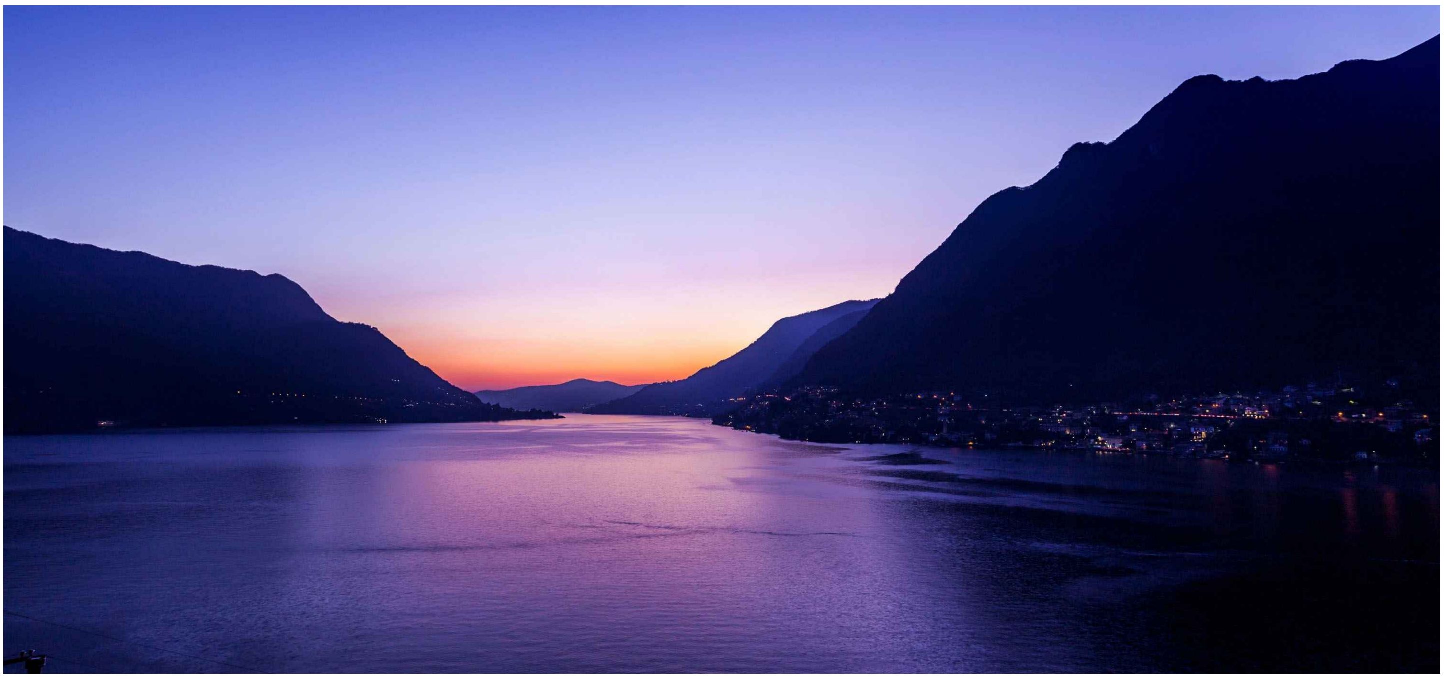

2-Spring Boot Configurations

Author: Vincent Lau

Learning Objectives

> Write an application.properties or application.yml file to configure a Spring Boot application

> Describe why the YAML format is preferred over the Properties format

> Access external configurations using various approaches in Spring Boot

Overview

- A `common practice` in Spring Boot is using an external configuration to define our properties. This allows us to use the same application code in different environments(dev/ test/ uat/ pre-prod/ prod).
- We can use **properties** files, **YAML** files, **environment variables** and **command-line arguments**.

Properties Configuration

- Although not absolutely necessary, if the `application.properties` **file exists**, it will be **auto-loaded** for configuring the Spring Boot Application.
- By default, Spring Boot can access configurations set in an `application.properties` file, which uses a **key-value format**:

```
spring.datasource.url=jdbc:h2:dev
spring.datasource.username=SA
spring.datasource.password=password
```

- Each line is a single configuration, so we need to express hierarchical data by using the same prefixes for our keys. In this example, every key belongs to *spring.datasource.*
- Other configurations may include server, database and security settings, etc.
- For example, the Spring Boot embedded Tomcat server listens to **port 8080 by default**. It is possible to change the default port by adding a property in the `application.properties` file.

```
server.port: 8081
```

Placeholder in Properties

- Within our values, we can use placeholders with the `${}` syntax to refer to the contents of other keys, system properties, or environment variables:

```
app.name=MyApp
app.description=${app.name} is a Spring Boot application
```

Multiple Profiles

- Spring Boot supports configurations for **different environments** (e.g. **dev, test, prod**), and will take in an environment variable upon startup to pick up the configurations based on the specified environment.

```
// Approach #1
```

```
mvn spring-boot:run -Dspring-boot.run.profiles=dev

// Approach #2
java -jar -Dspring.profiles.active=dev <JAR file name>.jar
```

- We can define the configurations for each profile all in the same file:

```
logging.file.name=myapplication.log
bael.property=defaultValue
#---
spring.config.activate.on-profile=dev
spring.datasource.password=password
spring.datasource.url=jdbc:h2:dev
spring.datasource.username=SA
bael.property=devValue
#---
spring.config.activate.on-profile=prod
spring.datasource.password=password
spring.datasource.url=jdbc:h2:prod
spring.datasource.username=prodUser
bael.property=prodValue
```

- Note we use the `#---` notation to indicate where we want to split the document.
- Alternatively, we can **store multiple profiles across different files**. We achieve this by putting the name of the profile in the file name - for example, `application-dev.yml` **or `application-dev.properties`.

YAML Configuration

YAML Format

- As well as Java properties files, we can also use YAML-based configuration files in our Spring Boot application. **YAML is a convenient format for specifying hierarchical configuration data**.

```
spring:
    datasource:
        password: password
        url: jdbc:h2:dev
        username: SA
```

Multiple Profiles

- We can also store configurations for different profiles in the same YAML file, except with more clarity than the `application.properties` file:

```yaml
logging:
  file:
    name: myapplication.log
---
spring:
  config:
    activate:
      on-profile: staging
  datasource:
    password: 'password'
    url: jdbc:h2:staging
    username: SA
bael:
  property: stagingValue
```

- It is **highly recommended not to include** both `application.properties` and `application.yml` in your project at the same time. **Stick to one format only.**

Accessing Configurations

- There are three ways to access external configurations in Spring Boot:
  - By using the `@Value` annotation
  - By using the `Environment` object
  - By using the `@ConfigurationProperties` annotation

Using @Value

- Here the property `server.port` is injected via field injection into one of our objects:

```java
@Value("${server.port}")
private String serverPort;
```

Using Environment Object

```java
@Autowired
private Environment env;

public String getSomeKey() {
    return env.getProperty("server.port");
}
```

Using @ConfigurationProperties

```java
@Configuration // from class to bean
@ConfigurationProperties(prefix = "server") // from yml to class
public class ServerConfig {
    String port;

    String anotherField;

    public String getPort() {
        return port;
    }

    public void setPort(String port) {
        this.port = port;
    }
}
```

- application.yml

```yaml
server:
  port: 8081
  anotherField: 'field value'
```

- We can also use the `@ConfigurationProperties` annotation to bind our properties to type-safe structured objects:
- Then we can access the configuration object just like any other beans:

```java
@Autowired
private ServerConfig serverConfig;

public String getSomeKey(){
    return serverConfig.getPort();
}
```

Questions

- How do we access an environment variable from the application.yml file?
- Why do we prefer using YAML format to write configuration file?
- What are the different ways to access external configuations in Spring Boot?