APPENDIX A

Upgrading to Spring Boot 2.0

Introduction

This book uses the Spring Boot version 1.5.7 for the different microservices that are part of the evolving application. Spring Boot 2.0, which is to be released after writing this book, introduces some breaking changes to the system since it's a major update.

This appendix helps you upgrade your applications to Spring Boot 2.0 in case you want to use it.

SOURCE CODE AVAILABLE WITH THE BOOK: V10

The repository v10 on GitHub (see `https://github.com/microservices-practical`) contains the system code, working with Spring Boot 2.0. It works exactly the same way as the previous version (v9), just with the new version. All changes made to the application are tagged with a comment starting with `BOOT2` so you can easily find them by performing a text search.

Note that, since there is no official version at the time of writing the book, we're using a milestone version: 2.0.0.M2.

Upgrading the Dependencies

If you want to use the latest version of Spring Boot, you need to change your `pom.xml` files. Keep in mind that you also need to upgrade the Spring Cloud release version, but it does not follow exactly the same schedule as Spring Boot—it's a few weeks delayed.

Therefore, the plan this appendix follows is to include the Spring Boot 2.0 Milestone 2 (`2.0.0.M2`), since it has an equivalent working version for Spring Cloud early releases: `Finchley.M2`. To be able to use milestone versions in your projects, you also need to append to your `pom.xml` the additional Spring repositories where these versions reside.

Listing A-1 shows the file for the multiplication microservice as a reference; you need to apply these same changes to all the other Spring Boot applications (upgrading the Spring Boot and Spring Cloud versions and adding the `repositories` and `pluginRepositories` blocks).

Listing A-1. pom.xml (multiplication v10)

```xml
<?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 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>microservices.book</groupId>
    <artifactId>social-multiplication-v10</artifactId>
    <version>0.10.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>social-multiplication-v10</name>
    <description>Social Multiplication App (Learn Microservices
    with Spring Boot)</description>
```

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.M2</version>
</parent>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.
    sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.
    reporting.outputEncoding>
    <java.version>1.8</java.version>
    <spring-cloud.version>Finchley.M2</spring-cloud.
    version>
</properties>

<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>

<dependencies>
<!-- All our existing dependencies... -->
</dependencies>
```

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin
            </artifactId>
        </plugin>
    </plugins>
</build>

<repositories>
    <repository>
        <id>spring-snapshots</id>
        <name>Spring Snapshots</name>
        <url>https://repo.spring.io/snapshot</url>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
    <repository>
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>

<pluginRepositories>
    <pluginRepository>
        <id>spring-snapshots</id>
        <name>Spring Snapshots</name>
```

```
            <url>https://repo.spring.io/snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>

</project>
```

Fixing the Breaking Changes

The CrudRepository Interface Does Not Include findOne()

The book's code uses the `findOne()` method included in the `CrudRepository` interface to retrieve a single entity from the database using Spring Data JPA. For some reason (documentation is not available yet at the time of writing), that method has been removed in an earlier version to the one included by the Spring Boot starter, which uses the version `2.0.0.M4` of `spring-data-commons` (you can see it by running `mvnw dependency:tree` from any of the root project folders).

That doesn't have a big impact on the code, but you need to change these method references to make everything compile again. The code uses `findOne()` in three classes inside the multiplication project: `MultiplicationServiceImpl`, `UserController`, and `UserControllerTest`.

Follow the same strategy to replace it in those three places: changing it to the preferred `findById()` method and using the returned `Optional` to throw an `IllegalArgumentException` if the identifier does not exist in your database. Listing A-2 shows one of the code snippets as a reference.

Listing A-2. MultiplicationServiceImpl.java (multiplication v10)

```
@Override
public MultiplicationResultAttempt getResultById(final Long
resultId) {
    // BOOT2: changed from findOne
    return attemptRepository.findById(resultId)
            .orElseThrow(() -> new IllegalArgumentException(
                    "The requested resultId [" + resultId +
                            "] does not exist."));
}
```

Actuator Endpoints Have Been Moved

We know from the release notes[1] that in Spring Boot 2.0, the actuator endpoints have been moved under `/application`. That means that you need to change the load balancing strategy configuration inside the API gateway to point to `/application/health` instead of just `/health`. If you don't change this, Ribbon will think all services are down and the API Gateway won't work. See Listing A-3.

[1]https://tpd.io/bootm1-rn

Listing A-3. RibbonConfiguration.java (gateway v10)

```
@Bean
public IPing ribbonPing(final IClientConfig config) {
    // BOOT2: changed from /health to /application/health
    return new PingUrl(false,"/application/health");
}
```

Applying Optional Updates

The WebMvcConfigurerAdapter Class Has Been Deprecated

The current code uses the class `WebMvcConfigurerAdapter` to disable CORS in the multiplication, gamification, and gateway microservices. That class has been deprecated in favor of the `WebMvcConfigurer` interface, since starting with Java 8, interfaces can include default implementations and Spring Boot 2.0 no longer supports earlier Java versions.

This change is optional here since it's just deprecated, but it's better to adapt to the change as soon as possible. Listing A-4 shows the change in one of the classes; be sure to apply this change to all three of the classes (inside the aforementioned microservices projects).

Listing A-4. WebConfiguration.java (gateway v10)

```
@Configuration
@EnableWebMvc
//BOOT2 changed to interface WebMvcConfigurer instead of
subclass of WebMvcConfigurerAdapter
public class WebConfiguration implements WebMvcConfigurer
{ //... }
```

Working with Spring Boot 2.0

Those are all the changes you should apply to make the system work with Spring Boot 2.0. Note that I couldn't try the release version, so it might happen (though it's unlikely) that there are other breaking changes to come, not covered by this appendix. If that is the case, you can visit the Spring Boot 2.0 release notes page to check. (See https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Release-Notes.)

On that same page, you'll find the new features coming with Spring Boot 2.0 as well. Some of the major new functionalities are support for Java 9 and some great features in Spring 5, such as the Reactive Web framework (Spring WebFlux). Have a look at them and keep learning!

Afterword

In this book, we covered the main topics related to microservices architecture. We started with a look *inside* a Spring Boot application, traveling from an empty project to a microservice properly structured in layers. To build it, we followed a test-driven development approach.

The book tried to explain from the beginning why it's a good idea to start with a small monolith. Actually, it's an idea supported by many people very experienced with microservices—start with a single project, identify boundaries, and decide if it's worthwhile to split your functionality. What happens frequently is that it's difficult to understand the *why* if you never worked with microservices and only with monolithic applications. However, at this point in time, I'm pretty sure you understand the pain you might suffer if you go for microservices from scratch. Setting up the ecosystem without having a strong knowledge baseline will cause chaos, at the least. Service discovery, routing, load balancing, communication between services, error handling—it's important that you know what you'll face on your way *before* you start your adventure with microservices.

In this particular adventure, you built a web application to allow users to practice multiplication every day without any help, to train their brains. That was the first microservice, but the real challenge started when we introduced the second one: a service that reacts to events happening in our previous logic and calculates the score and assigns badges to make the application look like a game. At that point, the book covered some basics about *gamification* and applied it to our system using an event-driven pattern.

Then we got deep into some core concepts of microservices, including how they can find each other with service discovery and apply load balancing to them when they scale up and down, how to route from the outside to the corresponding piece of the system using an API gateway, how to provide resilience using the circuit breaker pattern, etc. We made it practical, but I tried to put the focus on the concepts too. The reason is that nowadays you can find these patterns embedded in many cloud PaaS, including Cloud Foundry, Google App Engine, and Amazon AWS.

Some of those platforms will manage the microservice ecosystem for you: you just *push* your Spring Boot application to the cloud, set the number of instances and how they should be routed, and everything else is handled by the platform. *Pivotal's Cloud Foundry* even includes the same tools we used (Spring Cloud Netflix). However, you always need to understand what you're doing. Pushing your applications to the cloud without knowing what is happening behind the curtains is risky. If errors arise, it might be impossible for you to know which part of *the magic* (something that you don't understand) is not doing its job.

Finally, I stressed the importance of having a good test base in a system based on microservices. You saw how to implement end-to-end scenarios to protect your business flows using Cucumber and its human-friendly language, Gherkin.

This book doesn't cover *everything* you need to know to work with microservices. That would be impossible. There are some other important topics surrounding this type of architecture—containerization, centralized logging, continuous integration and deployment, etc. I recommend that you continue learning these topics following the same practical approach. You can keep evolving the architecture built in this book, focusing on the problems you want to solve and learning concepts incrementally, one by one.

I hope that, now that you've reached the end of the book, you have a better understanding of the topics and you can use them at work or in your personal projects. I've enjoyed writing this book a lot and, most importantly, I've learned along the way. Thanks.

Index