Spring Boot

Spring Boot

1. 스프링 부트 소개

1.1. 스프링 부트의 목적

- 스프링 부트는 설정의 간소화, 라이브러리들의 버전 관리, 애플리케이션 실행 및 배포의 편리성 등을 제공하여 개발자들이 더 쉽고 빠르게 스프링 기반 애플리케이션을 개발할 수 있도록 합니다. 이를 위해 스프링 부트는 다음과 같은 기능들을 제공합니다.
    - 스프링 부트는 스프링 기반 애플리케이션 개발을 빠르고 쉽게 할 수 있도록 다양한 기능을 제공합니다.
    - 자동 설정, 스타터 의존성, 내장형 서버를 통해 초기 설정, 라이브러리 관리, 실행 및 배포 등을 편리하게 할 수 있습니다.
    - 실행 가능한 JAR 파일로 패키징하여 배포하므로 배포 및 확장이 용이합니다.

1.2. 스프링 부트와 스프링 프레임워크의 차이점

- 스프링 프레임워크는 애플리케이션 개발을 위한 다양한 모듈과 기능을 제공합니다. 반면에, 스프링 부트는 스프링 프레임워크에서 지원하는 기능들을 더 쉽게 사용할 수 있도록 도와주는 도구입니다. 스프링 부트의 주요 특징은 다음과 같습니다.

    - 자동 설정: 스프링 부트는 애플리케이션의 필요에 따라 자동으로 설정을 완성해줍니다.
    - 스타터 의존성: 스프링 부트 스타터를 사용하면 프로젝트에 필요한 의존성을 간단하게 추가할 수 있습니다.
    - 내장형 서버: 스프링 부트는 내장형 서버를 사용하여 애플리케이션을 별도의 서버에 배포하지 않고도 실행 및 테스트가 가능합니다.
    - 실행 가능한 JAR 패키징: 스프링 부트는 애플리케이션을 실행 가능한 JAR 패키징하여 배포할 수 있습니다. 이를 통해 서버에 별도의 어플리케이션 서버가 필요 없이 바로 실행이 가능합니다. 또한, 서버의 설정과 의존성 관리를 손쉽게 할 수 있기 때문에 배포가 간편합니다. JAR 파일은 실행환경이 동일하다면 어디에서든 실행이 가능합니다. 이를 통해 서버가 다운되는 경우 다른 서버에서 빠르게 대처할 수 있습니다.

2. 스프링 부트 환경 구축

2.1. IDE 설치 (IntelliJ, Eclipse 등)

- 스프링 부트 개발을 위한 IDE(통합 개발 환경)를 선택하고 설치합니다. IntelliJ IDEA, Eclipse, Visual Studio Code 등이 인기 있는 선택지입니다. IntelliJ IDEA는 무료 버전인 Community Edition과 유료 버전인 Ultimate Edition이 있으며, 스프링 부트 개발에 필요한 기능은 대부분 무료 버전에서도 사용 가능합니다.

2.1. IDE 설치 (IntelliJ, Eclipse 등)

스프링 부트 개발을 위해 IDE(통합 개발 환경)를 설치해야 합니다. 아래는 IntelliJ IDEA를 설치하는 방법입니다.

1. IntelliJ IDEA 다운로드 페이지 (https://www.jetbrains.com/idea/download/)로 이동합니다.
2. Community 버전과 Ultimate 버전 중 선택합니다. 스프링 부트 개발에 필요한 기능은 대부분 Community 버전에서도 사용 가능합니다.
3. 다운로드한 설치 파일을 실행합니다.

4. 설치 과정에서 필요한 설정을 진행합니다.
5. 설치가 완료되면 IntelliJ IDEA를 실행합니다.
6. IntelliJ IDEA에서 새 프로젝트를 생성하고 스프링 부트 프로젝트를 설정합니다.

2.2. Maven 또는 Gradle 프로젝트 구성

- 스프링 부트 프로젝트를 생성할 때, 빌드 도구로 Maven 또는 Gradle을 선택할 수 있습니다. 두 도구 모두 의존성 관리와 빌드 자동화 기능을 제공하며, 선택한 도구에 따라 프로젝트 구성이 달라집니다. 자신이 선호하는 도구를 선택하거나, 이미 사용 중인 도구가 있다면 해당 도구를 사용하세요.

2.2. Maven 또는 Gradle 프로젝트 구성

- 스프링 부트 프로젝트를 생성할 때, 빌드 도구로 Maven 또는 Gradle을 선택할 수 있습니다. 두 도구 모두 의존성 관리와 빌드 자동화 기능을 제공하며, 선택한 도구에 따라 프로젝트 구성이 달라집니다. 자신이 선호하는 도구를 선택하거나, 이미 사용 중인 도구가 있다면 해당 도구를 사용하세요.

1. Maven
    - pom.xml 파일을 사용하여 프로젝트 구성 및 의존성 관리
    - XML 기반의 설정
    - 다음과 같은 기본 구조를 가짐:

```
my-app
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   └── resources
    └── test
        ├── java
        └── resources
```

2. Gradle
    - build.gradle 파일을 사용하여 프로젝트 구성 및 의존성 관리
    - Groovy 또는 Kotlin DSL 기반의 설정
    - 다음과 같은 기본 구조를 가짐:

```
my-app
├── build.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
└── src
    ├── main
    │   ├── java
```

```
    │   └── resources
    └── test
        ├── java
        └── resources
```

2.3. 스프링 부트 프로젝트 생성 및 실행

- 스프링 부트 프로젝트를 생성하는 방법에는 여러 가지가 있습니다. 가장 간편한 방법은 스프링 공식 웹사이트인 Spring Initializr를 사용하는 것입니다. 원하는 설정과 의존성을 선택한 후, 프로젝트를 생성하여 다운로드 받습니다. 그런 다음, 다운로드 받은 프로젝트를 IDE에 import하고 실행하면 스프링 부트 애플리케이션이 실행됩니다. 또한 대부분의 IDE들은 스프링 부트 프로젝트를 직접 생성하는 기능을 지원하므로, IDE의 해당 기능을 이용할 수도 있습니다.
    1. Spring Initializr로 프로젝트 생성 및 다운로드
    2. IDE에 프로젝트 import
    3. 애플리케이션 실행

3. 스프링 부트 핵심 기능

3.1. 스프링 부트 스타터 (Spring Boot Starter)

- 스프링 부트 스타터는 의존성 관리를 간편하게 해주는 라이브러리입니다.
- 예시:
- Maven (pom.xml)

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

- 스프링 부트는 애플리케이션의 필요에 따라 자동으로 설정을 완성해줍니다. 스프링 부트는 프로젝트에 포함된 라이브러리와 설정을 분석하여 최적의 설정을 자동으로 제공합니다. 예를 들어, 스프링 부트 스타터 웹을 사용하면 웹 애플리케이션에 필요한 기본 설정을 자동으로 적용해줍니다. 필요하다면 자동 설정을 사용자 정의 설정으로 변경할 수도 있습니다.

3.3. 내장 서버 (Embedded Server)

- 스프링 부트는 내장 서버(예: Tomcat, Jetty, Undertow 등)를 포함하고 있어, 애플리케이션을 별도의 서버에 배포할 필요 없이 실행할 수 있습니다. 이를 통해 개발 및 테스트 과정에서 시간을 절약할 수 있으며, 서버 설정을 건드릴 필요가 없어 관리가 편리해집니다.

3.4. 프로파일 (Profile)

- 스프링 부트에서는 서로 다른 환경(개발, 테스트, 운영 등)에 따라 다른 설정을 적용할 수 있는 프로파일 기능을 제공합니다. 프로파일을 사용하면 환경에 따라 적절한 설정 값을 쉽게 적용할 수 있으며, 여러 프로파일을 동시에 활성화할 수도 있습니다.

3.5. 의존성 관리 (Dependency Management)

- 스프링 부트는 프로젝트의 의존성 관리를 단순화하고 통합합니다. 스프링 부트는 관련된 라이브러리들의 호환되는 버전을 선택하여 의존성 충돌을 최소화합니다. 스프링 부트가 관리하는 의존성들은 'spring-boot-dependencies'라는 BOM(Bill of Materials)에 정의되어 있습니다. 이를 통해 개발자는 의존성 버전에 대한 걱정 없이 필요한 라이브러리를 프로젝트에 추가할 수 있습니다. 또한, 스프링 부트는 사용자 정의 의존성을 추가하거나 기존 의존성을 변경할 수 있도록 유연성을 제공합니다.

4. RESTful 웹 서비스 개발

4.1. 컨트롤러 (Controller)

- 컨트롤러는 클라이언트의 요청을 처리하고 응답을 반환하는 역할을 담당합니다. 스프링 부트에서는 '@RestController' 어노테이션을 사용해 클래스를 컨트롤러로 정의하고, '@RequestMapping', '@GetMapping', '@PostMapping', '@PutMapping', '@DeleteMapping' 등의 어노테이션을 사용해 요청을 처리하는 메서드를 정의할 수 있습니다.
    - 컨트롤러는 요청 처리 및 응답 반환을 담당합니다. '@RestController'와 요청 처리 어노테이션을 사용합니다.
    - 예시:

```
@RestController
public class MyController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!";
    }

    @PostMapping("/greeting")
    public String greeting(@RequestBody String name) {
        return "Hello, " + name + "!";
    }

    @PutMapping("/update")
    public String update(@RequestBody String newData) {
        // ... update logic ...
        return "Update successful!";
    }

    @DeleteMapping("/delete/{id}")
    public String delete(@PathVariable Long id) {
        // ... delete logic ...
        return "Delete successful!";
    }
}
```

4.2. 서비스 (Service)

- 서비스는 비즈니스 로직을 수행하며, 컨트롤러와 리포지터리 사이에서 동작합니다. 서비스는 일반적으로 인터페이스와 구현체로 구성되며, '@Service' 어노테이션을 사용해 스프링이 관리하는 빈(Bean)으로 등록할 수 있습니다.

- 예시:

```
public interface MyService {
   String doSomething(String input);
}

@Service
public class MyServiceImpl implements MyService {

   @Override
   public String doSomething(String input) {
      // ... business logic ...
      return "Processed: " + input;
   }
}
```

4.3. 리포지터리 (Repository)

- 리포지터리는 데이터 저장소와의 인터페이스를 제공하며, 데이터에 대한 CRUD(Create, Read, Update, Delete) 작업을 수행합니다. 스프링 부트에서는 '@Repository' 어노테이션을 사용해 클래스를 리포지터리로 정의하고, JPA, MyBatis 등의 데이터 접근 기술을 사용해 구현할 수 있습니다.
  - 예시 (JPA 사용):

```
@Entity
public class MyEntity {
   // ... fields and getters/setters ...
}

@Repository
public interface MyRepository extends JpaRepository<MyEntity, Long> {
}
```

4.4. 데이터 모델 (Data Model)

- 데이터 모델은 애플리케이션의 데이터 구조를 정의합니다. 스프링 부트에서는 데이터 모델을 POJO(Plain Old Java Object) 클래스로 정의하며, JPA를 사용하는 경우 '@Entity', '@Table', '@Id', '@GeneratedValue', '@Column' 등의 어노테이션을 사용해 데이터베이스 테이블과 매핑할 수 있습니다.
  - 예시:

```
   @Entity
   @Table(name = "my_table")
   public class MyEntity {
      @Id
      @GeneratedValue(strategy = GenerationType.IDENTITY)
      private Long id;

      @Column(name = "name")
```

```
        private String name;

        // ... getters/setters ...
    }
```

4.5. 예외 처리 (Exception Handling)

- RESTful 웹 서비스 개발 시 예외 처리는 중요한 부분입니다. 스프링 부트에서는 '@ExceptionHandler' 어노테이션을 사용해 컨트롤러에서 발생한 예외를 처리하는 메서드를 정의할 수 있습니다. 또한, '@ControllerAdvice' 어노테이션을 사용해 전역적으로 예외를 처리하는 클래스를 정의할 수도 있습니다. 이를 통해 클라이언트에게 적절한 오류 메시지와 응답 코드를 전달할 수 있습니다.

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(value = {MyException.class})
    public ResponseEntity<Object> handleMyException(MyException ex) {
        // ... error handling logic ...
        return new ResponseEntity<>(errorMessage, HttpStatus.BAD_REQUEST);
    }
}
```

5. 데이터 액세스

5.1. JPA (Java Persistence API)

- JPA는 Java Persistence API의 약자로, 자바 애플리케이션에서 관계형 데이터베이스를 사용하기 위한 표준 API입니다. JPA는 개발자로부터 데이터베이스와의 상호작용을 추상화하여, 객체 지향적인 방식으로 데이터를 처리할 수 있게 해줍니다. Hibernate, EclipseLink, OpenJPA 등의 구현체가 존재합니다.

5.2. 스프링 데이터 JPA

- 스프링 데이터 JPA는 스프링 프레임워크에서 JPA를 사용하기 위한 추가적인 편의 기능을 제공합니다. 스프링 데이터 JPA를 사용하면, 리포지터리 인터페이스만 정의하면 기본적인 CRUD 작업을 자동으로 구현해주는 기능을 제공합니다. 또한, 명명 규칙에 따라 메서드 이름만으로 쿼리를 생성하는 기능도 제공합니다.

```
public interface MyRepository extends JpaRepository<MyEntity, Long> {
}
```

5.3. 데이터베이스 연동 (MySQL, PostgreSQL, H2 등)

- 스프링 부트에서는 다양한 데이터베이스와 연동할 수 있습니다. MySQL, PostgreSQL, H2, Oracle 등의 데이터베이스를 사용할 수 있으며, 애플리케이션의 'application.properties' 또는 'application.yml' 파일에 데이터베이스 연결 정보를 설정하여 연동할 수 있습니다. 필요한 경우, 스프링 부트의 자동 설정 기능을 활용하여 데이터베이스 연결 풀 등의 세부 설정을 조정할 수도 있습니다.

```
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=myuser
spring.datasource.password=mypassword
```

6. 스프링 시큐리티 (Spring Security)

6.1. 인증 (Authentication)

- 스프링 시큐리티는 애플리케이션의 인증 과정을 관리합니다. 사용자의 아이디와 비밀번호를 확인하는 과정, 인증 성공 후 세션 생성 등의 작업을 담당합니다. 스프링 시큐리티는 여러 가지 인증 방식을 제공하며, 필요한 경우 커스텀 인증 과정을 구현할 수도 있습니다.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws
Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncode
r());
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

6.2. 인가 (

6.2. 인가 (Authorization)

- 인가는 사용자가 허용된 리소스에만 접근할 수 있도록 관리하는 과정입니다. 스프링 시큐리티는 권한 기반의 인가를 제공하며, 컨트롤러나 메서드 단위로 접근 권한을 지정할 수 있습니다. '@Secured', '@PreAuthorize', '@PostAuthorize' 등의 어노테이션을 사용해 인가 설정을 적용할 수 있습니다.

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
```

```
            http.authorizeRequests()
                .antMatchers("/admin/**").hasRole("ADMIN")
                .antMatchers("/user/**").hasAnyRole("USER", "ADMIN")
                .and()
                .formLogin();
        }
    }
```

6.3. OAuth2, JWT 토큰 기반 인증

- OAuth2는 인증 및 권한 부여를 위한 오픈 스탠다드 프로토콜입니다. 스프링 시큐리티는 OAuth2를 지원하여 외부 서비스(예: Google, Facebook 등)를 통한 인증 및 권한 부여를 쉽게 구현할 수 있습니다. OAuth2를 사용하면 사용자의 계정 정보를 안전하게 제공하고, 제3자 애플리케이션에 대한 접근 제어를 할 수 있습니다.
- JWT(JSON Web Token)는 웹 애플리케이션에서 인증 정보를 안전하게 전달하는 방법입니다. JWT는 암호화된 토큰 형태로 사용자 인증 정보를 저장하며, 서버와 클라이언트 사이에서 토큰을 주고받아 인증을 수행합니다. 스프링 시큐리티는 JWT 토큰 기반 인증을 지원하며, 필요한 경우 JWT 토큰을 생성하고 검증하는 로직을 커스터마이징할 수 있습니다.

```
@Configuration
@EnableResourceServer
public class ResourceServerConfig extends ResourceServerConfigurerAdapter
{

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/api/**").authenticated();
    }

    @Bean
    public JwtAccessTokenConverter accessTokenConverter() {
        JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
        converter.setSigningKey("your-signing-key");
        return converter;
    }

    @Bean
    public TokenStore tokenStore() {
        return new JwtTokenStore(accessTokenConverter());
    }
}
```

7. 테스트

7.1. 단위 테스트 (Unit Test)

- 단위 테스트는 애플리케이션의 개별 컴포넌트를 독립적으로 테스트하는 과정입니다. 스프링 부트에서는 JUnit과 같은 테스트 프레임워크를 사용하여 단위 테스트를 작성할 수 있습니다. '@Test' 어노테이션을 사용해 테스트 메서드를 정의하고, 필요한 경우 Mockito 등의 라이브러리를 사용해 의존성을 가짜로 대체하여 테스트를 수행할 수 있습니다.

7.2. 통합 테스트 (Integration Test)

- 통합 테스트는 애플리케이션의 여러 컴포넌트를 함께 테스트하는 과정입니다. 스프링 부트에서는 '@SpringBootTest' 어노테이션을 사용해 통합 테스트를 작성할 수 있습니다. 통합 테스트는 애플리케이션의 전체 동작을 검증하며, 데이터베이스, 외부 서비스 등과의 연동을 포함하여 테스트를 수행합니다.

8. 캐싱 (Caching)

8.1. 스프링 부트 캐시 추상화

- 스프링 부트는 캐시 추상화를 제공하여, 애플리케이션의 성능을 향상시키는 데 도움이 됩니다. 캐시 추상화를 사용하면 메서드의 실행 결과를 캐시에 저장하고, 동일한 입력이 주어질 때 캐시에서 결과를 반환할 수 있습니다. '@Cacheable', '@CachePut', '@CacheEvict' 등의 어노테이션을 사용해 캐싱 동작을 적용할 수 있습니다.

8.2. 캐시 구현체 (EhCache, Redis 등)

- 스프링 부트는 다양한 캐시 구현체를 지원합니다. EhCache, Redis, Hazelcast 등의 캐시 구현체를 사용할 수 있으며, 애플리케이션의 'application.properties' 또는 'application.yml' 파일에 캐시 구현체에 대한 설정을 지정하여 연동할 수 있습니다.

9. 마이크로 서비스

9.1. 스프링 클라우드 소개

- 스프링 클라우드는 마이크로 서비스 아키텍처를 구축하기 위한 스프링 프로젝트입니다. 스프링 클라우드는 서비스 디스커버리, API 게이트웨이, 서킷 브레이커 등의 기능을 제공하여 마이크로 서비스간의 통신과 관리를 쉽게 구현할 수 있습니다. 스프링 클라우드는 스프링 부트 기반 애플리케이션과 호환되며, 클라우드 네이티브 애플리케이션 개발에 도움을 줍니다.

9.2. 서비스 디스커버리 (Service Discovery)

- 서비스 디스커버리는 마이크로 서비스간의 통신을 관리하는 기능입니다. 서비스 디스커버리를 통해 서비스들은 서로의 위치를 알 수 있으며, 서비스에 대한 요청을 동적으로 라우팅할 수 있습니다. 스프링 클라우드에서는 Eureka, Consul, Zookeeper 등의 서비스 디스커버리 구현체를 사용할 수 있습니다.

9.3. API 게이트웨이 (API Gateway)

- API 게이트웨이는 클라이언트와 마이크로 서비스 사이의 중간 계층으로, 클라이언트의 요청을 적절한 마이크로 서비스로 라우팅하는 역할을 합니다. API 게이트웨이는 요청에 대한 인증, 로드 밸런싱, 캐싱 등의 기능을 제공하여 마이크로 서비스의 성능과 안정성을 향상시킬 수 있습니다. 스프링 클라우드에서는 Netflix Zuul, Spring Cloud Gateway 등의 API 게이트웨이 구현체를 사용할 수 있습니다.

9.4. 서킷 브레이커 (Circuit Breaker)

- 서킷 브레이커는 마이크로 서비스의 장애 전파를 방지하는 기능입니다. 서킷 브레이커는 서비스 호출에 대한 실패를 감지하고, 일정한 실패 횟수를 초과하면 서비스 호출을 차단합니다. 이렇게 함으로써, 장애가 다른 서비스로 전파되는 것

을 방지하고, 마이크로 서비스가 빠르게 복구될 수 있도록 돕습니다. 스프링 클라우드에서는 Netflix Hystrix, Resilience4j 등의 서킷 브레이커 구현체를 사용할 수 있습니다.

9.2. 서비스 디스커버리 (Service Discovery)

서비스 디스커버리는 마이크로 서비스간의 통신을 관리하는 기능입니다. 서비스 디스커버리를 통해 서비스들은 서로의 위치를 알 수 있으며, 서비스에 대한 요청을 동적으로 라우팅할 수 있습니다. 스프링 클라우드에서는 Eureka, Consul, Zookeeper 등의 서비스 디스커버리 구현체를 사용할 수 있습니다.

9.3. API 게이트웨이 (API Gateway)

API 게이트웨이는 클라이언트와 마이크로 서비스 사이의 중간 계층으로, 클라이언트의 요청을 적절한 마이크로 서비스로 라우팅하는 역할을 합니다. API 게이트웨이는 요청에 대한 인증, 로드 밸런싱, 캐싱 등의 기능을 제공하여 마이크로 서비스의 성능과 안정성을 향상시킬 수 있습니다. 스프링 클라우드에서는 Netflix Zuul, Spring Cloud Gateway 등의 API 게이트웨이 구현체를 사용할 수 있습니다.

9.4. 서킷 브레이커 (Circuit Breaker)

서킷 브레이커는 마이크로 서비스의 장애 전파를 방지하는 기능입니다. 서킷 브레이커는 서비스 호출에 대한 실패를 감지하고, 일정한 실패 횟수를 초과하면 서비스 호출을 차단합니다. 이렇게 함으로써, 장애가 다른 서비스로 전파되는 것을 방지하고, 마이크로 서비스가 빠르게 복구될 수 있도록 돕습니다. 스프링 클라우드에서는 Netflix Hystrix, Resilience4j 등의 서킷 브레이커 구현체를 사용할 수 있습니다.

10. 배포 및 모니터링

10.1. 도커 (Docker)를 이용한 컨테이너화

도커는 애플리케이션과 그 실행 환경을 격리시키는 가상화 기술을 사용하여, 독립적인 컨테이너 내에서 실행되도록 만드는 플랫폼입니다. 도커를 사용하면 애플리케이션의 배포와 관리를 쉽게 할 수 있습니다. 스프링 부트 애플리케이션을 도커 컨테이너로 만들기 위해 'Dockerfile'을

10. 배포 및 모니터링

10.1. 도커 (Docker)를 이용한 컨테이너화

도커는 애플리케이션과 그 실행 환경을 격리시키는 가상화 기술을 사용하여, 독립적인 컨테이너 내에서 실행되도록 만드는 플랫폼입니다. 도커를 사용하면 애플리케이션의 배포와 관리를 쉽게 할 수 있습니다. 스프링 부트 애플리케이션을 도커 컨테이너로 만들기 위해 'Dockerfile'을 작성하고, 도커 이미지를 빌드 및 배포하는 과정을 학습합니다.

10.2. 쿠버네티스 (Kubernetes) 환경

쿠버네티스는 컨테이너화된 애플리케이션의 배포, 스케일링 및 관리를 자동화하기 위한 오픈소스 플랫폼입니다. 쿠버네티스를 사용하면 마이크로 서비스를 쉽게 관리할 수 있습니다. 쿠버네티스 환경에서 스프링 부트 애플리케이션을