2025/09/29~30

MSA 이해와 구현 방법

박창렴 / myanjini@gmail.com / 010-2982-7033

# 내용 공유 >>>

**http://bit.ly/3Wioln7**

숫자만 붉은색이고 대소문자를 구분해서 입력해 주세요. 수업 종료 후 삭제될 수 있으니, 필요한 내용은 미리 백업해 주세요.

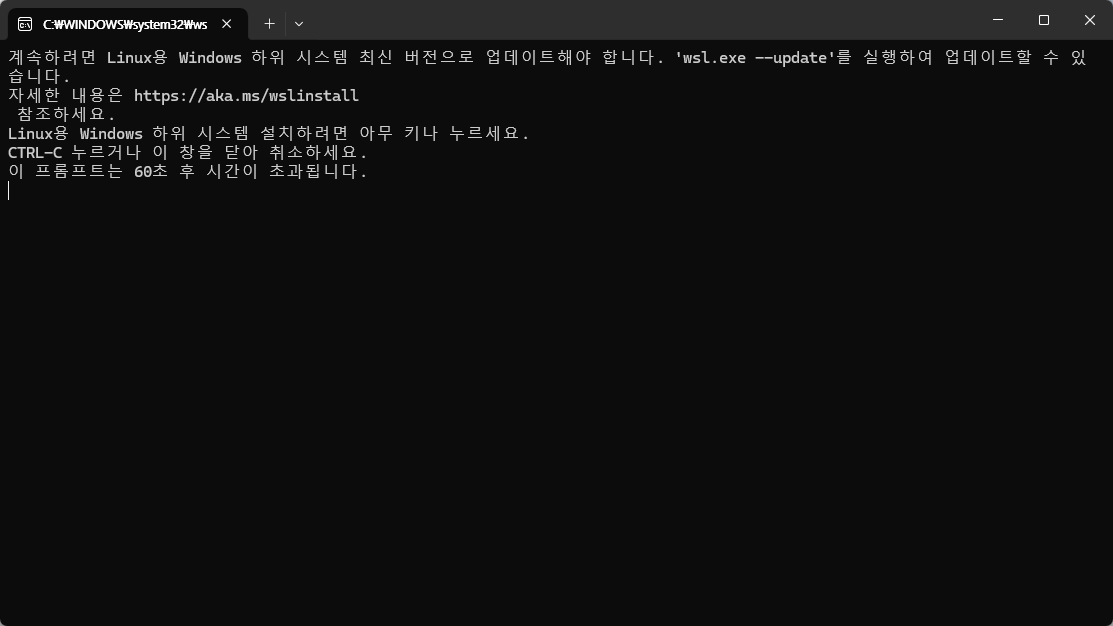
# 실습 환경 설정

## Docker Desktop (리부팅 필요)

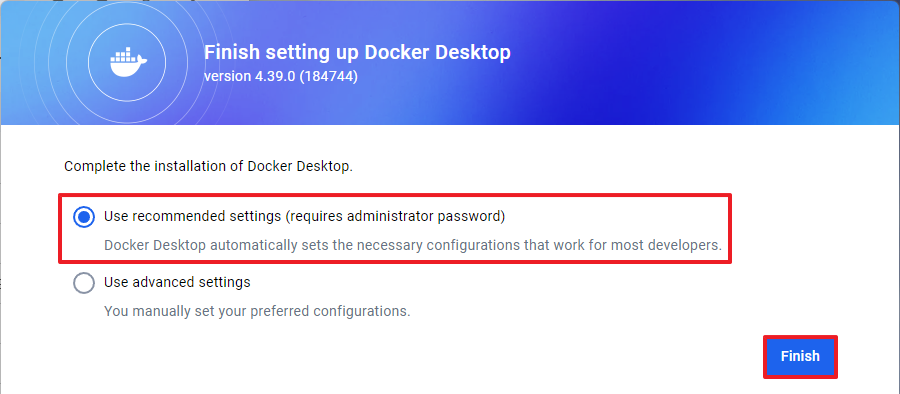
https://www.docker.com/get-started/



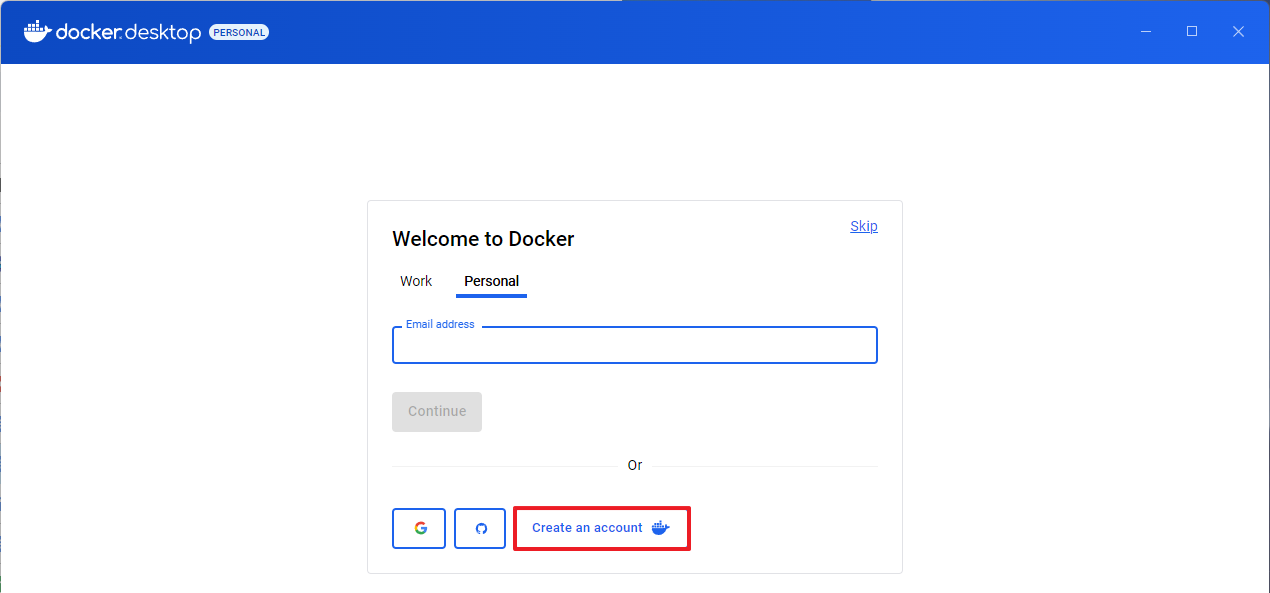
리부팅 후 아래와 같은 메시지가 출력되면 아무 키나 눌러서 WSL 최신 버전으로 업데이트 합니다.



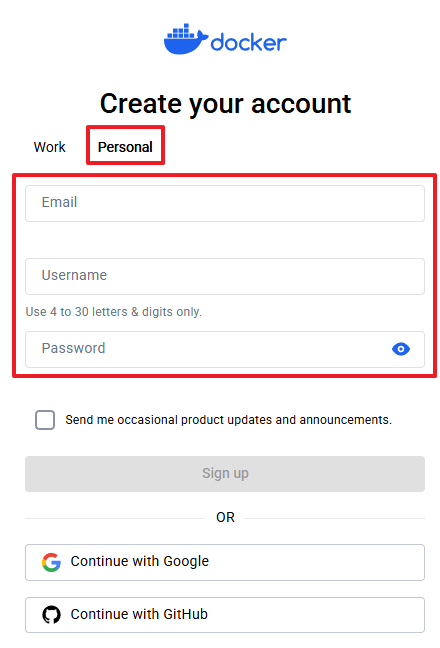
도커 데스크탑 콘솔 설정을 마무리 합니다.



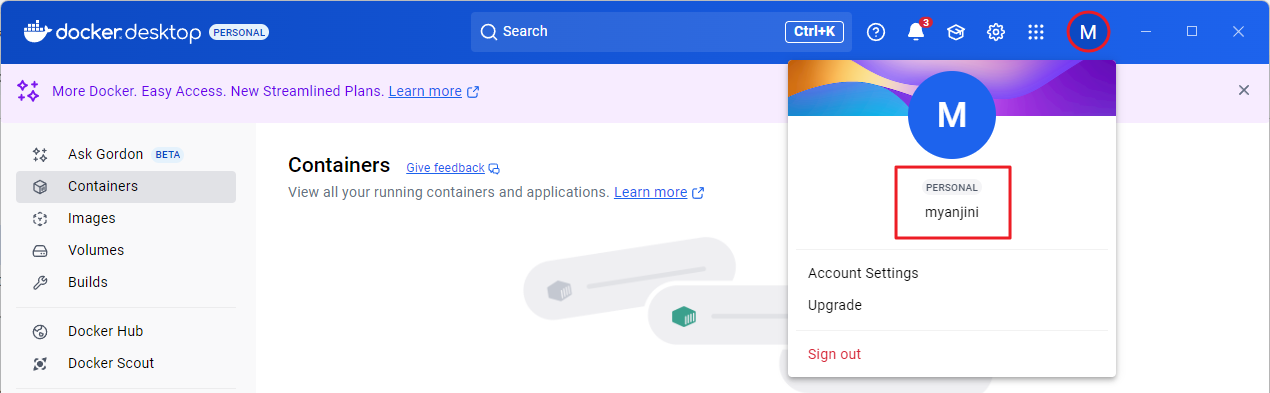
도커 허브 계정이 없는 경우, "Create an account" 버튼을 클릭해서 계정을 생성합니다. 계정이 있는 경우, 로그인합니다.



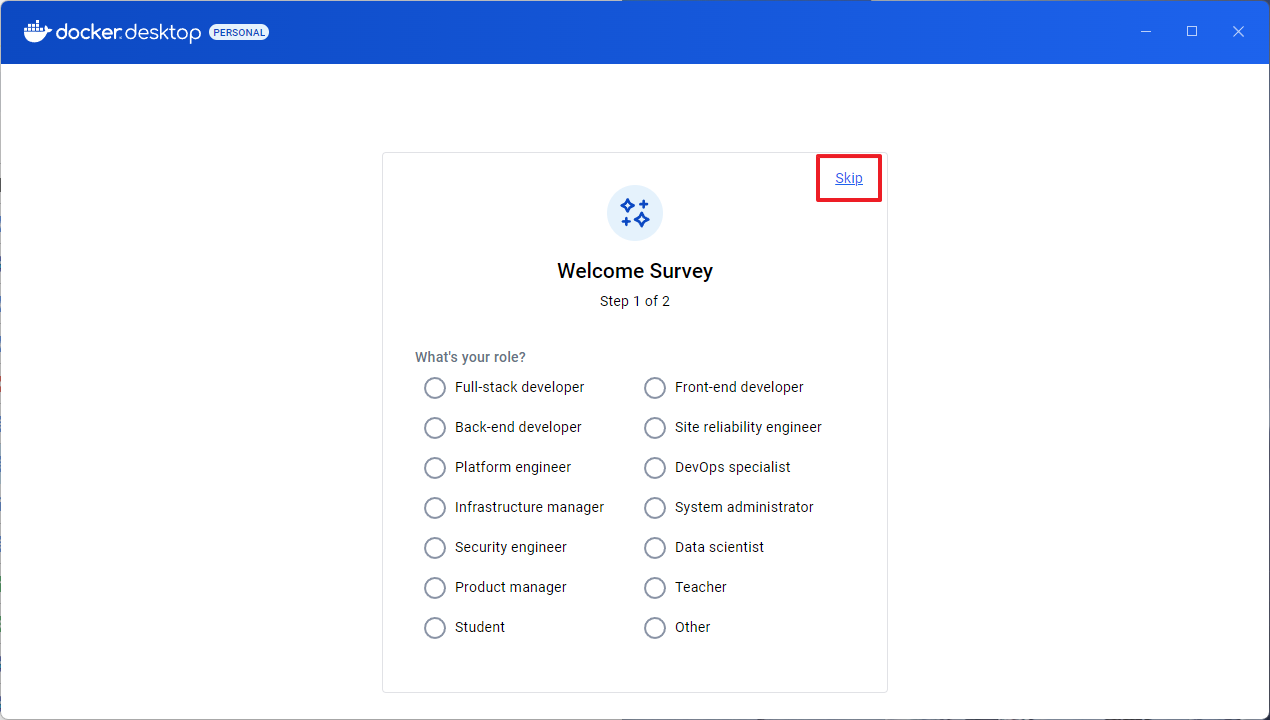
회원 가입은 소셜 로그인 방식이 아닌, Email, Username, Password를 입력해서 가입해 주세요.

****

가입한 계정으로 로그인 ⇒ 정상적으로 로그인하면, 도커 허브와 도커 데스크탑 오른쪽 상단에 프로필 아이콘이 나타납니다.

****

설문은 생략해도 됩니다.



### 도커 실행 확인

C:\Users\myanj> docker version

Client:

Version: 28.3.0

API version: 1.51

Go version: go1.24.4

Git commit: 38b7060

Built: Tue Jun 24 15:44:57 2025

OS/Arch: windows/amd64

Context: desktop-linux

Server: Docker Desktop 4.43.1 (198352)

Engine:

Version: 28.3.0

API version: 1.51 (minimum version 1.24)

Go version: go1.24.4

Git commit: 265f709

Built: Tue Jun 24 15:44:25 2025

OS/Arch: linux/amd64

Experimental: false

containerd:

Version: 1.7.27

GitCommit: 05044ec0a9a75232cad458027ca83437aae3f4da

runc:

Version: 1.2.5

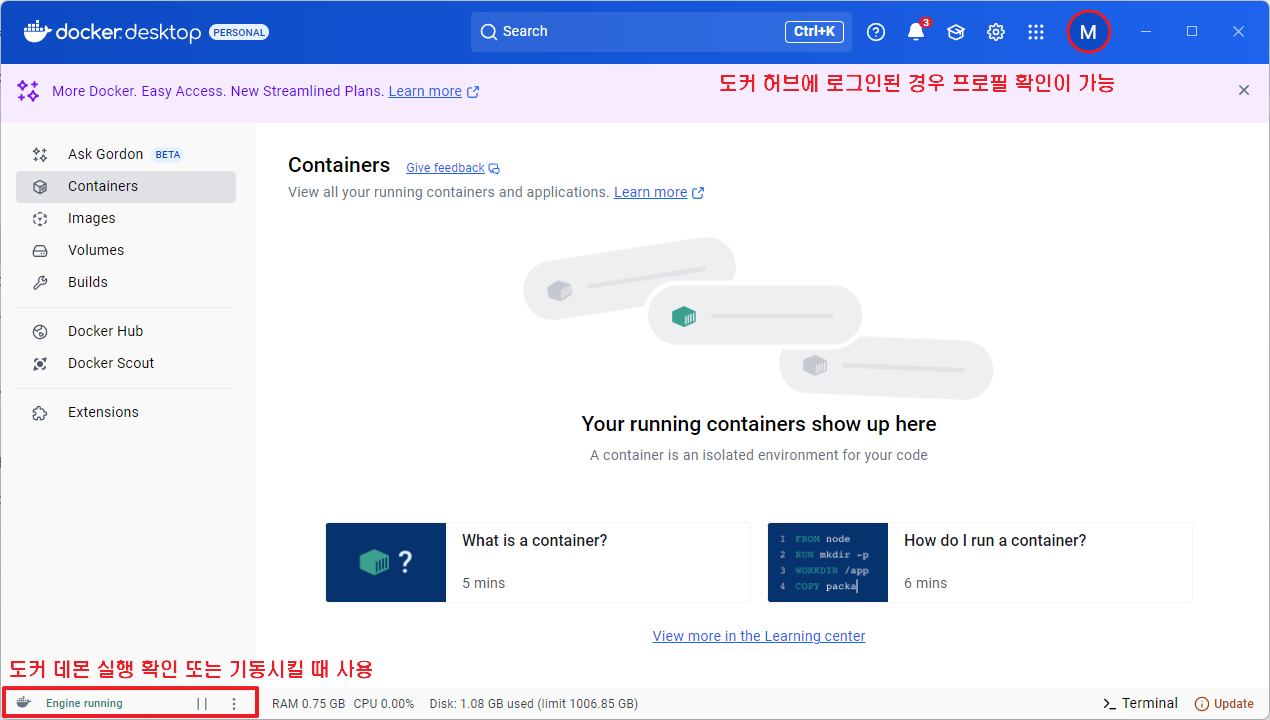
GitCommit: v1.2.5-0-g59923ef

docker-init:

Version: 0.19.0

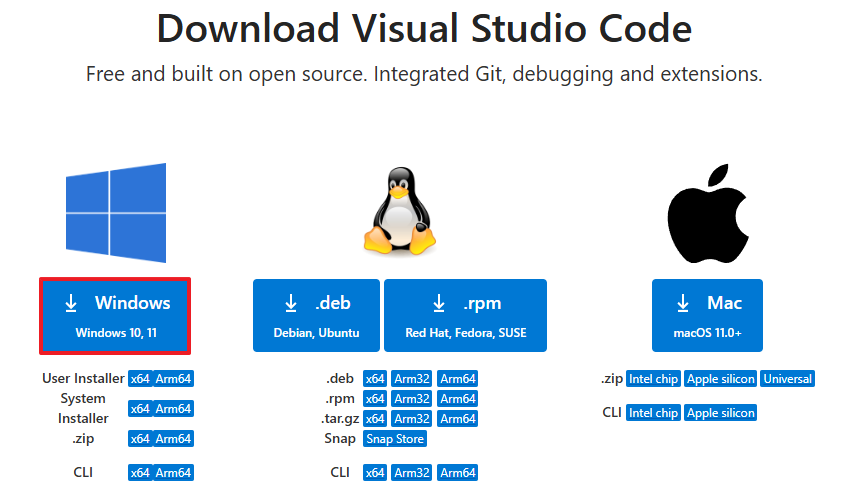
GitCommit: de40ad0

### 서버 정보가 나오지 않는 경우 → 시작 메뉴에서 Docker Desktop을 실행해서 도커 데몬 실행을 확인



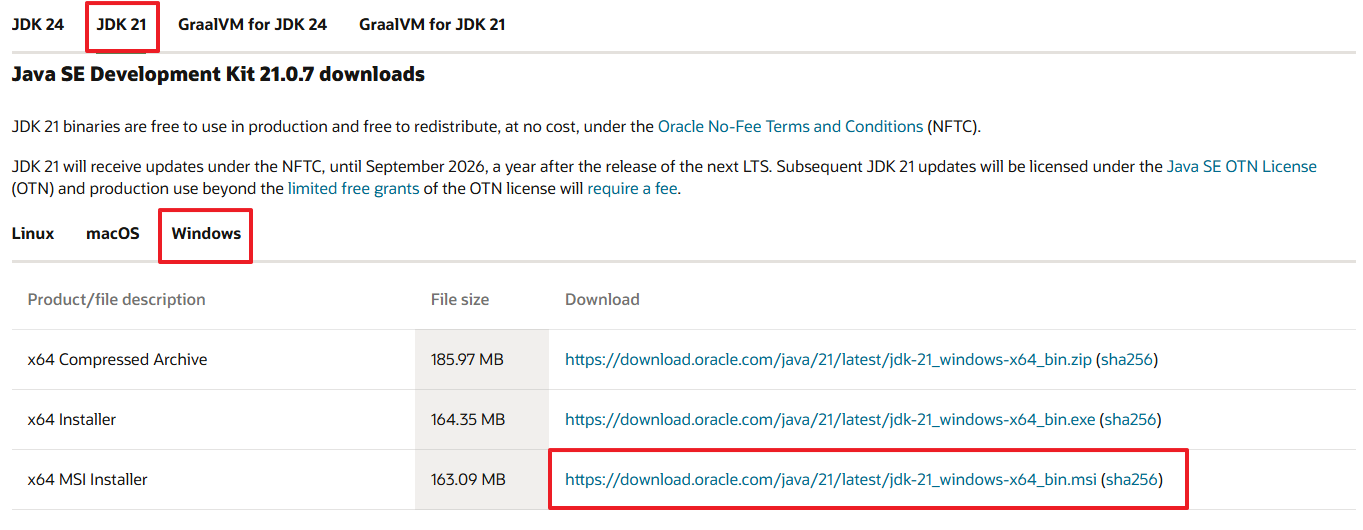
## Visual Studio Code

https://code.visualstudio.com/download



## JDK

https://www.oracle.com/kr/java/technologies/downloads/#java21



기본 설정 상태로 설치를 진행합니다. 설치가 완료되명 명령 프롬프트를 실행해서 설치를 확인합니다.

C:\Users\myanjini> java --version

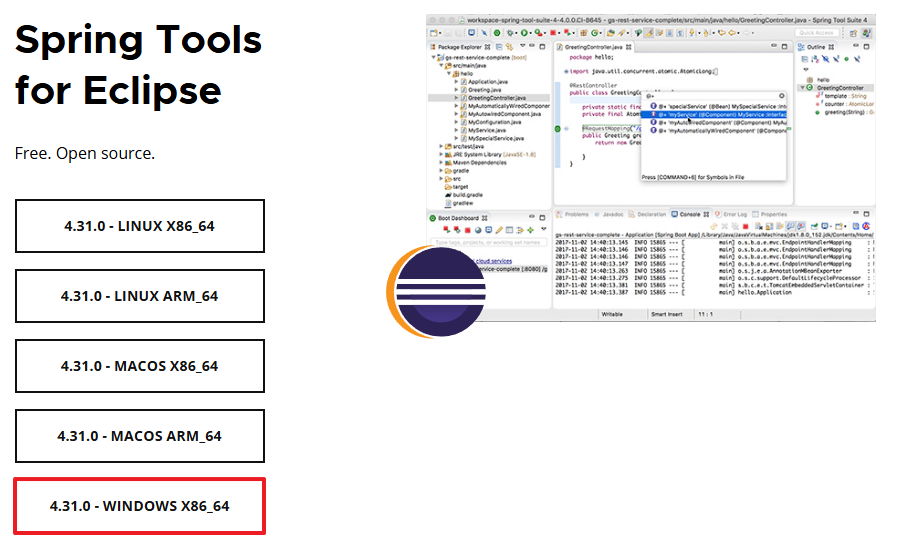
java 21.0.7 2025-04-15 LTS

Java(TM) SE Runtime Environment (build 21.0.7+8-LTS-245)

Java HotSpot(TM) 64-Bit Server VM (build 21.0.7+8-LTS-245, mixed mode, sharing)

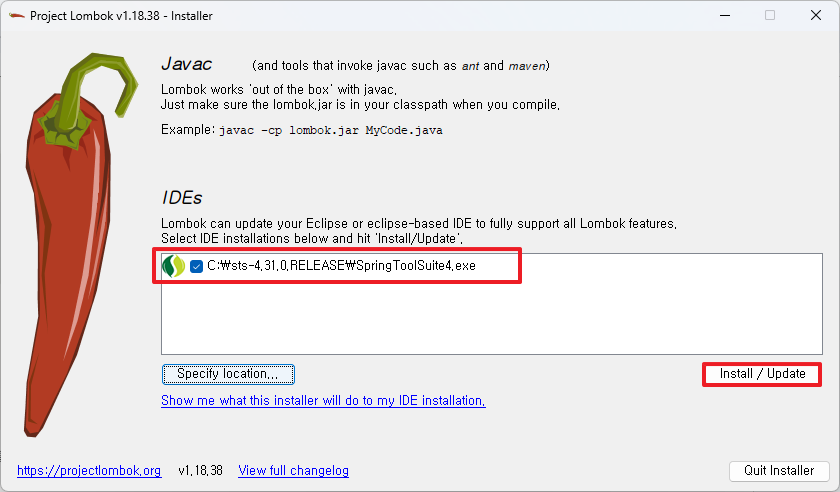
## STS 다운로드 후 압축해제

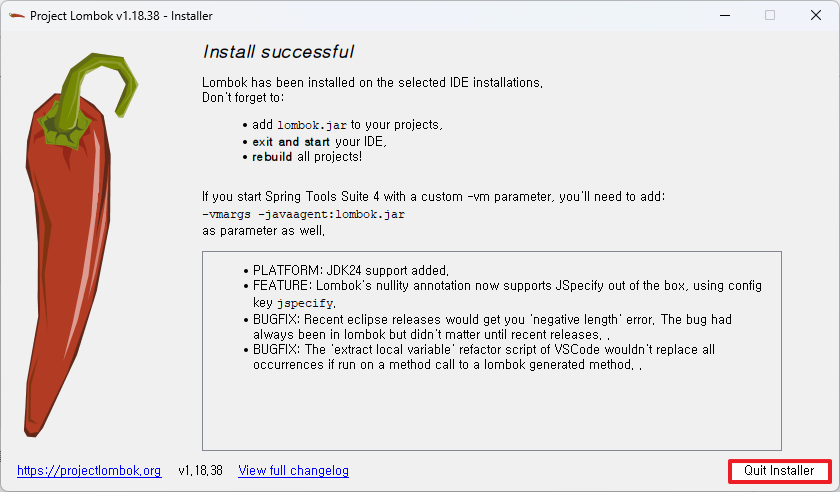
https://spring.io/tools

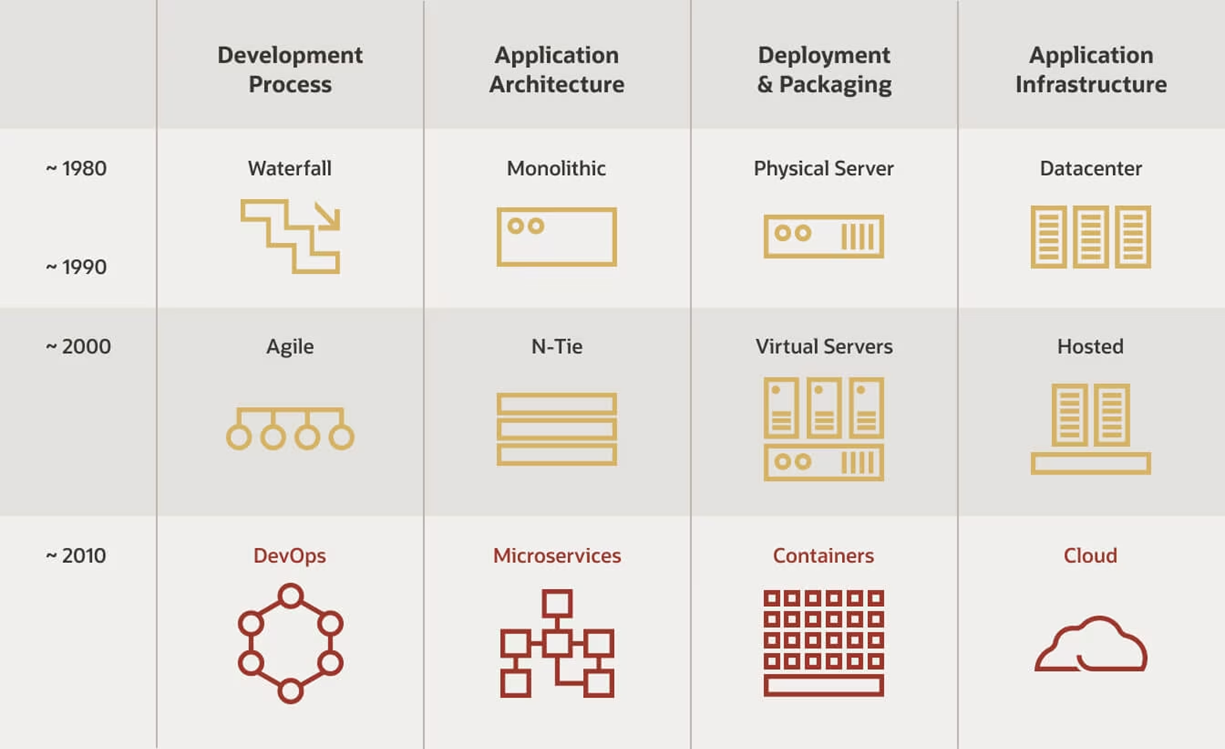


## Lombok 다운로드 후 설치

https://projectlombok.org/downloads/lombok.jar







# 참고자료

## 표준프레임워크 MSA 적용 가이드

https://www.egovframe.go.kr/home/ntt/nttRead.do?pagerOffset=0&searchKey=&searchValue=&menuNo=76&bbsId=171&nttId=1814

## [클라우드네이티브정보시스템구축을위한 - 발주자안내서](https://www.egovframe.go.kr/cmm/file/readDownloadFile.do?fileId=FILE_000000000018351&fileSn=6)

https://www.egovframe.go.kr/cmm/file/readDownloadFile.do?fileId=FILE\_000000000018351&fileSn=6

## [클라우드네이티브정보시스템구축을위한 - 개발자안내서](https://www.egovframe.go.kr/cmm/file/readDownloadFile.do?fileId=FILE_000000000018351&fileSn=6)

https://www.egovframe.go.kr/cmm/file/readDownloadFile.do?fileId=FILE\_000000000018351&fileSn=7

## 스프링 마이크로서비스 코딩 공작소 [ 개정 2판 ]

https://www.yes24.com/Product/Goods/110243944

## 마이크로서비스 패턴

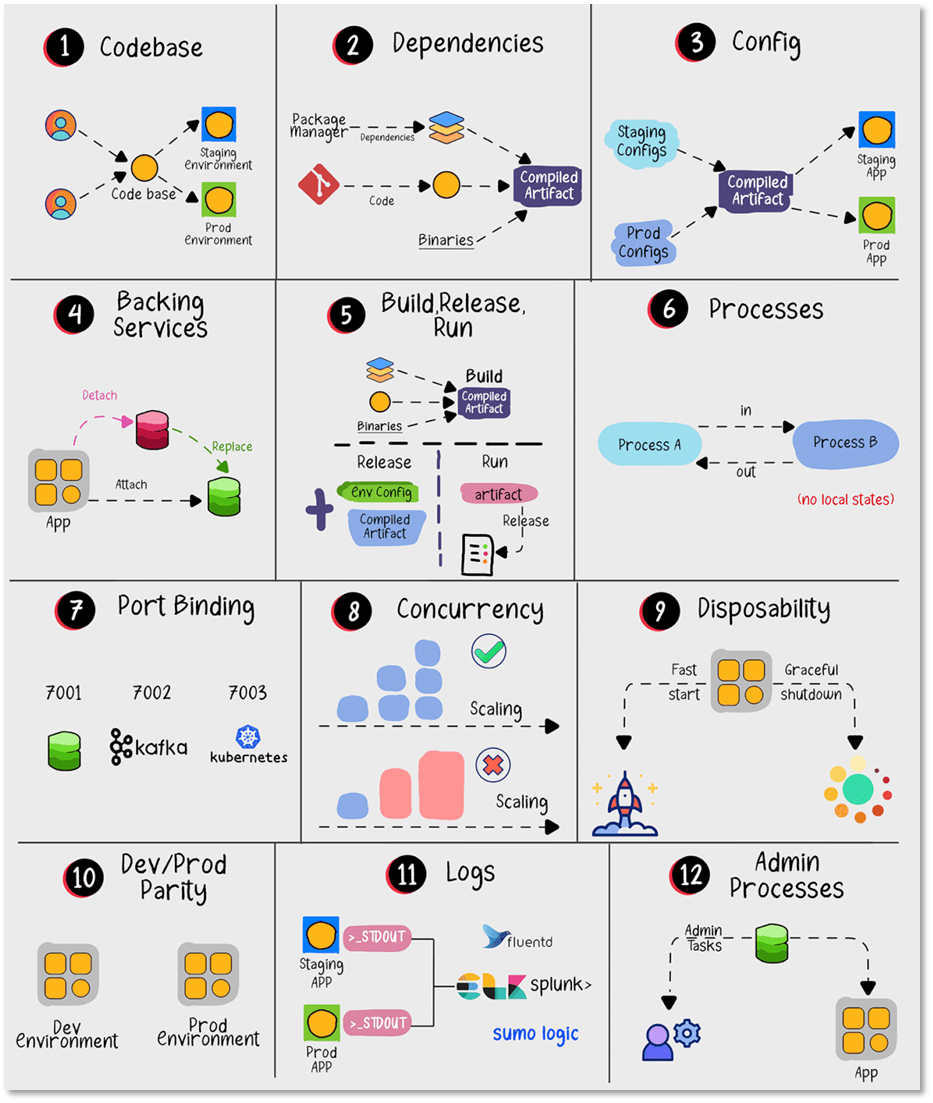
https://microservices.io/

https://www.yes24.com/Product/Goods/86542732

# REST 성숙도 모델

https://g4daclom.tistory.com/163

# 12가지 요소(12 Factors)



# 잠시 쉬고, 15시 15분에 이어서 하겠습니다.

https://developer.mozilla.org/ko/docs/Web/Security/Same-origin\_policy

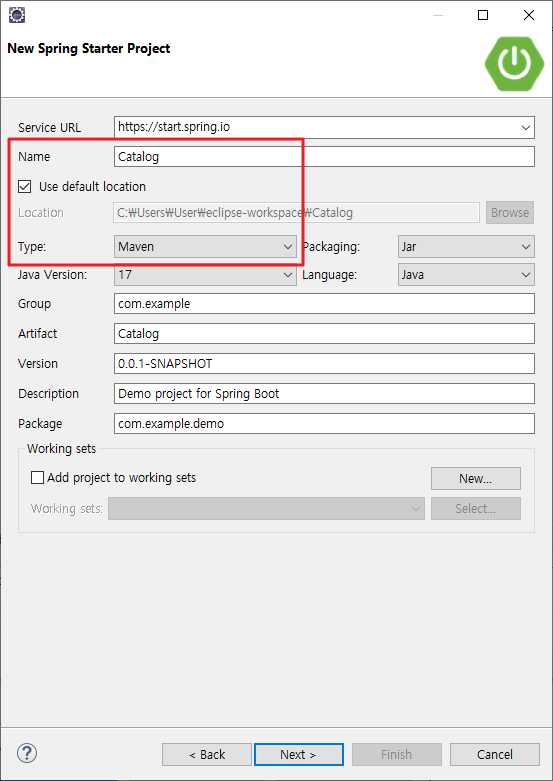
https://developer.mozilla.org/ko/docs/Web/HTTP/Guides/CORS

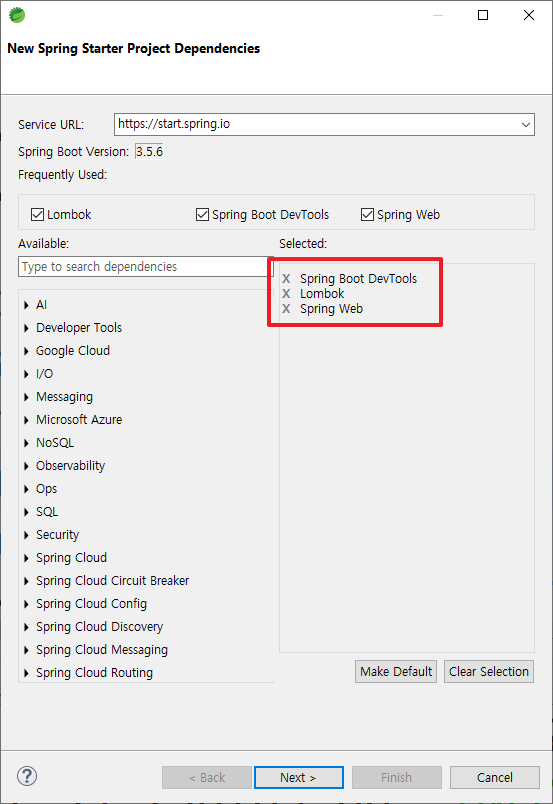


# 카탈로그 서비스 생성

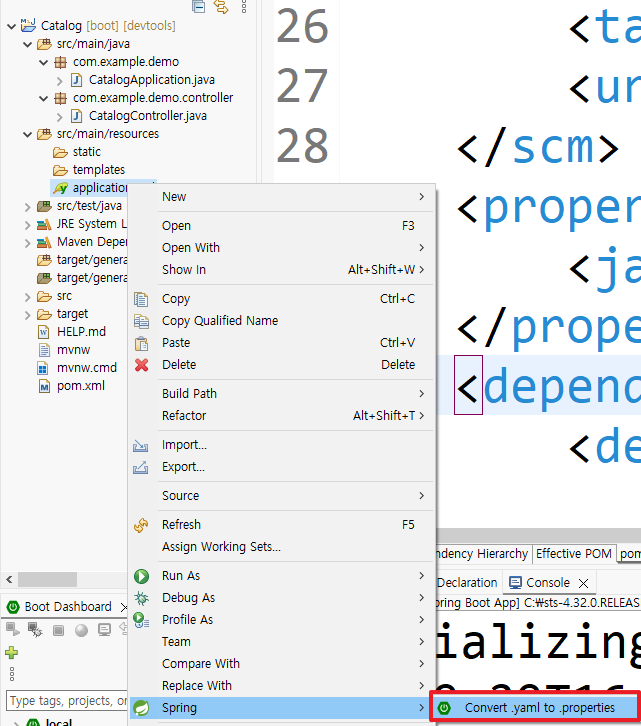
## #1 프로젝트 생성







## #2 설정 파일(application.properties)에 yaml 형식으로 변경하고, 서버 서비스 포트를 추가



spring:

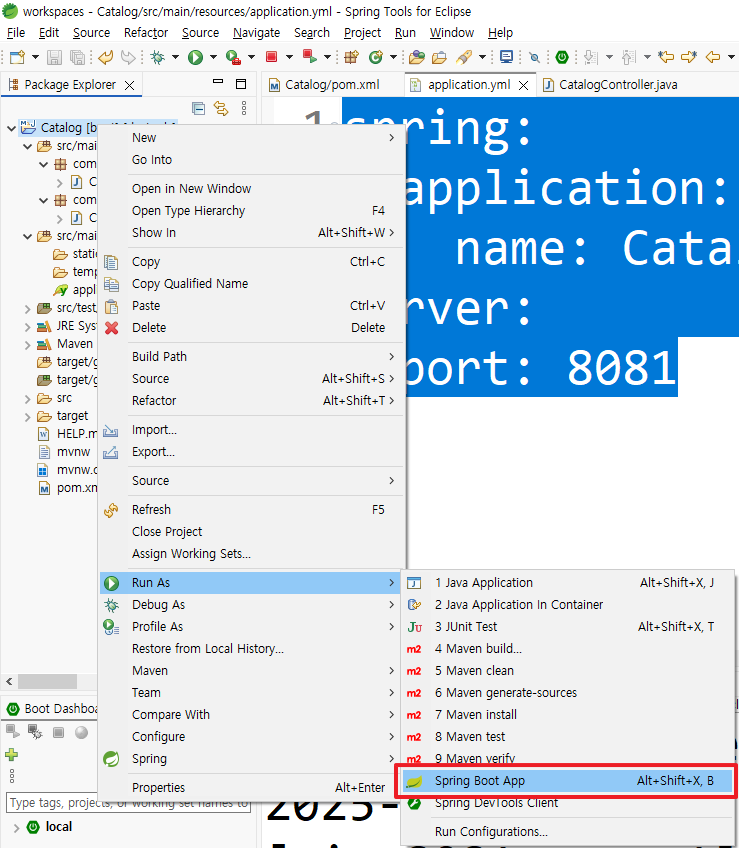
application:

name: Catalog

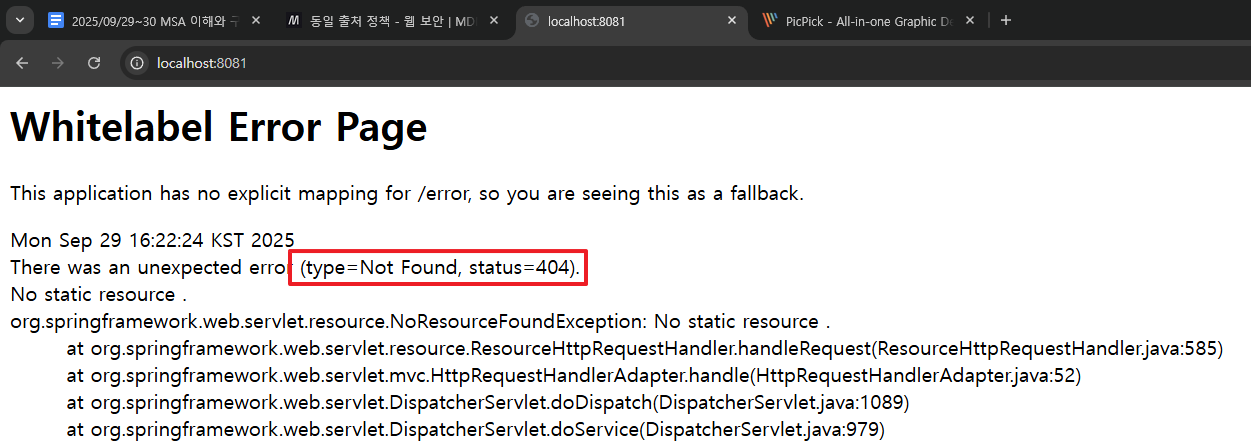
server:

port: 8081

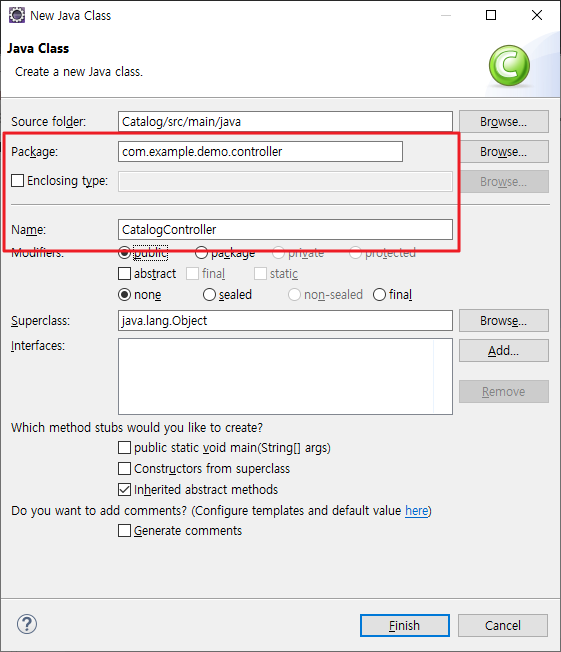
## #3 애플리케이션 실행



## #4 웹 브라우저를 이용해서 서비스 요청 ⇒ 요청을 처리할 엔드포인트가 정의되지 않았기 때문에 404 오류를 반환



## #5 컨트롤러 추가



package com.example.demo.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

**@RestController**

public class CatalogController {

**@GetMapping("/")**

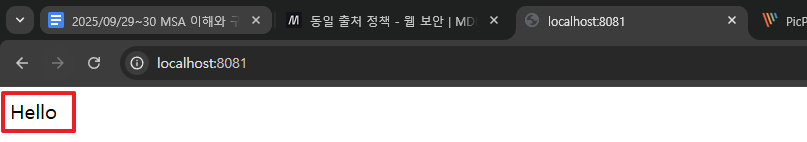
public String hello() {

return "Hello";

}

}

## #6 다시 웹 브라우저를 이용해서 요청 ⇒ 서버가 반환한 값을 출력하는 것을 확인



# REST API 테스트 도구

## GUI 기반

POST MAN

https://insomnia.rest/

https://chromewebstore.google.com/detail/talend-api-tester-free-ed/aejoelaoggembcahagimdiliamlcdmfm?hl=ko&pli=1

## CLI 기반

curl

## curl을 이용해서 서비스를 요청

C:\Users\kt ds\_210> **curl http://localhost:8081/**

Hello

C:\Users\kt ds\_210> **curl -X GET http://localhost:8081/**

Hello

## 컨트롤러에 PostMapping을 추가

package com.example.demo.controller;

import org.springframework.web.bind.annotation.GetMapping;

**import org.springframework.web.bind.annotation.PostMapping;**

import org.springframework.web.bind.annotation.RestController;

@RestController

public class CatalogController {

@GetMapping("/")

public String hello() {

return "Hello";

}

**@PostMapping("/")**

**public String world() {**

**return "World";**

**}**

}

## curl 을 이용해서 POST 요청을 테스트

C:\Users\kt ds\_210> **curl -X POST http://localhost:8081/**

World

C:\Users\kt ds\_210> **curl -X POST http://localhost:8081/ -v**

\* Trying [::1]:8081...

\* Connected to localhost (::1) port 8081 ⇐ 연결과정

> POST / HTTP/1.1 ⇐ 요청 시작 → 방식 URL 프로토콜

> Host: localhost:8081 ⇐ 요청 헤더

> User-Agent: curl/8.4.0

> Accept: \*/\*

> ⇐ 요청 헤더 끝 → 뒤에 나오는 요청 본문은 생략

< HTTP/1.1 200 ⇐ 응답 시작 → 프로토콜 상태코드 상태코드설명

< Content-Type: text/plain;charset=UTF-8 ⇐ 응답 헤더

< Content-Length: 5

< Date: Mon, 29 Sep 2025 07:38:46 GMT

< ⇐ 응답 헤더 끝

World ⇐ 응답 본문

\* Connection #0 to host localhost left intact ⇐ 연결 해제

## 요청 본문을 추가하는 경우

C:\Users\kt ds\_210> **curl -X POST http://localhost:8081/ -d "{\"name\": \"홍길동\", \"age\": 23}" -H "Content-Type: application/json" -v**

{\"name\": \"홍길동\", \"age\": 23}" -H "Content-Type: application/json" -v

Note: Unnecessary use of -X or --request, POST is already inferred.

\* Trying [::1]:8081...

\* Connected to localhost (::1) port 8081

> POST / HTTP/1.1

> Host: localhost:8081

> User-Agent: curl/8.4.0

> Accept: \*/\*

> Content-Type: application/json

> Content-Length: 32

>

< HTTP/1.1 200

< Content-Type: text/plain;charset=UTF-8

< Content-Length: 5

< Date: Mon, 29 Sep 2025 07:45:06 GMT

<

World\* Connection #0 to host localhost left intact

# 컨테이너 이미지 생성

## pom.xml 파일에 도커 이미지 생성에 필요한 정보를 추가

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.6</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>Catalog</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Catalog</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

**<docker.image.prefix>myanjini</docker.image.prefix> ⇐ 도커 허브 계정(사용자 이름)**

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<annotationProcessorPaths>

<path>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</path>

</annotationProcessorPaths>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

**<artifactId>spring-boot-maven-plugin</artifactId>**

<configuration>

**<image>**

**<name>${docker.image.prefix}/catalog-service</name>**

**</image>**

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

## 

## 명령 프롬프트에서 컨테이너 이미지를 빌드 ⇒ 프로젝트 디렉터리에서 실행

C:\Users\kt ds\_210> **cd c:\workspaces\Catalog**

c:\workspaces\Catalog> **dir pom.xml**

C 드라이브의 볼륨에는 이름이 없습니다.

볼륨 일련 번호: C27A-EB7A

c:\workspaces\Catalog 디렉터리

2025-09-29 오후 04:53 2,388 pom.xml

1개 파일 2,388 바이트

0개 디렉터리 147,583,520,768 바이트 남음

c:\workspaces\Catalog> **mvnw spring-boot:build-image**

:

[INFO] Successfully built image 'docker.io/myanjini/catalog-service:latest'

[INFO]

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESS

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 01:47 min

[INFO] Finished at: 2025-09-29T16:59:42+09:00

[INFO] ------------------------------------------------------------------------

## 이미지 생성을 확인

c:\workspaces\Catalog> **docker image ls**

REPOSITORY TAG IMAGE ID CREATED SIZE

paketobuildpacks/ubuntu-noble-run-tiny 0.0.31 864c3227dab1 9 days ago 34.7MB

myanjini/catalog-service latest fc7523f6c974 45 years ago 525MB

paketobuildpacks/builder-noble-java-tiny latest ebf7ba80b6f9 45 years ago 1.18GB

## 이미지를 이용해서 컨테이너를 실행

c:\workspaces\Catalog> **docker container run --name my-catalog-service --rm -p 8282:8081 -itd myanjini/catalog-service**

ab48dbdeae8599124a6efad7c89f299a999978930201f4a8b08afa2d04ff2cef ⇐ 컨테이너 ID

## 컨테이너 실행 상태를 확인

c:\workspaces\Catalog> **docker container ls**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ab48dbdeae85 myanjini/catalog-service "/cnb/process/web" 27 seconds ago Up 27 seconds 0.0.0.0:8282->8081/tcp, [::]:8282->8081/tcp my-catalog-service

## 컨테이너로 요청을 전달

c:\workspaces\Catalog> **curl http://localhost:8282/**

Hello

## 컨테이너를 추가로 실행

c:\workspaces\Catalog> **docker container run --name my-catalog-service2 --rm -p 8283:8081 -itd myanjini/catalog-service**

cb93bf7f9abc31d7fcdc1073d26248d21a35743cf5ca218d3208591b4c05d9d7

c:\workspaces\Catalog> **docker container ls**

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

cb93bf7f9abc myanjini/catalog-service "/cnb/process/web" 5 seconds ago Up 5 seconds 0.0.0.0:8283->8081/tcp, [::]:8283->8081/tcp my-catalog-service2

ab48dbdeae85 myanjini/catalog-service "/cnb/process/web" 2 minutes ago Up 2 minutes 0.0.0.0:8282->8081/tcp, [::]:8282->8081/tcp my-catalog-service

c:\workspaces\Catalog> **curl http://localhost:8283/**

Hello

## 도커 허브에 이미지를 등록

c:\workspaces\Catalog> **docker image push myanjini/catalog-service**

Using default tag: latest

The push refers to repository [docker.io/myanjini/catalog-service]

: ~~~~~~~~~ ~~~~~~~~~ ~~~~~~~~~~~~~~

| | |

| | +-- image name (tag 생략 = latest)

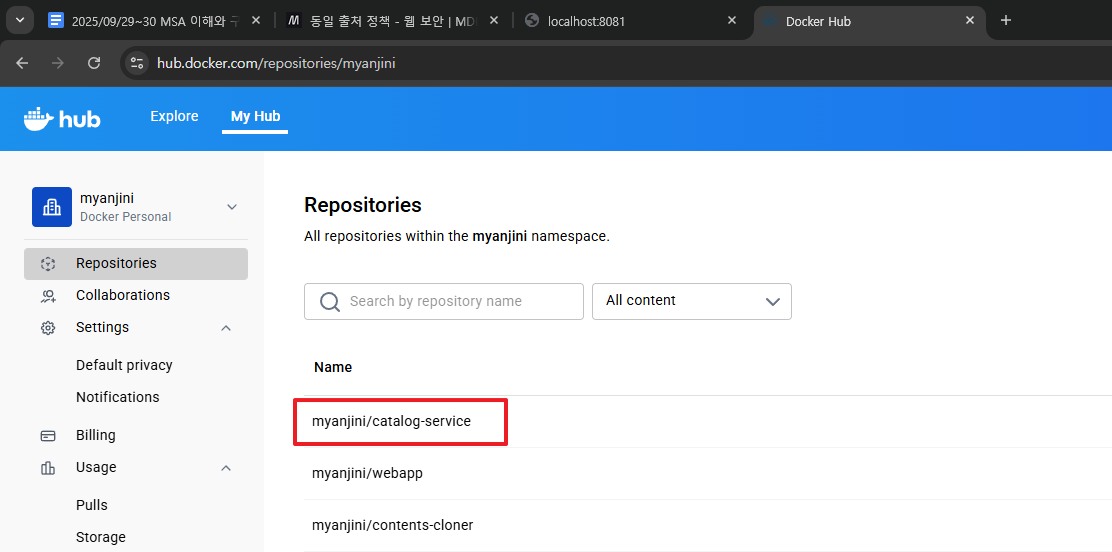
| +-- repository (registry 계정명) ⇐ docker login 시 부여 받은 액세스 키를

+-- registry 이용해서 자격 증명

a1309ce57d8b: Mounted from paketobuildpacks/ubuntu-noble-run-tiny

latest: digest: sha256:25e8f319e0693d7bff05f9f7005da286ebca60f2a1460ff3d58ffe4d7caebc80 size: 3431

## 도커 허브에서 확인 ⇒ https://hub.docker.com



## 도커 허브에 등록된 이미지를 이용해서 컨테이너를 실행

c:\workspaces\Catalog> **docker container run --name your-catalog-service --rm -p 9999:8081 -itd myanjini/catalog-service**

c:\workspaces\Catalog> **curl http://localhost:9999/**

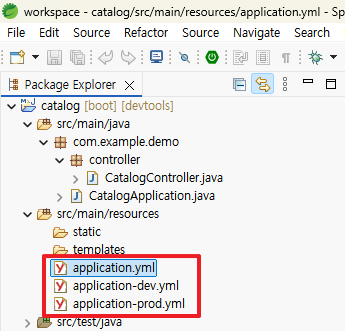
Hello ^..^

# 스프링 프로파일

환경(개발, 테스트, 운영 등)에 따라 서로 다른 설정이나 Bean을 적용할 수 있도록 도와주는 환경 분리 기능

* 환경별 설정을 분리해 유지보수를 쉽게 하기 위해
* 운영/개발/테스트에서 서로 다른 데이터베이스, 포트, 로그 설정 등을 적용
* 코드나 설정을 조건적으로 활성화

## 설정 파일 분리



application.yml (.properties)

application-dev.yml

application-prod.yml

~~~~

프로파일 이름

## application.yml

spring:

application:

name: **catalog-service**

server:

port: 8081

## application-dev.yml

spring:

application:

name: **catalog-service-DEV**

**custom:**

**name: develop**

## application-prod.yml

**custom:**

**name: production**

## 설정 파일을 참조하도록 컨트롤러 메서드 추가

package com.example.demo.controller;

**import org.springframework.beans.factory.annotation.Value;**

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class CatalogController {

@GetMapping("/")

public String hello() {

return "Hello ^..^";

}

@PostMapping("/")

public String world() {

return "World";

}

**@Value("${spring.application.name}")**

**private String applicationName;**

**@Value("${custom.name:anonymouse}")**

**private String customName;**

**@GetMapping("/whoami")**

**public String profileTest() {**

**return customName + "@" + applicationName;**

**}**

}

## 테스트

D:\workspace> **curl -X GET http://localhost:8081/whoami**

anonymouse@catalog-service

~~~~~~~~~~ ~~~~~~~~~~~~~~~

| |

| +-- application.yml 파일의 설정값

+-- @Value("${custom.name:anonymouse}") 기본값

## application.yml 파일에 프로파일을 지정

spring:

application:

name: catalog-service

**profiles:**

**active:**

**- dev**

server:

port: 8081

## 테스트

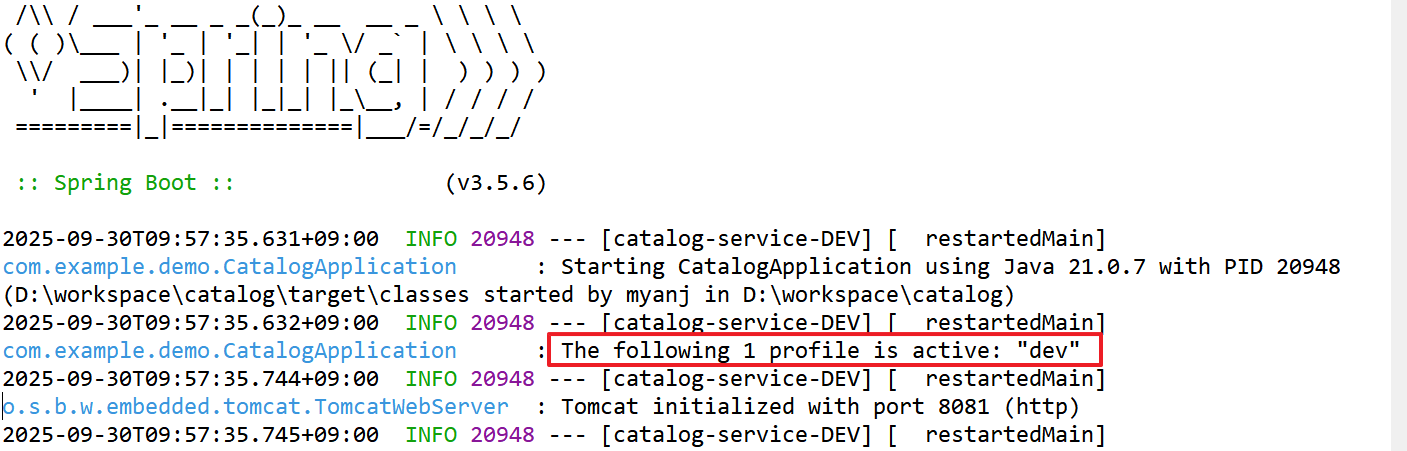
D:\workspace> **curl -X GET http://localhost:8081/whoami**

develop@catalog-service-DEV

~~~~~~~ ~~~~~~~~~~~~~~~~~~~

| |

+-------+---- application-dev.yml 파일의 설정값



## maven의 spring-boot:run 플러그인을 실행할 때 -D 옵션을 사용해 시스템 프로퍼티 전달

D:\workspace\catalog>

~~~~~~~~~~~~~~~~~~~~

프로젝트 디렉터리

D:\workspace\catalog> **mvnw spring-boot:run -Dspring-boot.run.profiles=prod -Dspring-boot.run.arguments="--server.port=8082"**

:

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v3.5.6)

2025-09-30T10:06:29.557+09:00 INFO 6476 --- [catalog-service] [ restartedMain] com.example.demo.CatalogApplication : Starting CatalogApplication using Java 21.0.8 with PID 6476 (D:\workspace\catalog\target\classes started by myanj in D:\workspace\catalog)

2025-09-30T10:06:29.560+09:00 INFO 6476 --- [catalog-service] [ restartedMain] com.example.demo.CatalogApplication : The following 1 profile is active: "prod"

2025-09-30T10:06:29.603+09:00 INFO 6476 --- [catalog-service] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable

2025-09-30T10:06:29.603+09:00 INFO 6476 --- [catalog-service] [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'

2025-09-30T10:06:30.285+09:00 INFO 6476 --- [catalog-service] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8082 (http)

2025-09-30T10:06:30.297+09:00 INFO 6476 --- [catalog-service] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2025-09-30T10:06:30.298+09:00 INFO 6476 --- [catalog-service] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]

2025-09-30T10:06:30.336+09:00 INFO 6476 --- [catalog-service] [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

2025-09-30T10:06:30.337+09:00 INFO 6476 --- [catalog-service] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 733 ms

2025-09-30T10:06:30.617+09:00 WARN 6476 --- [catalog-service] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : Unable to start LiveReload server

2025-09-30T10:06:30.645+09:00 INFO 6476 --- [catalog-service] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8082 (http) with context path '/'

2025-09-30T10:06:30.651+09:00 INFO 6476 --- [catalog-service] [ restartedMain] com.example.demo.CatalogApplication : Started CatalogApplication in 1.531 seconds (process running for 1.878)

## 테스트

C:\Users\myanj> **curl -X GET http://localhost:8082/whoami**

production@catalog-service

~~~~~~~~~~ ~~~~~~~~~~~~~~~

| |

| +-- application.yml 파일의 설정값

+-- application-prod.yml 파일의 설정값

## java 명령어를 사용해서 이미 빌드된 JAR 파일을 실행할 때 -- 옵션을 사용해 애플리케이션 인자로 전달

d:\workspace\catalog>

d:\workspace\catalog> **mvnw clean package**

d:\workspace\catalog> **dir .\target**

D 드라이브의 볼륨: work

볼륨 일련 번호: 48EA-D1C2

d:\workspace\catalog\target 디렉터리

2025-09-30 오전 10:13 <DIR> .

2025-09-30 오전 10:13 <DIR> ..

2025-09-30 오전 10:13 20,997,283 catalog-0.0.1-SNAPSHOT.jar

2025-09-30 오전 10:13 4,277 catalog-0.0.1-SNAPSHOT.jar.original

2025-09-30 오전 10:13 <DIR> classes

2025-09-30 오전 10:13 <DIR> generated-sources

2025-09-30 오전 10:13 <DIR> generated-test-sources

2025-09-30 오전 10:13 <DIR> maven-archiver

2025-09-30 오전 10:13 <DIR> maven-status

2025-09-30 오전 10:13 <DIR> surefire-reports

2025-09-30 오전 10:13 <DIR> test-classes

2개 파일 21,001,560 바이트

9개 디렉터리 237,546,639,360 바이트 남음

d:\workspace\catalog> **java -jar .\target\catalog-0.0.1-SNAPSHOT.jar --spring.profiles.active=prod --server.port=8083**

. \_\_\_\_ \_ \_\_ \_ \_

/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \

( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \

\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )

' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /

=========|\_|==============|\_\_\_/=/\_/\_/\_/

:: Spring Boot :: (v3.5.6)

2025-09-30T10:16:17.691+09:00 INFO 31932 --- [catalog-service] [ main] com.example.demo.CatalogApplication : Starting CatalogApplication v0.0.1-SNAPSHOT using Java 21.0.8 with PID 31932 (D:\workspace\catalog\target\catalog-0.0.1-SNAPSHOT.jar started by myanj in d:\workspace\catalog)

2025-09-30T10:16:17.695+09:00 INFO 31932 --- [catalog-service] [ main] com.example.demo.CatalogApplication : The following 1 profile is active: "prod"

2025-09-30T10:16:18.644+09:00 INFO 31932 --- [catalog-service] [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8083 (http)

2025-09-30T10:16:18.659+09:00 INFO 31932 --- [catalog-service] [ main] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2025-09-30T10:16:18.659+09:00 INFO 31932 --- [catalog-service] [ main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]

2025-09-30T10:16:18.696+09:00 INFO 31932 --- [catalog-service] [ main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

2025-09-30T10:16:18.698+09:00 INFO 31932 --- [catalog-service] [ main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 943 ms

2025-09-30T10:16:19.111+09:00 INFO 31932 --- [catalog-service] [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8083 (http) with context path '/'

2025-09-30T10:16:19.132+09:00 INFO 31932 --- [catalog-service] [ main] com.example.demo.CatalogApplication : Started CatalogApplication in 1.944 seconds (process running for 2.465)

## 테스트

C:\Users\myanj> **curl -X GET http://localhost:8083/whoami**

production@catalog-service

## Java 명령어를 사용해 이미 빌드된 JAR 파일을 실행할 때 -D 옵션을 사용해 시스템 프로퍼티를 전달

D:\workspace\catalog>

D:\workspace\catalog> **java -Dspring.profiles.active=prod -Dserver.port=8084 -jar .\target\catalog-0.0.1-SNAPSHOT.jar**

2025-09-30T10:21:55.989+09:00 INFO 5856 --- [catalog-service] [ main] com.example.demo.CatalogApplication : Starting CatalogApplication v0.0.1-SNAPSHOT using Java 21.0.8 with PID 5856 (D:\workspace\catalog\target\catalog-0.0.1-SNAPSHOT.jar started by myanj in D:\workspace\catalog)

2025-09-30T10:21:55.992+09:00 INFO 5856 --- [catalog-service] [ main] com.example.demo.CatalogApplication : The following 1 profile is active: "prod"

2025-09-30T10:21:57.002+09:00 INFO 5856 --- [catalog-service] [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8084 (http)

2025-09-30T10:21:57.018+09:00 INFO 5856 --- [catalog-service] [ main] o.apache.catalina.core.StandardService : Starting service [Tomcat]

2025-09-30T10:21:57.018+09:00 INFO 5856 --- [catalog-service] [ main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.46]

2025-09-30T10:21:57.055+09:00 INFO 5856 --- [catalog-service] [ main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

2025-09-30T10:21:57.058+09:00 INFO 5856 --- [catalog-service] [ main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1005 ms

2025-09-30T10:21:57.467+09:00 INFO 5856 --- [catalog-service] [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8084 (http) with context path '/'

2025-09-30T10:21:57.484+09:00 INFO 5856 --- [catalog-service] [ main] com.example.demo.CatalogApplication : Started CatalogApplication in 2.025 seconds (process running for 2.518)

# 잠시 쉬고, 10시 35분에 이어서 하겠습니다.

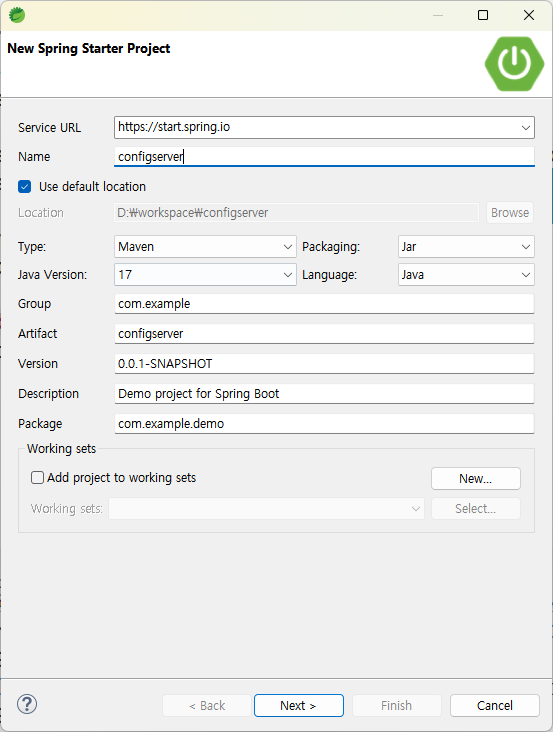
# 스프링 컨피그 서버 구축

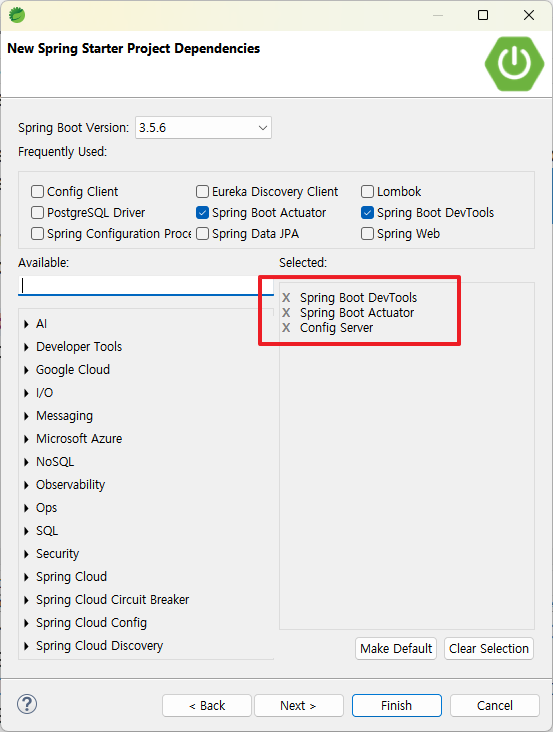
스프링 컨피그 서버는 분산 시스템, 특히 마이크로서비스 아키텍처에서 여러 애플리케이션의 설정 정보를 중앙에서 관리하고 제공하는 역할을 하는 서버입니다. 각 마이크로서비스는 독립적인 설정 파일을 가지는 대신, 이 중앙 집중식 컨피그 서버로부터 필요한 설정 정보를 가져옵니다.

## 주요 기능 및 특징

* **중앙 집중식 관리**: 모든 애플리케이션의 설정 정보를 한곳에서 관리하여, 각 서비스의 설정 파일을 개별적으로 관리해야 하는 번거로움을 없애줍니다.
* **버전 관리**: 설정 정보는 기본적으로 Git 리포지토리에 저장되므로, 설정 변경 내역을 추적하고 필요에 따라 롤백하는 등 버전 관리가 용이합니다.
* **환경별 설정 분리**: 개발, 테스트, 운영 환경별로 다른 설정 파일을 관리할 수 있습니다.
* **동적 업데이트**: 서비스 재배포 없이 설정 파일을 변경하고, 변경된 설정을 즉시 적용할 수 있습니다. Spring Cloud Bus와 같은 도구를 함께 사용하면 여러 인스턴스에 변경 사항을 한 번에 전파할 수 있습니다.
* **외부 저장소 사용**: Git 외에도 파일 시스템, HashiCorp Vault 등 다양한 외부 저장소를 백엔드로 사용할 수 있습니다.

## 프로젝트 생성





## 부트스트랩 클래스에 @EnableConfigServer 어노테이션을 추가

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

**import org.springframework.cloud.config.server.EnableConfigServer;**

**@EnableConfigServer**

@SpringBootApplication

public class ConfigserverApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigserverApplication.class, args);

}

}

## 컨피그 서버 설정 ⇒ application.properties → .yml ⇒ 클래스 패스에 설정 파일을 저장하도록 설정

spring:

application:

name: config-server

profiles:

active: **native**

cloud:

config:

server:

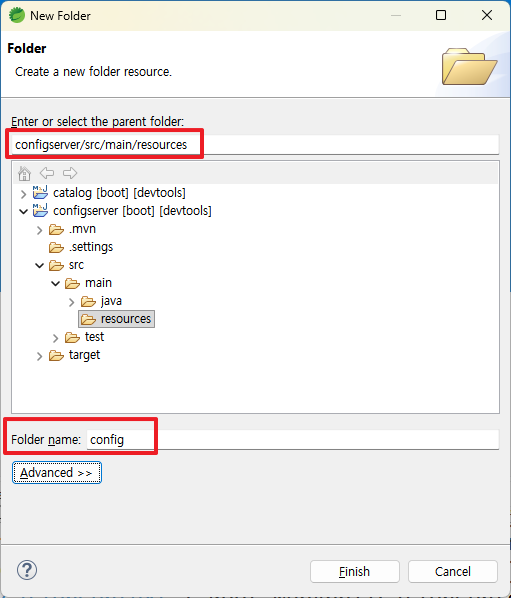
native:

search-locations: classpath:/config

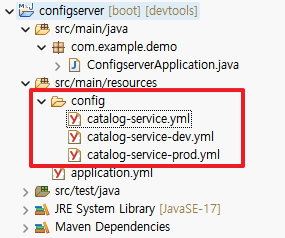
server:

port: 8071

## 설정 파일을 저장할 폴더를 추가 ⇒ src\main\resources\config\



## catalog 서비스의 설정 파일(3개)을 복사해서 catalog-service\*.yml 파일로 저장



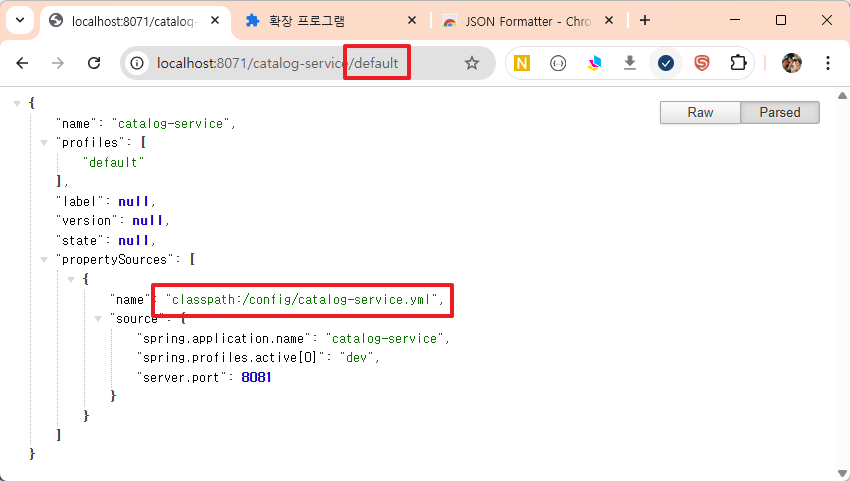
## 컨피그 서버를 구동 후 설정 정보를 요청

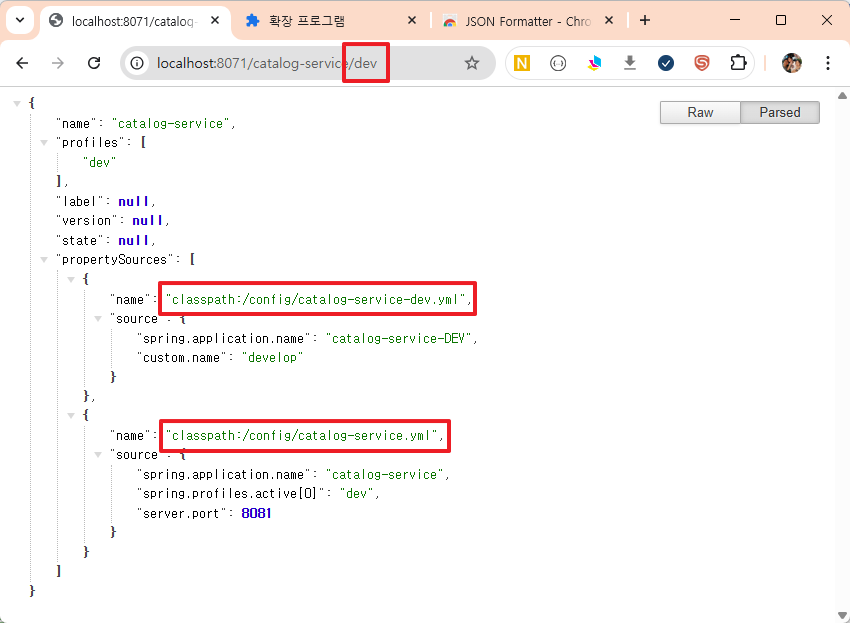
http://컨피그서버주소/어플리케이션이름/프로파일/라벨(옵션)

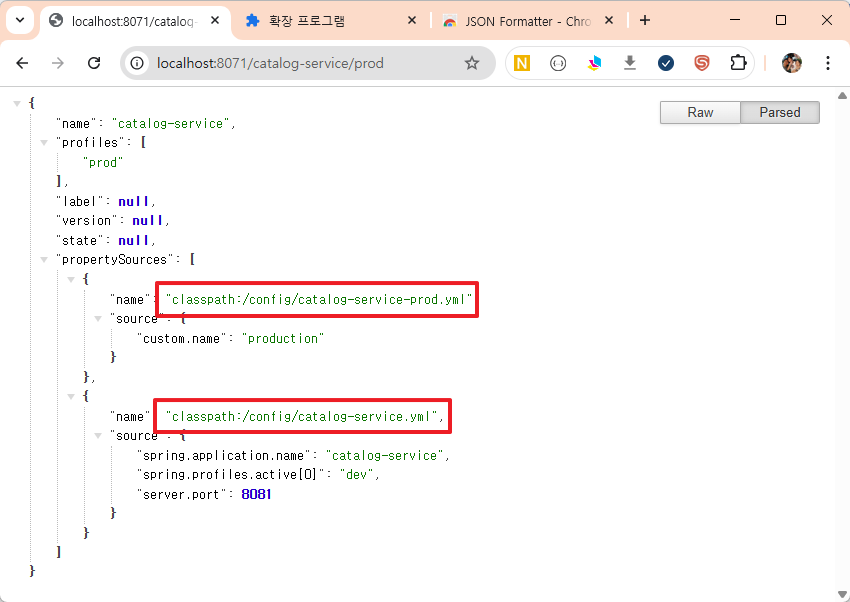
http://localhost:8071/catalog-service/default

http://localhost:8071/catalog-service/dev

http://localhost:8071/catalog-service/prod







## 설정 파일을 컨피그 서버의 특정 디렉터리에 저장하도록 변경

spring:

application:

name: config-server

profiles:

active: native

cloud:

config:

server:

native:

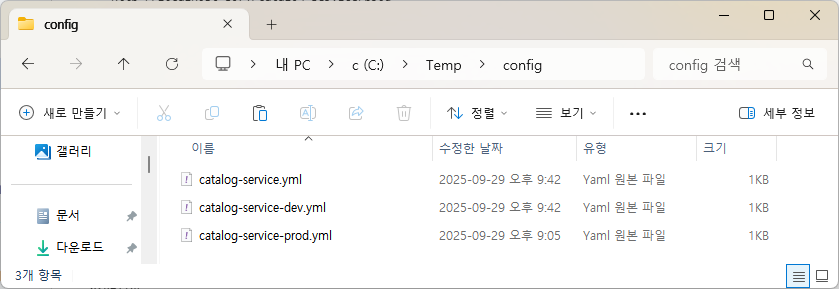
**# search-locations: classpath:/config**

**search-locations: file:///c:\temp\config**

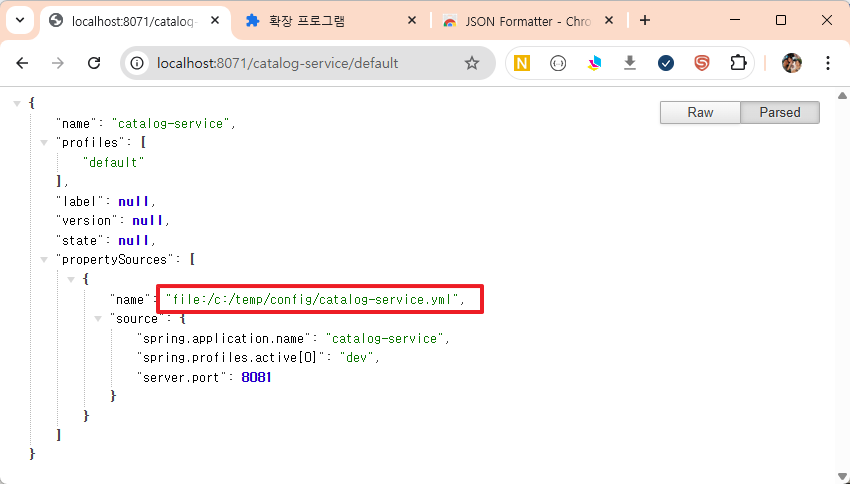
server:

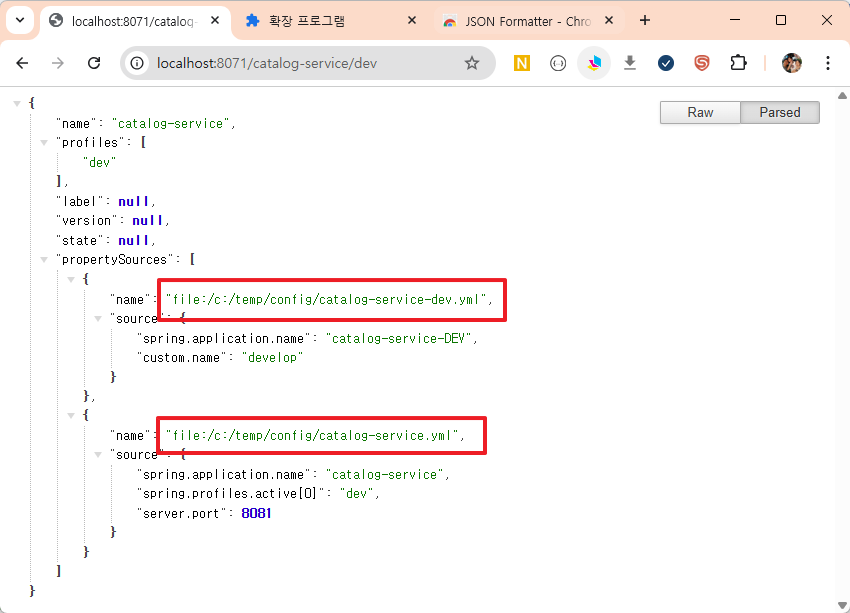
port: 8071

## c:\temp\config 디렉터리를 만들고 카탈로그 서비스의 설정 파일을 복사



## 테스트

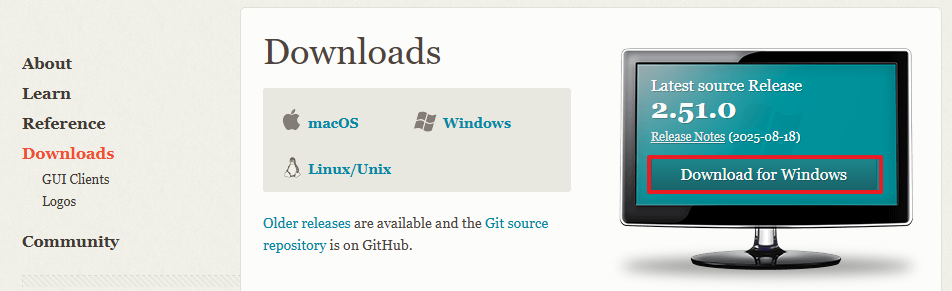




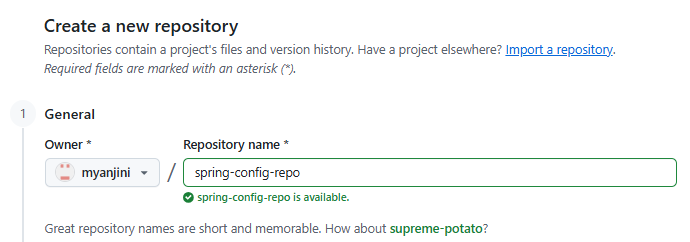


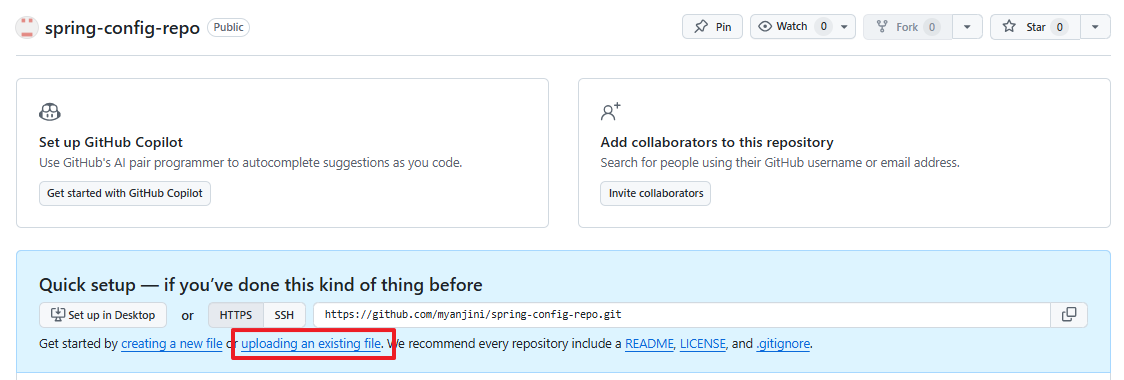
## 깃 설치

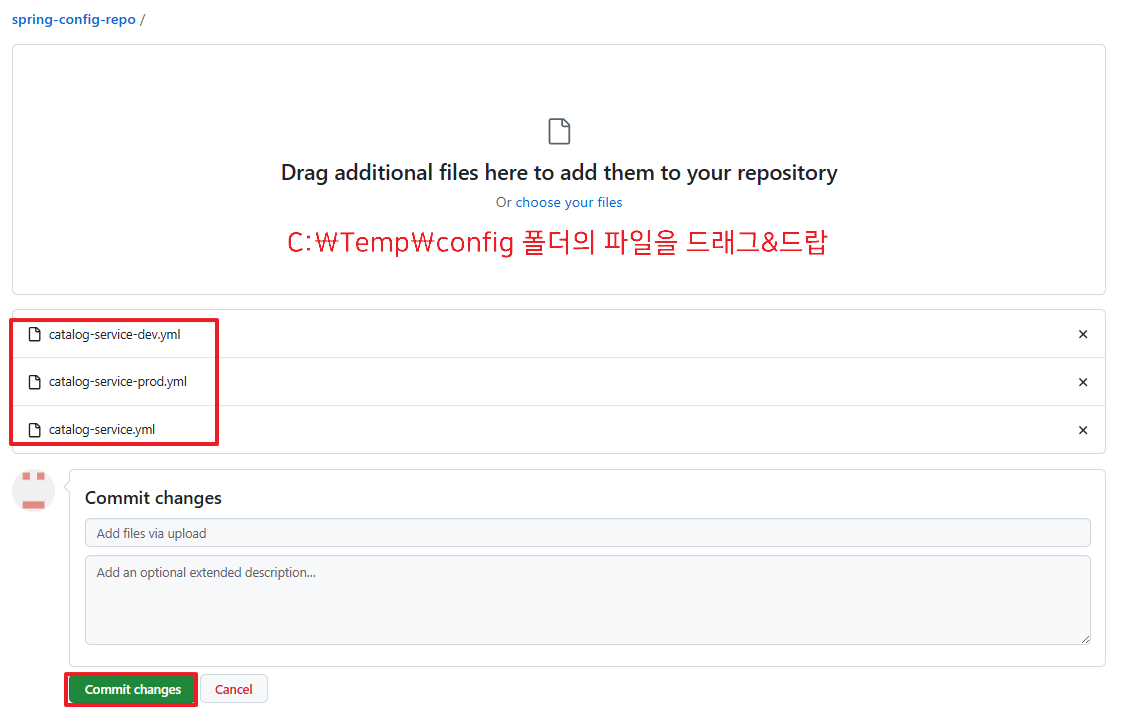
https://git-scm.com/downloads

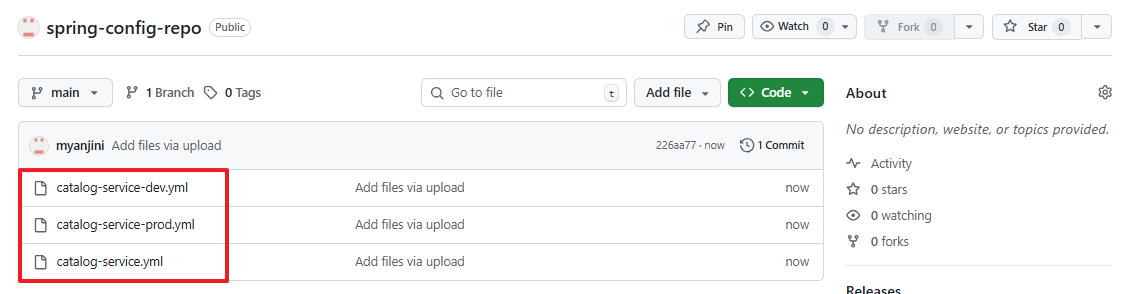


## 깃허브에 spring-config-repo 레포지터리를 생성하고 설정 파일을 저장









## 컨피그 서버 설정을 변경

spring:

application:

name: config-server

profiles:

**# active: native**

**active: git**

cloud:

config:

server:

native:

# search-locations: classpath:/config

search-locations: file:///c:\temp\config

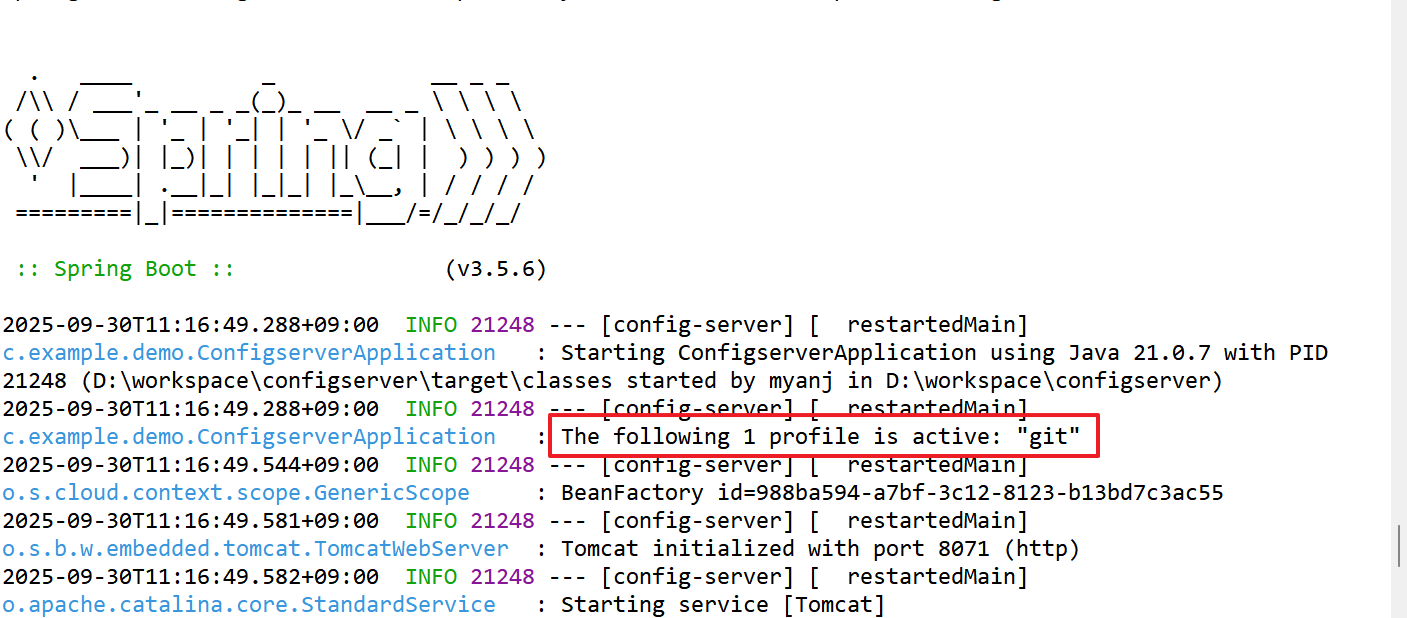
**git:**

**uri: https://github.com/myanjini/spring-config-repo.git**

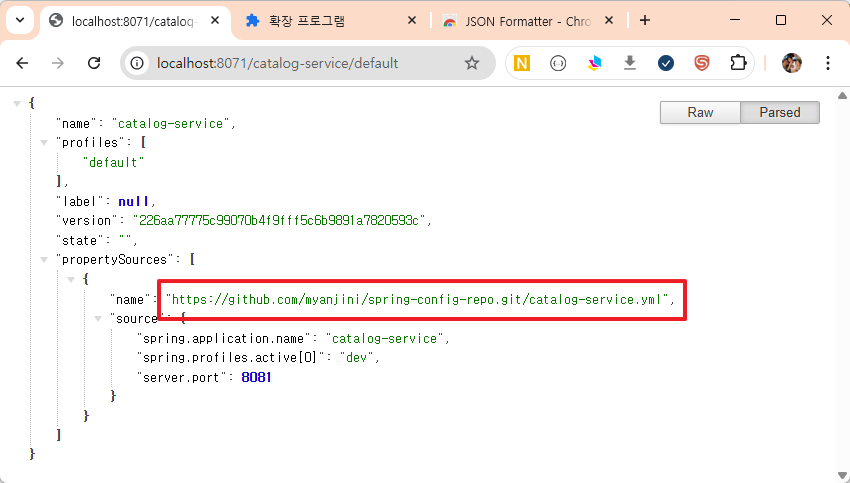
server:

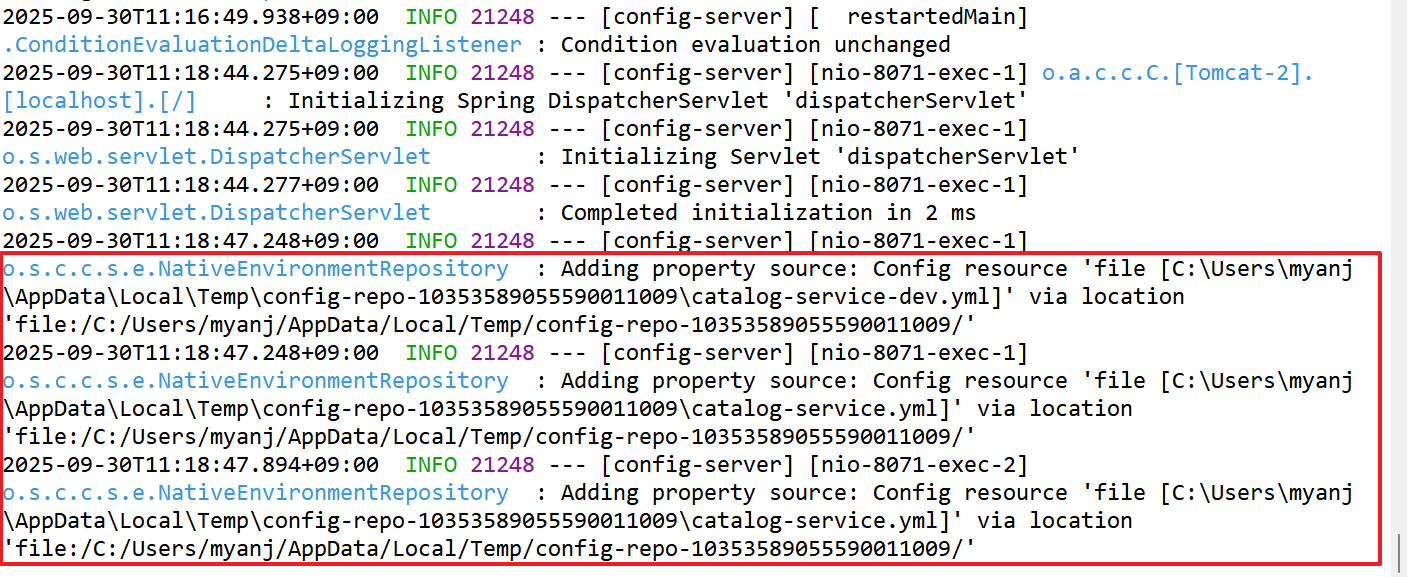
port: 8071

## 컨피그 서버 시작 시 로그를 확인



## 설정 정보를 요청했을 때 컨피그 서버의 로그를 확인

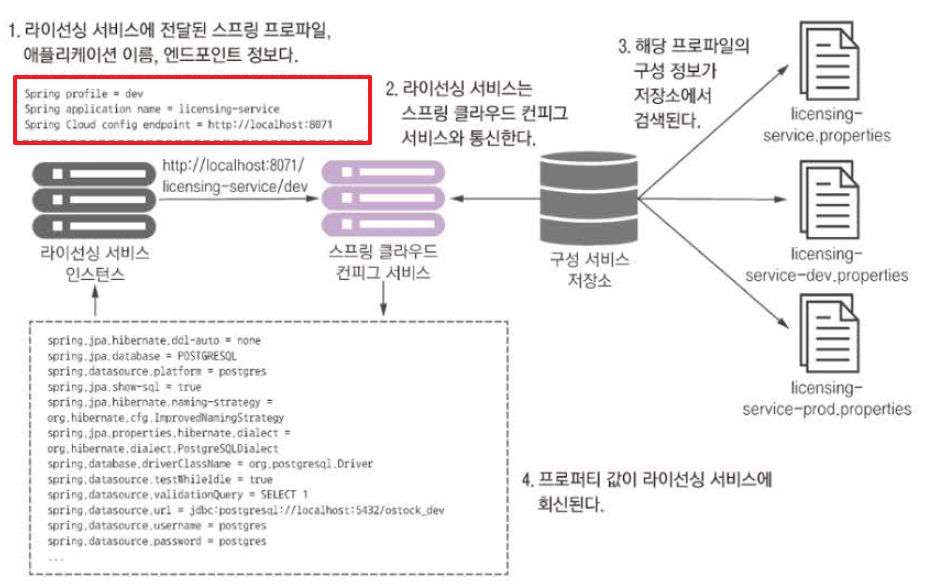




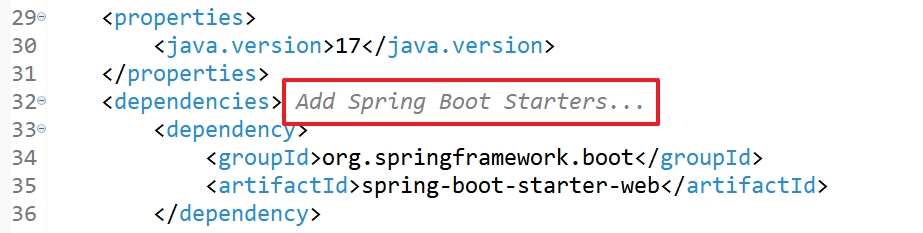
참고 ⇒ https://docs.spring.io/spring-cloud-config/docs/current/reference/html/#\_environment\_repository

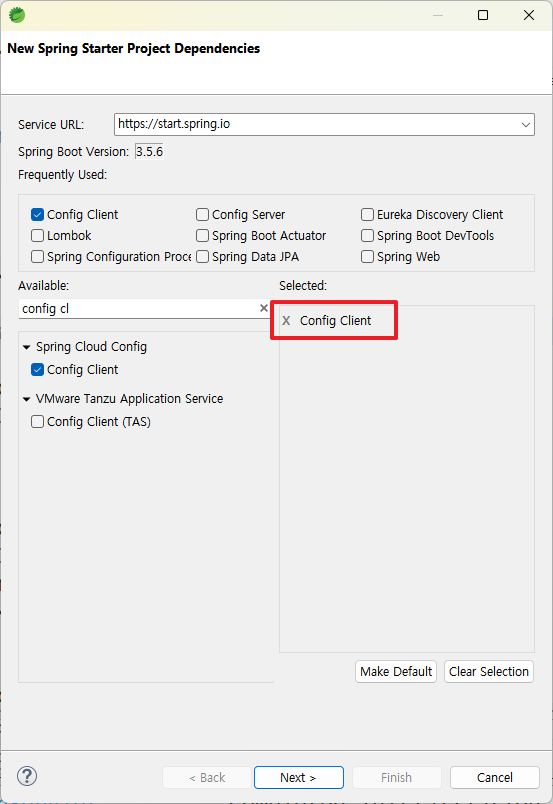
# 점심 식사하고, 13시에 이어서 하겠습니다.

# 스프링 클라우드 컨피그와 스프링 부트 클라이언트를 통합



## 카탈로그 서비스에 의존을 추가





<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.6</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>catalog</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>catalog</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

**<spring-cloud.version>2025.0.0</spring-cloud.version>**

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-config</artifactId>**

**</dependency>**

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<annotationProcessorPaths>

<path>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</path>

</annotationProcessorPaths>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

**<dependencyManagement>**

**<dependencies>**

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-dependencies</artifactId>**

**<version>${spring-cloud.version}</version>**

**<type>pom</type>**

**<scope>import</scope>**

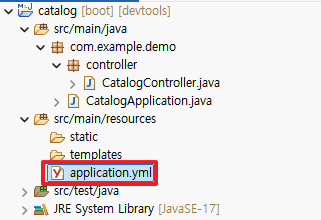
**</dependency>**

**</dependencies>**

**</dependencyManagement>**

</project>

## 카탈로그 서비스의 설정 파일(application.yml)을 수정하고, -dev.yml, -prod.yml 파일을 삭제



spring:

application:

name: catalog-service

profiles:

active:

- dev

**config:**

**import: "optional:configserver:http://localhost:8071"**

**# ~~~~~~ ~~~~~~~~ ~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~~~~~**

**# | | | 컨피그 서버의 엔드포인트**

**# | | +-- Spring Cloud Config Server**

**# | +-- 컨피그 서버가 비활성화된 경우 로컬 설정을 사용해서 실행**

**# +-- Spring Boot 2.4 부터 spring.cloud.config.uri 대신 사용**

server:

port: 8081

## 깃허브에 등록된 catalog-service.yml 파일에 profiles를 삭제

spring:

application:

name: catalog-service

**# profiles:**

**# active:**

**# - dev**

server:

port: 8081

## 컨피그 서버 구동 → 카탈로그 서비스 구동 → 테스트

c:\Temp> **curl -X GET http://localhost:8081/whoami**

develop@catalog-service-DEV

# 스프링 부트의 기본 설정 우선 순위

#1(우선순위 낮음) <<<<< #7(우선순위 높음)

## #1 Spring Boot 기본 설정

spring:

application:

name: catalog-service

# profiles:

# active:

# - dev

config:

import: "optional:configserver:http://localhost:8071"

#server:

# port: 8081

**⇒ server.port를 지정하지 않으면 기본 서비스 포트 8080을 사용**

## #2 로컬 설정 파일 ⇒ application.yml

spring:

application:

name: catalog-service

# profiles:

# active:

# - dev

config:

import: "optional:configserver:http://localhost:8071"

**server:**

**port: 8081**

**⇒ 8081 포트로 구동**

## #3 프로파일이 적용된 로컬 설정 파일

spring:

application:

name: catalog-service

**profiles:**

**active:**

**- dev**

config:

import: "optional:configserver:http://localhost:8071"

server:

port: 8081

**⇒ 애플리케이션 이름이 catalog-service-DEV로 설정 ⇐ application.yml 내용을 application-dev.yml 내용이 덮어씀**

## #4 Spring Cloud Config 서버에서 가져온 설정

**깃 허브에 저장된 catalog-service.yml 파일의 내용을 수정**

spring:

application:

name: catalog-service

# profiles:

# active:

# - dev

server:

port: **8082**

**깃 허브에 저장된 catalog-service-Dev.yml 파일의 내용을 수정**

spring:

application:

name: catalog-service-DEV

custom:

name: development

**server:**

**port: 8083**

## #5 OS 환경 변수

### 배포 파일 생성 → 로컬 설정 파일이 있고, Config 서버로부터 설정 파일을 가져오도록 설정한 상태로 Jar 파일을 생성

D:\workspace\catalog> **mvnw clean package**

D:\workspace\catalog> **dir .\target**

D 드라이브의 볼륨: work

볼륨 일련 번호: 48EA-D1C2

D:\workspace\catalog\target 디렉터리

2025-09-30 오후 01:51 <DIR> .

2025-09-30 오후 01:50 <DIR> ..

2025-09-30 오후 01:51 32,444,452 catalog-0.0.1-SNAPSHOT.jar

2025-09-30 오후 01:51 4,094 catalog-0.0.1-SNAPSHOT.jar.original

2025-09-30 오후 01:50 <DIR> classes

2025-09-30 오후 01:50 <DIR> generated-sources

2025-09-30 오후 01:50 <DIR> generated-test-sources

2025-09-30 오후 01:51 <DIR> maven-archiver

2025-09-30 오후 01:50 <DIR> maven-status

2025-09-30 오후 01:51 <DIR> surefire-reports

2025-09-30 오후 01:50 <DIR> test-classes

2개 파일 32,448,546 바이트

9개 디렉터리 237,513,289,728 바이트 남음

### STS에서 카탈로그 서비스를 중지하고, Config 서버만 실행

### SERVER\_PORT 환경변수의 값으로 8084를 설정

D:\workspace\catalog> **set SERVER\_PORT=8084**

D:\workspace\catalog> **echo %SERVER\_PORT%**

8084

### (환경 변수를 설정한 같은 터미널에서) 애플리케이션 실행

D:\workspace\catalog> **java -jar .\target\catalog-0.0.1-SNAPSHOT.jar**

**⇒ 8084 포트로 기동되는 것을 확인**

## #6 JVM 시스템 프로퍼티 (-D이름=값) ⇐ 환경변수 설정한 창에서 실행

d:\workspace\catalog> **java -Dserver.port=8085 -jar .\target\catalog-0.0.1-SNAPSHOT.jar**

**⇒ 8085 포트로 기동되는 것을 확인**

## #7 Command-Line 인자

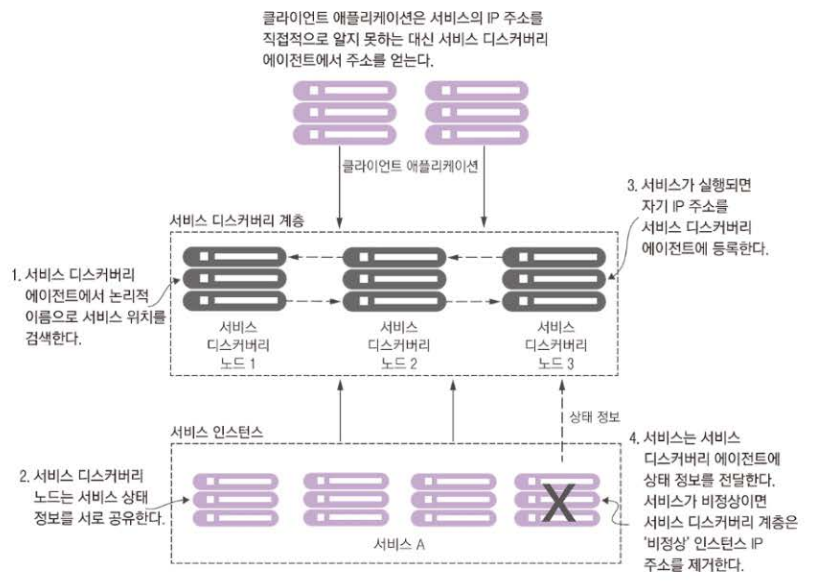
D:\workspace\catalog> **java -Dserver.port=8085 -jar .\target\catalog-0.0.1-SNAPSHOT.jar --server.port=8086**

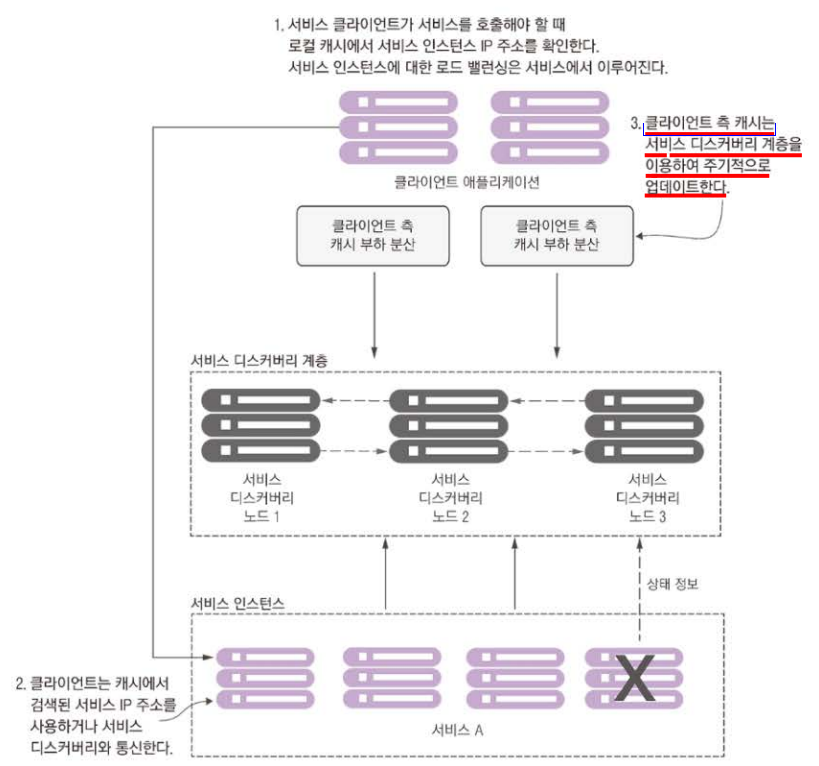
**⇒ 8086 포트로 기동되는 것을 확인**

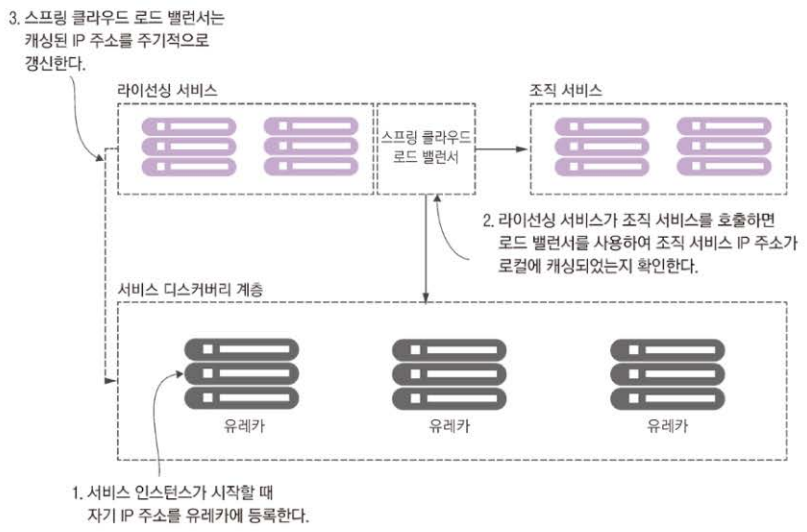
# 잠시 쉬고 14시 15분에 이어서 하겠습니다.

# 서비스 디스커버리(Service Discovery)

* 분산 시스템이나 마이크로서비스 아키텍처에서 각 서비스의 위치(IP 주소, 포트 등)를 자동으로 찾고 통신할 수 있도록 도와주는 메커니즘
* 스프링 클라우드와 넷플릭스 유레카의 서비스 디스커버리 엔진을 사용해 서비스 디스커버리 패턴을 구현
* 클라이언트 측 로드 밸런싱을 위해 스프링 클라우드 로드 밸런서를 사용

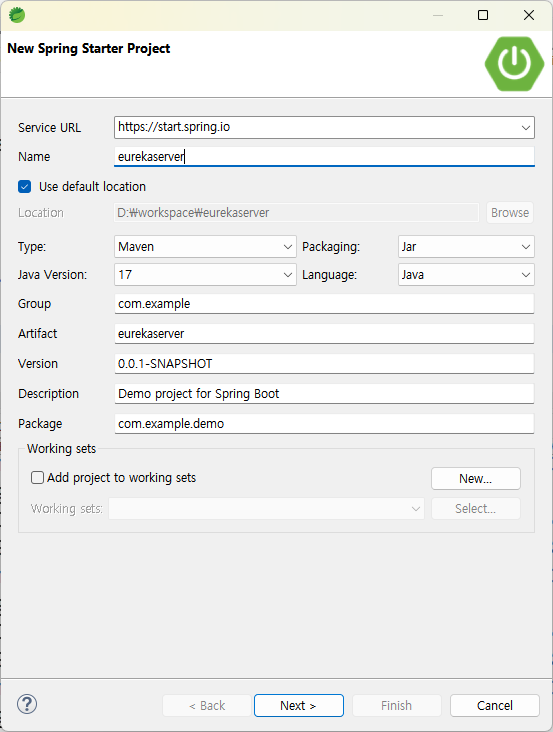


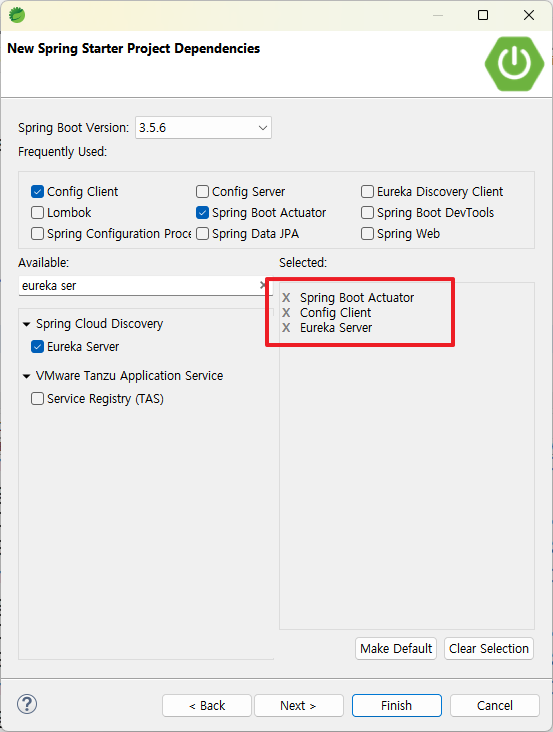




## 유레카 서버 구축

### 프로젝트 생성





### application.properties → application.yml ⇒ Config 서버 정보를 등록

spring:

application:

name: eureka-server

**config:**

**import: "optional:configserver:http://localhost:8071"**

### 깃 허브에 유레카 서버 설정 파일을 추가



server:

port: 8070

eureka:

instance:

hostname: localhost

client:

registerWithEureka: false

fetchRegistry: false

serviceUrl:

defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/

server:

waitTimeInMsWhenSyncEmpty: 5

### 유레카 서버의 부트 스트랩 클래스에 @EnableEurekaServer 어노테이션을 추가

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

**import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;**

**@EnableEurekaServer**

@SpringBootApplication

public class EurekaserverApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaserverApplication.class, args);

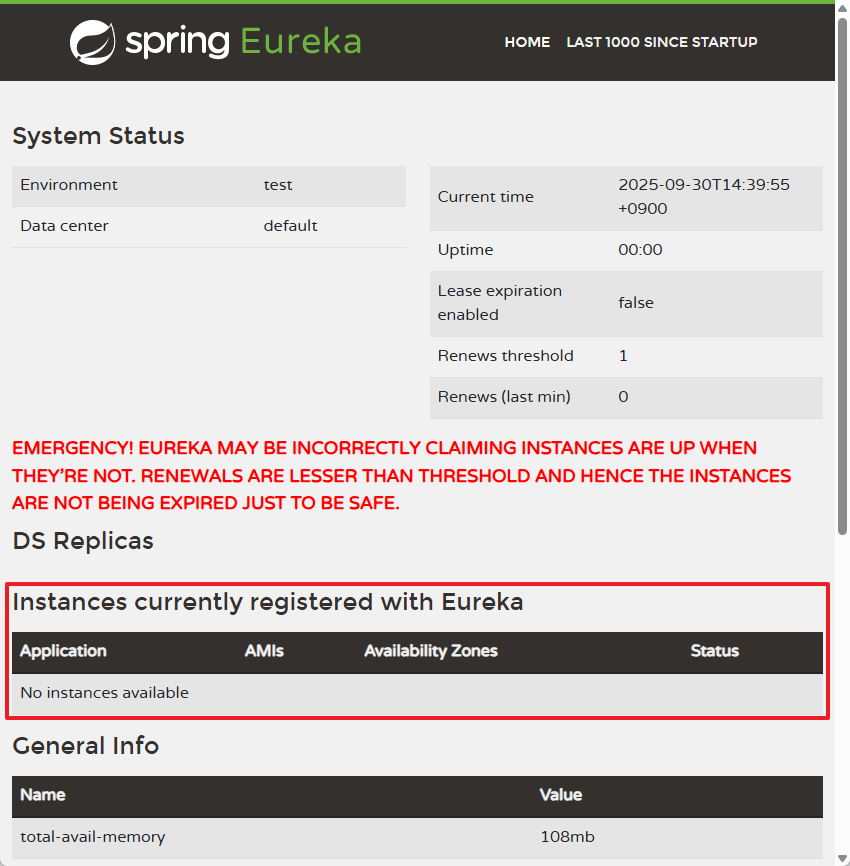
}

}

## 테스트

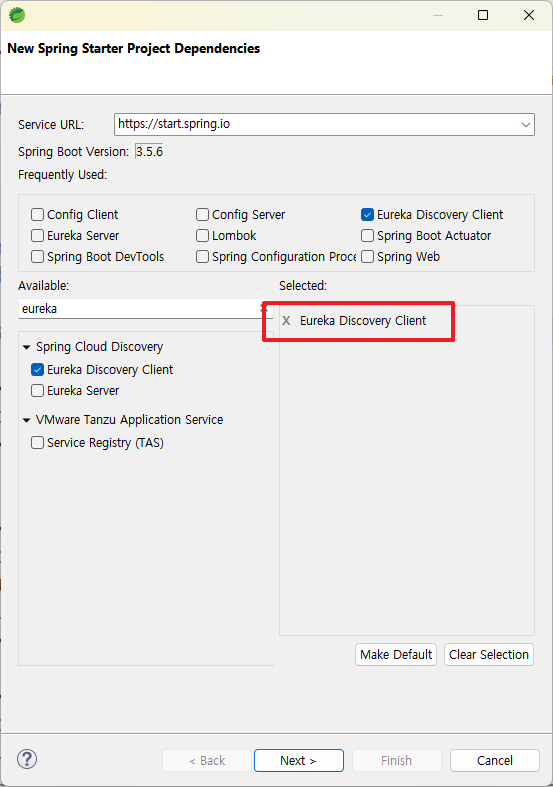
컨피그 서버 기동 → 유레카 서버 기동 → 카탈로그 서비스 기동

## 유레카 서버 대시보드로 접속 ⇒ http://localhost:8070 로 접속



## 카탈로그 서비스에 유레카 클라이언트 기능을 추가

### pom.xml 파일에 유레카 클라이언트 의존을 추가



<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.6</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>catalog</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>catalog</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

<spring-cloud.version>2025.0.0</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-config</artifactId>

</dependency>

**<dependency>**

**<groupId>org.springframework.cloud</groupId>**

**<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>**

**</dependency>**

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<annotationProcessorPaths>

<path>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</path>

</annotationProcessorPaths>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

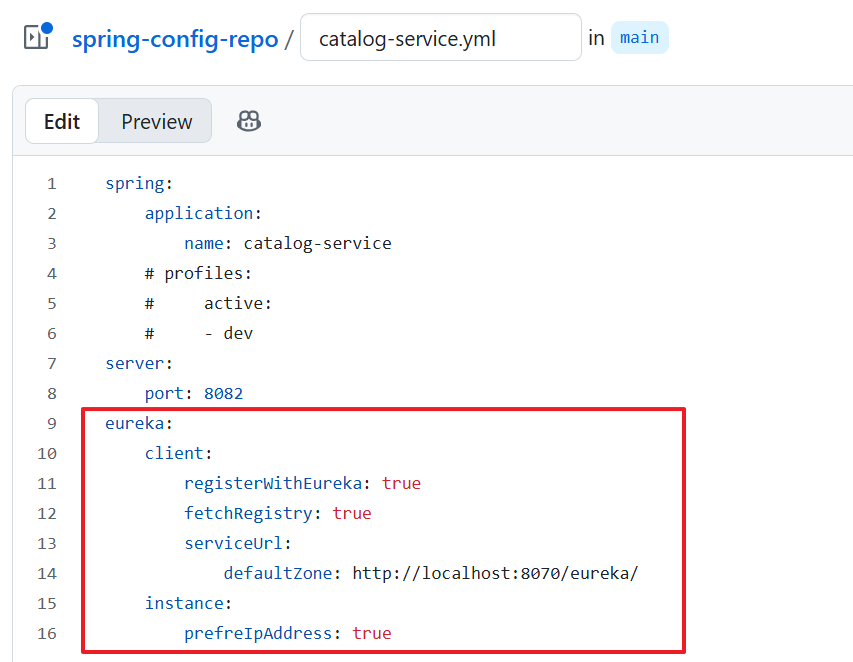
</dependency>

</dependencies>

</dependencyManagement>

</project>

### 카탈로그 서비스의 설정 파일에 유레카 클라이언트 설정을 추가 ⇒ 깃허브에서



spring:

application:

name: catalog-service

# profiles:

# active:

# - dev

server:

port: 8082

**eureka:**

**client:**

**registerWithEureka: true**

**fetchRegistry: true**

**serviceUrl:**

**defaultZone: http://localhost:8070/eureka/**

**instance:**

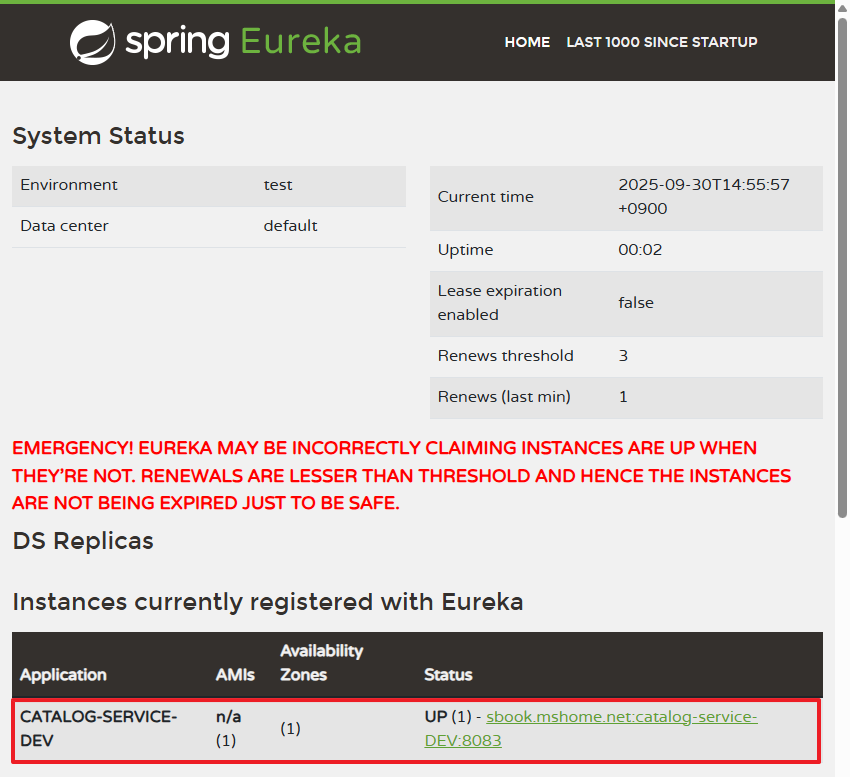
**prefreIpAddress: true**

## 테스트 ⇒ STS에서 실행 중인 인스턴스를 모두 종료 후 명령 프로프트에서 실행 ⇐ 개별 서비스의 로그 확인을 위해

D:\workspace\configserver> **mvnw spring-boot:run**

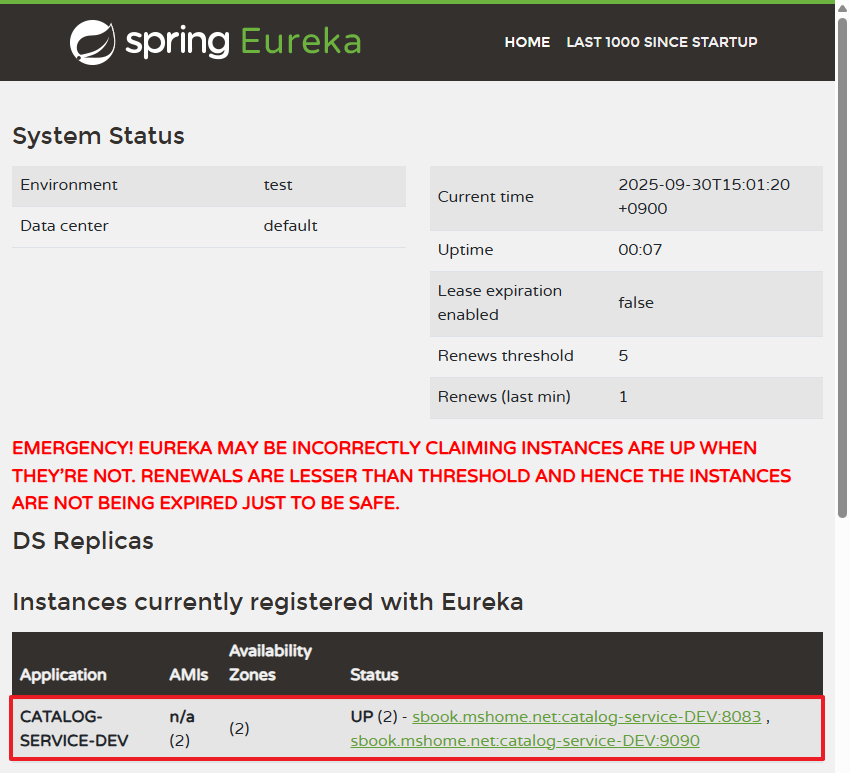
d:\workspace\eurekaserver> **mvnw spring-boot:run**

D:\workspace\catalog> **mvnw spring-boot:run**

****

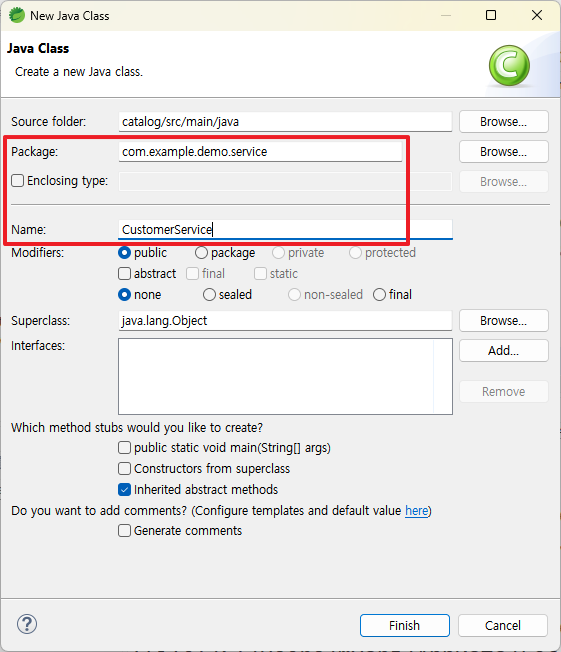
## 카탈로그 서비스를 추가로 실행

D:\workspace\catalog> **mvnw spring-boot:run -Dspring-boot.run.arguments="--server.port=9090"**

****

# 잠시 쉬고, 15시 15분에 이어서 하겠습니다.

## 카탈로그 서비스에 Serivce(이후에 커스터머 서비스를 호출)를 추가



package com.example.demo.service;

import org.springframework.stereotype.Service;

@Service

public class CustomerService {

public String getCustomerDetail(String customerId) {

// TODO. 커스터머 서비스를 호출해서 반환받은 값을 반환

return customerId;

}

}

## CatalogController를 수정

package com.example.demo.controller;

**import java.time.LocalDateTime;**

**import java.time.format.DateTimeFormatter;**

**import org.springframework.beans.factory.annotation.Autowired;**

import org.springframework.beans.factory.annotation.Value;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RestController;

**import com.example.demo.service.CustomerService;**

**import lombok.extern.slf4j.Slf4j;**

**@Slf4j**

@RestController

public class CatalogController {

@GetMapping("/")

public String hello() {

return "Hello ^..^";

}

@PostMapping("/")

public String world() {

return "World";

}

@Value("${spring.application.name}")

private String applicationName;

@Value("${custom.name:anonymouse}")

private String customName;

@GetMapping("/whoami")

public String profileTest() {

return customName + "@" + applicationName;

}

**@Autowired**

**private CustomerService customerService;**

**@GetMapping("/catalogs/{customerId}")**

**public String getCustomerInfo(@PathVariable String customerId) {**

**String customerInfo = customerService.getCustomerDetail(customerId);**

**log.debug("<<< customerId = " + customerId);**

**log.debug(">>> customerInfo = " + customerInfo);**

**return String.format("customer id %s's customer info is %s at %s\n",**

**customerId, customerInfo, LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SSS"))**

**);**

**}**

}

## 테스트

C:\Users\myanj> **curl -X GET http://localhost:8083/catalogs/123456789**

customer id 123456789's customer info is 123456789 at 2025-09-30 15:30:35.633

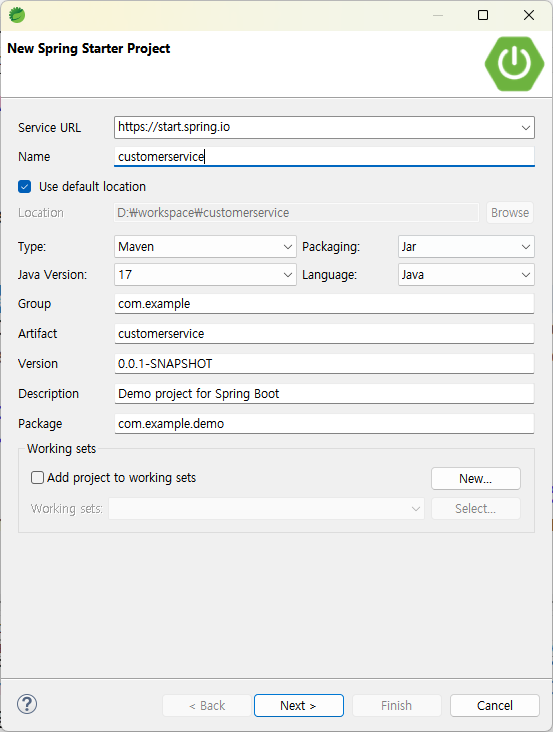
~~~~~~~~~ ~~~~~~~~~ ~~~~~~~~~~~~~~~~~~~~~~~

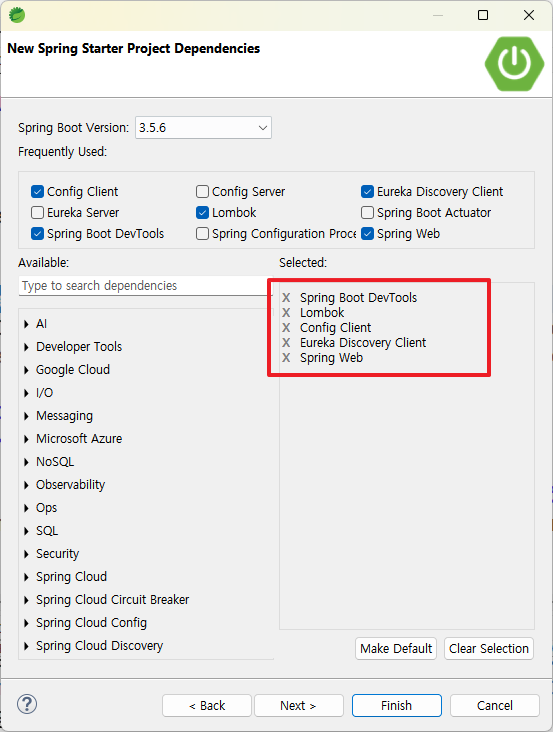
| | 호출된 시간

+-- 경로변수로 전달된 값 +-- CustomerService.getCustomerDetail() 메서드의 반환값

# 커스터머 서비스 프로젝트 생성

## 프로젝트 생성





## application.properties → application.yml

spring:

application:

name: **customer-service**

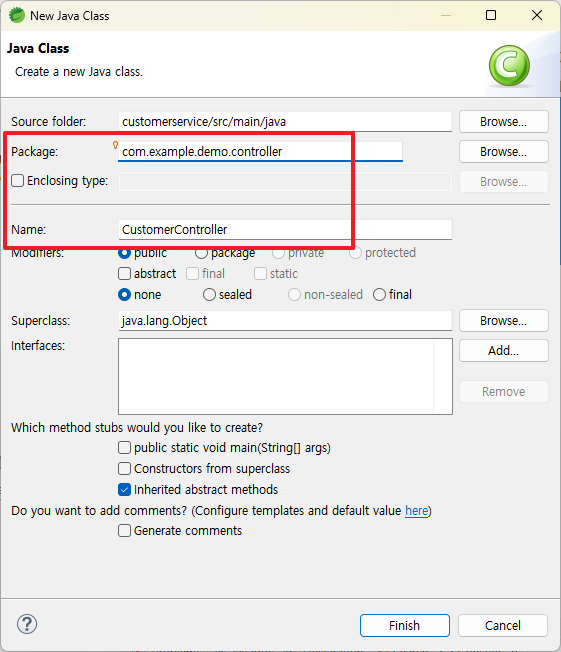
**config:**

**import: optional:configserver:http://localhost:8071**

**server:**

**port: 8082**

## 컨트롤러 추가



package com.example.demo.controller;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

import lombok.extern.slf4j.Slf4j;

@Slf4j

@RestController

public class CustomerController {

@Autowired

private CustomerService customerService;

@GetMapping("/customerInfo/{customerId}")

public String getCustomerInfo(@PathVariable String customerId) {

String customerInfo = customerService.getCustomerDetail(customerId);

log.debug("<<< customerId: " + customerId);

log.debug(">>> customerInfo: " + customerInfo);

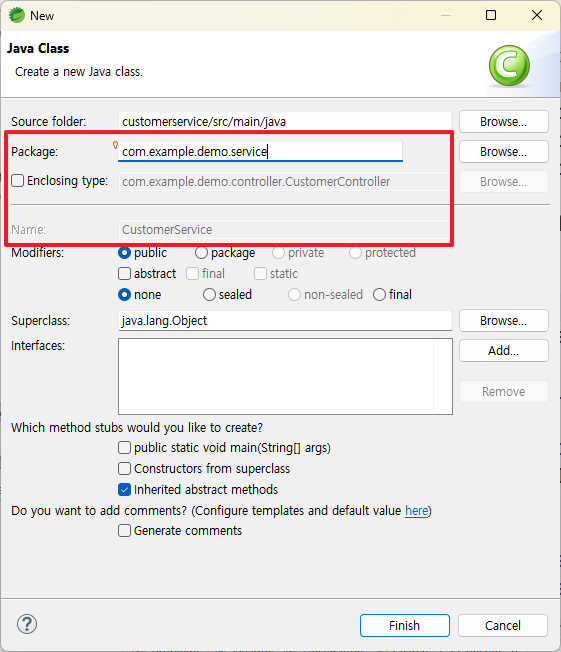
return String.format("\*\*\* customer id %s's customer info is %s at %s \*\*\*",

customerId, customerInfo, LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SSS"))

);

}

}



package com.example.demo.service;

import org.springframework.stereotype.Service;

@Service

public class CustomerService {

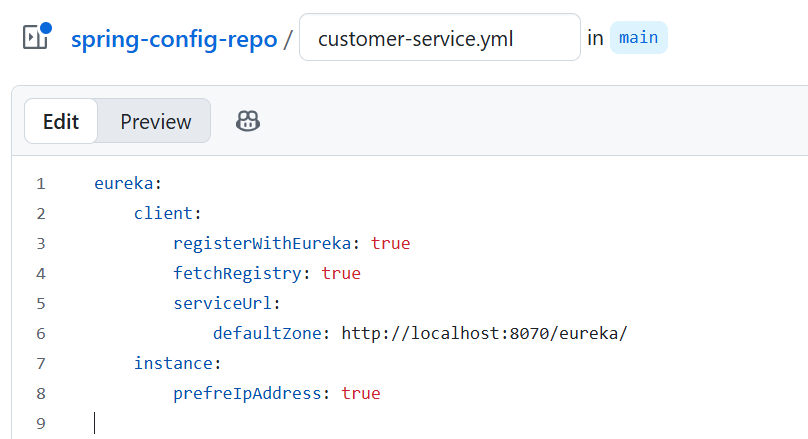
public String getCustomerDetail(String customerId) {

return String.format("고객 ID %s의 상세정보", customerId);

}

}

## 설정 파일을 깃 허브에 생성(등록)



eureka:

client:

registerWithEureka: true

fetchRegistry: true

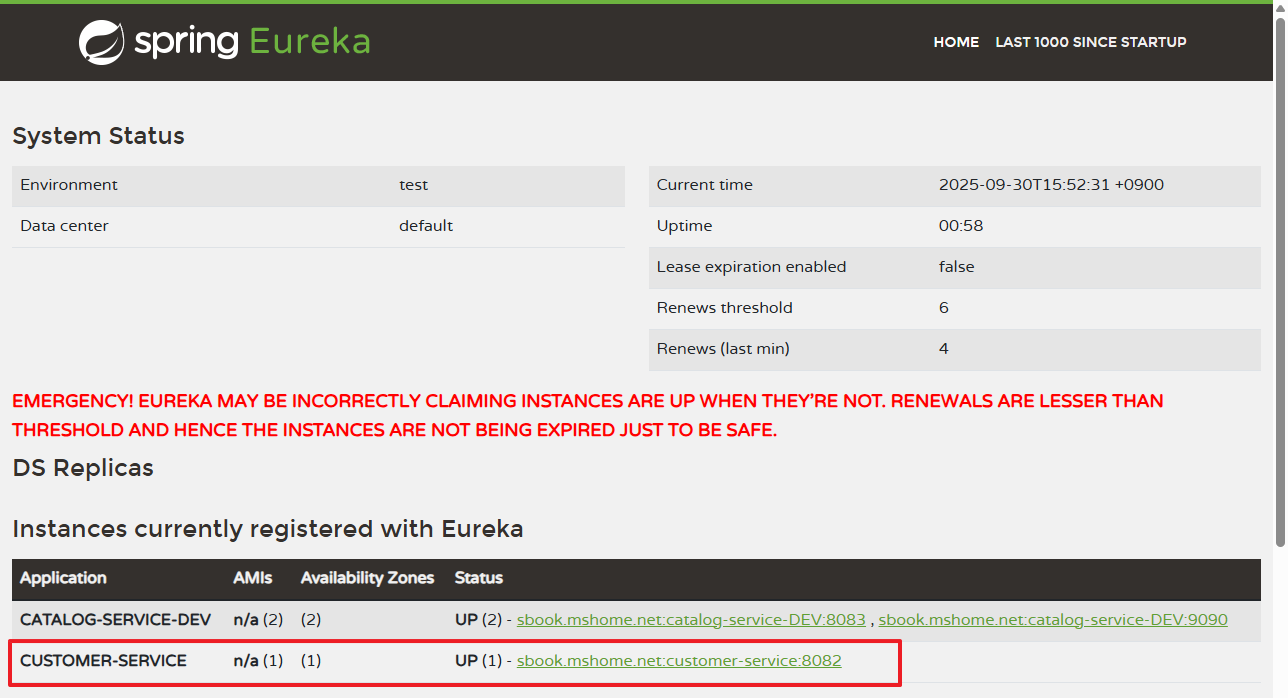
serviceUrl:

defaultZone: http://localhost:8070/eureka/

instance:

prefreIpAddress: true

## 테스트 ⇒ Config 서버 → Eureka 서버 → 커스터머 서비스 순으로 실행



C:\Users\myanj> **curl -X GET http://localhost:8082/customerInfo/123456789**

\*\*\* customer id 123456789's customer info is 고객 ID 123456789의 상세정보 at 2025-09-30 15:53:59.297 \*\*\*

# 서비스 디스커버리를 이용한 서비스 검색(호출)

| 기능 | DiscoveryClient | RestTemplate | Feign Client |
| --- | --- | --- | --- |
| 역할 | 유레카 서버에서 서비스 목록을 조회하는 데 사용  일반적으로 RestTemplate 또는 WebClient와 함께 사용  서비스 디스커버리 직접 호출 | 스프링에서 기본적으로 제공하는 REST API 호출 도구  Spring Boot 2.4 부터는 WebClient 또는 Feign 사용을 권장 (deprecated)  REST API 호출 (수동 설정 필요) | 인터페이스 기반의 REST API 호출 라이브러리  Spring Boot 2.x 이상에서 가장 권장하는 방식  Eureka 서비스 디스커버리를 자동으로 지원  REST API 호출 (자동 설정 필요) |
| 유레카 연동 | 가능 | 가능 | 가능 |
| 부하 분산 지원 | 직접 구현 | Ribbon 필요 | 내장 |
| 설정 난이도 | 복잡 | 중간 | 쉬움 |
| 성능 최적화 | 직접 관리 필요 | HTTP 호출 직접 수행 | 동적 프록시로 최적화 |

# 잠시 쉬고, 16시 10분에 이어서 하겠습니다.