<http://localhost:8080/swagger-ui.html>

The screenshot shows the Swagger UI interface for a REST API. The main navigation bar at the top has tabs for 'API', 'Definitions', 'Models', and 'Logs'. Below the navigation, there's a search bar and a dropdown menu for selecting the API version (v1). The main content area displays the 'GET /api/v1/playlist-groups' operation.

**Parameters:**

Name	Description
sort	string (query) DEFAULT

**Responses:**

**Curl:**

```
curl -X 'GET' \
  'http://localhost:8080/api/v1/playlist-groups?sort=DEFAULT' \
  -H 'accept: */*'
```

**Request URL:**

```
http://localhost:8080/api/v1/playlist-groups?sort=DEFAULT
```

**Server response:**

**Code**    **Details**

200    Response body

```
{
  "groups": [
    {
      "id": 1,
      "title": "Spotify 플레이리스트",
      "playlists": [
        {
          "id": 1,
          "title": "Today's Top Hits",
          "uri": "https://api.spotify.com/v1/playlists/1"
        }
      ]
    }
  ]
}
```

# Swagger를 통한 문서화

## ArgumentResolver를 통한 인자 주입 시에 발생하는 이슈

```
@GetMapping("/api/v1/users/me")
fun me(
    @Authenticated user: User,
): UserMeResponse {
    return UserMeResponse(
        username = user.username,
        image = user.image
    )
}
```

커스텀화하여 해결해야 함.

The screenshot shows a Swagger UI interface for a REST API. At the top, it displays a code snippet in Java-like syntax for a controller method named 'me'. This method uses an 'Authenticated' ArgumentResolver to inject a 'User' object into the 'user' parameter of the 'UserMeResponse' return type. A lightbulb icon is present next to the code snippet, likely indicating a tool tip or suggestion.

Below the code snippet, a message in Korean reads "커스텀화하여 해결해야 함." (Needs to be customized and solved).

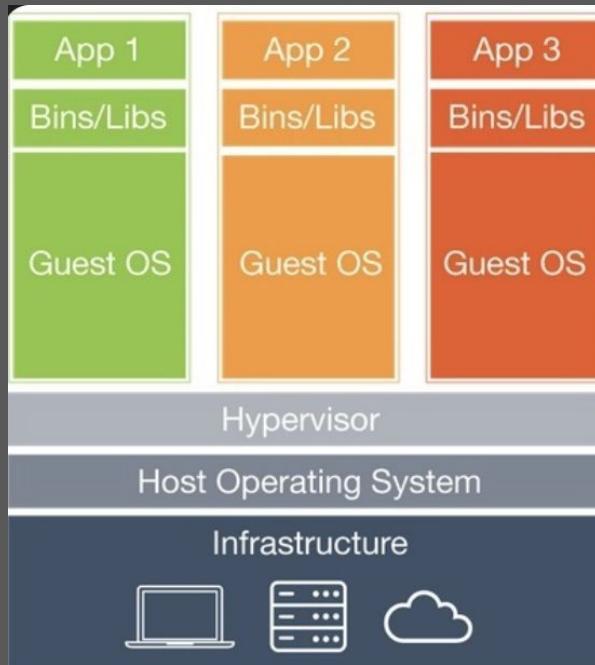
The main part of the screenshot is the Swagger UI interface. It shows a 'GET /api/v1/users/me' endpoint. Under the 'Parameters' section, there is a table with one row. The row contains a column labeled 'Name' with the value 'user' and a column labeled 'Description' with the value 'object (query)'. A red asterisk and the text 'required' are placed next to the 'user' name. Below the table, the parameter schema is shown as a JSON object:

```
{ "id": 0, "username": "string", "image": "string", "accessToken": "string" }
```

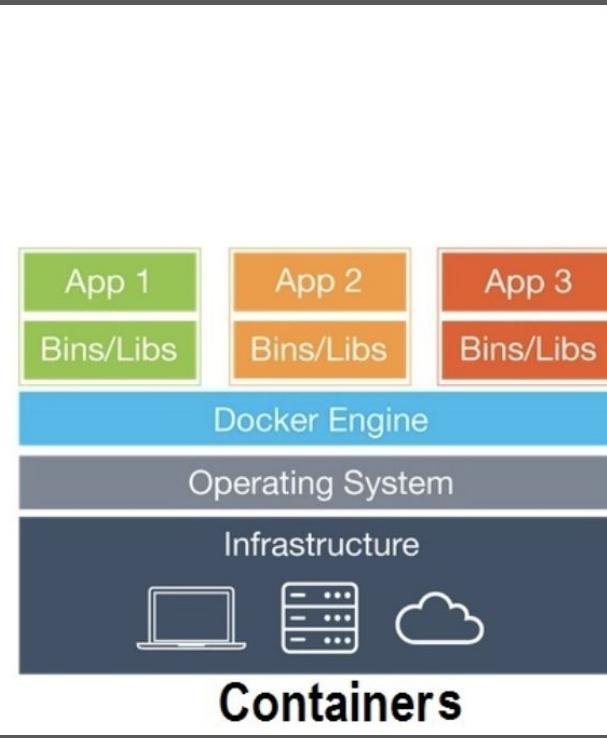
At the bottom of the interface, there is a 'Responses' section. It contains a table with two rows. The first row has columns 'Code' (200) and 'Description' (OK). The second row has columns 'Code' (No links) and 'Description' (No links).

# Jar 배포 / Docker 배포

## VM vs Container



**Virtual Machines**



**Containers**

# Jar 배포 / Docker 배포

## VM vs Container

### ● VM

- hypervisor (VirtualBox,,)

- Guest OS

### ● Container

- 컨테이너 가상화 (Docker,,)

- namespace, cgroup을 통한 격리

- Host OS 공유

	하이퍼바이저 형 가상화	컨테이너 형 가상화
시작 시간	길다 (분)	짧다 (초)
컨테이너 무게	 수 GB ~ 수백 GB OS + 애플리케이션 + 런타임 소프트웨어	 ~ 수백 MB 애플리케이션 + 런타임 소프트웨어
Guest OS	Windows/Linux 등 다양한 선택 가능	호스트 OS와 동일한 OS
이식성	대부분 가상 이미지에 대한 변화가 필요	컨테이너 이미지 그대로 사용 가능
데이터 관리	VM 내부 또는 연결된 스토리지에 저장	컨테이너 내부에 있는 데이터는 종료시 소멸되며, 필요에 따라 스토리지를 이용하여 저장
Guest OS 와 관계	Guest OS는 하드웨어(가상)로 인식	Host OS를 커널 수준으로 분리하여 OS를 가상화 형태로 사용, 필요에 따라 호스트와 리소스 공유 가능
시스템 성능	각 가상 머신마다 전용 운영 체제가 있기 때문에 가상 머신에 구축된 애플리케이션을 실행할 때 메모리 사용량이 필요 이상으로 많아져 가상 머신이 호스트에 필요 한 리소스를 모두 사용할 수 있다.	컨테이너화된 애플리케이션은 원전한 가상 머신보다 리소스를 더 적게 사용하고 호스트 메모리에 가해지는 부담을 줄일 수 있도록 운영 체제 환경(커널)을 공유한다.
유지관리와 업데이트	 운영 체제를 업데이트하거나 패치할 경우 기존 컴퓨터를 하나씩 업데이트해야 하고 각 게스트 OS를 별개적으로 패치해야 한다.	 컨테이너 호스트(컨테이너를 호스트하는 컴퓨터)의 운영 체제만 업데이트하면 됩니다. 따라서 유지관리가 매우 간소화된다.

# Jar 배포

## jar 파일 빌드

# Jar 배포

## AWS ec2

▼ Application and OS Images (Amazon Machine Image) 정보

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Q 수천 개의 애플리케이션 및 OS 이미지를 포함하는 전체 카탈로그 검색

Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux 더 많은 AMI 찾아보기 AWS, Marketplace 및 커뮤니티의 AMI 포함

Amazon Machine Image(AMI)

Amazon Linux 2023 AMI

ami-01123b84e2a4fba05 (64비트(x86)) / ami-0c68ab5091e5f073a (64비트(Arm))  
가상화: hvm ENA enabled: true 루트 디바이스 유형: ebs

설명 AWS EC2는 VM입니다.

Amazon Linux 2023 AMI 2023.2.20231113.0 x86\_64 HVM kernel-6.1

아키텍처 64비트(x86) AMI ID ami-01123b84e2a4fba05 확인된 공급 업체

# Jar 배포

## ssh로 EC2에서 jar 배포

```
chujonghyun@chujonghyeon-ui-MacBookPro:~/Desktop/sshkey chmod 400 sshKey.pem  
chujonghyun@chujonghyeon-ui-MacBookPro:~/Desktop/sshkey ssh -i "sshKey.pem" ubuntu@ec2-13-125-18-214.ap-northeast-2.compute.amazonaws.com  
The authenticity of host 'ec2-13-125-18-214.ap-northeast-2.compute.amazonaws.com (13.125.18.214)' can't be established.  
ED25519 key fingerprint is SHA256:eNKWcUiDNvVP7dyIx2S/dZ+jdI/ymlmaVqBxB/wfIDE.  
This key is not known by any other names  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
```

**ubuntu@ip-172-31-2-94:~\$ sudo apt-get install git**

Reading package lists... Done

Building dependency tree... Done

Reading state information... Done

git is already the newest version (1:2.34.1-1ubuntu1.4).

git set to manually installed.

0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

**ubuntu@ip-172-31-2-94:~\$ git --version**

git version 2.34.1

**ubuntu@ip-172-31-2-94:~\$**

1. ssh로 ec2 접근
2. git pull
3. ./gradle bootJar
4. java -jar ..

# Docker 배포

## docker 빌드

The screenshot shows a Java project structure in an IDE. The left pane displays the project tree:

- build
- classes
- kotlin
- libs (selected)
- reports
- resources
- snapshot
- test-results
- tmp
- resolvedMainClassName
- gradle
- src
- .gitignore
- build.gradle.kts
- Dockerfile (selected)
- gradlew

The right pane shows the content of the selected Dockerfile:

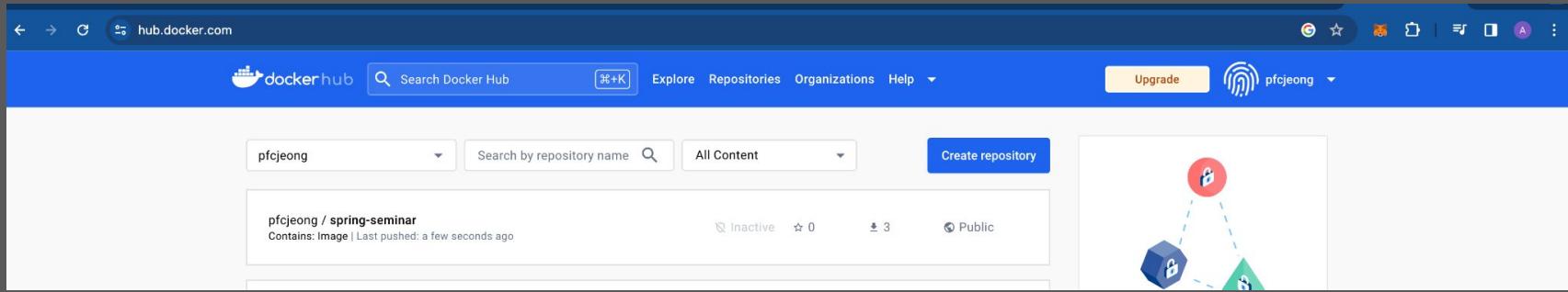
```
1 FROM openjdk:17-jdk-slim
2
3 ARG JAR_FILE=/build/libs/*.jar
4 COPY ${JAR_FILE} app.jar
5 ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Below the IDE window, a terminal window shows the command and output of the docker build process:

```
→ seminar-2023-springboot-assignment git:(master) ✘ docker build -t pfcjeong/spring-seminar .
[+] Building 1.5s (7/7) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 73B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/openjdk:17-jdk-slim
=> [internal] load build context
=> => transferring context: 230B
=> [1/2] FROM docker.io/library/openjdk:17-jdk-slim@sha256:aaa3b3cb27e3e520b8f116863d0580c438ed55ecfa0bc126b41f68c3f62f9774
```

# Docker 배포

## docker 푸시 (도커 허브)



```
→ seminar-2023-springboot-assignment git:(master) docker push pfcjeong/spring-seminar
Using default tag: latest
The push refers to repository [docker.io/pfcjeong/spring-seminar]
f8faefcca4b3: Layer already exists
6be690267e47: Layer already exists
13a34b6fff78: Layer already exists
9c1b6dd6c1e6: Layer already exists
latest: digest: sha256:25fdf232fbbae9163ed3d2af0566ad70a22608d1ddaeaf8cb76bde4113f4f702 size: 1165
```

# Docker 배포

## docker 이미지 실행

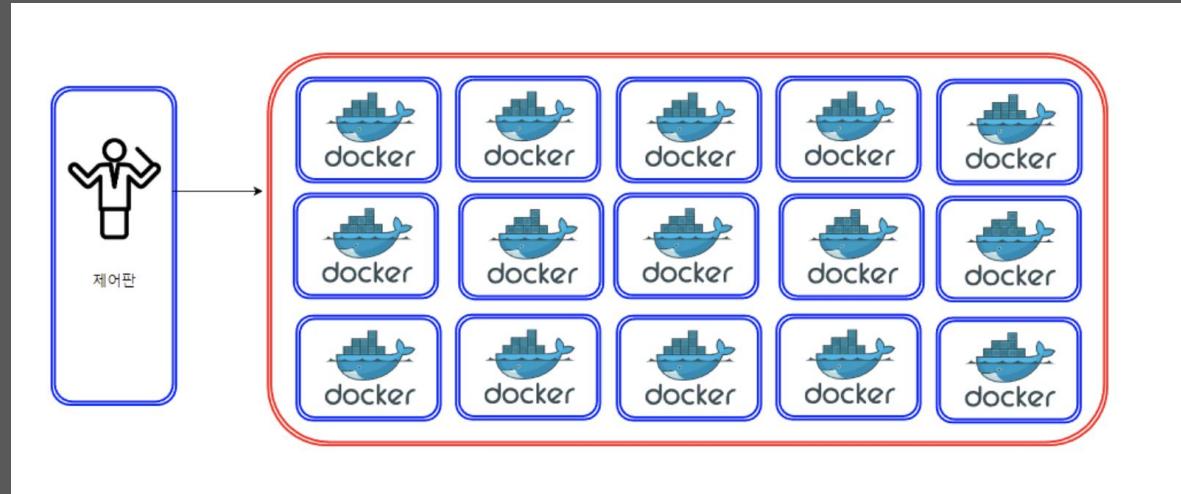
jar 배포와 마찬가지로 ssh로 EC2 접근 후, docker run ..

# Docker 배포

쿠버네티스를 통한 배포



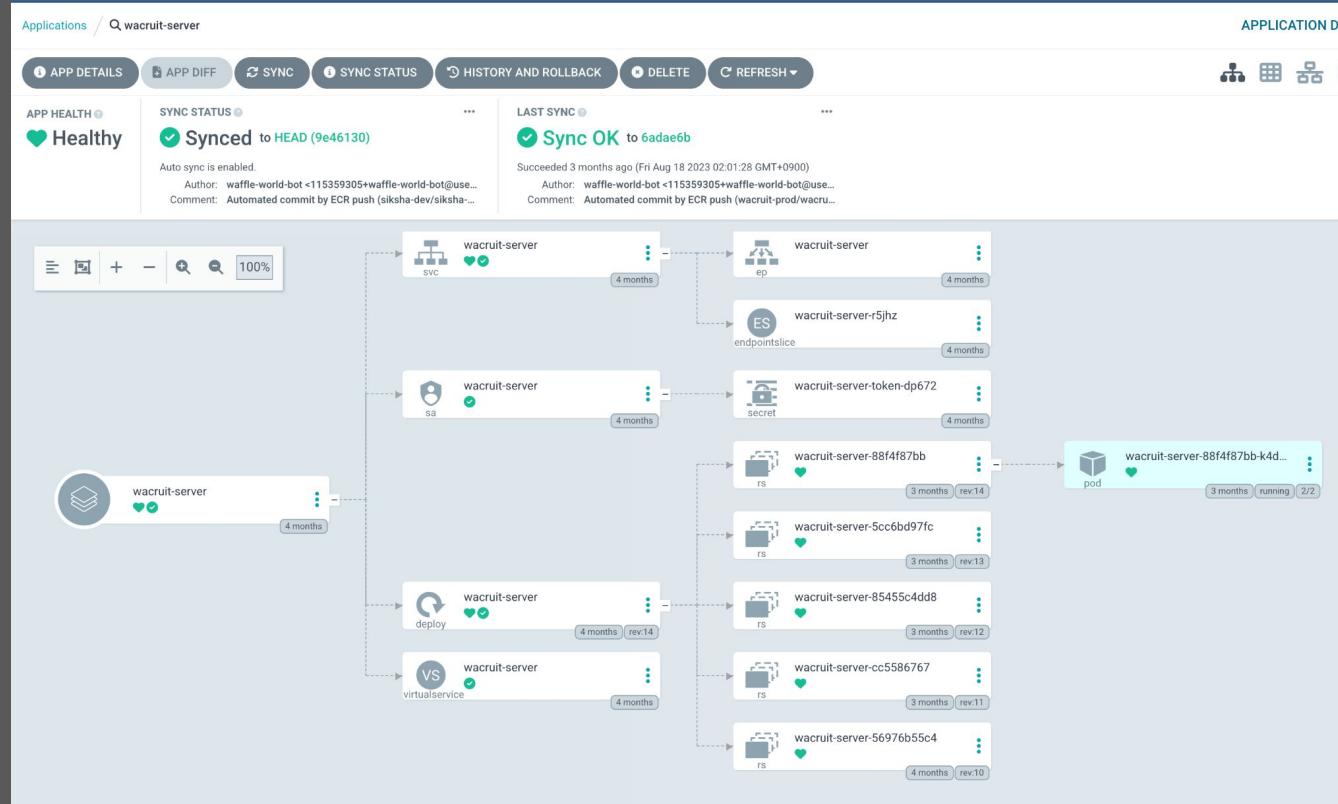
**kubernetes**



컨테이터 오케스트레이션 툴 (Container Orchestration Tool)

# Docker 배포

## 쿠버네티스를 통한 배포



# Week4 과제

## 1. 로컬 MySQL 연결

The screenshot shows a Java IDE interface with a project tree on the left and a code editor on the right.

**Project Tree:**

- seminar-2023-springboot-assignment [spring2023] (~/waffle/seminar)
- .gradle
- .idea
- build
- gradle
- src
  - main
    - kotlin
    - resources
      - static
      - application.yaml
      - application-db.yaml
      - data.sql
      - schema.sql
  - test
- .gitignore
- build.gradle.kts
- Dockerfile
- gradlew
- gradlew.bat
- README.md
- settings.gradle.kts
- External Libraries

**Code Editor (application-db.yaml):**

```
spring.config.activate.on-profile: local

spring:
  h2:
    console:
      enabled: true
      path: /h2-console
  datasource:
    url: jdbc:h2:mem:testdb;MODE=MySQL
    driver-class-name: org.h2.Driver
    username: sa
    password: 1234
  jpa:
    defer-datasource-initialization: true
  ---
  spring.config.activate.on-profile: prod

spring:
  datasource:
    url: jdbc:mysql://localhost:3306/spring_seminar
    driver-class-name: com.mysql.jdbc.Driver
    username: root
```

1. 로컬 환경에서 MySQL 서버 구동
2. prod 프로필로 서버 실행
3. 깃허브 닉네임으로 회원 가입

# Week4 과제

## 1. 로컬 MySQL 연결

```
~ mysql -u root

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 51
Server version: 8.1.0 Homebrew

Copyright (c) 2000, 2023, 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 |
| mysql          |
| performance_schema |
| spring_seminar   |
| sys            |
+-----+
5 rows in set (0.00 sec)

mysql> use spring_seminar;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from users;
+----+-----+-----+-----+
| id | username | password | image           |
+----+-----+-----+-----+
| 1  | PFCjeong | 12341234 | https://wafflestudio.com/images/icon_intro.svg |
+----+-----+-----+-----+
1 row in set (0.00 sec)
```

이미지와 같이 캡쳐하여 제출

# Week4 과제

## 2. Docker

The screenshot shows the Docker Hub interface for creating a new repository. The user is navigating through the 'General' tab of the 'pfcjeong/spring-seminar' repository. The repository has no description, was last pushed a few seconds ago, and contains one tag, 'latest'. The 'Tags' section lists the 'latest' tag.

Tag	OS	Type	Pulled	Pushed
latest		Image	--	a few seconds ago

1. 도커허브 계정 생성
2. spring-seminar라는 이름의 레포지토리 생성
3. 현재 master 브랜치 형상으로 도커 이미지를 spring-seminar에 푸시
4. 이미지와 같이 캡쳐하여 제출
5. 푸시된 도커 이미지는 정상 동작해야 함.