

CLOUDZ LABS

Spring Boot Training

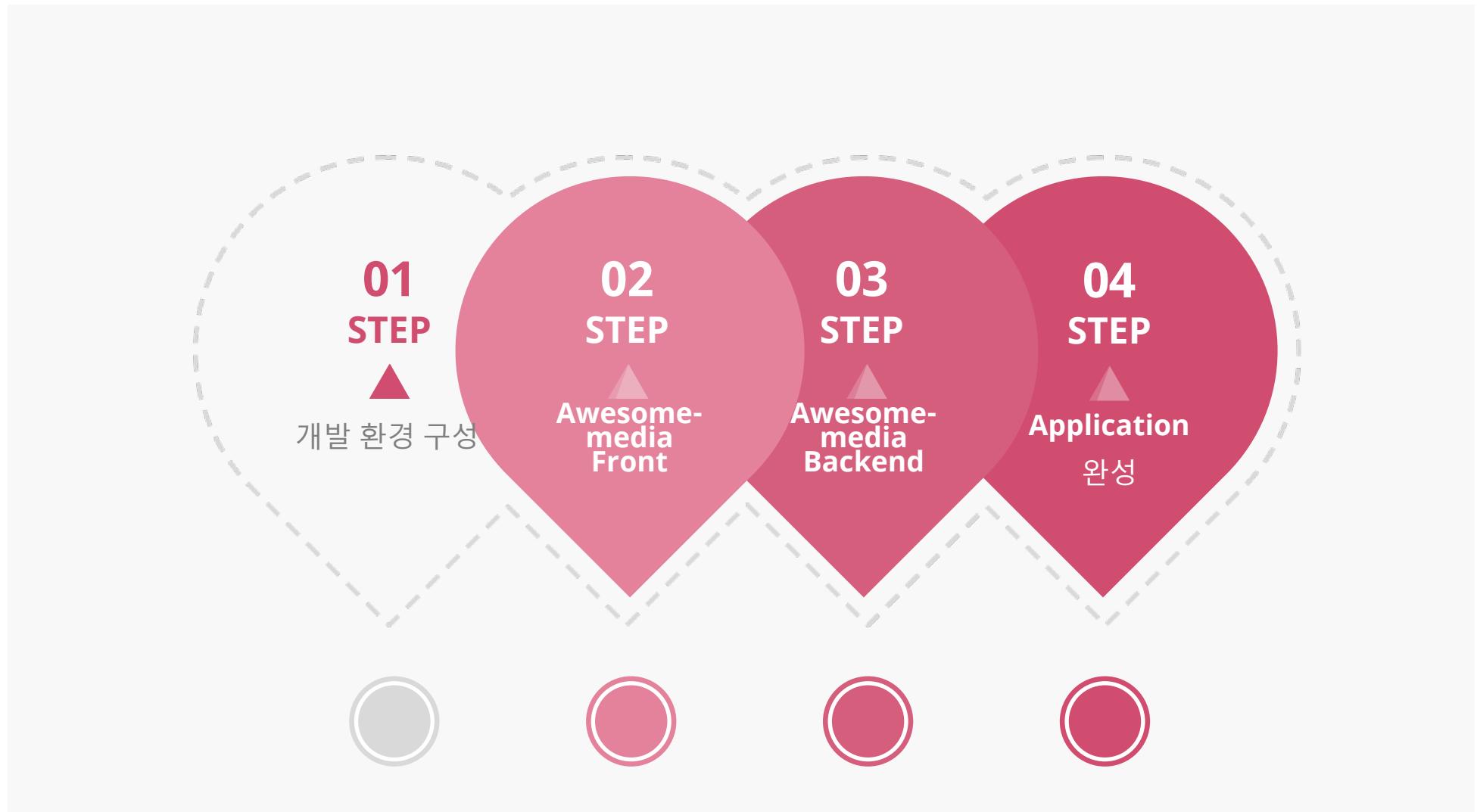


Table of Contents

01 | Spring Boot Application

02 | Deploying to the Cloud(Container)

PART 01 Spring Boot Application



개발환경 구성

개발환경을 구축 합니다.

시스템 요구 사항

Name	Servlet	Java	Apache Maven
Tomcat 8.5	3.1	Java 8	3.2 +

통합 개발 환경(IDE)

- Spring Tool Suite 3.9.2.RELEASE : <https://spring.io/tools/>
- JDK 8 : <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

개발환경 구성

개발환경을 구축 합니다.

Docker 설치

OS	링크
Windows10	https://docs.docker.com/docker-for-windows/install/#download-docker-for-windows
Windows7	https://docs.docker.com/toolbox/overview/#whats-in-the-box
Mac	https://docs.docker.com/docker-for-mac/install/

CLI 설치

- Kubectl : <https://kubernetes.io/docs/tasks/tools/install-kubectl>

Windows 환경변수 설정

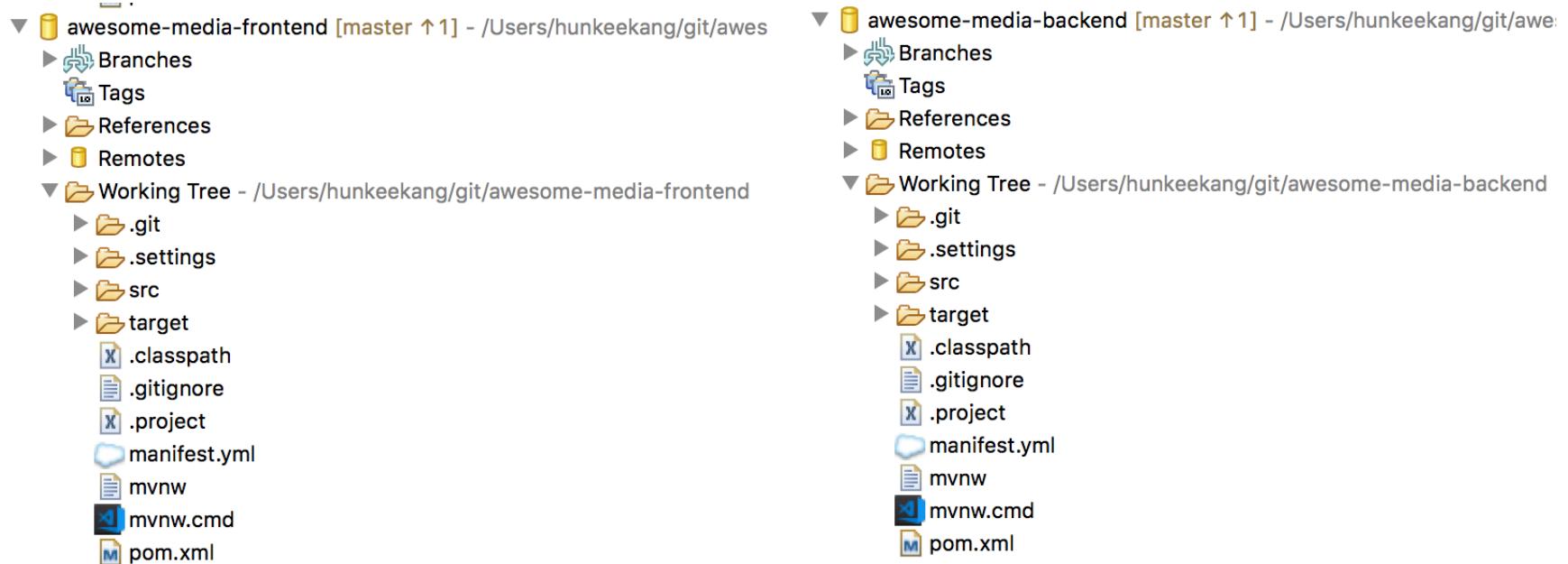
- JDK & Kubectl

개발환경 구성

Awesome Media 프로젝트를 Git에서 Clone 받습니다.

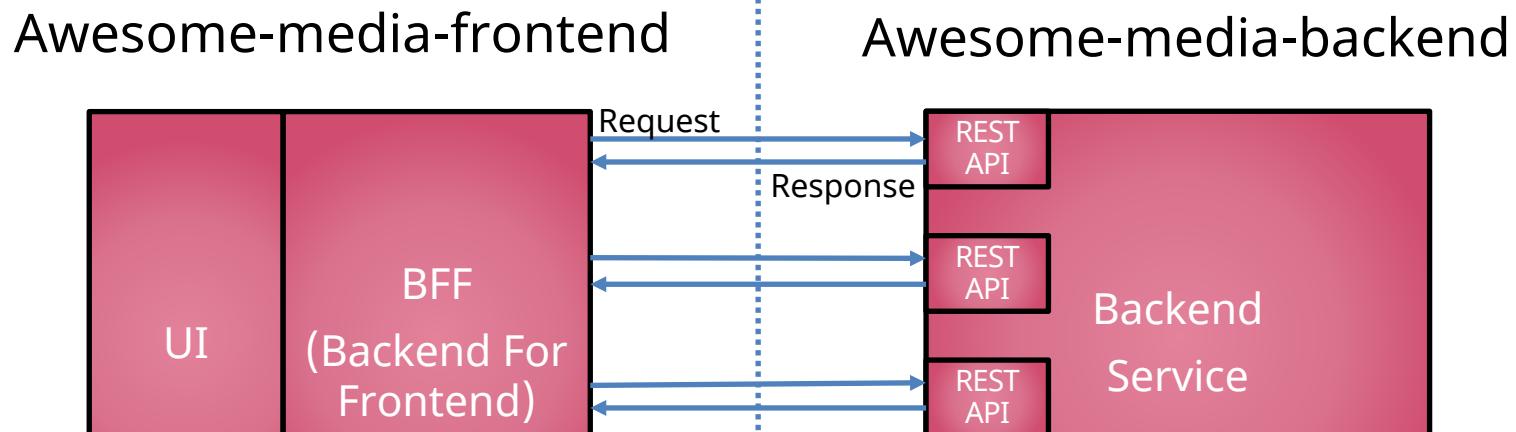
Git 주소

- <https://mygit.skcc.com/scm/cna/awesome-media-frontend.git>
- <https://mygit.skcc.com/scm/cna/awesome-media-backend.git>



Awesome Media 구조

Architecture



Awesome Media Frontend

Awesome Media Frontend 프로젝트 구조는 다음과 같습니다.

```
awesome-media-frontend
+- src
  +- main
    +- java
      +- com
        +- skcc
          +- account
            +- controller
            +- service
            +- dao
            +- vo
          +- category
          +- contents
          +- ...
          +- AwesomeMediaFrontendApplication.java
          +- Config.java
    +- resources
  +- test
```

Awesome Media Frontend

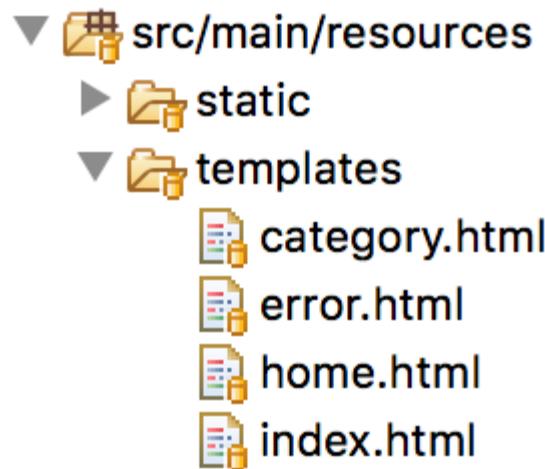
Awesome Media Frontend 프로젝트의 폴더구조는 다음과 같은 의미를 가집니다.

workspace를 기준으로 한 위치	의미	비고
<code> \${project_home}</code>	프로젝트 루트	
<code> \${project_home}/src/main/java</code>	업무 클래스의 파일 저장소	Base Package : main 및 configuration 존재
<code> \${project_home}/src/main/resources</code>	프로젝트의 리소스 저장소	
<code> \${project_home}/src/test/java</code>	단위 테스트 파일 저장소	

Awesome Media Frontend

Template Engine을 사용하기 위해, Dependency를 추가하고 아래의 경로로 View를 구성합니다.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```



application.yml

```
spring:
  thymeleaf:
    prefix: classpath:/static/html/
```

Awesome Media Frontend

Backend Service의 Rest API를 호출하기 위해 Spring에서 제공하는 RestTemplate을 설정합니다.

```
import org.springframework.cloud.client.loadbalancer.LoadBalanced;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.web.client.RestTemplate;  
  
@Configuration  
public class Config {  
    @Bean  
    @LoadBalanced  
    public RestTemplate restTemplate() {  
        return new RestTemplate();  
    }  
}
```

Awesome Media Front

현재까지 구현된 **Application**은 이런 모습일 것입니다!(Backend까지 완성된 후 모습)

The screenshot shows a movie poster for '퍼시픽 림' (Pacific Rim) as the main content. At the top left is a cartoon character icon with the text 'banana?'. To the right are a search bar with the placeholder '검색어를 입력하세요' and a user profile icon labeled '두국만'.

퍼시픽 림

바다 외계 생명체에 맞선 세계 동맹이 실패로 끝나면서 전직 파일럿과 새내기 파일럿이 힘을 합쳐 지구를 구하려 한다.

액션 모험

아퀼리브리엄 퍼시픽 림 나는 전설이다 다크아워

Awesome Media Backend

Awesome Media Backend 프로젝트 구조는 다음과 같습니다.

```
awesome-media-backend
+- src
  +- main
    +- java
      +- com
        +- skcc
          +- account
            +- controller
            +- service
            +- dao
            +- vo
          +- category
          +- contents
          +- profile
          +- ...
          +- AwesomeMediaBackendApplication.java
    +- resources
  +- test
```

Awesome Media Backend

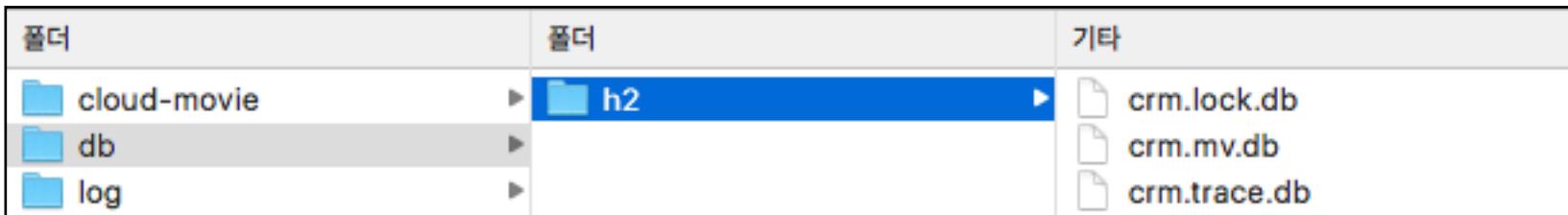
Awesome Media Backend 프로젝트의 폴더구조는 다음과 같은 의미를 가집니다.

workspace를 기준으로 한 위치	의미	비고
<code> \${project_home}</code>	프로젝트 루트	
<code> \${project_home}/src/main/java</code>	업무 클래스의 파일 저장소	Base Package : main 및 configuration 존재
<code> \${project_home}/src/main/resources</code>	프로젝트의 리소스 저장소	
<code> \${project_home}/src/test/java</code>	단위 테스트 파일 저장소	

Awesome Media Backend

Local에 H2 Database 연동을 위해, **application.yml**에 다음과 같이 **Datasource** 설정을 추가합니다(File에 저장).

```
spring:
  application:
    name: awesome-media-localhost
  datasource:
    initialize: true
    platform: h2
    driver-class-name: org.h2.Driver
    url: jdbc:h2:./db/h2/amdb;AUTO_SERVER=TRUE
    username: sa
    password: null
  h2:
    console:
      enabled: true
```



Awesome Media Backend

Local에 H2 Database 연동을 위해, `application.yml`에 다음과 같이 **Datasource** 설정을 추가합니다(Memory에 저장).

```
spring:
  application:
    name: awesome-media-localhost
  datasource:
    initialize: true
    platform: h2
    driver-class-name: org.h2.Driver
    url: jdbc:h2:mem:mediadb;
    username: sa
    password: null
  h2:
    console:
      enabled: true
```

Awesome Media Backend

지정된 DBMS에 따라서 초기 테이블 생성과 데이터를 **insert**할 수 있습니다.

```
src
+- main
  +- java
  +- resource
    +- data-h2.sql
    +- schema-h2.sql
```

```
-- accounts
DROP TABLE IF EXISTS accounts;
CREATE TABLE accounts (
  username varchar(255) NOT NULL,
  password varchar(255)
);
```

```
-- categories
DROP TABLE IF EXISTS categories;
CREATE TABLE categories (
  id varchar(255) NOT NULL,
  name varchar(255)
);
```

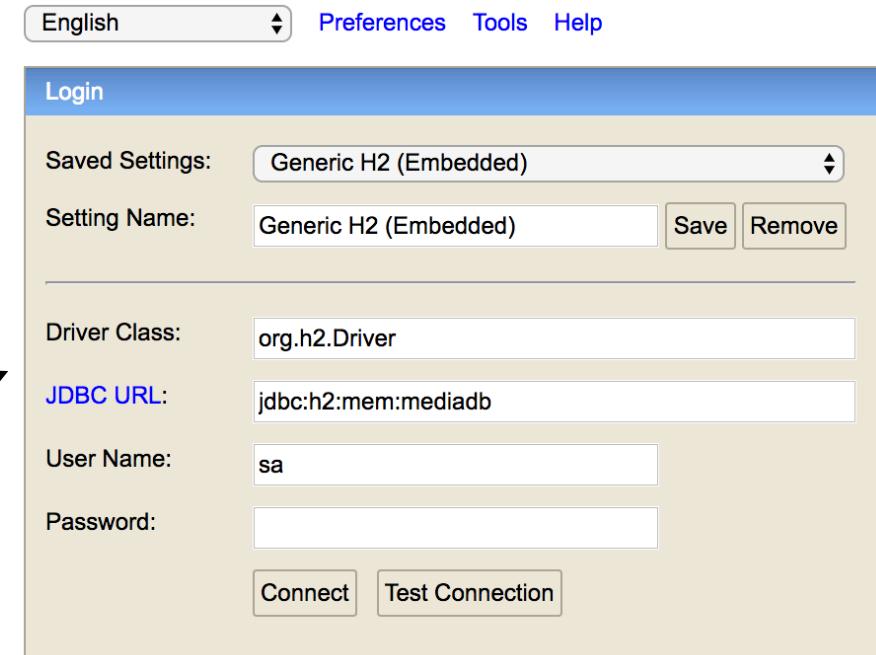
...

```
INSERT INTO categories (id, name)
VALUES
  ('action','액션 모험'),
  ('comedy','코미디'),
  ('drama','드라마 영화'),
  ('family_kids','가족/아동 영화'),
  ('horror','공포 영화'),
  ...
  ('winner','수상작');
```

Awesome Media Backend

생성된 DB는 Spring Boot에서 제공하는 “/h2-console”로 접속할 수 있습니다.

```
spring:  
  application:  
    name: awesome-media-localhost  
  datasource:  
    initialize: true  
    platform: h2  
    url: jdbc:h2:mem:mediadb;  
    username: sa  
    password: null  
  
  h2:  
    console:  
      enabled: true
```



Awesome Media Backend

H2 DB가 성공적으로 접속되면 다음과 같은 화면을 볼 수 있습니다.

The screenshot shows the H2 Database Browser interface. On the left, there is a tree view of database objects:

- jdbc:h2:mem:mediadb
- ACCOUNTS
- CATEGORIES
- CONTENTS
- EPISODES
- PROFILES
- PROMOTIONS
- INFORMATION_SCHEMA
- Sequences
- Users

Below the tree view, it says "H2 1.4.196 (2017-06-10)".

On the right, there is a toolbar with buttons: Run, Run Selected, Auto complete, Clear, and SQL statement:.

The SQL statement input field contains: `SELECT * FROM CATEGORIES|`

The results pane displays the output of the query:

```
SELECT * FROM CATEGORIES;
```

ID	NAME
action	액션 모험
comedy	코미디
drama	드라마 영화
family_kids	가족/아동 영화
horror	공포 영화
korean	한국 영화
romance	로맨스 영화
sf_fantasy	SF 및 판타지 영화
tv	TV 프로그램
winner	수상작

(10 rows, 2 ms)

Awesome Media Backend

실습에서 작성할 Awesome Media 애플리케이션의 **Layer**구성은 아래와 같습니다.



Awesome Media Backend

Spring Boot Starter에서 제공하는 MyBatis Autoconfiguration은 패키지 내에 Mapper를 발견하면 자동으로 Bean을 등록합니다.

```
@Mapper
public interface ContentsMapper {
    @Select("select * from contents where category=#{category}")
    @Results({
        @Result(property = "hasEpisodes", column = "has_episodes"),
        @Result(property = "regDate", column = "reg_date")
    })
    public List<Content> findByCategory(String category);

    @Insert("insert into contents(category, title, grade, poster, stillcut, rate,
year, summary, video, runtime, has_episodes, view, reg_date) values
(#{category}, #{title}, #{grade}, #{poster}, #{stillcut}, #{rate}, #{year},
#{summary}, #{video}, #{runtime}, #{hasEpisodes}, #{view}, #{regDate})")
    public int insertContent(Content content);
}
```

Awesome Media Backend

Service Layer에서는 **ContentMapper** 를 사용하여 데이터를 접근하고, 비즈니스 로직을 처리하여 결과를 **Controller**에 전달합니다.

```
@Service("contentsService")
public class ContentsService {

    @Autowired
    private ContentsMapper contentsMapper;

    public List<Content> getContents(String category) {
        return contentsMapper.findByCategory(category);
    }

    public int addContent(Content content) {
        return contentsMapper.insertContent(content);
    }
}
```

Awesome Media Backend

Controller Layer에서는 **@RestController** 어노테이션을 이용해서 REST API를 접근 가능하게 합니다.

```
@RestController
@RequestMapping("/v1")
public class ContentsRestController {
    private static Logger logger = LoggerFactory.getLogger(EpisodeRestController.class);

    @Autowired
    private ContentsService contentsService;

    @RequestMapping(path="/contents", method=RequestMethod.GET, name="getContents")
    public List<Content> getContents(@RequestParam(value = "category") String category) {
        logger.debug("getContents() called!!!");
        return contentsService.getContents(category);
    }

    @RequestMapping(path="/contents", method=RequestMethod.POST, name="addContents")
    public int addContents(@RequestBody Content content) {
        return contentsService.addContent(content);
    }
}
```

Awesome Media Backend

(GET)contents API를 호출한 결과입니다.(크롬 JSON Formatter)

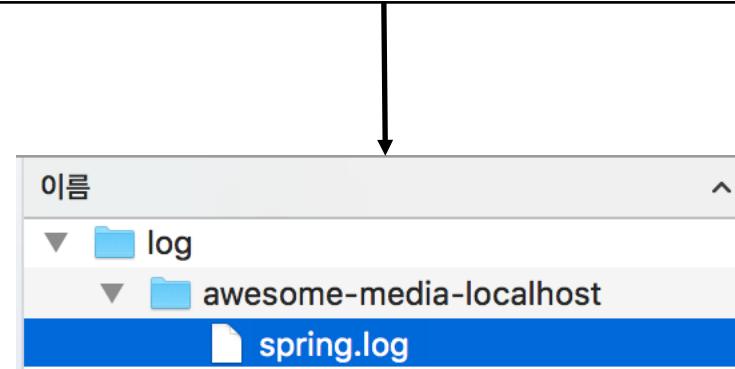
<http://localhost:8090/v1/contents?category=action>

```
[  
  {  
    "id": 61,  
    "category": "action",  
    "title": "아클리브리엄",  
    "grade": 0,  
    "poster": "https://art-s.nflximg.net/babeb/cf266c5d538ef7b48221c5e6f941adb470ebabeb.webp",  
    "stillcut": "https://art-s.nflximg.net/69a47/8526d5254108173a3f71e9e2d5eaf92043669a47.webp",  
    "rate": "15",  
    "year": 2002,  
    "summary": "반 이상향적 미래에서는 전체주의 정권이 약물로 대중을 조종하고 감정의 표현을 사형으로 처벌함으로써 평화를 유지한다.",  
    "video": "RBnJ0wzKOD",  
    "runtime": 107,  
    "hasEpisodes": false,  
    "view": 99,  
    "regDate": "2015-10-26"  
  },  
  {  
    "id": 62,  
    "category": "action",  
    "title": "퍼시픽 림",  
    "grade": 2,  
    "poster": "https://art-s.nflximg.net/09934/245a98e19d095c5b662e4ef469aec5d35db09934.webp",  
    "stillcut": "http://awesomewallpaper.files.wordpress.com/2013/08/pacific-rim-wallpapers-7.jpg",  
    "rate": "12"  
  }]
```

Awesome Media Backend

Log는 Application에 이름을 부여하고, “log/{application 이름}” 폴더에 Spring Log가 파일로 남겨지도록 설정합니다.

```
spring:  
  application:  
    name: awesome-media-localhost  
  
logging:  
  path: ../log/${spring.application.name}
```



RESTful API를 통한 Frontend와 Backend 연동

awesome-media-backend에서 Contents 목록을 조회하기 위해 구현한 API의 Spec은 다음과 같습니다.

Request URL → <http://localhost:8090/v1/contents?category=action>

HTTP	URI Template
GET	/v1/contents?category={id}

Paramters

Parameter	Parameter 명	Parameter Type	Data Type	Required
id	카테고리 ID	query	String	true

RESTful API를 통한 Frontend와 Backend 연동

awesome-media-backend에서 Contents 목록을 조회하기 위해 구현한 API의 Spec은 다음과 같습니다.

Response → application/json

```
[  
  {  
    "id": 61,  
    "category": "action",  
    "title": "아킬리브리엄",  
    "grade": 0,  
    "poster": "https://art-s.nflximg.net/babeb/cf266c5d538ef7b48221c5e6f941adb470ebabeb.webp",  
    "stillcut": "https://art-s.nflximg.net/69a47/8526d5254108173a3f71e9e2d5eaf92043669a47.webp",  
    "rate": "15",  
    "year": 2002,  
    "summary": "반 이상향적 미래에서는 전체주의 정권이 약물로 대중을 조종하고 감정의 표현을 사형으로 처벌함으로써 평화를 유지한다.",  
    "video": "RBnJ0wzK0dE",  
    "runtime": 107,  
    "hasEpisodes": false,  
    "view": 99,  
    "regDate": "2015-10-26"  
  },  
  {  
    "id": 62,  
    "category": "action",  
    "title": "퍼시픽 림",  
    "grade": 2,  
    "poster": "https://art-s.nflximg.net/09934/245a98e19d095c5b662e4ef469aec5d35db09934.webp",  
    "stillcut": "http://awesomewallpaper.files.wordpress.com/2013/08/pacific-rim-wallpapers-7.jpg",  
    "rate": "12"  
  }]
```

RESTful API를 통한 Frontend와 Backend 연동

awesome-media-frontend와 awesome-media-backend 애플리케이션을 연동하기 위해 주소를 설정하고 REST 형태로 backend를 호출합니다.

awesome-media-frontend 프로젝트의 application.yml

```
api:  
  bff:  
    path: /  
  services:  
    url: http://localhost:8090
```

```
@Value("${api.services.url}")  
private String serviceUrl;
```

RESTful API를 통한 Frontend와 Backend 연동

RestTemplate을 이용해서 Rest를 호출할 수 있습니다.

```
@Service("contentsService")
public class ContentsService {

    @Autowired
    private RestTemplate restTemplate;

    @Value("${api.services.url}")
    private String serviceUrl;                                api:  
services:  
url: http://localhost:8090

    public List<Content> getContentsByCategory(String category) {
        return
    Arrays.asList(restTemplate.getForObject(String.format("%s/v1/contents?category=%s",
    serviceUrl, category), Content[].class));
    }
}
```

RESTful API를 통한 Frontend와 Backend 연동

ContentsService를 사용하기 위한 RestController를 작성합니다(ContentsController)

```
@RestController
@RequestMapping("/v1")
public class ContentsController {

    @Autowired
    private ContentsService contentsService;

    @RequestMapping(path="/contents", method = RequestMethod.GET, name="getContents")
    public List<Content> getContents(@RequestParam(value = "category") String category){
        return contentsService.getContentsByCategory(category);
    }
}
```

Awesome Media Application

Application이 완성됐습니다!

The screenshot shows a web application interface. At the top left is a cartoon character icon with the text "banana?". Next to it is a "메뉴" (Menu) button. On the right side is a search bar with the placeholder "검색어를 입력하세요" (Enter search term) and a user profile icon with the text "두국만".

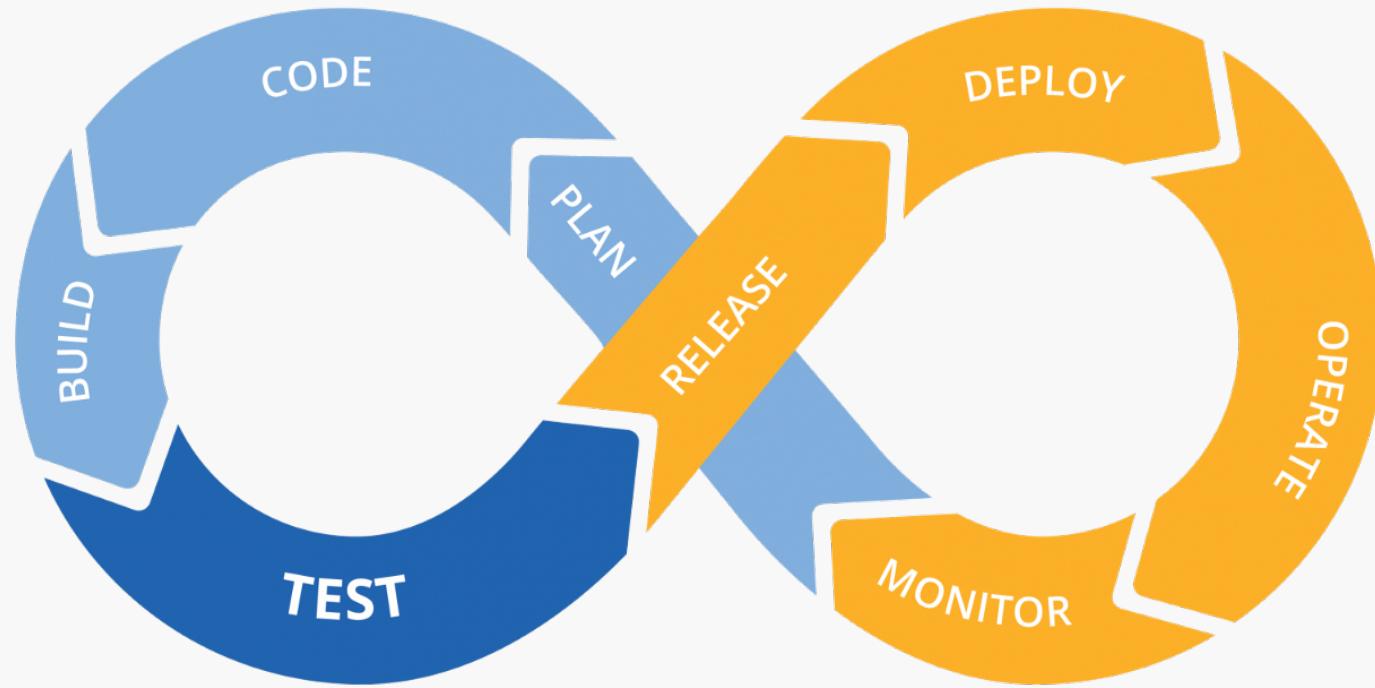
퍼시픽 림

바다 외계 생명체에 맞선 세계 동맹이 실패로 끝나면서 전직 파일럿과 새내기 파일럿이 힘을 합쳐 지구를 구하려 한다.

액션 모험

아퀼리브리엄 퍼시픽 림 나는 전설이다 다크아워

PART 02 Deploying to the Cloud(Container)



Ready

CaaS 환경으로의 애플리케이션 배포를 준비합니다.

- ✓ Infra환경(Kubernetes, Backing Service 등등..)이 갖추어 졌는가?
- ✓ Application이 12 Factors를 만족하는가?
- ✓ 외부 연동서비스는 어떤 것들이 있는가?
- ✓ 배포 이미지는(JDK, 버전 등) 어떤 것을 선택할 것인가?

설정 분리

application.yml을 배포환경 별로 다음과 같이 추가합니다.

```
src
+- main
  +- java
  +- resource
    +- application-k8s.yml
    +- application-default.yml
    +- application.yml
```

설정분리

배포 환경별 적용된 설정은 다음과 같습니다.

Configuration	default	k8s
application	O	O
application-k8s	X	O
application-default	O	X

설정분리

환경별로 적용된 설정은 다음과 같습니다.

application-default.yml

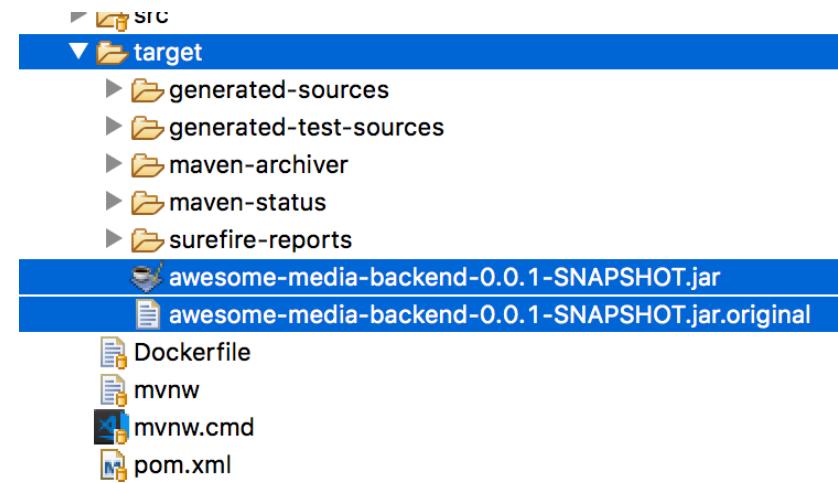
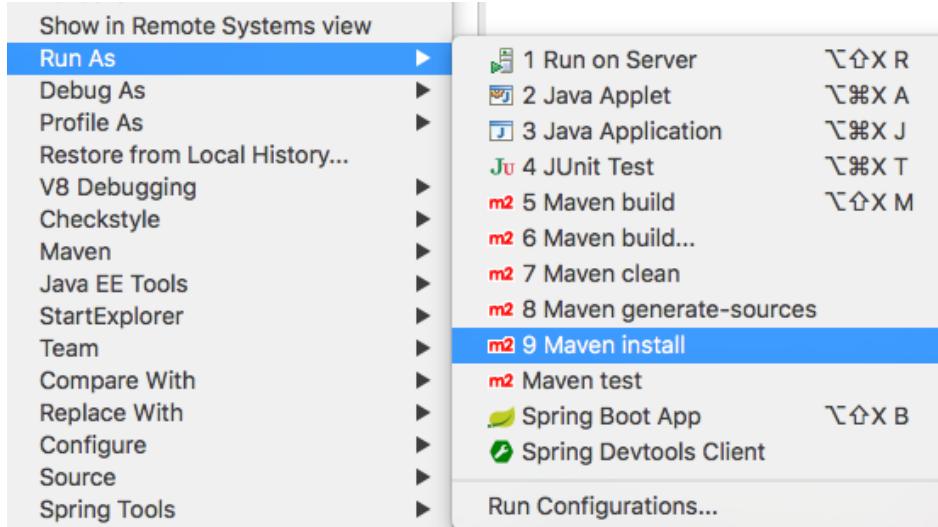
```
spring:  
  application:  
    name: awesome-media-localhost  
  datasource:  
    initialize: true  
    platform: h2  
    url: jdbc:h2:mem:mediadb;  
    username: sa  
    password: null  
  h2:  
    console:  
      enabled: true
```

application-k8s.yml

```
spring:  
  application:  
    name: awesome-media-k8s  
  datasource:  
    initialize: true  
    platform: mysql  
    url:  
      jdbc:mysql://${AWESOME_MEDIA_DB_SERVICE_HOST}  
      ${AWESOME_MEDIA_DB_SERVICE_PORT}/${DATABASE}  
      ?username=${MYSQL_USER}&password=${MYSQL_PASSWORD}
```

Maven Build(jar Packaging)

Maven을 통해서 애플리케이션을 패키징합니다. 패키징된 결과는 /target 폴더에서 확인하실 수 있습니다.



감사합니다

