Spring Boot Developer course

3 Day Instructor-led Training

Version 2.0.0.M7.a

Copyright Notice

Course Introduction

Spring Boot Developer

Logistics

- Student introductions
- Self introduction
- Course registration (if needed)
- Courseware
- Internet access
- Phones on silent

Working hours

Lunch and breaks

Toilets/Restrooms

Fire alarms

Emergency exits

Any other questions?

Course Objectives

- Learn to use Spring Boot for web and other applications
- Gain hands-on experience
- Generous mixture of presentation and labs

Covered in this section

- Agenda
- Spring and Pivotal

Agenda: Day 1

- Spring Framework
- Spring Boot Overview
- Spring Boot Internals
- Spring Boot Features
- Web Development with Spring Boot

Agenda: Day 2

- Data Access with Spring Boot
- Spring Boot Testing
- Spring Boot Actuator
- Spring Boot Security
- Spring Boot Messaging

Agenda: Day 3

- Spring Integration
- Spring Cloud Stream
- Spring Boot Microservices
- Custom Spring Boot Starters
- and more...

Covered in this section

- Agenda
- Spring and Pivotal

Spring and Pivotal

- SpringSource, the company behind Spring
  - acquired by VMware in 2009
  - transferred to Pivotal joint venture 2013
- Spring projects key to Pivotal's big-data and cloud strategies
  - Virtualize your Java Apps
    - Save license cost
    - Deploy to private, public, hybrid clouds
  - Real-time analytics
    - Spot trends as they happen
    - Spring Data, Spring Hadoop, Spring XD & Pivotal HD

The Pivotal World

Cloud Foundry

Cloud Independence
Microservices
Continuous Delivery
Dev Ops

CLOUD FOUNDRY™

Development

Frameworks
Services
Analytics

spring

RabbitMQ™

Pivotal Tracker

Big Data Suite

High Capacity
Real-time Ingest
SQL Query
Scale-out Storage

HAWQ

GREENPLUM

GEMFIRE®

hadoop

Pivotal **Labs** *Working with clients to build better apps more quickly*

Spring Projects

Spring Security

Spring Data

Spring Batch

Spring Integration

Spring (SOAP) Web Services

Spring Social

Spring Cloud

Spring Framework

Spring AMQP

Spring Session

Spring Hateoas

Spring Android

Spring Web Flow

Spring Reactor

Spring Cloud Data Flow

Spring Boot

Spring Mobile

Demos and Labs

- There will be **DEMOS** in some topics
  - Demos will help you to understand better how to use the technology
- **Repetition** is the key of mastering!
- Better to start fresh
  - You will use the http://start.spring.io/ for every project/lab.
- **Maven** or **Gradle**, pick your building tool.
  - Every project/lab has instructions for both tools.
- Some times copy/paste is good (just be careful)
  - You will copy/paste some of the Domain classes over new projects, just be careful where.

Pivotal

A NEW PLATFORM FOR A NEW ERA

Spring Framework

Spring Boot Developer

A quick introduction

Agenda

- Spring Framework
- Spring Application Development
- What’s new in Spring Framework 5.0

Spring Framework

- Open Source
- Lightweight
- Container
- Framework

Spring Framework

Open Source

- Binary and Source freely available
- Apache 2 License
- Maven central
- Well documented

Spring Framework

Lightweight

- A J2EE Server is not required
- Is not invasive
- Low overhead

Spring Framework

Container

- Spring serves as a container for your application objects
- Uses dependency injection to instantiate your objects

Spring Framework

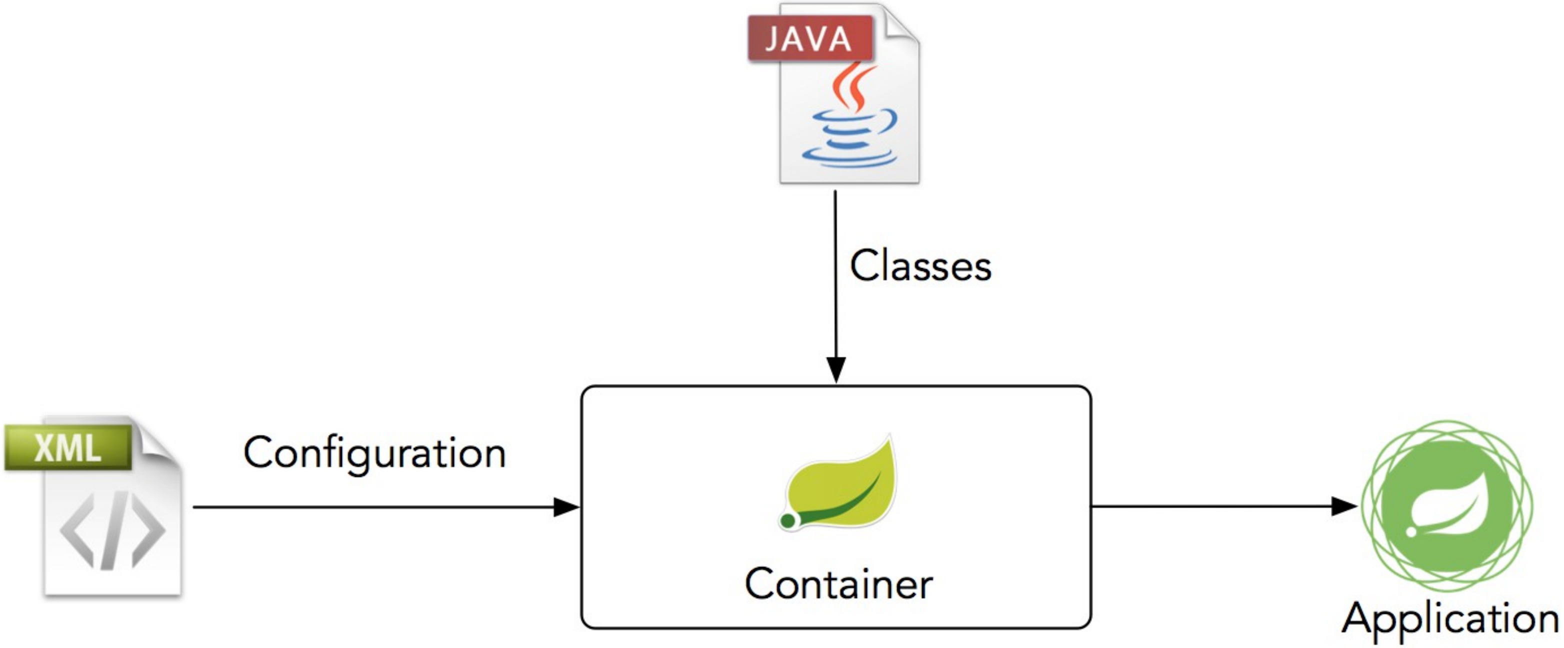
Framework

- Provides framework classes to simplify working with lower-level technologies

Agenda

- Spring Framework
- Spring Application Development
- What’s new in Spring Framework 5.0

Spring Application Development



Spring Application Development

Configuration

- XML
- Java Config
- Annotations

Classes

- POJOs

Agenda

- Spring Framework
- Spring Application Development
- What’s new in Spring Framework 5.0

What’s new in Spring Framework 5.0

- JDK 8+ and Java EE 7+ Baseline
  - Entire framework codebase based on Java 8 source code level now
  - Full compatibility with JDK 9 for development and deployment.
  - Java EE 7 API level required in Spring's corresponding features now.
  - Compatibility with Java EE 8 API level at runtime.
- Removed Packages, Classes and Methods
  - Dropped support: Portlet, Velocity, JasperReports, XMLBeans, JDO, Guava
- General Core Revision
  - JDK 8+ enhancements, JDK 9 compatibility

What’s new in Spring Framework 5.0

- Core Container
  - Support for any ***@Nullable*** annotations as indicators for optional injection points.
  - Functional style on ***GenericApplicationContext***/***AnnotationConfigApplicationContext***
  - Consistent detection of transaction, caching, async annotations on interface methods.
- Spring Web MVC
  - Support for Servlet 4.0 ***PushBuilder*** argument in Spring MVC controller methods
  - Data binding with immutable objects (***Kotlin*** / ***Lombok*** / ***@ConstructorProperties***)
  - Support for Reactor 3.1 ***Flux*** and ***Mono*** as well as ***RxJava*** 1.3 and 2.1 as return values from Spring MVC controller methods.

What’s new in Spring Framework 5.0

- Spring WebFlux
  - New spring-webflux module, an alternative to spring-webmvc built on a reactive foundation
  - Fully asynchronous and non-blocking
  - @Controller style, annotation-based, programming model, similar to Spring MVC, but supported in WebFlux, running on a reactive stack.
  - New functional programming model ("WebFlux.fn") as an alternative to the ***@Controller***, annotation-based, programming model.
  - New WebClient with a functional and reactive API for HTTP calls
- Kotlin support
- Testing Improvements
- HTTP 2 support

Demo

REST API with Spring

Lab - optional

A simple web application with Spring

Summary

- Spring Framework
  - open source, lightweight, container, framework
  - Spring 5: webflux module, Java 9 ready
- Spring Application Development
  - configuration + classes >> container = application
  - web Application
    - DispatcherServlet, JPA and XML Config, deployment

Pivotal

A NEW PLATFORM FOR A NEW ERA

Spring Boot

Spring Boot Developer

An overview of Spring Boot

Spring Framework

Remember?

- Spring Web development requirements:
  - web:
    - DispatcherServlet, XML Context
  - data:
    - DataSource, TransactionManager, EntityManagerFactory
  - dependency:
    - maven, gradle, ant, ivy
  - logging, property files, monitoring, metrics, security?

Agenda

- Spring Boot
- Spring Boot Application Development

Spring Boot

What is Spring Boot?

- **OPINIONATED** runtime for Spring projects
- Next generation of Spring applications
- **R**apid **A**pplication **D**evelopment
- Easy to use features

Spring Boot

Provides

- Sensible defaults
- **Auto Configuration**
- Ability to create ***stand-alone*** (server-less: runnable) and ***deployable*** applications
- Full control over any configuration:
  - xml, java config, annotations, ***application.properties/yml***

Spring Boot

Supports different project types:

- web, batch, jdbc, integration, messaging, cloud, and more...

Spring Boot

It is not

- IDE plugin
- Code generator
- Scaffolding

Agenda

- Spring Boot
- Spring Boot Application Development

Spring Boot Application Development

Spring Boot Components

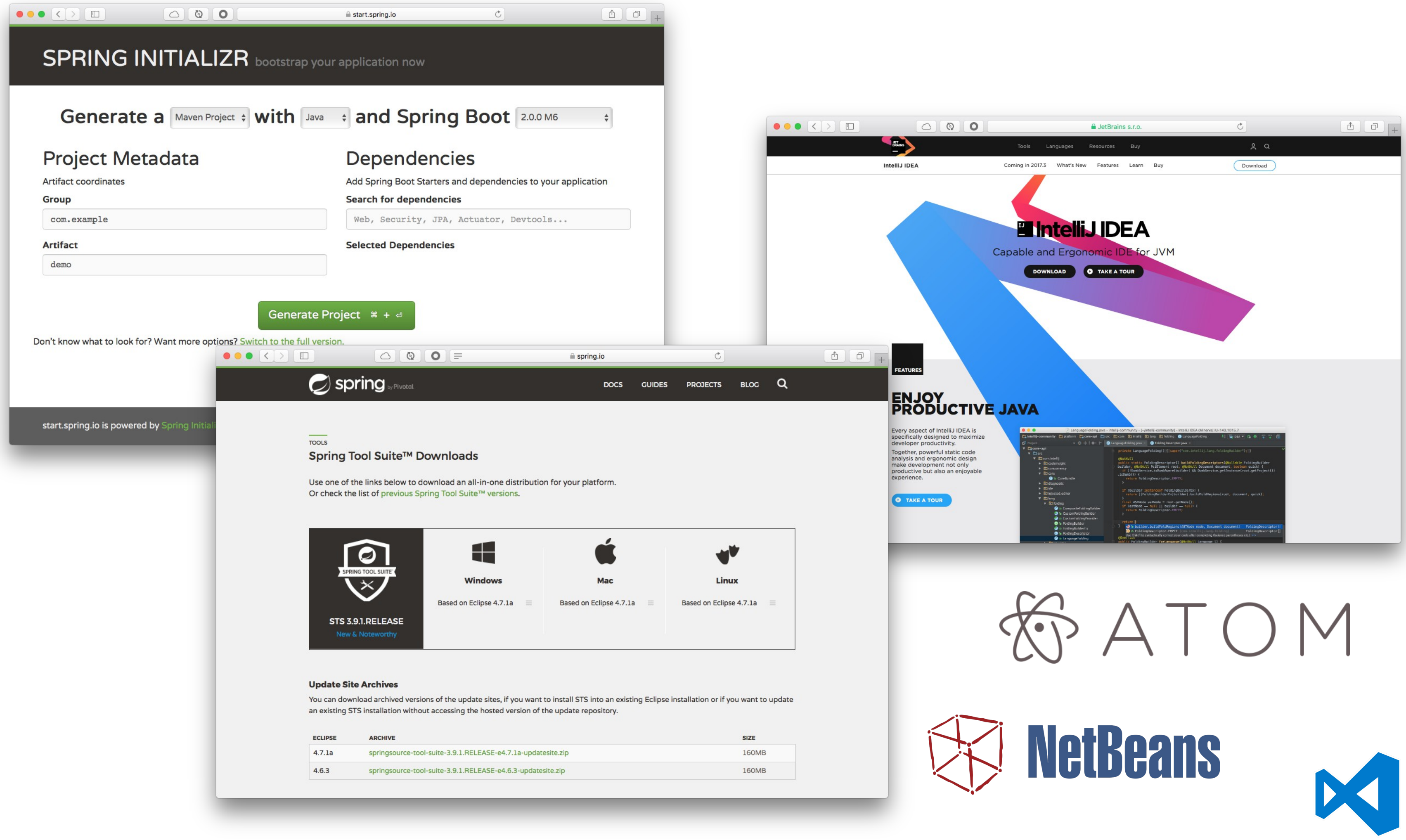
- Dependency Management:
  - maven, gradle, ant, ivy
    - ***<parent/>***
    - ***<dependency/>***:
      - ***spring-boot-starter*** technology
    - ***<plugin/>***
- main application
  - ***@SpringBootApplication***
  - ***SpringApplication.run***

Spring Boot Application Development

Ways to create a Spring Boot application

- Spring Boot Initializr - https://start.spring.io/
- IDE
  - Spring Tool Suite - https://spring.io/tools/sts/all
  - Intellij IDEA - https://www.jetbrains.com/idea/download/
  - Netbeans - https://netbeans.org/downloads/
  - Atom - https://atom.io/
  - VSCode - https://code.visualstudio.com/
- Spring Boot CLI

Spring Boot Application Development



start.spring.io

SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.0.0 M6

Project Metadata

Artifact coordinates

Group: com.example

Artifact: demo

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies: Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Generate Project ⌘ + ⏎

Don't know what to look for? Want more options? Switch to the full version.

start.spring.io is powered by Spring Initializ

spring.io

TOOLS

Spring Tool Suite™ Downloads

Use one of the links below to download an all-in-one distribution for your platform. Or check the list of previous Spring Tool Suite™ versions.

STS 3.9.1.RELEASE — New & Noteworthy

Windows	Mac	Linux
Based on Eclipse 4.7.1a	Based on Eclipse 4.7.1a	Based on Eclipse 4.7.1a

Update Site Archives

You can download archived versions of the update sites, if you want to install STS into an existing Eclipse installation or if you want to update an existing STS installation without accessing the hosted version of the update repository.

ECLIPSE	ARCHIVE	SIZE
4.7.1a	springsource-tool-suite-3.9.1.RELEASE-e4.7.1a-updatesite.zip	160MB
4.6.3	springsource-tool-suite-3.9.1.RELEASE-e4.6.3-updatesite.zip	160MB

JetBrains s.r.o.

IntelliJ IDEA

Capable and Ergonomic IDE for JVM

FEATURES

ENJOY PRODUCTIVE JAVA

Every aspect of IntelliJ IDEA is specifically designed to maximize developer productivity.

Together, powerful static code analysis and ergonomic design make development not only productive but also an enjoyable experience.

Demo

Simple Spring Boot app

Lab

Create a REST Spring Boot Web App

Summary

- Spring Boot
  - ***Opinionated*** Runtime for spring projects
  - Provides ***sensible defaults*** (best practices)
  - Components:
    - dependency, starter, @SpringBootApplication, SpringApplication.run
  - Spring Initializr, IDE, Spring Boot CLI

Pivotal

A NEW PLATFORM FOR A NEW ERA

Spring Boot Internals

Spring Boot Developer

An overview of auto-configuration

Spring Boot Internals

Remember?

- Spring Boot is an ***OPINIONATED*** runtime for Spring projects

Agenda

- Spring Boot auto-configuration

Spring Boot auto-configuration

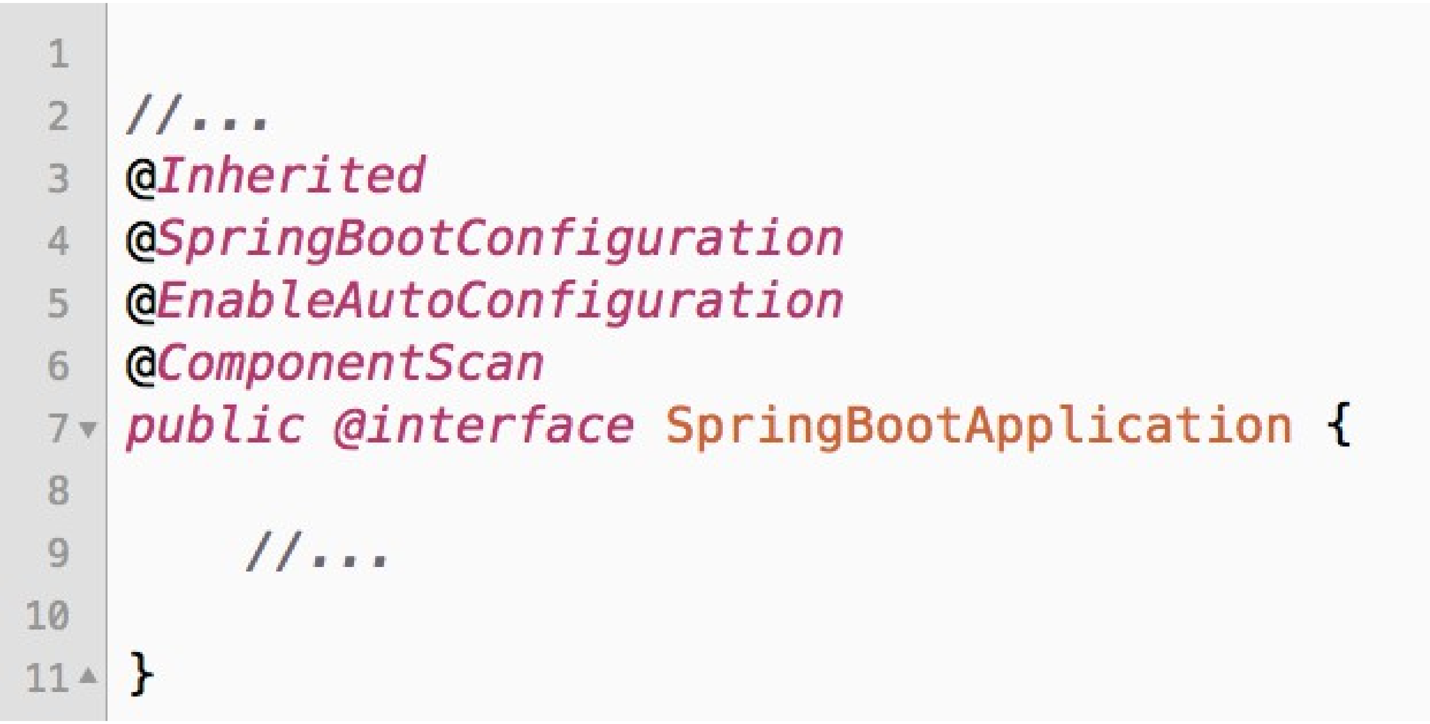
- Spring Boot uses sensible defaults based on what dependencies are on the classpath

- **auto-configuration** is enabled by using the *@EnableAutoConfiguration* annotation

- Where or how to use this annotation?

Spring Boot auto-configuration

- ***@SpringBootApplication*** is a composite annotation.



```

//...
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public @interface SpringBootApplication {

    //...

}
```

Spring Boot auto-configuration

- ***@EnableAutoConfiguration*** reads the ***spring-boot-autoconfigure/META-INF/spring.factories***

- The ***spring.factories*** file contains a list of classes (****AutoConfiguration***) that have all the logic to be executed accordingly to the dependencies that an application has in the classpath.

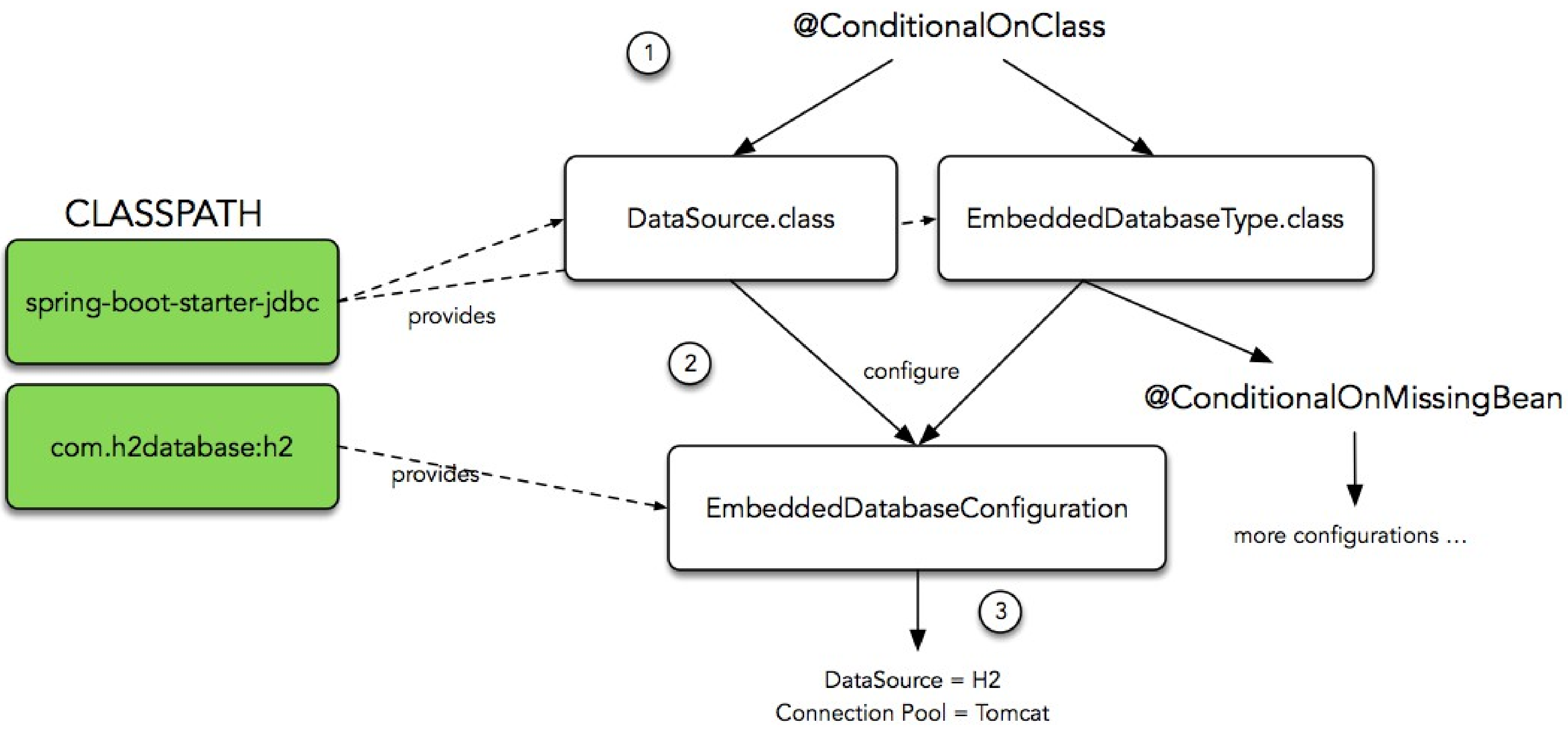
Spring Boot auto-configuration

The ***AutoConfiguration** classes use:

- *@ConditionalOnClass*
- *@ConditionalOnBean*
- *@ConditionalOnProperty*
- *@ConditionalOnMissingBean*
- *@ConditionalOnMissingClass*

and more ... to set the defaults for the Spring application, the necessary ***Spring Beans***.

Spring Boot auto-configuration



1. Is there a DataSource and EmbeddedDatabaseType classes in the classpath? Is there any DataSource Bean defined?

Demo

DataSourceAutoConfiguration review

Lab

Using @Conditional annotations...

Summary

- Spring Boot Internals
  - Opinionated runtime for spring projects
  - Provides sensible defaults (best practices)
- auto-configuration
  - Based on annotations: *@Conditional**

Pivotal

A NEW PLATFORM FOR A NEW ERA

Spring Boot Features

Spring Boot Developer

Discovering Spring Boot features

Agenda

- Packaging
- SpringApplication
- External Configuration
- Profiles
- Logging

Spring Boot Features: packaging

Spring Boot can create executable applications

Maven:

```
./mvnw package
```

Gradle:

```
./gradlew build
```

Run:

```
java -jar myapp.jar
```

Spring Boot Features: packaging

- A Spring Boot web application will have an embedded servlet container

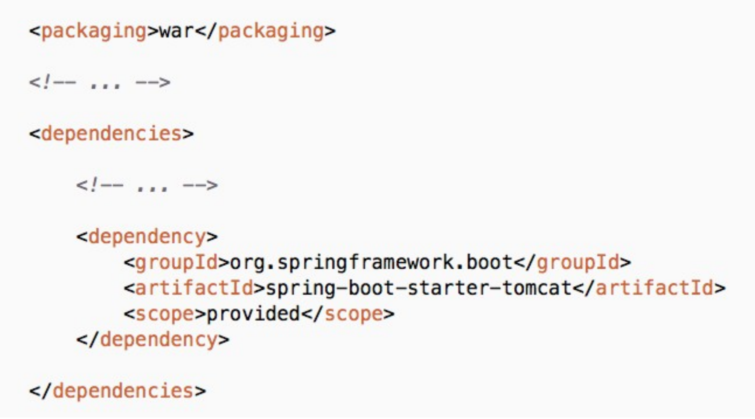
- Spring Boot supports: ***Tomcat***, ***Undertow*** and ***Jetty***

- ***Tomcat*** is the default

Spring Boot Features: packaging

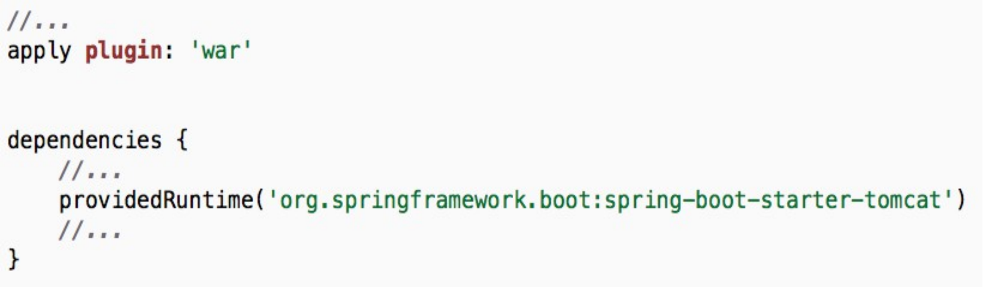
An executable / deployable WAR must have:

maven

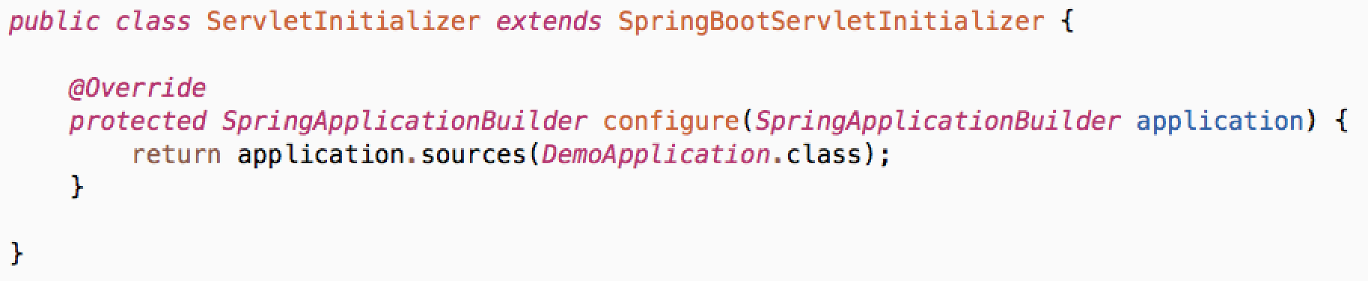


```xml
<packaging>war</packaging>

<!-- ... -->

<dependencies>

    <!-- ... -->

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>

</dependencies>
```

gradle



```groovy
//...
apply plugin: 'war'


dependencies {
    //...
    providedRuntime('org.springframework.boot:spring-boot-starter-tomcat')
    //...
}
```

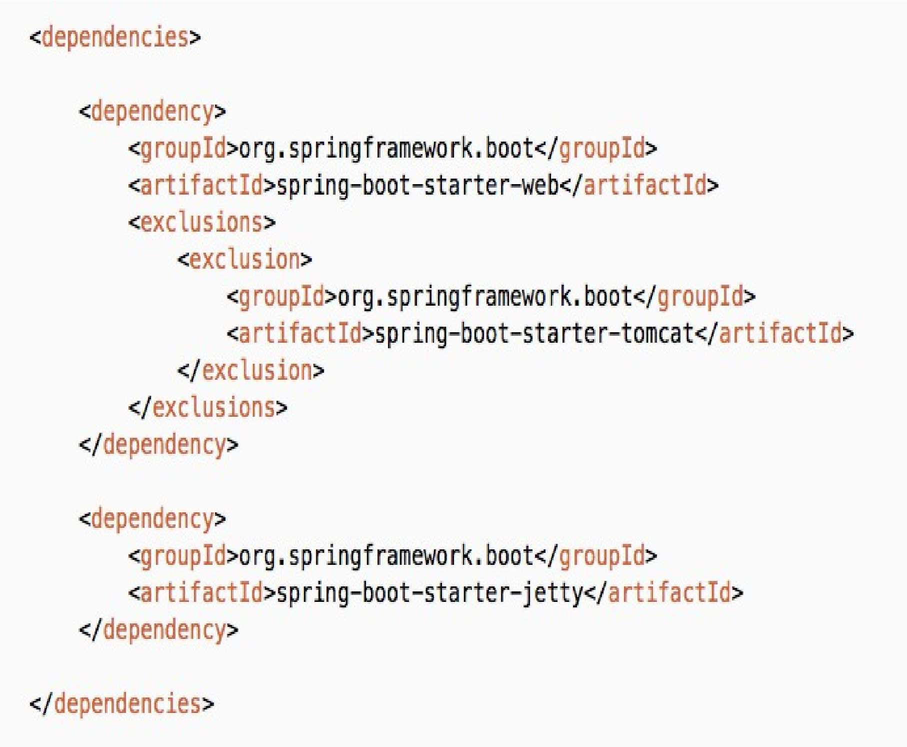


```java
public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(DemoApplication.class);
    }

}
```

Spring Boot Features: packaging

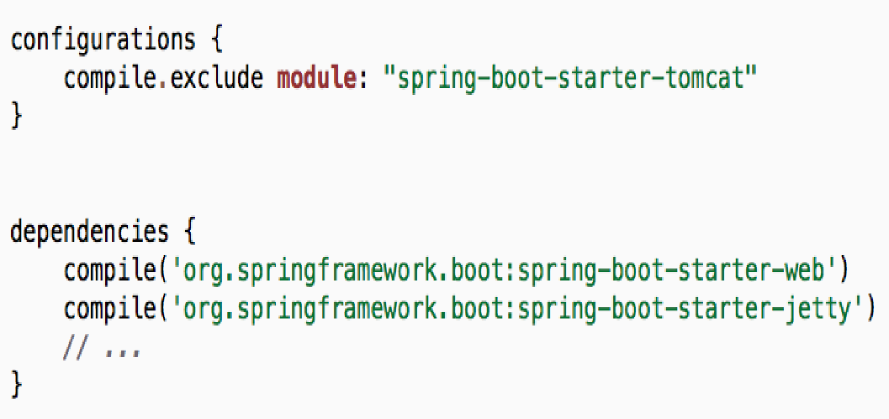
Spring Boot allows to override the defaults:

maven



```xml
<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-tomcat</artifactId>
            </exclusion>
        </exclusions>
    </dependency>

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

</dependencies>
```

gradle



```groovy
configurations {
    compile.exclude module: "spring-boot-starter-tomcat"
}


dependencies {
    compile('org.springframework.boot:spring-boot-starter-web')
    compile('org.springframework.boot:spring-boot-starter-jetty')
    // ...
}
```

Demo

packaging

Agenda

- Packaging
- SpringApplication
- External Configuration
- Profiles
- Logging

Spring Boot Features: *SpringApplication*

SpringApplication class bootstraps a Spring application and it provides:

- A way to customize the banner
- Customize the application through ***application.properties***/***yml***
- A fluent API builder: ***SpringApplicationBuilder***
- Events and Listeners
- Application Type: ***setWebApplicationType***
  - ***WebApplicationType.NONE, WebApplicationType.SERVLET, WebApplicationType.REACTIVE***
- Access to application arguments
- Run specific code once the SpringApplication has started
- Admin features: ***MBeanServer***

Demo

SpringApplication class

Agenda

- Packaging
- SpringApplication
- External Configuration
- Profiles
- Logging

Spring Boot Features: External Configuration

Spring Boot allows externalize configuration to use the same code in different environments through:

- ***application.properties*** / ***application.yml***
- Environment variables
- Command line arguments

Spring Boot Features: External Configuration

- Property values can be injected using ***@Value*** annotation or can be bound to structured objects via ***@ConfigurationProperties***

- Spring Boot uses a ***PropertySource*** order to allow value overriding

- Spring Boot uses ***relaxed binding rules*** for binding

Spring Boot Features: External Configuration

Spring Boot uses a very particular ***PropertySource*** order that is designed to allow sensible overriding of values; to name a few:

...

Command line arguments.

Properties from SPRING_APPLICATION_JSON

...

Java System properties (System.getProperties()).

OS environment variables.

...

Profile-specific application properties outside of your packaged jar

Profile-specific application properties packaged inside your jar

Application properties outside of your packaged jar

Application properties packaged inside your jar

...

Spring Boot Features: External Configuration

SpringApplication will load properties from ***application.properties*** files in the following locations and add them to the spring ***Environment***:

- */config* subdirectory of the current directory.
- current directory
- classpath */config* package
- classpath root

Demo

External Configuration

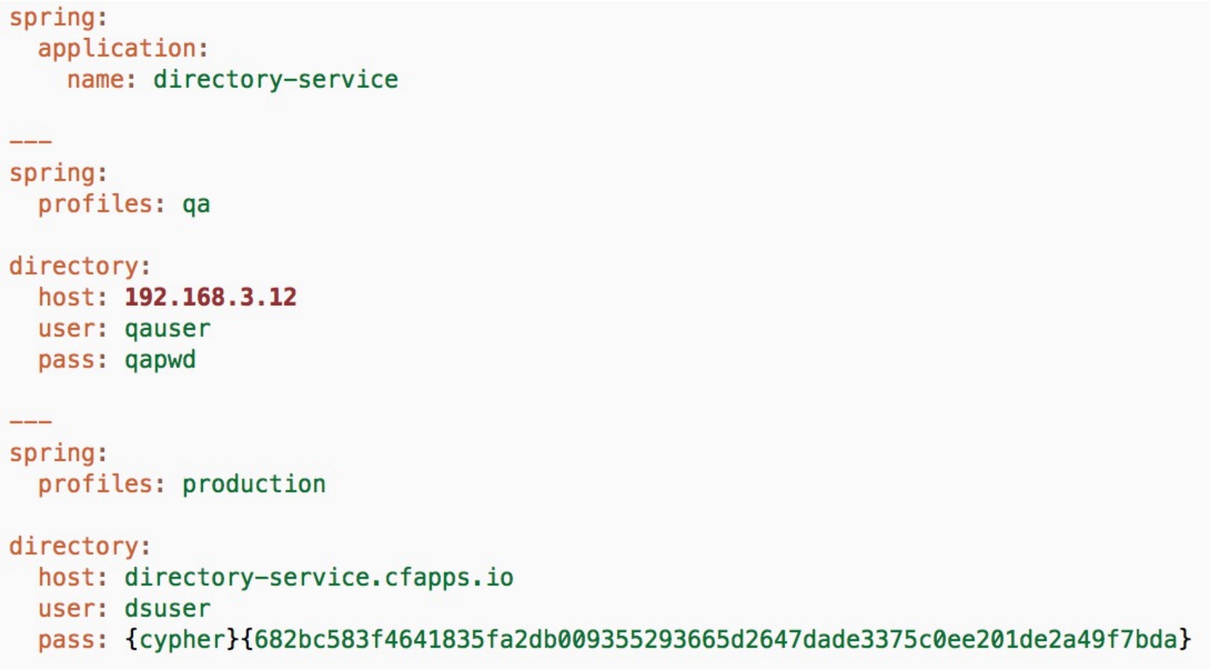
Agenda

- Packaging
- SpringApplication
- External Configuration
- Profiles
- Logging

Spring Boot Features: Profiles

Spring Boot allows to use profile-specific properties:

- ***application-{profile}.properties***
- a single ***application.yml*** that contains profiles blocks:



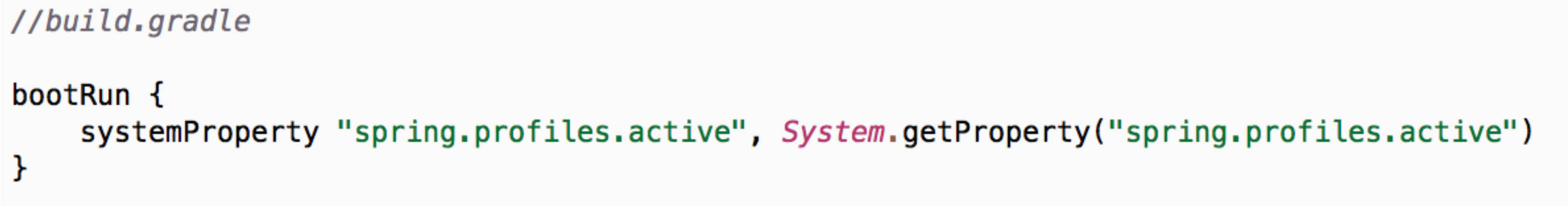
```
spring:
  application:
    name: directory-service

---
spring:
  profiles: qa

directory:
  host: 192.168.3.12
  user: qauser
  pass: qapwd

---
spring:
  profiles: production

directory:
  host: directory-service.cfapps.io
  user: dsuser
  pass: {cypher}{682bc583f4641835fa2db009355293665d2647dade3375c0ee201de2a49f7bda}
```

Spring Boot Features: Profiles

maven:

./mvnw spring-boot:run -Dspring.profiles.active=dev

gradle:



```
//build.gradle

bootRun {
    systemProperty "spring.profiles.active", System.getProperty("spring.profiles.active")
}
```

./gradlew bootRun -Dspring.profiles.active=dev

JAR:

SPRING_PROFILES_ACTIVE=production java -jar myapp.jar

or

java –Dspring.profiles.active=qa -jar myapp.jar

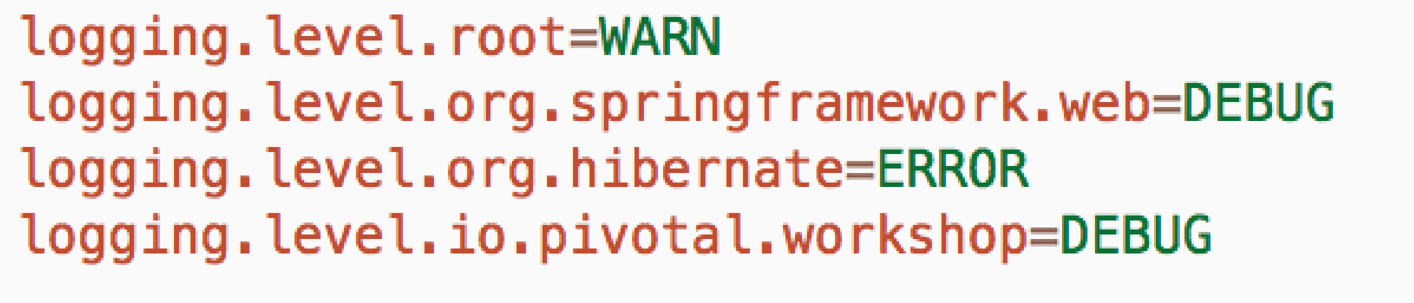
Agenda

- Packaging
- SpringApplication
- External Configuration
- Profiles
- Logging

Spring Boot Features: Logging

- Spring Boot uses ***Commons Logging*** for all internal logging, ***Logback*** will be used by default.

- Spring Boot support logger levels configuration: *TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF* through the ***logging.level.**** properties in the ***application.properties/yml*** file



```
logging.level.root=WARN
logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=ERROR
logging.level.io.pivotal.workshop=DEBUG
```

Lab

Spring Boot Features

Summary

- Packaging: **JAR** and **WAR**
  - **WAR**: executable and deployable
- ***SpringApplication***
  - Customizable: banner, fluent API builder, etc
- External Configuration
- Profiles
- Logging

Pivotal

A NEW PLATFORM FOR A NEW ERA

Web Development with Spring Boot

Spring Boot Developer

Spring MVC

Pivotal™

Spring Web Development

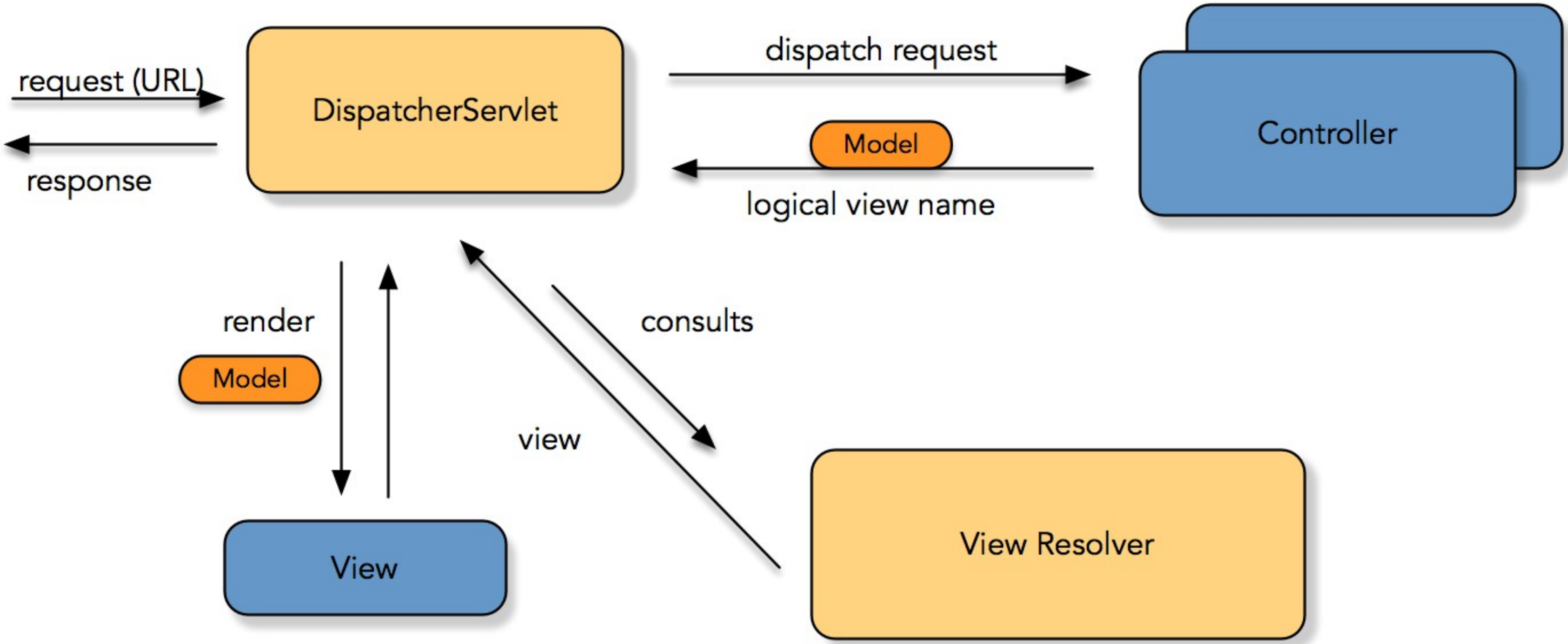
Remember?

- web.xml
- DispatcherServlet
- <context:component-scan />
- View resolvers

Agenda

- Spring Web MVC
- Spring Boot Web Development

Spring Web MVC



Agenda

- Spring Web MVC
- Spring Boot Web Development

Spring Boot Web Development

- Spring Boot uses the power of **Spring Web MVC** to create powerful web applications with ease
- Spring Boot Web applications can be created by adding the ***spring-boot-starter-web*** dependency
- The ***spring-boot-starter-web*** dependency brings the ***spring-web***, ***spring-webmvc*** jars and other additional libraries like *tomcat*, *jackson*, etc.

Spring Boot Web Development

- Spring Boot will auto-configure the ***DispatcherServlet***, content and view resolvers

- You can use all the ***spring-mvc*** annotations:
  - @Controller / @RestController
  - @RequestMapping
  - @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping
  - @PathVariable, @RequestParam, @RequestHeader, @RequestBody, @RequestAttribute, @ModelViewAttribute
  - @SessionAttributes, @ModelAttribute, @CookieValue
  - @ControllerAdvice, @RestControllerAdvice
  - @RequestPart, @ExceptionHandler,
  - ServletRequest, HttpServletRequest, Principal, and more...

Spring Boot Web Development

Supports for serving static resources:

- */static*, */public*, */META-INF/resources*
- */webjars/***
- *index.html* and custom *favicon*

Spring Boot Web Development

- Support for **JSON** and **XML** serialization
- Multiple template engine support:
  - Groovy Server Pages
  - Freemaker
  - Velocity
  - Mustache
  - Thymeleaf

Spring Boot Web Development

Support embedded servlet containers:

- Servlet 3.x engines
- Access and compatibility with J2EE annotations:
  - @WebServlet
  - @WebFilter
  - @WebListener

Spring Boot Web Development

Customization through *application.properties/yml*

Custom network configuration:

- *server.port*, *server.address*

Custom embedded servlet container configuration:

- *server.session.**, *server.compression.**, *server.servlet.**

Custom error pages by providing:

- *src/main/resources/public/error/<status-code>.html*

Demo

Web App with Spring Boot

Lab

Code Snippet Manager application

Summary

- Spring Boot Web Development
  - Opinionated Runtime for Web Projects
  - Uses the power of ***Spring Web MVC***
  - Highly Customizable

Pivotal

A NEW PLATFORM FOR A NEW ERA

Data Access with Spring Boot

Spring Boot Developer

Data access with JDBC, JPA and REST

Spring Boot Data Access

Remember?

- persistence.xml
- DataSource
- TransactionManager
- EntityFactoryManager

Agenda

- JDBC
- Spring Data JPA
- Spring Data Rest
- NoSQL
- Additional features

Spring Boot Data Access: JDBC

- Spring Boot uses the extensive support from the Spring Framework for working with SQL databases

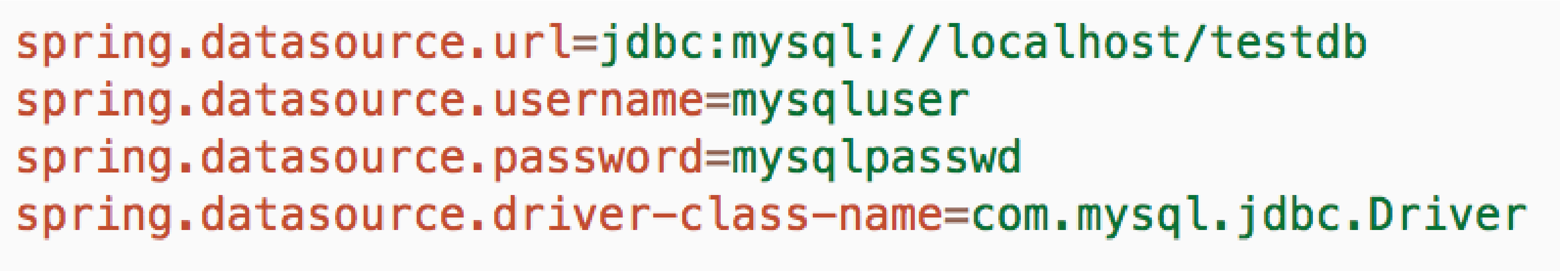
- Spring Boot uses direct access from ***JdbcTemplate*** to complete **ORM** technologies like **Hibernate**

- Spring Boot JDBC applications can be created by adding the ***spring-boot-starter-jdbc*** and the **SQL** driver dependencies

Spring Boot Data Access: JDBC

- Spring Boot will auto-configure the ***DataSource*** based on default properties or any existing configuration

- ***DataSource*** properties can be overridden in the *application.properties*/*yml* file



```
spring.datasource.url=jdbc:mysql://localhost/testdb
spring.datasource.username=mysqluser
spring.datasource.password=mysqlpasswd
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

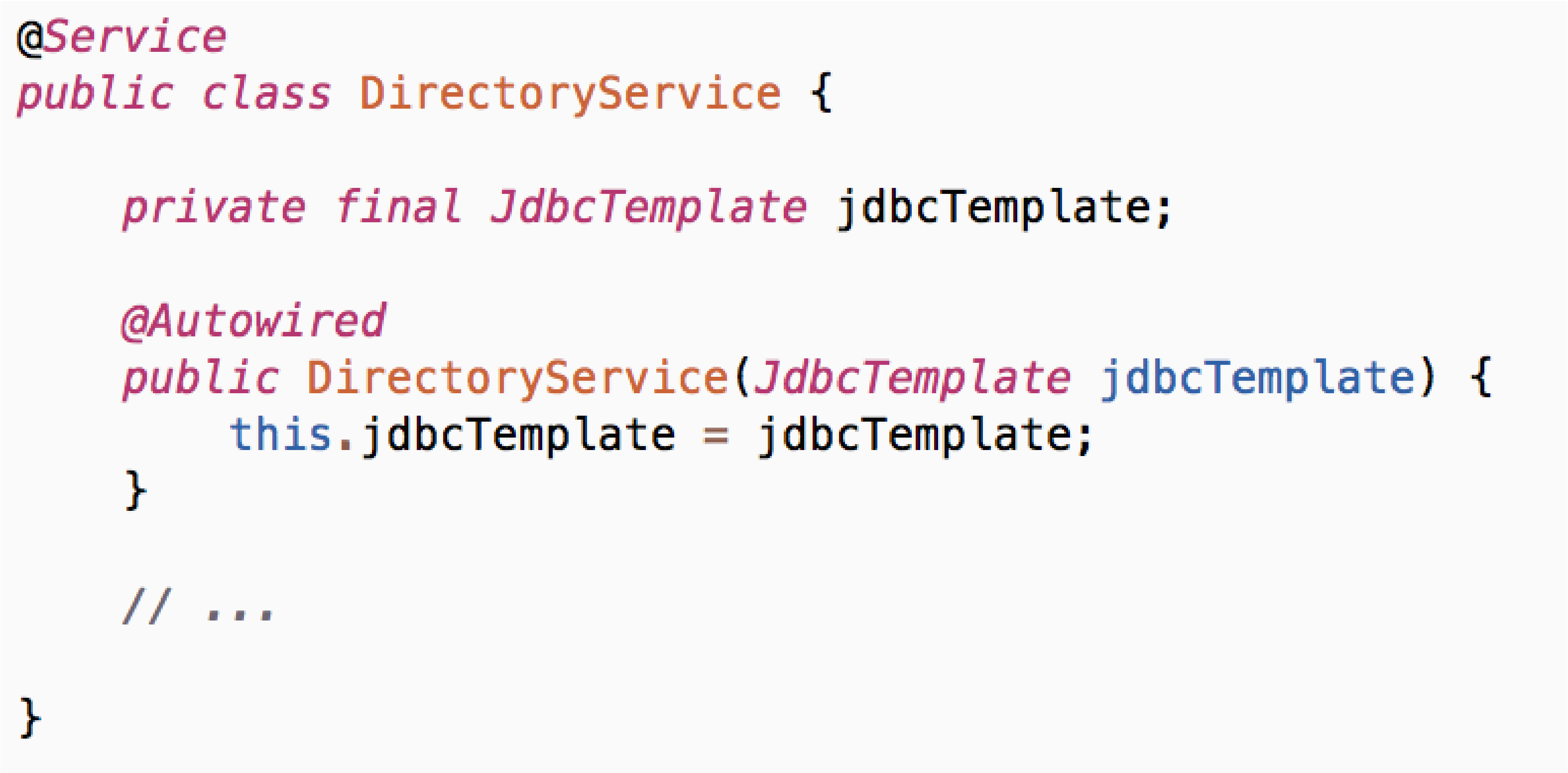
- Multiple ***DataSource*** bean definitions can exist

Spring Boot Data Access: JDBC

- Spring Boot uses **HikariCP** as the default connection pool
- Spring Boot support embedded databases: **H2**, **HSQL** and **Derby**
- Spring Boot uses the Spring JDBC initializer feature, it loads SQL from *schema.sql* and *data.sql*
- Spring Boot also supports the *schema-${platform}.sql* and *data-${platform}.sql*

Spring Boot Data Access: JDBC

- Spring Boot auto-configures the *JdbcTemplate* so it's easy to use in any spring bean



```
@Service
public class DirectoryService {

    private final JdbcTemplate jdbcTemplate;

    @Autowired
    public DirectoryService(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    // ...

}
```

Demo

JDBC Demo with Spring Boot

Lab

JDBC

Agenda

- JDBC
- Spring Data JPA
- Spring Data Rest
- NoSQL
- Additional features

Spring Boot Data Access: JPA

- Spring Boot uses the power of the ***Spring Data*** project to create data applications with ease

- Spring Boot JPA applications can be created by adding the *spring-boot-starter-data-jpa* and the **SQL** driver dependencies.

Spring Boot Data Access: JPA

Spring Data

- Relies on the Java Persistence API
- Repository generation base on interfaces: *Repository*, *CrudRepository*, *JpaRepository*
- Custom object mapping
- Dynamic query derivation from repository method names
- Schema generation through *spring.jpa.** properties
- Initialization through an import.sql file

Lab

JPA

Agenda

- JDBC
- Spring Data JPA
- Spring Data Rest
- NoSQL
- Additional features

Spring Boot Data Access: Rest

- Spring Boot will use ***Spring Data Rest*** project to create hypermedia-driven REST web services on top of repositories

- Spring Boot data-rest applications can be created by adding the *spring-boot-starter-data-jpa*, *spring-boot-starter-data-rest* and the **SQL** driver dependencies

Spring Boot Data Access: data-rest

Spring Data Rest

- Exposes a discoverable REST API for your domain model using HAL as media type
- Exposes collection, item and association resources representing your model
- Supports pagination via navigational links
- Allows to dynamically filter collection resources
- Ships a customized variant of the HAL Browser
- Currently supports JPA, MongoDB, Neo4j, Solr, Cassandra, Gemfire
- Allows advanced customizations of the default resources exposed
- and more ...

Lab

Data Rest

Agenda

- JDBC
- Spring Data JPA
- Spring Data Rest
- NoSQL
- Additional features

Spring Boot Data Access: NoSQL

- Spring Boot will use ***Spring Data*** project to create NoSQL data applications with ease

- Spring Boot will provide auto-configuration for: **Redis**, **MongoDB**, **Neo4j**, **Elasticsearch**, **Solr**, **Cassandra**, **Couchbase** and **LDAP**, so is easy to use its respective *<data-technology>Template* class.

- Spring Boot NoSQL applications can be created by adding the necessary data starter technology dependency

Agenda

- JDBC
- Spring Data JPA
- Spring Data Rest
- NoSQL
- Additional features

Spring Boot Data Access: Additional features

Spring Boot provides additional tools and functionality:

- Higher-level database migration tool: *flyway* and *liquibase*
- H2's web console through: /h2-console endpoint
  - *spring.h2.console.enable = true* to enable it
  - it can be secured
- Support for *jOOQ* (Java Object Oriented Querying)
  - Code generation through *jooq-codegen-maven* plugin
  - auto-configuration of the ***DSLContext*** interface
  - Customization of *jOOQ* by setting *spring.jooq.sql-dialect* property

Summary

- Spring Boot Data Access
- JDBC: auto-configuration of the DataSource / JdbcTemplate
- JPA: based on spring data - repositories, mapping, query methods
- Rest: based on spring data rest - restful implementation of the domain object through HATEOAS
- NoSQL: based on spring data
- Additional features: h2-console, flyway and liquibase, jooq

Pivotal

A NEW PLATFORM FOR A NEW ERA

Testing with Spring Boot

Spring Boot Developer

TDD with Spring Boot

Agenda

- Testing
- Spring Boot Testing

Testing

- Structure your code with clean separation of concerns so that individual parts can be unit tested.

- TDD is a good way to achieve this.

- Use constructor injection to ensure that objects can be instantiated directly. Don’t use field injection as it just makes your tests harder to write.

Agenda

- Testing
- Spring Boot Testing

Spring Boot Testing

- Spring Boot uses the ***Spring Test*** project to provide an easy way to execute unit and integration tests, facilitating a TDD approach

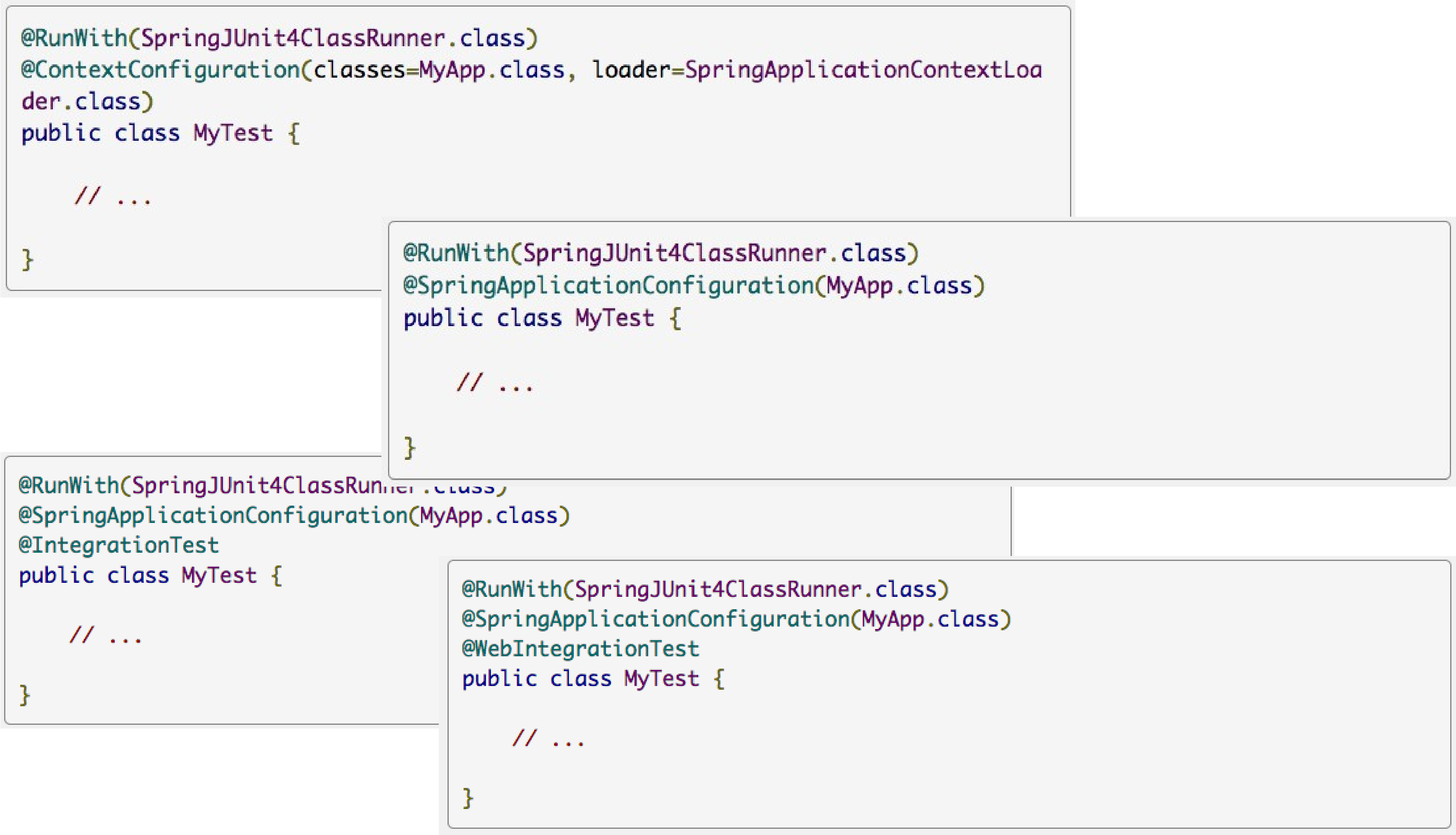
- Spring Boot Tests can be created by adding the *spring-boot-starter-test* dependency

Spring Boot Testing

The ***spring-boot-starter-test*** dependency provides:

- JUnit 5
- Spring Test & Spring Boot Test
- Assertj
- Hamcrest
- Mockito
- JsonAssert
- JsonPath

Old Spring Boot Testing



```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=MyApp.class, loader=SpringApplicationContextLoa
der.class)
public class MyTest {

    // ...

}
```

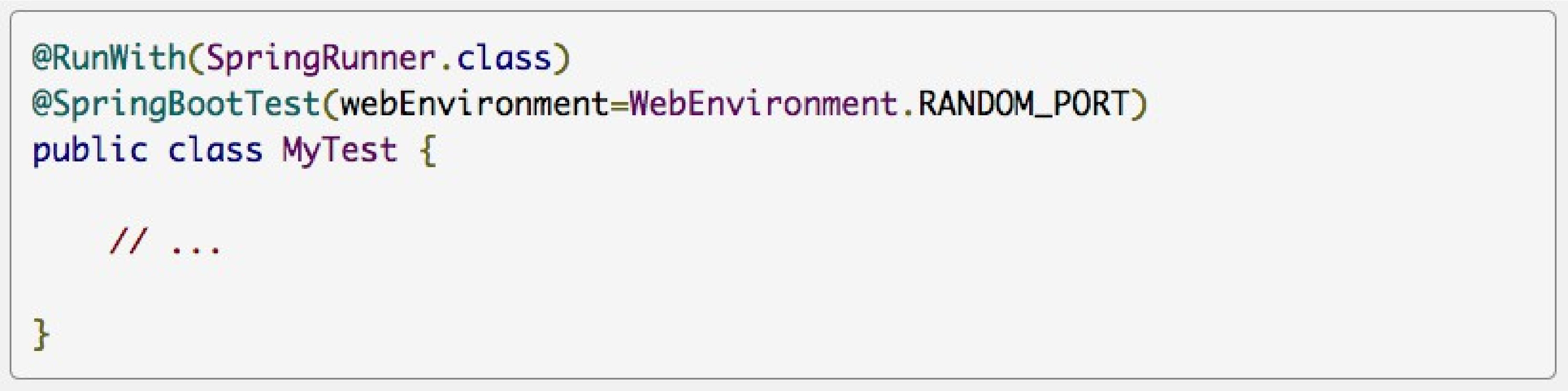
```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
public class MyTest {

    // ...

}
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
@IntegrationTest
public class MyTest {

    // ...

}
```

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(MyApp.class)
@WebIntegrationTest
public class MyTest {

    // ...

}
```

Spring Boot Testing

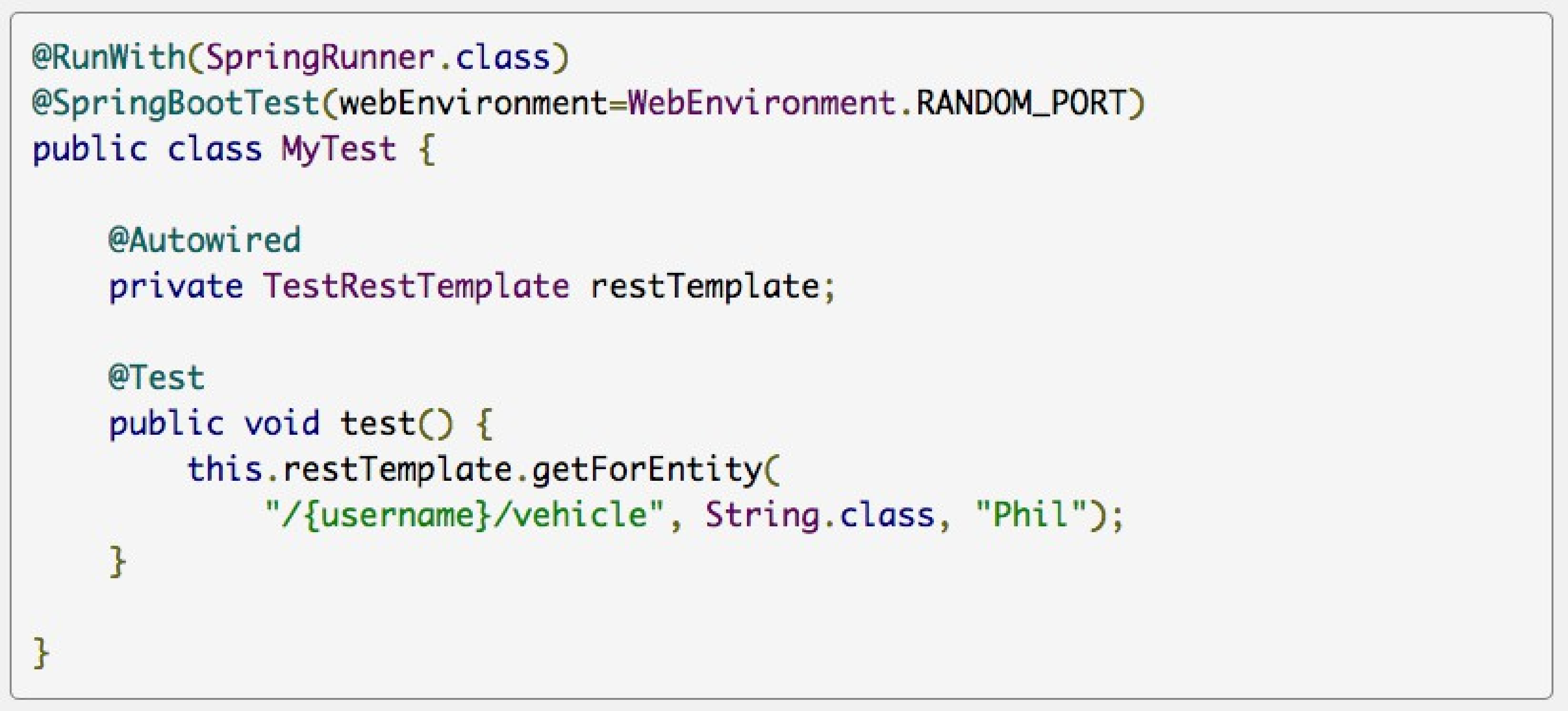
A new spring boot integration test will look like this:



```java
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.RANDOM_PORT)
public class MyTest {

    // ...

}
```

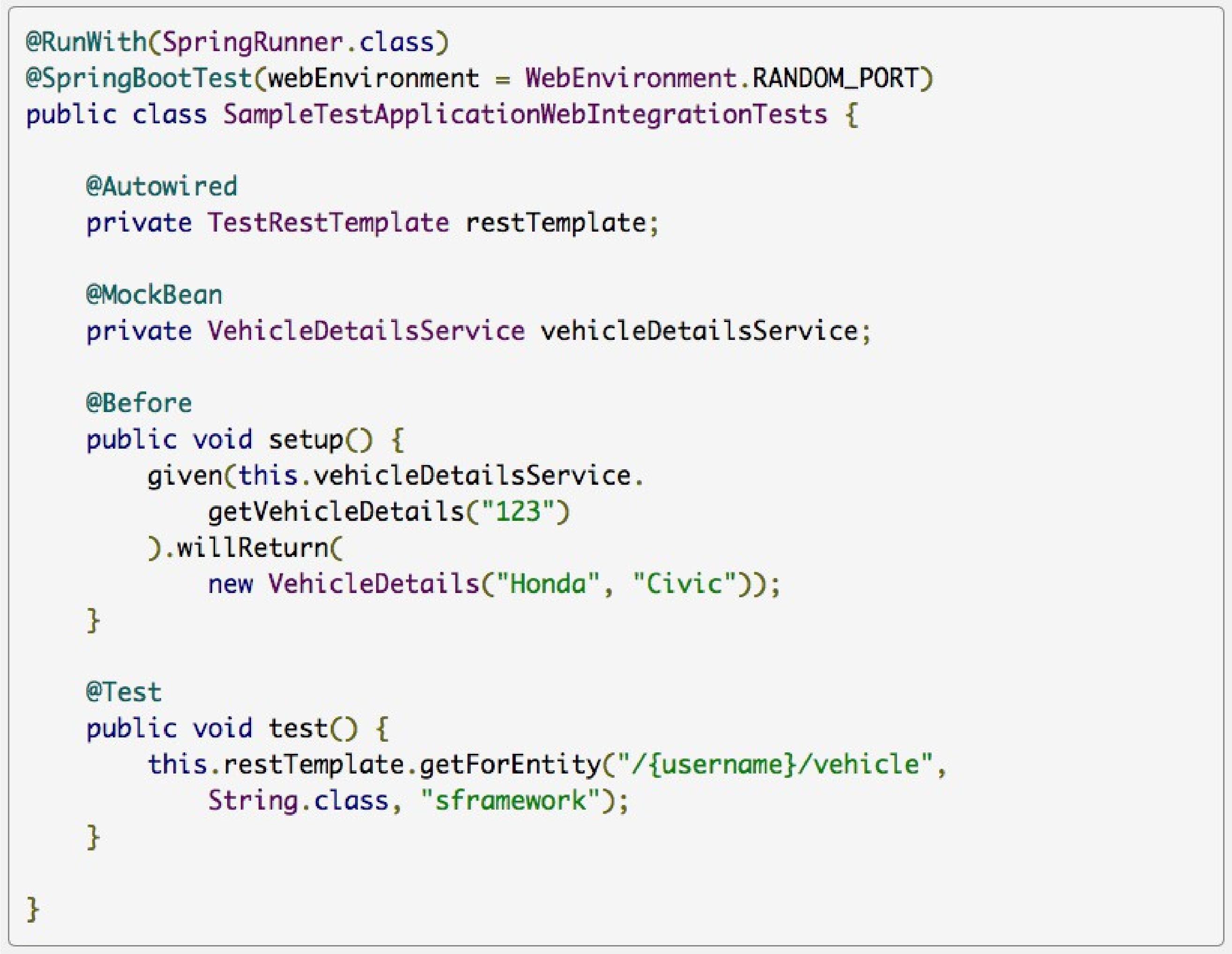
Spring Boot Testing

A more concrete example that actually hits a real REST endpoint:



```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.RANDOM_PORT)
public class MyTest {

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void test() {
        this.restTemplate.getForEntity(
            "/{username}/vehicle", String.class, "Phil");
    }

}
```

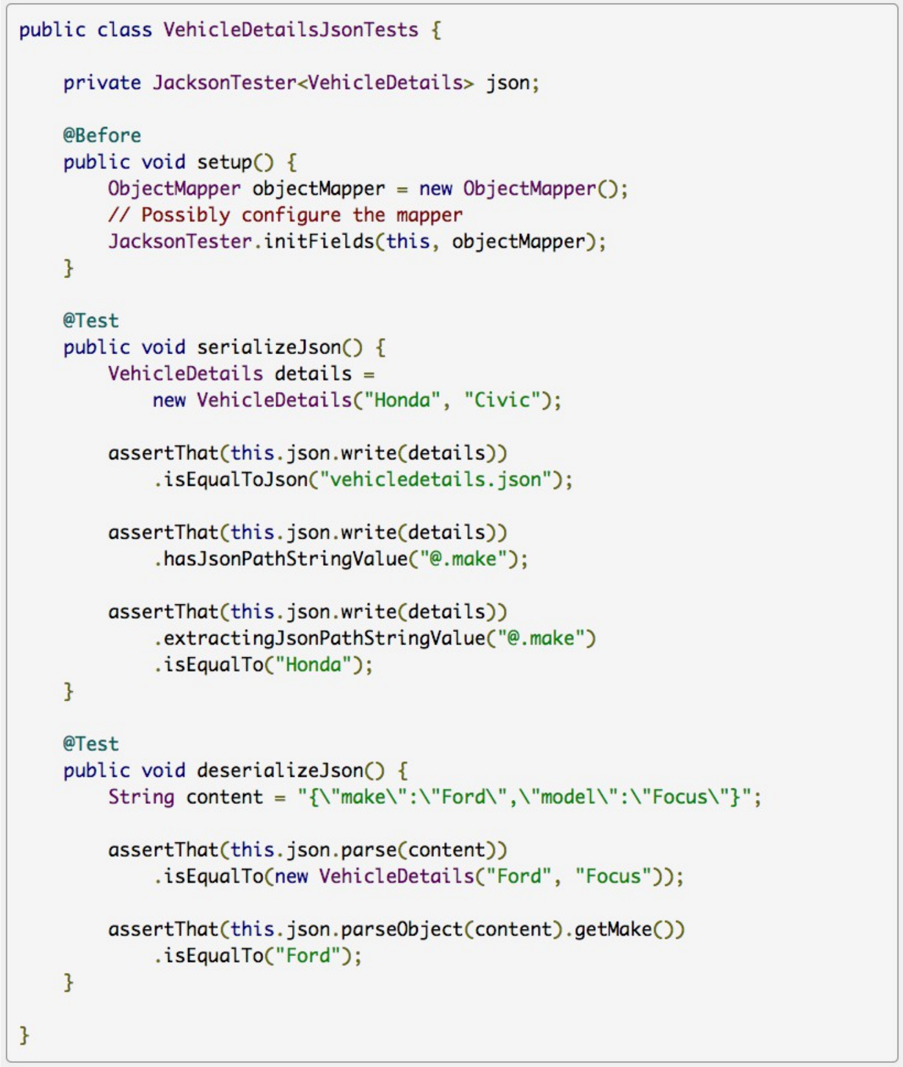
Spring Boot Testing



```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
public class SampleTestApplicationWebIntegrationTests {

    @Autowired
    private TestRestTemplate restTemplate;

    @MockBean
    private VehicleDetailsService vehicleDetailsService;

    @Before
    public void setup() {
        given(this.vehicleDetailsService.
            getVehicleDetails("123")
        ).willReturn(
            new VehicleDetails("Honda", "Civic"));
    }

    @Test
    public void test() {
        this.restTemplate.getForEntity("/{username}/vehicle",
            String.class, "sframework");
    }

}
```

Mocking and Spying

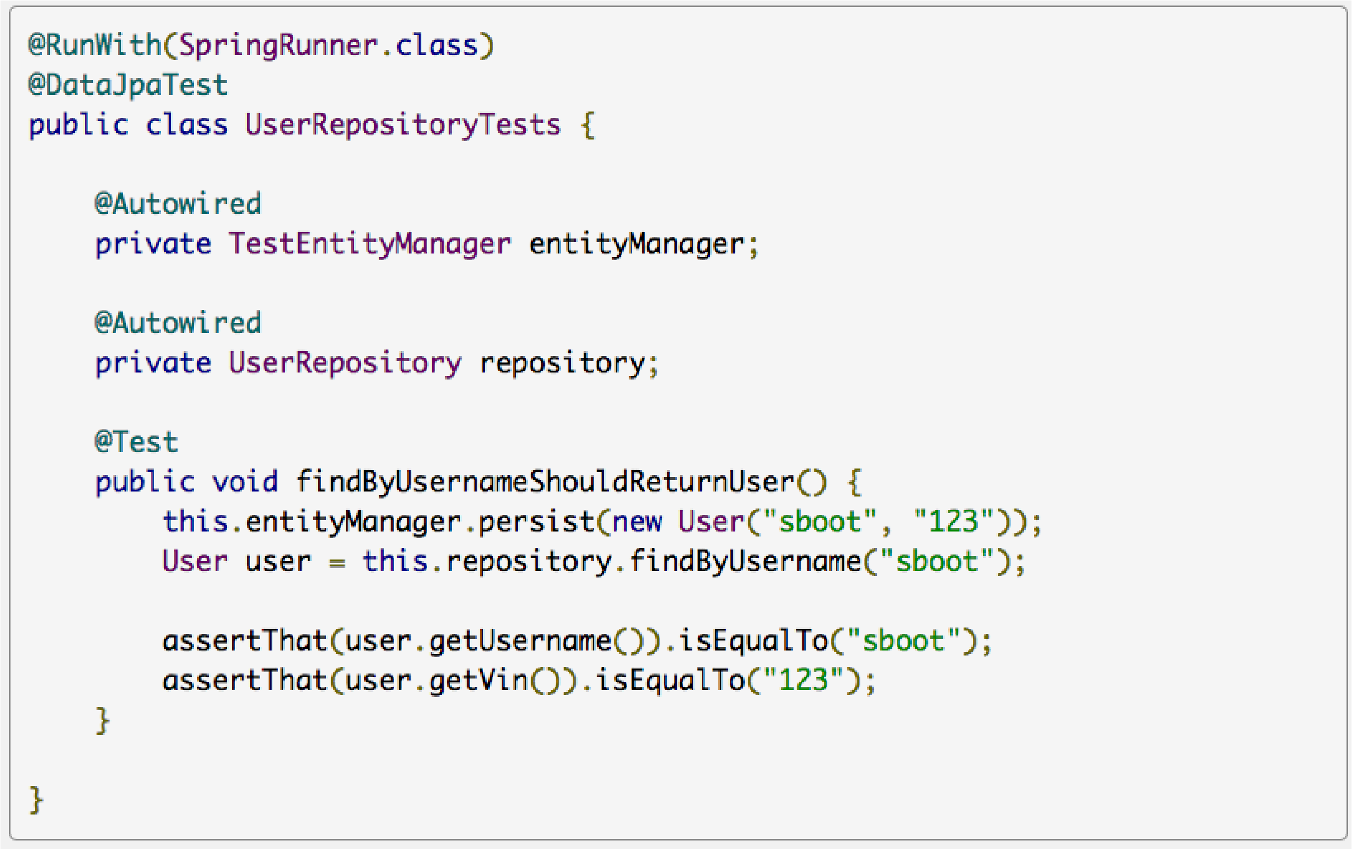
Spring Boot Testing



```
public class VehicleDetailsJsonTests {

    private JacksonTester<VehicleDetails> json;

    @Before
    public void setup() {
        ObjectMapper objectMapper = new ObjectMapper();
        // Possibly configure the mapper
        JacksonTester.initFields(this, objectMapper);
    }

    @Test
    public void serializeJson() {
        VehicleDetails details =
            new VehicleDetails("Honda", "Civic");

        assertThat(this.json.write(details))
            .isEqualToJson("vehicledetails.json");

        assertThat(this.json.write(details))
            .hasJsonPathStringValue("@.make");

        assertThat(this.json.write(details))
            .extractingJsonPathStringValue("@.make")
            .isEqualTo("Honda");
    }

    @Test
    public void deserializeJson() {
        String content = "{\"make\":\"Ford\",\"model\":\"Focus\"}";

        assertThat(this.json.parse(content))
            .isEqualTo(new VehicleDetails("Ford", "Focus"));

        assertThat(this.json.parseObject(content).getMake())
            .isEqualTo("Ford");
    }

}
```

JSON assertions

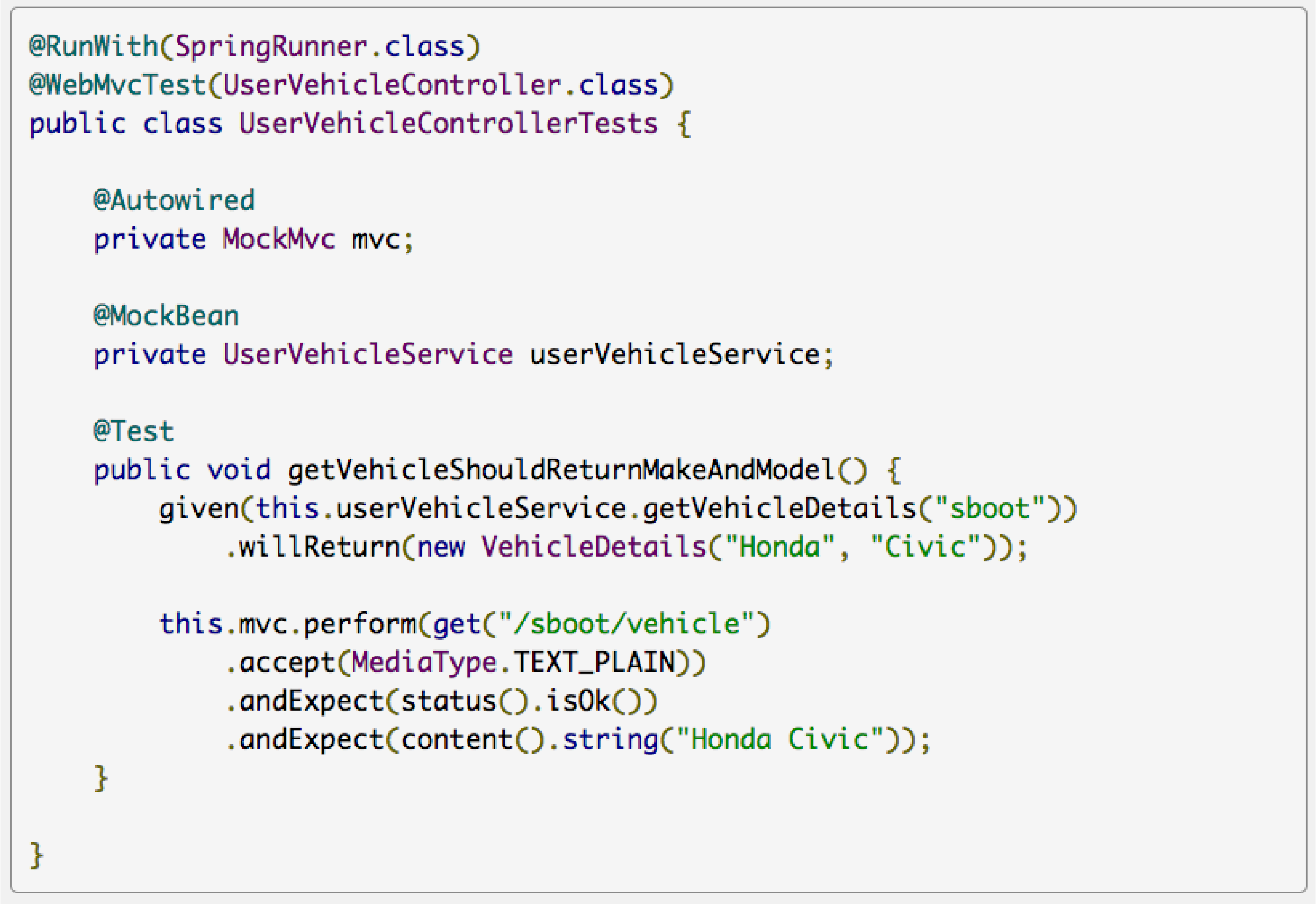
Spring Boot Testing



```
@RunWith(SpringRunner.class)
@DataJpaTest
public class UserRepositoryTests {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private UserRepository repository;

    @Test
    public void findByUsernameShouldReturnUser() {
        this.entityManager.persist(new User("sboot", "123"));
        User user = this.repository.findByUsername("sboot");

        assertThat(user.getUsername()).isEqualTo("sboot");
        assertThat(user.getVin()).isEqualTo("123");
    }

}
```

JPA slice

Spring Boot Testing

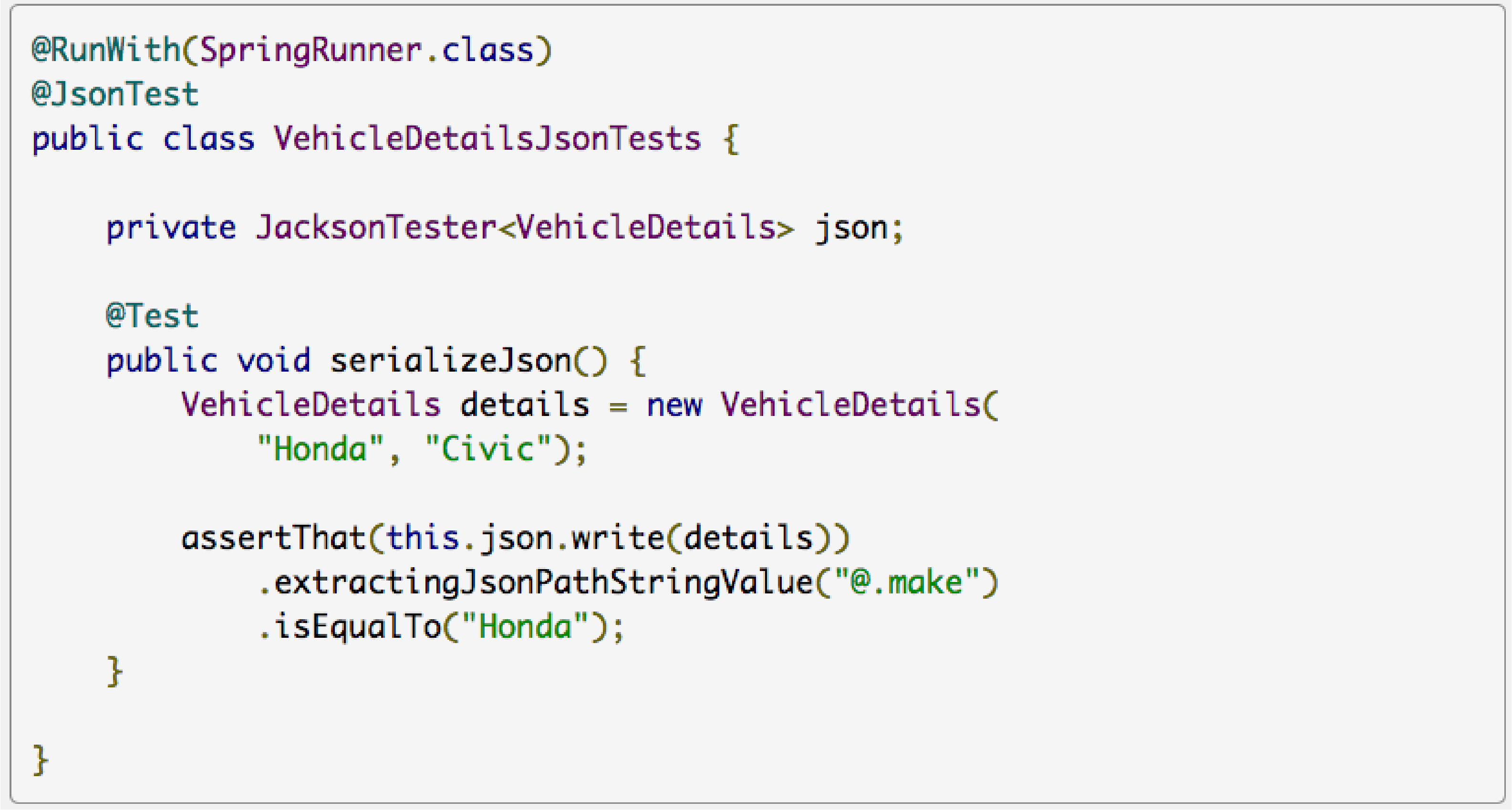


```java
@RunWith(SpringRunner.class)
@WebMvcTest(UserVehicleController.class)
public class UserVehicleControllerTests {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private UserVehicleService userVehicleService;

    @Test
    public void getVehicleShouldReturnMakeAndModel() {
        given(this.userVehicleService.getVehicleDetails("sboot"))
            .willReturn(new VehicleDetails("Honda", "Civic"));

        this.mvc.perform(get("/sboot/vehicle")
            .accept(MediaType.TEXT_PLAIN))
            .andExpect(status().isOk())
            .andExpect(content().string("Honda Civic"));
    }

}
```

MVC slice

Spring Boot Testing

JSON slice



```
@RunWith(SpringRunner.class)
@JsonTest
public class VehicleDetailsJsonTests {

    private JacksonTester<VehicleDetails> json;

    @Test
    public void serializeJson() {
        VehicleDetails details = new VehicleDetails(
            "Honda", "Civic");

        assertThat(this.json.write(details))
            .extractingJsonPathStringValue("@.make")
            .isEqualTo("Honda");
    }

}
```

Pivotal

Lab

Testing

Summary

- Spring Boot Testing
- Provides different libraries: mockito, jsonassert, etc
- Provides: @RunWith and @SpringBootTest annotations
- Slices: JPA, MVC, JSON

Pivotal

A NEW PLATFORM FOR A NEW ERA

Spring Boot Actuator

Spring Boot Developer

Out-of-the-box production-ready features

Non-Functional Requirements

- Every application nowadays required non-functional requirements, like monitoring, health checks and management

Agenda

- Spring Boot Actuator
- Metrics
- Health Indicators

Spring Boot Actuator

- Spring Boot includes a number of additional production-ready features to help you monitor and manage your application when it’s pushed to production

- Adding these production-ready features to a Spring Boot application is as easy as including the *spring-boot-starter-actuator*

Spring Boot Actuator

- Spring Boot ***Actuator*** provides HTTP endpoints through a Spring MVC based application

- ***Actuator*** endpoints allow you to monitor and interact with your application

- ***Actuator*** endpoints can be exposed also through **JMX** using **jolokia**

Spring Boot Actuator

- ***Actuator*** endpoint are mapped under ***/actuator***
- Spring Boot includes a number of built-in endpoints:

– conditions	displays an auto-configuration report
– beans	displays a complete list of all the spring beans
– dump	performs a thread dump
– env	exposes Spring’s ConfigurableEnvironment
– health	shows application health information
– info	displays arbitrary application info
– metrics	shows metrics information for the current application
– mappings	displays a collated list of all @RequestMapping paths
– shutdown	allows the application to be gracefully shutdown
– trace	displays trace information

…

- By default, only the **info** (*/actuator/info*)and the **health**(*/actuator/health*) are available.

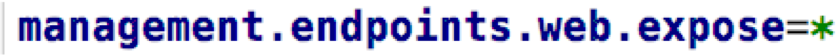
Spring Boot Actuator

- Endpoints can be customized using the *application.properties*/*yml*, you can change if an endpoint is enabled, with a particular path, accessible through JMX or/and Web and how long it will be cached.

syntax:

  - *management.endpoint.[endpoint-name].enabled*
  - *management.endpoint.[endpoint-name].cache.time-to-live*

- You can change the defaults (enable all endpoints) using the:

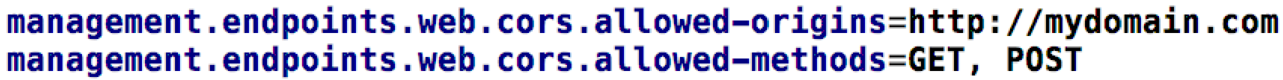


```
management.endpoints.web.expose=*
```

- By default the *spring-boot-actuator* uses the role ACTUATOR to get access to the endpoints if secured.

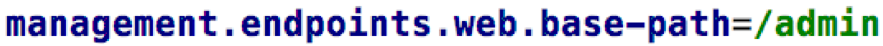
Spring Boot Actuator

- ***Actuator*** has *CORS* support, endpoints can be configure what kind of cross domain request are authorized



```
management.endpoints.web.cors.allowed-origins=http://mydomain.com
management.endpoints.web.cors.allowed-methods=GET, POST
```

- You can change the default base path (***/actuator***) with:

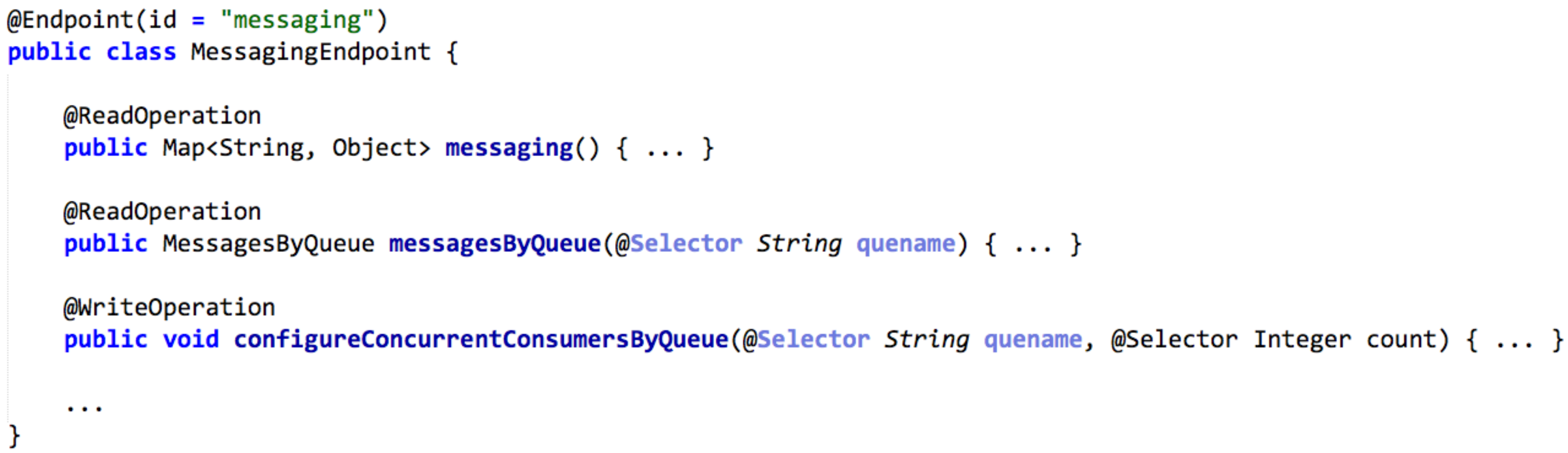


```
management.endpoints.web.base-path=/admin
```

- ***Actuator*** brings an easy way to implement custom endpoints with ***@Endpoint***, ***@ReadOperation***, ***@WriteOperation*** and ***@Selector*** annotations that can be enabled and used for web (Spring MVC and Jersey) and/or JMX with the same code.

Spring Boot Actuator

- The following actuator endpoint will be expose:



```
@Endpoint(id = "messaging")
public class MessagingEndpoint {

    @ReadOperation
    public Map<String, Object> messaging() { ... }

    @ReadOperation
    public MessagesByQueue messagesByQueue(@Selector String quename) { ... }

    @WriteOperation
    public void configureConcurrentConsumersByQueue(@Selector String quename, @Selector Integer count) { ... }

    ...
}
```

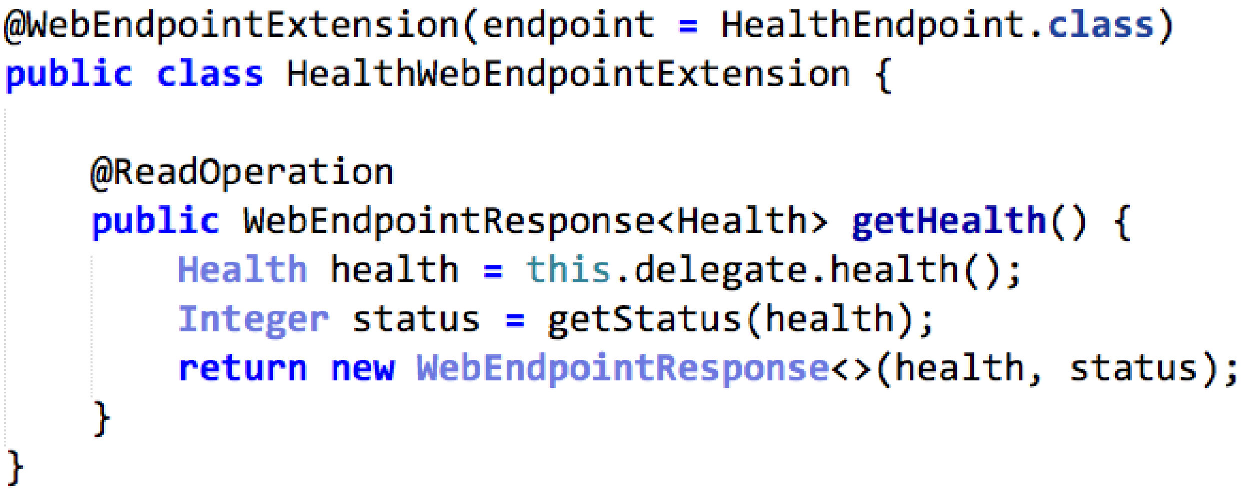
by default as web: */actuator/messaging*

and JMX object name:

org.springframework.boot:type=Endpoint,name=Messaging

Spring Boot Actuator

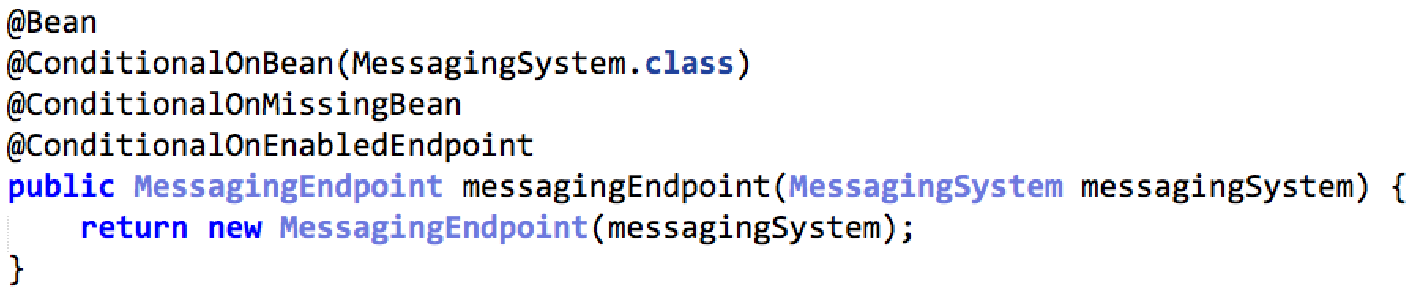
- ***Actuator*** brings **extensions** to override endpoints operation for a given technology by using the *WebEndpointResponse<Health>* as response instead of **Health**:



```
@WebEndpointExtension(endpoint = HealthEndpoint.class)
public class HealthWebEndpointExtension {

    @ReadOperation
    public WebEndpointResponse<Health> getHealth() {
        Health health = this.delegate.health();
        Integer status = getStatus(health);
        return new WebEndpointResponse<>(health, status);
    }
}
```

Spring Boot Actuator

- With a custom endpoint, now is necessary to configure it with the *@ConditionalOnEnabledEndpoint* that makes sure that the endpoint is not created (or exposed) according to the current configuration:



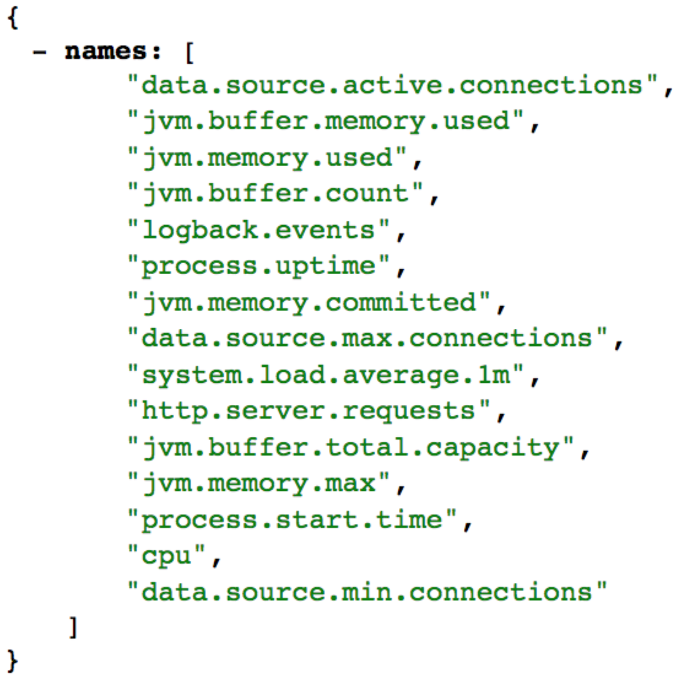
```
@Bean
@ConditionalOnBean(MessagingSystem.class)
@ConditionalOnMissingBean
@ConditionalOnEnabledEndpoint
public MessagingEndpoint messagingEndpoint(MessagingSystem messagingSystem) {
    return new MessagingEndpoint(messagingSystem);
}
```

Agenda

- Spring Boot Actuator
- Metrics
- Health Indicators

Spring Boot Actuator: metrics

Actuator includes a *metrics* endpoint that exposes system metrics:

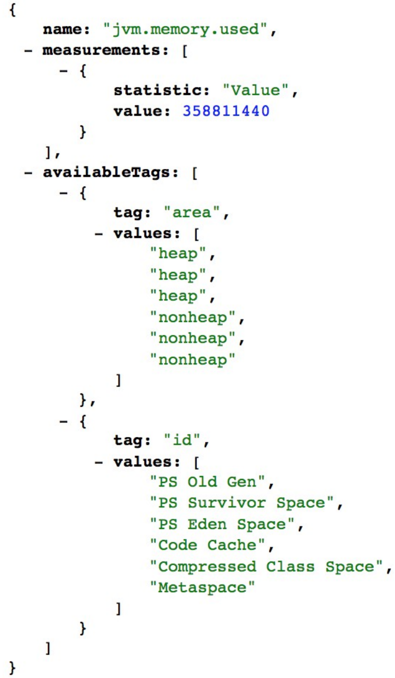


```
{
  - names: [
        "data.source.active.connections",
        "jvm.buffer.memory.used",
        "jvm.memory.used",
        "jvm.buffer.count",
        "logback.events",
        "process.uptime",
        "jvm.memory.committed",
        "data.source.max.connections",
        "system.load.average.1m",
        "http.server.requests",
        "jvm.buffer.total.capacity",
        "jvm.memory.max",
        "process.start.time",
        "cpu",
        "data.source.min.connections"
    ]
}
```

You can query this metrics

using the name and selector tags:

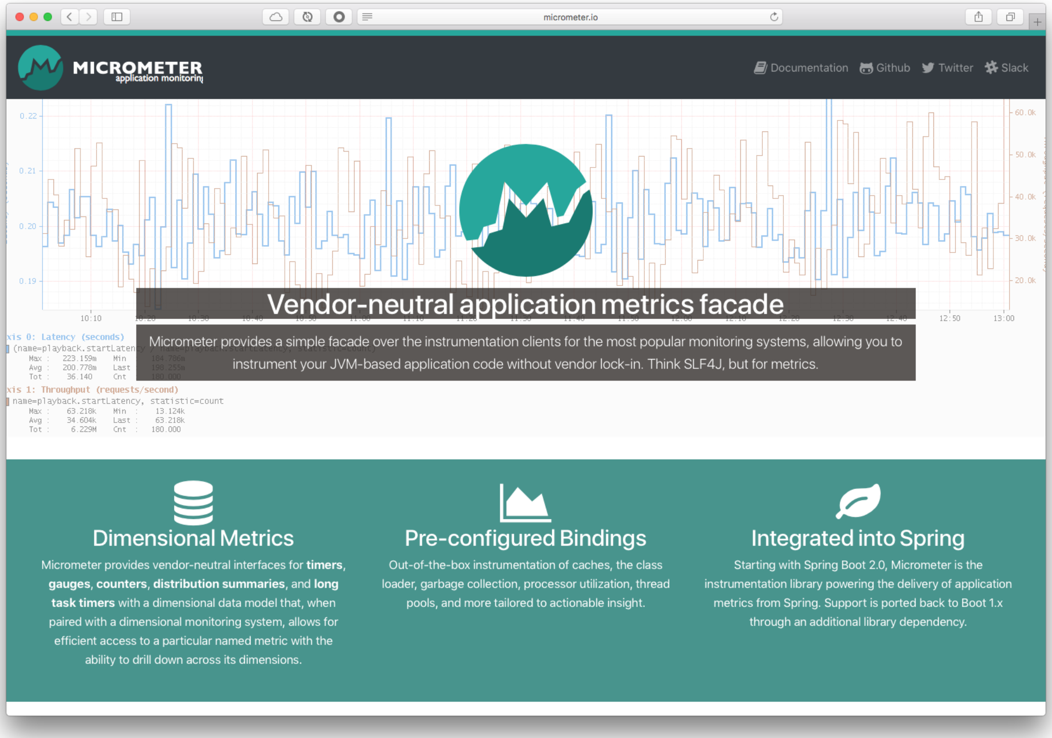
metrics/jvm.memory.used?tag=heap



```
{
    name: "jvm.memory.used",
  - measurements: [
      - {
            statistic: "Value",
            value: 358811440
        }
    ],
  - availableTags: [
      - {
            tag: "area",
          - values: [
                "heap",
                "heap",
                "heap",
                "nonheap",
                "nonheap",
                "nonheap"
            ]
        },
      - {
            tag: "id",
          - values: [
                "PS Old Gen",
                "PS Survivor Space",
                "PS Eden Space",
                "Code Cache",
                "Compressed Class Space",
                "Metaspace"
            ]
        }
    ]
}
```

Spring Boot Actuator: metrics

- ***Actuator* metrics** support ***Micrometer*** for dimensional and hierarchical metrics.
- ***Micrometer*** (http://micrometer.io/) provides a simple facade over the instrumentation clients for the most popular monitoring systems, allowing you to instrument your JVM-based application code without vendor lock-in.



Vendor-neutral application metrics facade

Micrometer provides a simple facade over the instrumentation clients for the most popular monitoring systems, allowing you to instrument your JVM-based application code without vendor lock-in. Think SLF4J, but for metrics.

```
xis 0: Latency (seconds)
 (name=playback.startLatency
    Max :   223.159m    Min
    Avg :   200.778m    Last
    Tot :    36.140     Cnt
xis 1: Throughput (requests/second)
 name=playback.startLatency, statistic=count
    Max :    63.218k    Min  :    13.124k
    Avg :    34.604k    Last :    63.218k
    Tot :     6.229M    Cnt  :   180.000
```

Dimensional Metrics

Micrometer provides vendor-neutral interfaces for **timers**, **gauges**, **counters**, **distribution summaries**, and **long task timers** with a dimensional data model that, when paired with a dimensional monitoring system, allows for efficient access to a particular named metric with the ability to drill down across its dimensions.

Pre-configured Bindings

Out-of-the-box instrumentation of caches, the class loader, garbage collection, processor utilization, thread pools, and more tailored to actionable insight.

Integrated into Spring

Starting with Spring Boot 2.0, Micrometer is the instrumentation library powering the delivery of application metrics from Spring. Support is ported back to Boot 1.x through an additional library dependency.

Pivotal™

Spring Boot Actuator: metrics

- Using **Micrometer**, Spring Boot auto-configures a composite meter registry and adds a registry to the composite for each of the supported implementations that it finds on the classpath.

- **Micrometer** support several monitoring systems:
  - (**Dimensional**) **Atlas**, **Prometheus**, **Datadog**, **Influx**, **StatsD**, **Telegraf**
  - (**Hierarchical**) **Graphite**, **Ganglia**, **JMX**, **Etsy StatsD**

- **Micrometer** provide a set of *Meter* (registry) primitives: *Timer*, *Counter*, *Gauge*, *DistributionSummary* and *LongTaskTimer*.

Agenda

- Spring Boot Actuator
- Metrics
- Health Indicators

Spring Boot Actuator: health indicators

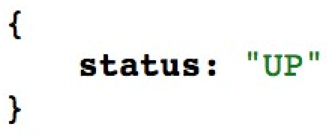
- Health information can be used to check the status of your running application

- ***Actuator*** provides the */actuator/health* endpoint that shows the health (***status***) or the details of every component of your application

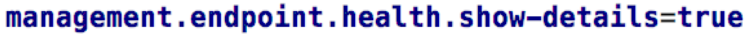
- ***Actuator*** include a number of auto-configured ***health indicators*** and provides you an easy way to create a custom one

Spring Boot Actuator: health indicators

/actuator/health

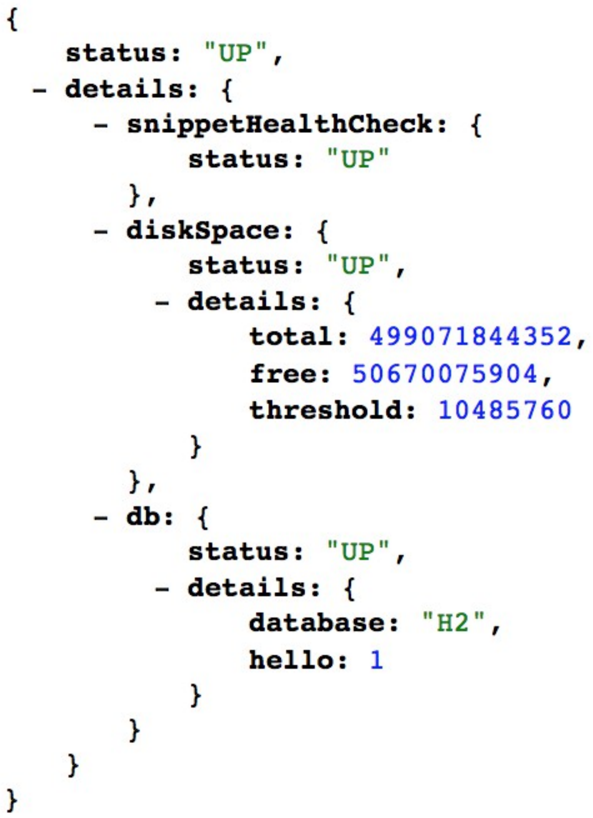


```
{
    status: "UP"
}
```



```
management.endpoint.health.show-details=true
```

/actuator/health



```
{
     status: "UP",
   - details: {
       - snippetHealthCheck: {
              status: "UP"
         },
       - diskSpace: {
              status: "UP",
            - details: {
                  total: 499071844352,
                  free: 50670075904,
                  threshold: 10485760
              }
         },
       - db: {
              status: "UP",
            - details: {
                  database: "H2",
                  hello: 1
              }
         }
     }
}
```

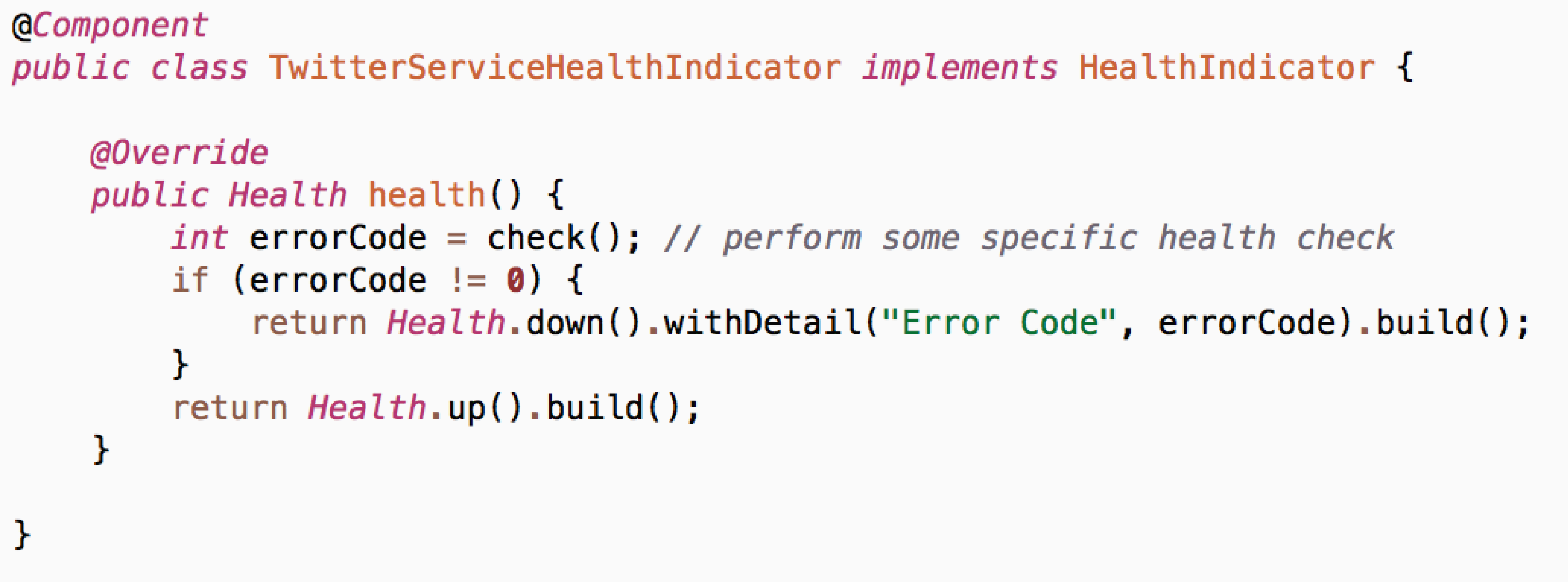
Spring Boot Actuator: health indicators

out-of-the-box health indicators:

- *CassandraHealthIndicator*
- *DiskSpaceHealthIndicator*
- *DataSourceHealthIndicator*
- *ElasticsearchHealthIndicator*
- *JmsHealthIndicator*
- *MailHealthIndicator*
- *MongoHealthIndicator*
- *RabbitHealthIndicator*
- *RedisHealthIndicator*
- *SolrHealthIndicator*

Spring Boot Actuator: health indicators

- ***Actuator*** provides the *HealthIndicator* interface and the *AbstractHealthIndicator* class to create a custom health indicator



```
@Component
public class TwitterServiceHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
        int errorCode = check(); // perform some specific health check
        if (errorCode != 0) {
            return Health.down().withDetail("Error Code", errorCode).build();
        }
        return Health.up().build();
    }

}
```

Demo

Actuator

Lab

Custom HealthIndicator

Summary

- Spring Boot Actuator
- Actuator exposes built-in endpoints
- Actuator can be used with Micrometer
- Built-in health indicators
- Allows creation of custom health indicators

Pivotal

A NEW PLATFORM FOR A NEW ERA

Security with Spring Boot

Spring Boot Developer

Securing Web Applications

Agenda

- Security with Spring Boot

Security with Spring Boot

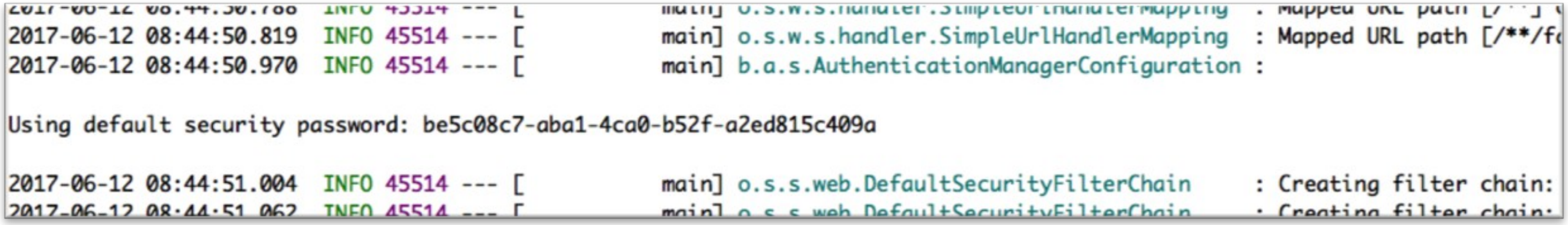
- Spring Boot uses the *spring-security* project to simplify the protection of applications

- To create secured spring boot applications it is necessary to add the *spring-boot-starter-security* dependency

- Spring Boot will **auto-configure** basic security by default

Security with Spring Boot

- Spring Boot will **auto-configure** a basic security by default and print out a *default security password* on application startup

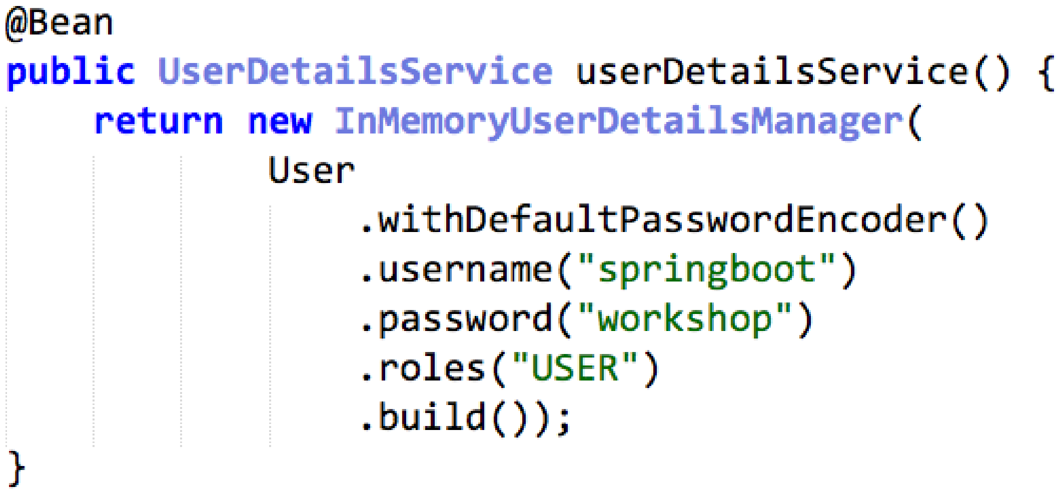


```
2017-06-12 08:44:50.788  INFO 45514 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] o
2017-06-12 08:44:50.819  INFO 45514 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/fa
2017-06-12 08:44:50.970  INFO 45514 --- [           main] b.a.s.AuthenticationManagerConfiguration :

Using default security password: be5c08c7-aba1-4ca0-b52f-a2ed815c409a

2017-06-12 08:44:51.004  INFO 45514 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain:
2017-06-12 08:44:51.062  INFO 45514 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain:
```

- Spring Security provides a more secure defaults and the ability to migrate how passwords are stored. The default *PasswordEncoder* is a *DelegatingPasswordEncoder* which encode passwords using **BCrypt** by default.

Security with Spring Boot

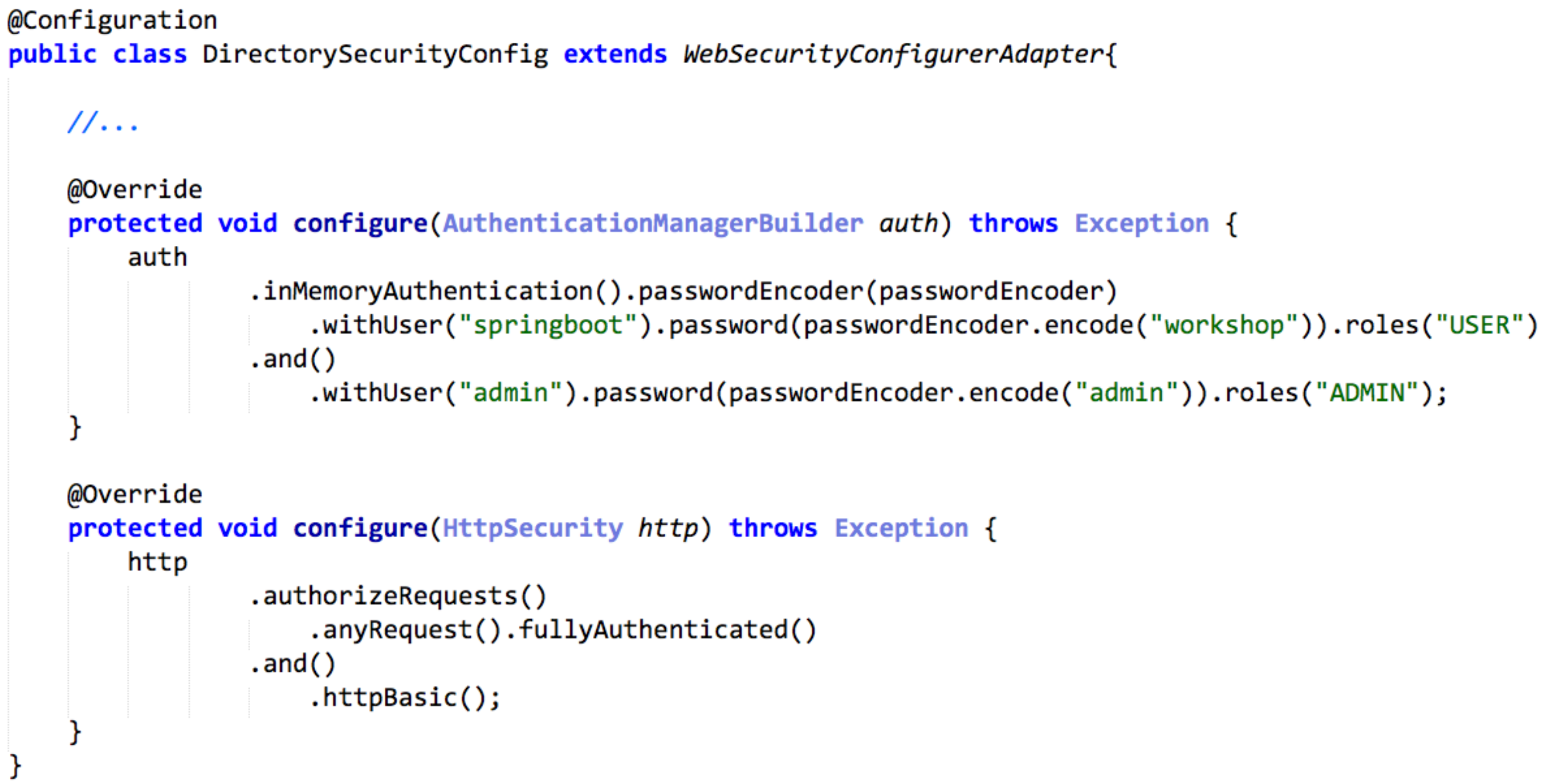
You can change the default **user** and the **generated password** by providing a *UserDetailsService* and returning a *InMemoryUserDetailsManager* instance.



```
@Bean
public UserDetailsService userDetailsService() {
    return new InMemoryUserDetailsManager(
            User
                .withDefaultPasswordEncoder()
                .username("springboot")
                .password("workshop")
                .roles("USER")
                .build());
}
```

Security with Spring Boot

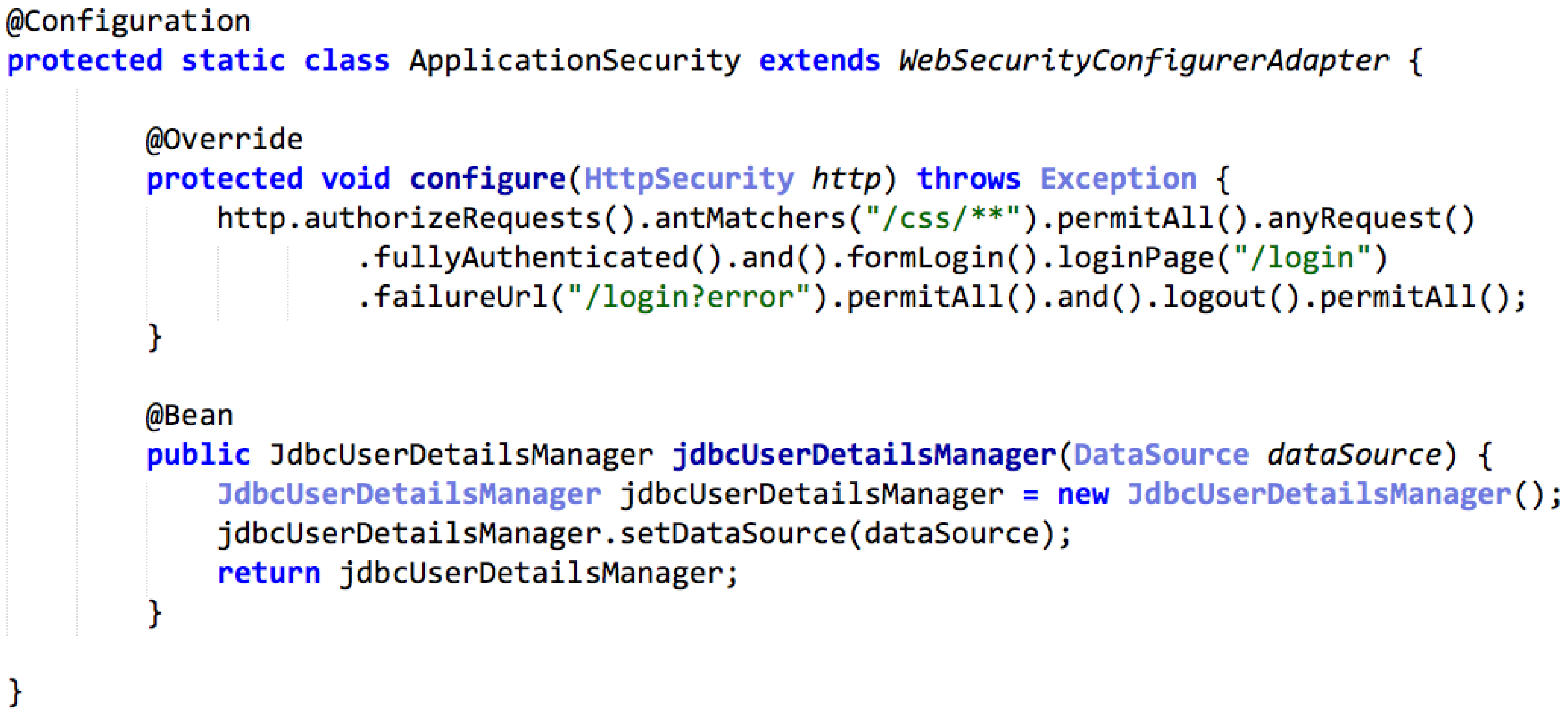
Spring Boot allows you to configure security programmatically by extending the *WebSecurityConfigurerAdapter* and controlling access



```
@Configuration
public class DirectorySecurityConfig extends WebSecurityConfigurerAdapter{

    //...

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
                .inMemoryAuthentication().passwordEncoder(passwordEncoder)
                    .withUser("springboot").password(passwordEncoder.encode("workshop")).roles("USER")
                .and()
                    .withUser("admin").password(passwordEncoder.encode("admin")).roles("ADMIN");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                    .anyRequest().fullyAuthenticated()
                .and()
                    .httpBasic();
    }
}
```

Security with Spring Boot

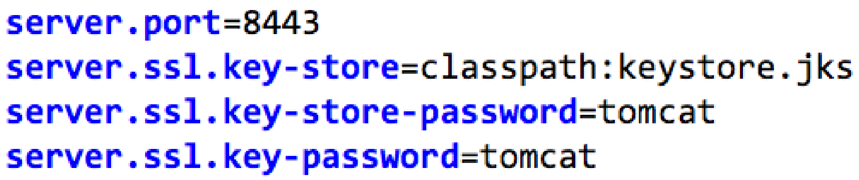
JDBC:



```
@Configuration
protected static class ApplicationSecurity extends WebSecurityConfigurerAdapter {

        @Override
        protected void configure(HttpSecurity http) throws Exception {
            http.authorizeRequests().antMatchers("/css/**").permitAll().anyRequest()
                    .fullyAuthenticated().and().formLogin().loginPage("/login")
                    .failureUrl("/login?error").permitAll().and().logout().permitAll();
        }

        @Bean
        public JdbcUserDetailsManager jdbcUserDetailsManager(DataSource dataSource) {
            JdbcUserDetailsManager jdbcUserDetailsManager = new JdbcUserDetailsManager();
            jdbcUserDetailsManager.setDataSource(dataSource);
            return jdbcUserDetailsManager;
        }

}
```

Security with Spring Boot

- SSL can be configured with the *server.ssl.** properties



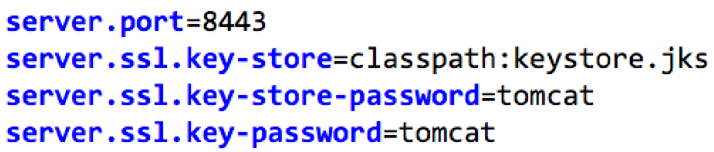
```
server.port=8443
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-password=tomcat
server.ssl.key-password=tomcat
```

- Spring Boot Actuator requires an “***ACTUATOR***” role

- You can use a custom persistence mechanism to hold user information for authentication and authorization by implementing *UserDetailsService*

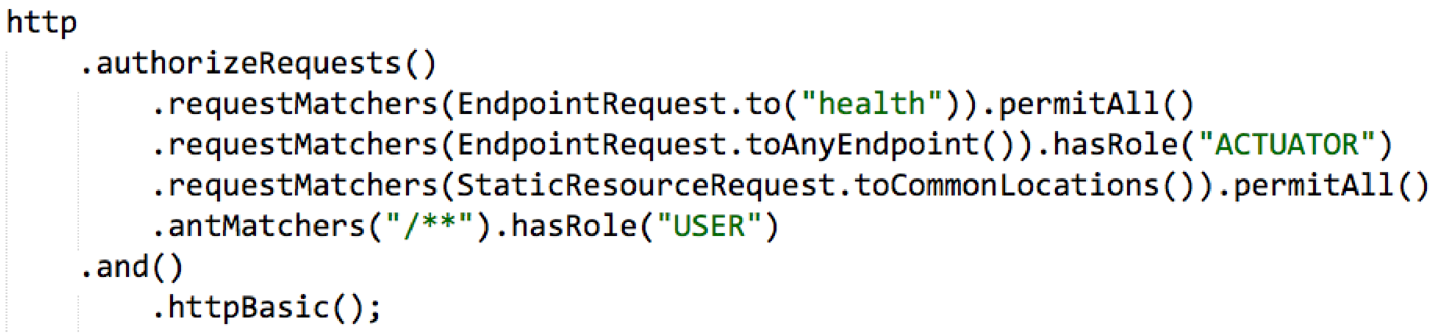
Security with Spring Boot

- SSL can be configured with the *server.ssl.** properties



```
server.port=8443
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-password=tomcat
server.ssl.key-password=tomcat
```

- Spring Boot Actuator requires an “***ACTUATOR***” role
- Spring Security provides several utility classes that can be use with the request matchers: *EndpointRequest*, *StaticResourceRequest*



```
http
    .authorizeRequests()
        .requestMatchers(EndpointRequest.to("health")).permitAll()
        .requestMatchers(EndpointRequest.toAnyEndpoint()).hasRole("ACTUATOR")
        .requestMatchers(StaticResourceRequest.toCommonLocations()).permitAll()
        .antMatchers("/**").hasRole("USER")
    .and()
        .httpBasic();
```

Demo

Security a Web App

Lab

Jdbc Security

Summary

- Security with Spring Boot
- Spring boot uses spring-security project for securing applications
- Include spring-boot-starter-security for basic security
- Highly customizable: in-memory, jdbc, ldap, custom

Pivotal

A NEW PLATFORM FOR A NEW ERA

Messaging with Spring Boot

Spring Boot Developer

RabbitMQ

Agenda

- Spring Messaging
- Spring Boot Messaging with RabbitMQ
  - Quick overview
  - Exchanges, Bindings and Queues
  - Sending messages
  - Consuming messages

Spring Messaging

- The Spring Framework provides extensive support for integrating with messaging systems: from simplified use of the **JMS API** using *JmsTemplate* to a complete infrastructure to receive messages asynchronously

- The *spring-amqp* project provides a similar feature set for the '***A**dvanced **M**essage **Q**ueuing **P**rotocol*' providing a *RabbitTemplate* class for sending and receiving message plus some useful annotations

- There is also support for STOMP messaging natively in spring, WebSockets and Kafka

Agenda

- Spring Messaging
- Spring Boot Messaging with RabbitMQ
  - Quick overview
  - Exchanges, Bindings and Queues
  - Sending messages
  - Consuming messages

Spring Boot Messaging with RabbitMQ

- *spring-amqp* provides the *@EnableRabbit* that scans for annotations like *@RabbitListener* and *@SendTo*, for listening and reply

- Spring Boot uses the power of **Spring Messaging** by adding several auto-configuration options for *RabbitTemplate* and defaults for *ConnectionFactory* classes

- Spring Boot defaults can be controlled by external configuration properties in *spring.rabbitmq.**

Spring Boot Messaging with RabbitMQ

- Spring Boot messaging applications with RabbitMQ can be created by adding the *spring-boot-starter-amqp* dependency

- If the *spring-boot-actuator* is in the classpath, the ***RabbitMQHealthIndicator*** is auto-configured.

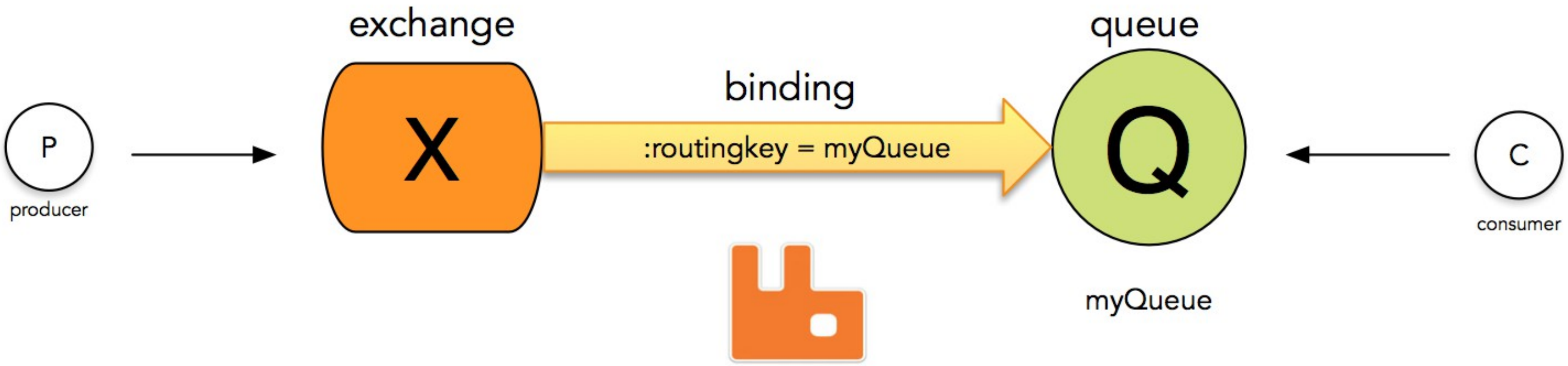
RabbitMQ: overview

- Rabbitmq is an amqp message broker
- Platform agnostic, broadly applicable for enterprise, totally open source
- Implemented with *erlang*
- Distributed: cluster ready, reliability/scalability out of the box
- High Availability: mirror queues, data/state replication with full ACID, routing capabilities
- Multiple protocol support: *amqp*, *mqtt*, *stomp*, *smtp*, *xmpp*
- Security: TLS, LDAP
- Plugin Based: federation, shovel, consistent hash, sharding, ...
- Multiple client libraries: java, .net, ruby, erlang, python, php, ...

RabbitMQ: overview

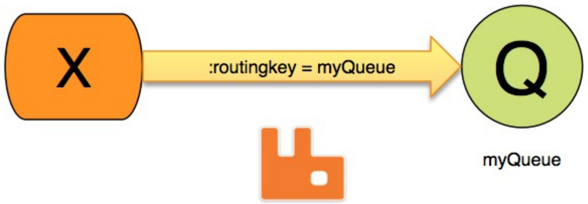
Google
NOKIA
hp
amazon
QUALCOMM
EA
LG
NVIDIA
VISA
CISCO
i'm lovin' it

RabbitMQ: exchanges, bindings, queues



RabbitMQ: exchanges types

Default Exchange



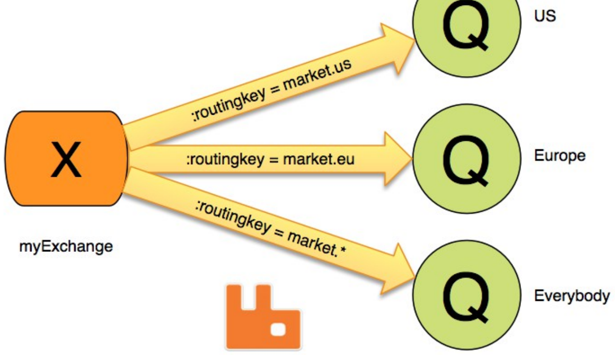
X

:routingkey = myQueue

Q

myQueue

Topic Exchange



X

myExchange

:routingkey = market.us

:routingkey = market.eu

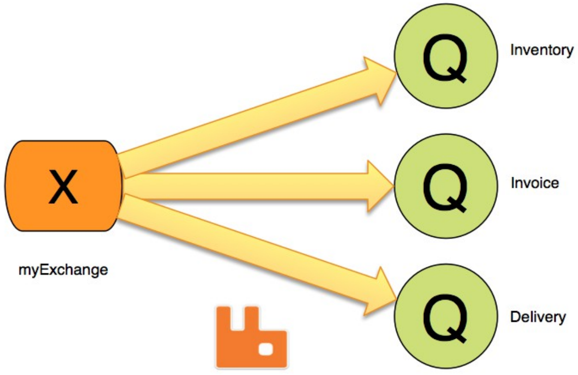
:routingkey = market.*

Q US

Q Europe

Q Everybody

Fanout Exchange



X

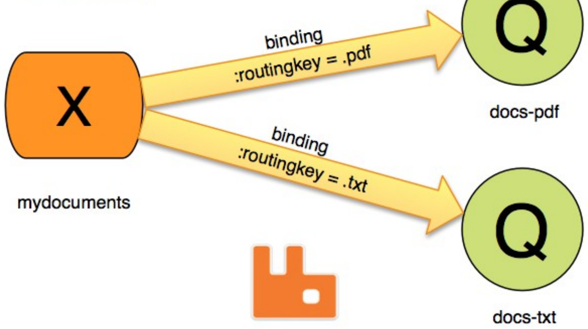
myExchange

Q Inventory

Q Invoice

Q Delivery

Direct Exchange



X

mydocuments

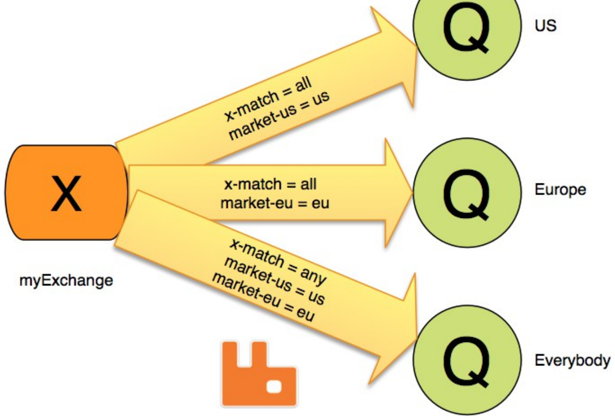
binding
:routingkey = .pdf

binding
:routingkey = .txt

Q docs-pdf

Q docs-txt

Headers Exchange



X

myExchange

x-match = all
market-us = us

x-match = all
market-eu = eu

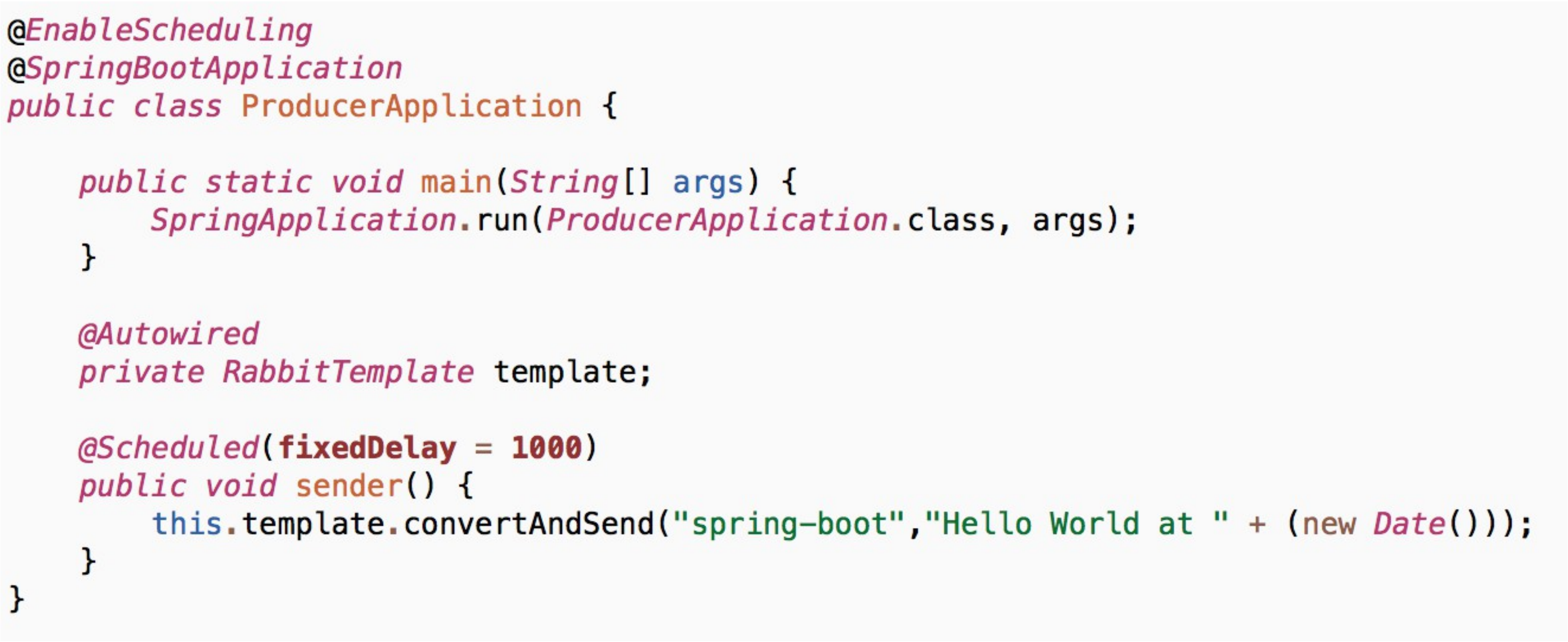
x-match = any
market-us = us
market-eu = eu

Q US

Q Europe

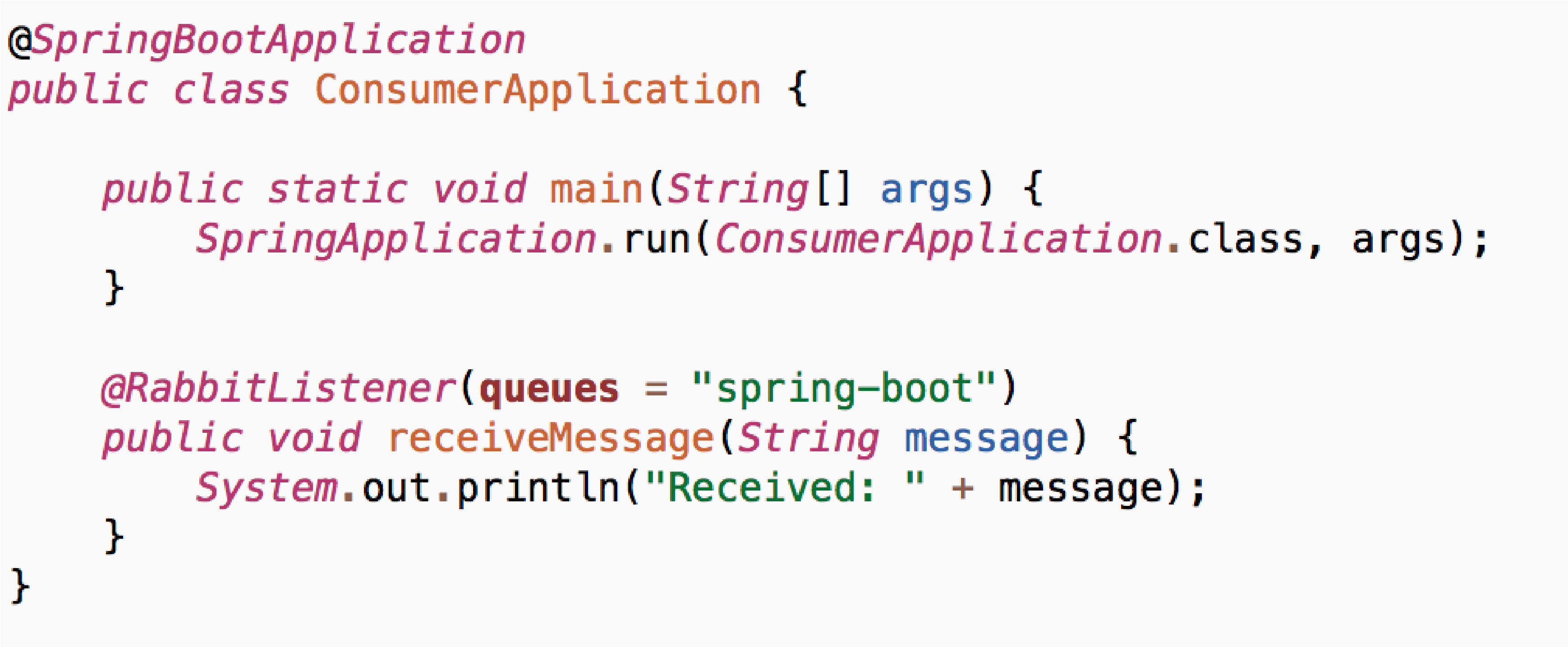
Q Everybody

RabbitMQ: sending messages



```
@EnableScheduling
@SpringBootApplication
public class ProducerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ProducerApplication.class, args);
    }

    @Autowired
    private RabbitTemplate template;

    @Scheduled(fixedDelay = 1000)
    public void sender() {
        this.template.convertAndSend("spring-boot","Hello World at " + (new Date()));
    }
}
```

RabbitMQ: consuming messages



```java
@SpringBootApplication
public class ConsumerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConsumerApplication.class, args);
    }

    @RabbitListener(queues = "spring-boot")
    public void receiveMessage(String message) {
        System.out.println("Received: " + message);
    }
}
```

Demo

RabbitMQ

Lab

Messaging using RabbitMQ

Summary

- Spring Boot simplifies messaging by providing multiple auto-configuration options for jms, amqp, websockets (stomp) and kafka

Pivotal

A NEW PLATFORM FOR A NEW ERA

Reactor and WebFlux with Spring Boot

Spring Boot Developer

Reactive Programming

Agenda

- Reactive Programming
- Reactor
- Spring WebFlux
- Spring Boot with Reactor and WebFlux

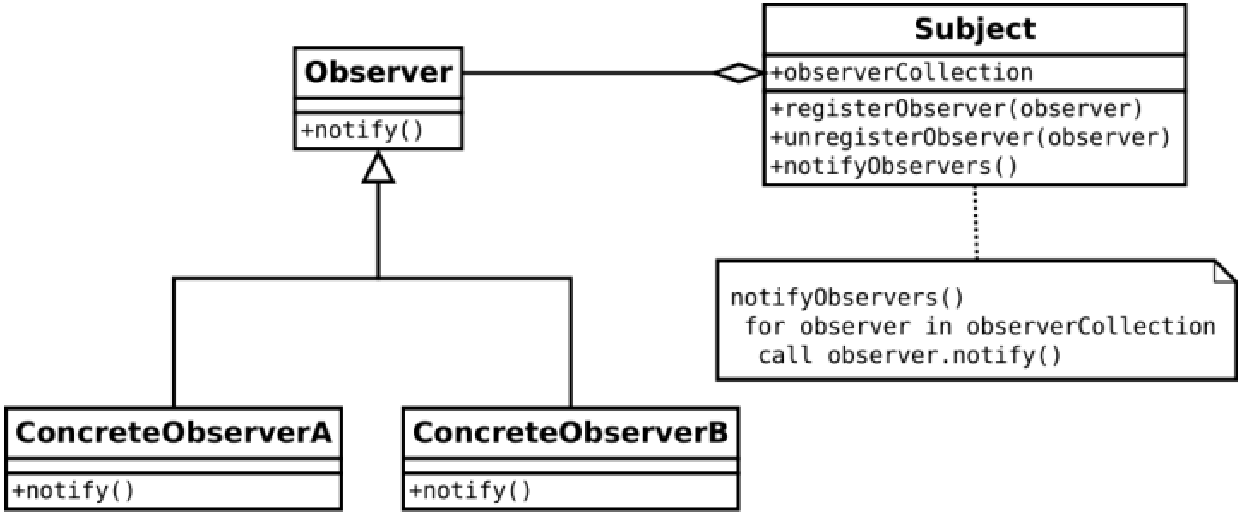
Reactive Programming

- ***Reactive Programming*** *is an* ***asynchronous*** *paradigm concerned with data streams and the propagation of change*

- where can we use ***Reactive Programming***?
  - Spreadsheets/cells (event-driven architectures)
  - High concurrent messaging (synchronously / asynchronously)
  - External service calls
  - Async processing

Reactive Programming

- ***Reactive Programming*** paradigm is often presented in object-oriented languages as an extension of the ***Observer Design Pattern***



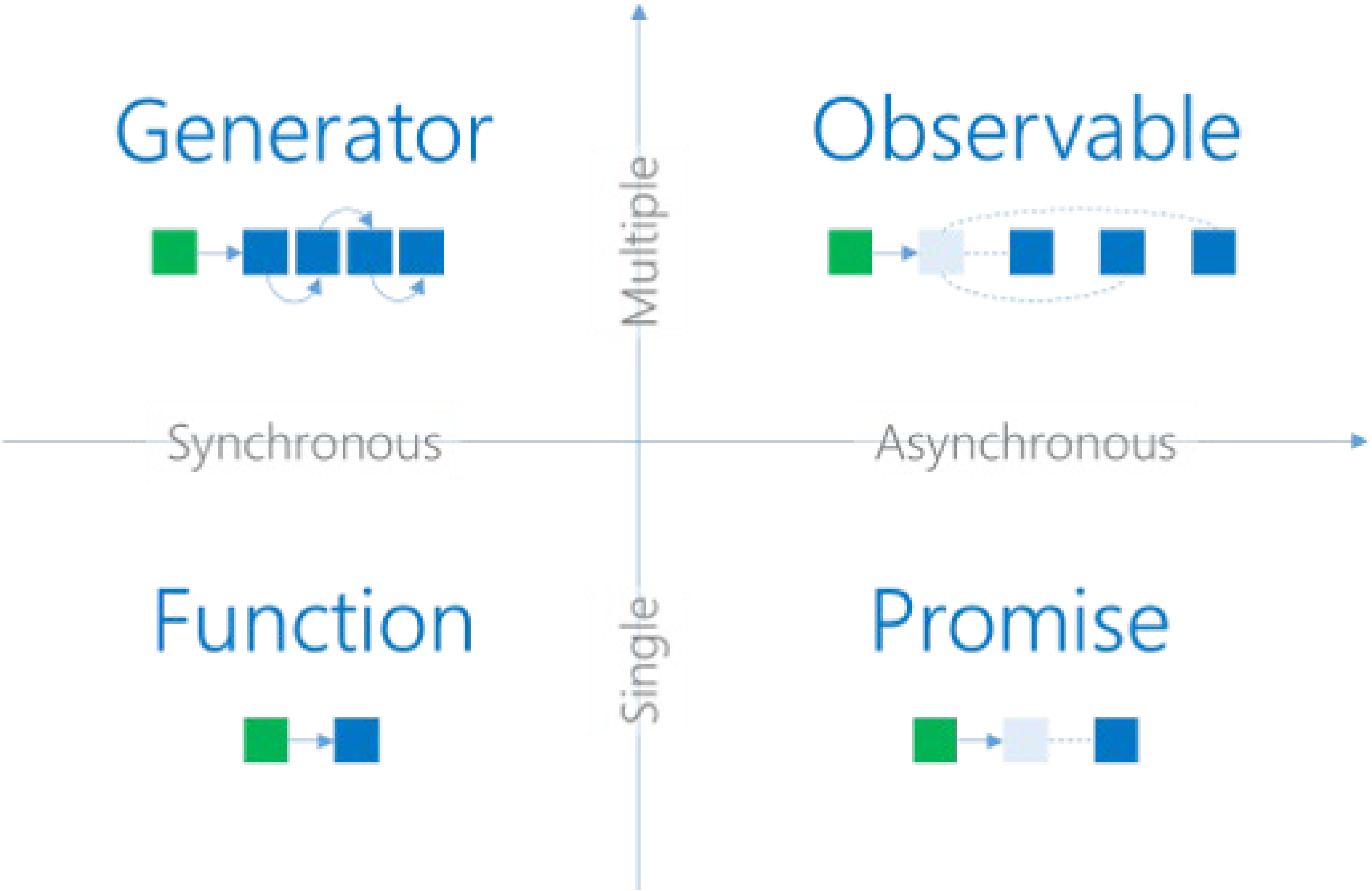
Pivotal

Reactive Programming

- There are solutions (libraries) for ***non-blocking I/O*** like:
  - ruby: event-machine
  - java: Future/CompletableFuture (java.util.concurrent), Observable
  - big data: map-reduce / fork-join
  - akka: actor models

- ***Reactive Programming*** is the next step in creating a system that are *responsive, resilient, elastic* and *message-driven* in a ***asynchronous*** way:
  - ***flow control***
  - ***back-pressure***

Reactive Programming



Agenda

- Reactive Programming
- Reactor
- Spring WebFlux
- Spring Boot with Reactor and WebFlux

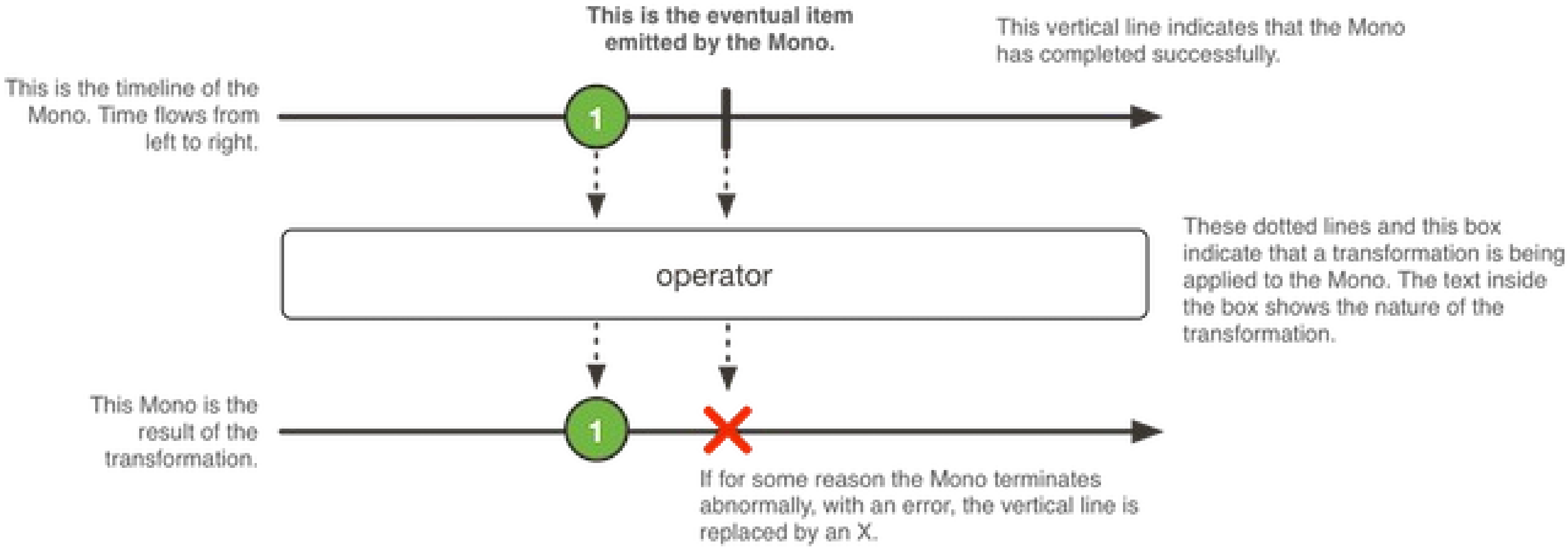
Reactor

- ***Reactor*** is an implementation of the ***Reactive Programming paradigm***

- ***Reactor*** offers ***non-blocking*** and ***backpressure-ready*** embeddable solutions including local and remote unicast/multicast messaging or TCP/HTTP/UDP client and servers.

- ***Reactor*** offers 2 reactive composable API **Flux** [N] and **Mono** [0|1] extensively implementing ***Reactive Extensions***.

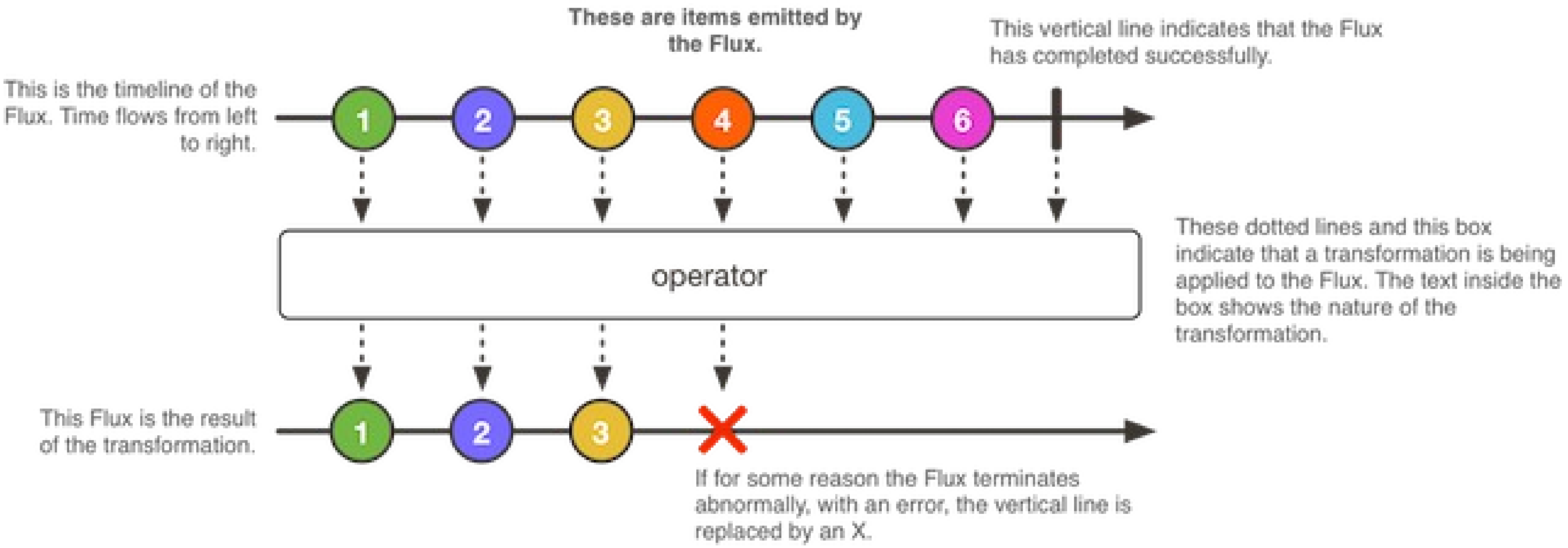
Reactor



This is the eventual item emitted by the Mono.
This vertical line indicates that the Mono has completed successfully.
This is the timeline of the Mono. Time flows from left to right.
1
operator
These dotted lines and this box indicate that a transformation is being applied to the Mono. The text inside the box shows the nature of the transformation.
This Mono is the result of the transformation.
1
If for some reason the Mono terminates abnormally, with an error, the vertical line is replaced by an X.

MONO

Reactor



These are items emitted by the Flux.

This vertical line indicates that the Flux has completed successfully.

This is the timeline of the Flux. Time flows from left to right.

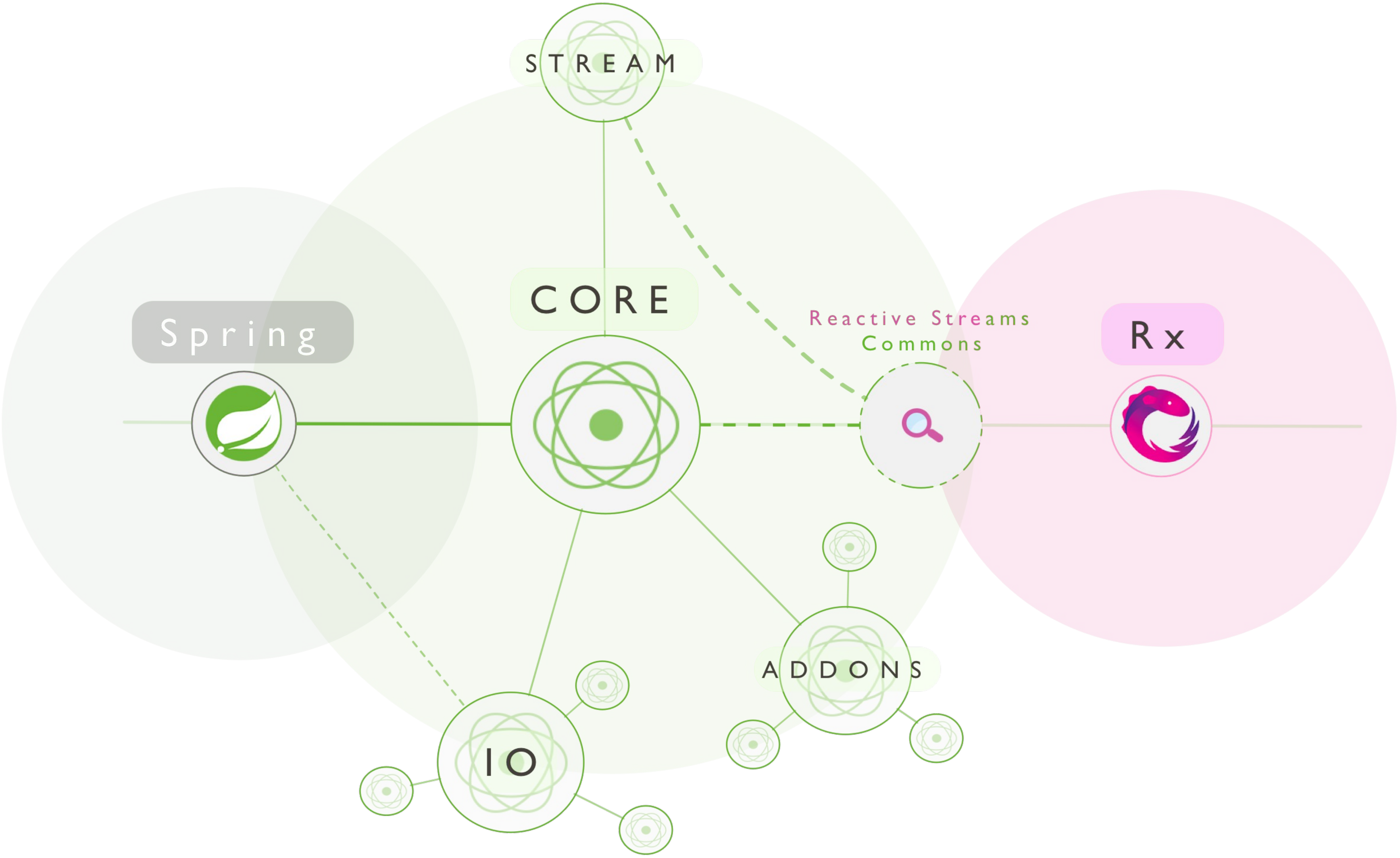
These dotted lines and this box indicate that a transformation is being applied to the Flux. The text inside the box shows the nature of the transformation.

This Flux is the result of the transformation.

If for some reason the Flux terminates abnormally, with an error, the vertical line is replaced by an X.

FLUX

Reactor



Agenda

- Reactive Programming
- Reactor
- Spring WebFlux
- Spring Boot with Reactor and WebFlux

Spring WebFlux

- Spring Framework 5 embraces ***Reactive Streams*** as the contract for communicating backpressure across async components and libraries

- Spring Framework 5 includes a new ***spring-webflux*** module.

- The module contains support for ***Reactive HTTP*** and ***WebSocket clients*** as well as for ***Reactive Server*** web applications including REST, HTML browser, and WebSocket style interactions

- Exposes the ***Reactor*** types: **Flux** [N] and **Mono** [0|1].

Spring WebFlux

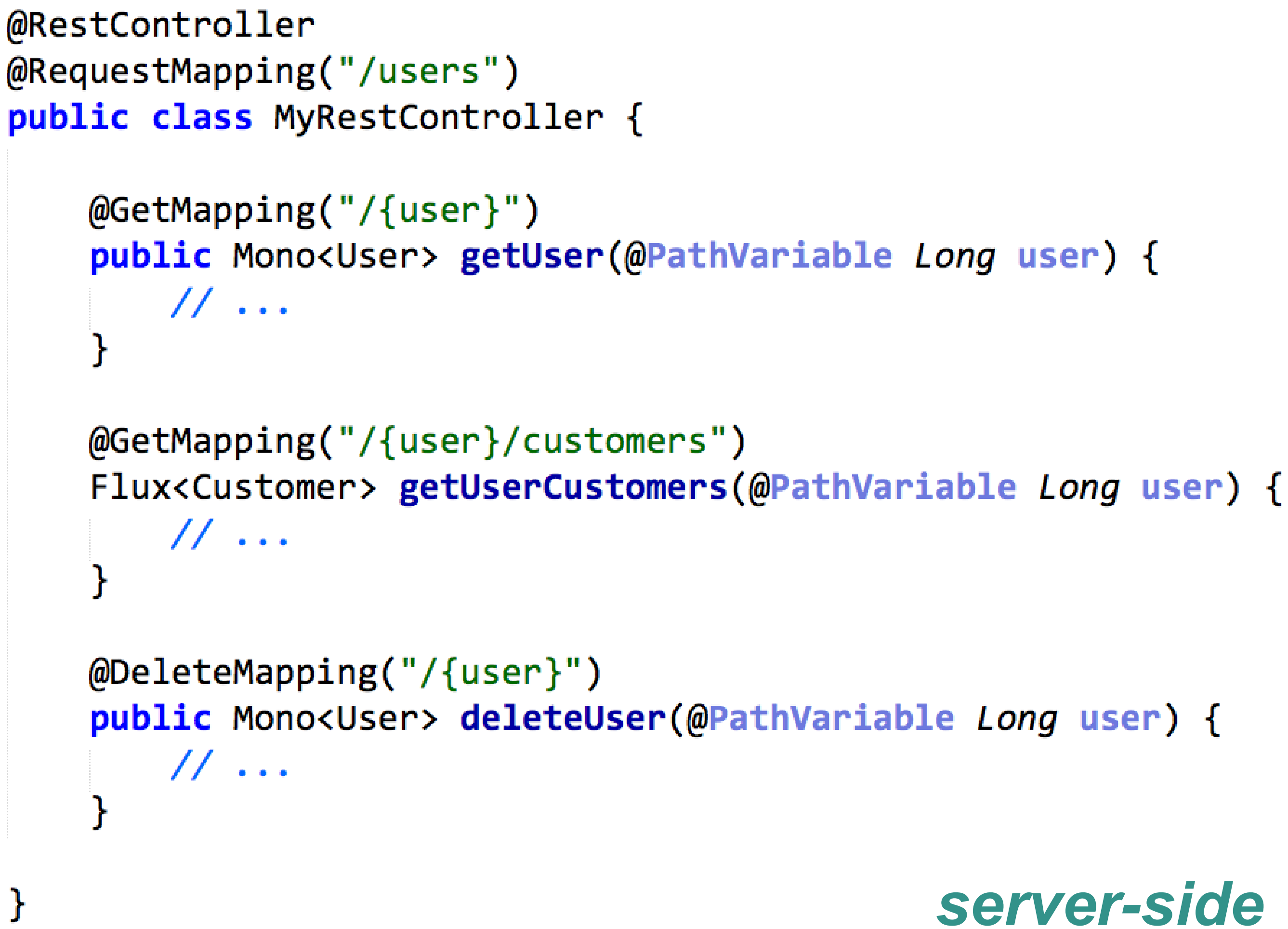
- On the ***server-side*** WebFlux supports 2 distinct programming models:
  - annotation-based (@Controller)
  - functional (Java 8 lambda style routing and handling)

- On the ***client-side*** WebFlux includes a functional, reactive ***WebClient*** that offers a fully non-blocking and reactive alternative to the ***RestTemplate***

- Support for ***reactive WebSockets*** and testing (with ***WebTestClient***)

Spring WebFlux

annotation-based

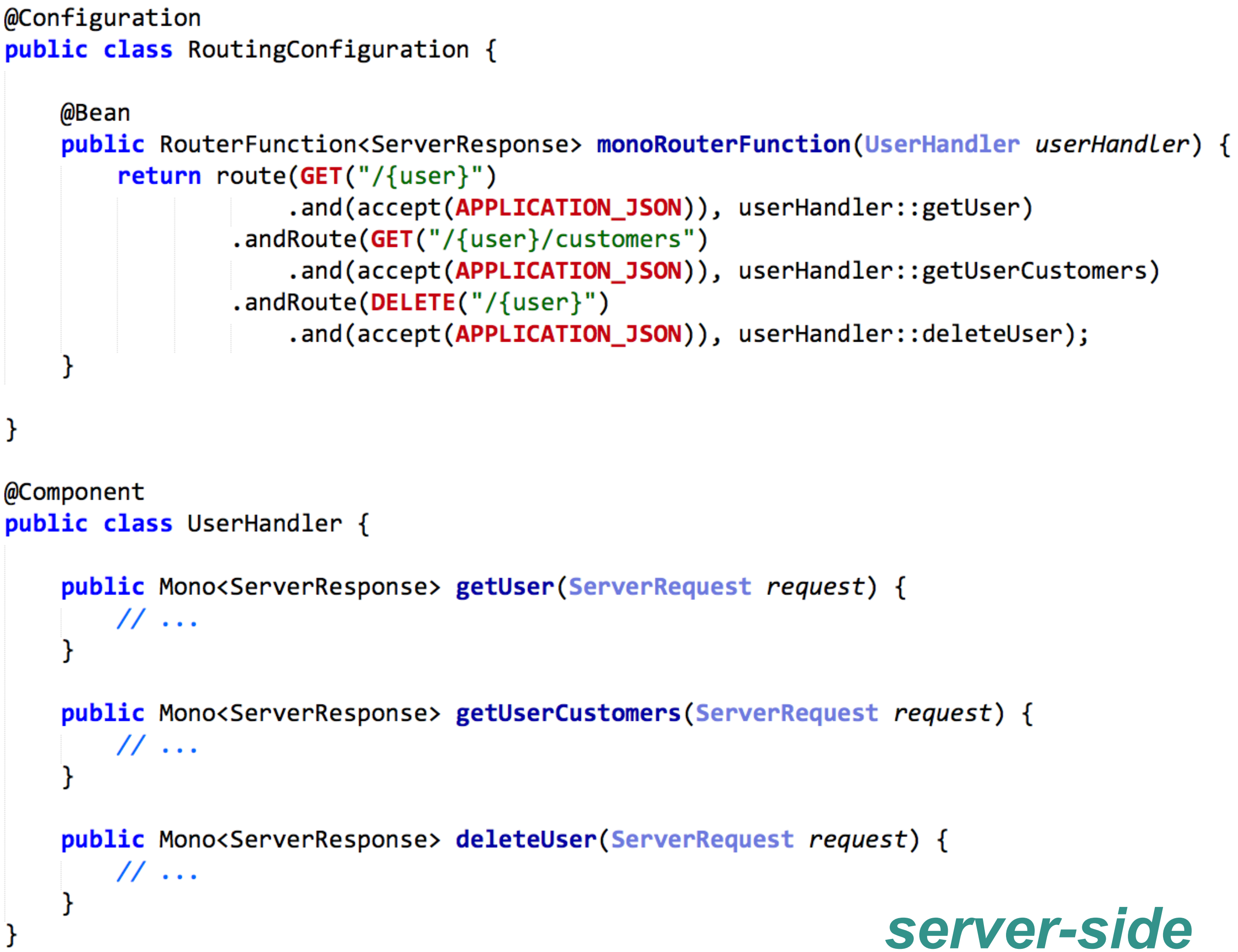


```
@RestController
@RequestMapping("/users")
public class MyRestController {

    @GetMapping("/{user}")
    public Mono<User> getUser(@PathVariable Long user) {
        // ...
    }

    @GetMapping("/{user}/customers")
    Flux<Customer> getUserCustomers(@PathVariable Long user) {
        // ...
    }

    @DeleteMapping("/{user}")
    public Mono<User> deleteUser(@PathVariable Long user) {
        // ...
    }

}
```

server-side

Spring WebFlux

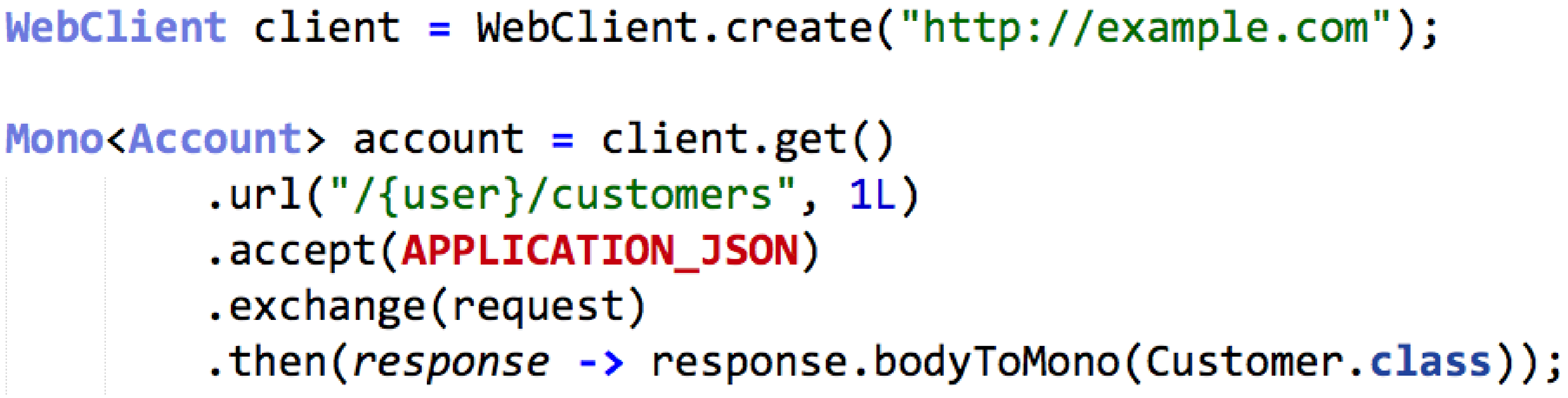
functional



```
@Configuration
public class RoutingConfiguration {

    @Bean
    public RouterFunction<ServerResponse> monoRouterFunction(UserHandler userHandler) {
        return route(GET("/{user}")
                    .and(accept(APPLICATION_JSON)), userHandler::getUser)
                .andRoute(GET("/{user}/customers")
                    .and(accept(APPLICATION_JSON)), userHandler::getUserCustomers)
                .andRoute(DELETE("/{user}")
                    .and(accept(APPLICATION_JSON)), userHandler::deleteUser);
    }

}

@Component
public class UserHandler {

    public Mono<ServerResponse> getUser(ServerRequest request) {
        // ...
    }

    public Mono<ServerResponse> getUserCustomers(ServerRequest request) {
        // ...
    }

    public Mono<ServerResponse> deleteUser(ServerRequest request) {
        // ...
    }
}
```

server-side

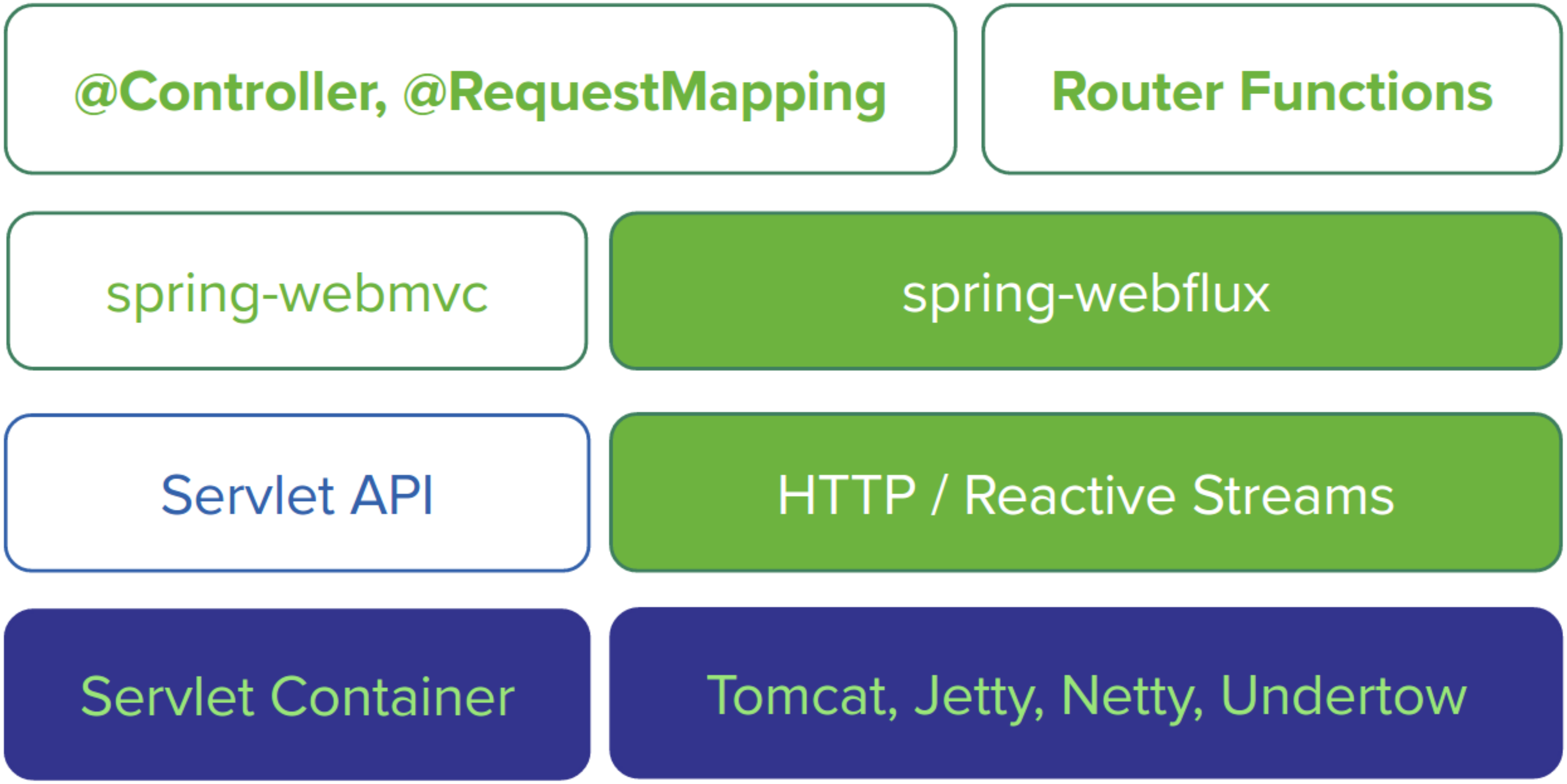
Spring WebFlux



```
WebClient client = WebClient.create("http://example.com");

Mono<Account> account = client.get()
        .url("/{user}/customers", 1L)
        .accept(APPLICATION_JSON)
        .exchange(request)
        .then(response -> response.bodyToMono(Customer.class));
```

client-side

Spring WebFlux



Agenda

- Reactive Programming
- Reactor
- Spring WebFlux
- Spring Boot with Reactor and WebFlux

Spring Boot with Reactor and WebFlux

- To get started, add the ***spring-boot-starter-webflux*** module dependency to your application

- Spring Boot provides ***auto-configuration*** for ***Spring WebFlux***:
  - Configuring codecs for ***HttpMessageReader*** and ***HttpMessageWriter*** instances
  - Support for serving static resources, including support for WebJars

Spring Boot with Reactor and WebFlux

- Easy override through ***application.properties*** and a ***@Configuration*** class of type ***WebFluxConfigurer***

- Spring ***WebFlux*** supports a variety of templating technologies including ***Thymeleaf***, ***FreeMarker*** and ***Mustache***

- Error Handling with ***AbstractErrorWebExceptionHandler***, a WebFlux functional way

Spring Boot with Reactor and WebFlux

- By adding *spring-starter-webflux* and *spring-boot-starter-actuator*, the Actuator endpoints can expose ***Mono*** or ***Flux*** types and can be expose and use along with **Micrometer** (http://micrometer.io/)

Lab

Reactive Programming with Spring Boot and WebFlux

Summary

- ***reactive programming*** is about non-blocking applications that are asynchronous and event-driven and require a small number of threads to scale vertically
- ***reactor*** is a fully non-blocking reactive programming foundation for the JVM, with efficient demand management
- ***spring framework 5*** embraces ***Reactive Streams*** as the contract for communicating backpressure across async components and libraries
- spring boot brings the ***auto-configuration*** for ***WebFlux***

Pivotal

A NEW PLATFORM FOR A NEW ERA

Spring Integration and Cloud Stream with Spring Boot

Spring Boot Developer

Moving to the Cloud

Pivotal™

Agenda

- Spring Integration
- Spring Boot Cloud Stream

Spring Integration

- ***Spring Integration*** is an extension of the spring framework's messaging domain model that provides an Enterprise Integration support with a higher level of abstraction:
  - Provide a simple model for implementing complex enterprise integration solutions.
  - Facilitate asynchronous, message-driven behavior within a spring-based application.
  - Promote intuitive, incremental adoption for existing Spring users.

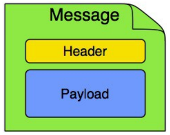
Spring Integration

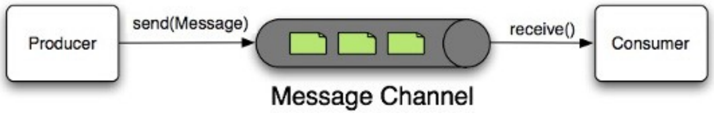
- ***Spring Integration*** is guided by the following principles:
  - Components should be *loosely coupled* for modularity and testability.
  - The framework should enforce *separation of concerns* between business logic and integration logic.
  - Extension points should be abstract in nature but within well-defined boundaries to promote *reuse* and *portability*.

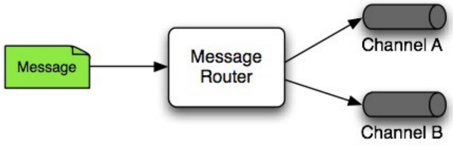
Spring Integration

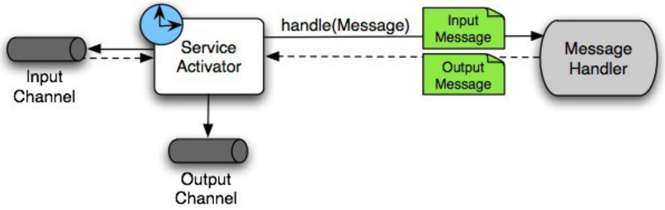
Main Components:

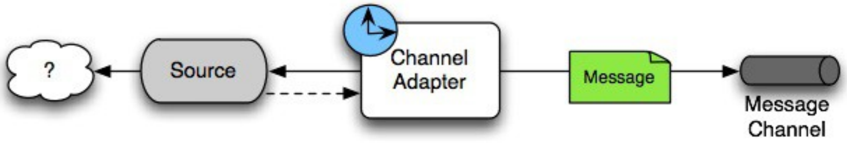
- Message
- Message Channel
- Message Endpoint:
  - Transformer
  - Filter
  - Router
  - Splitter
  - Aggregator
  - Service Activator
  - Channel Adapter

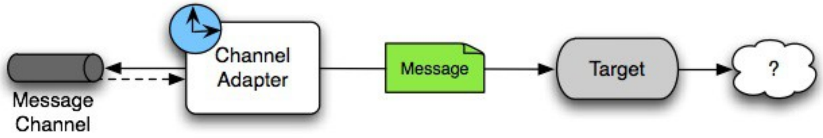










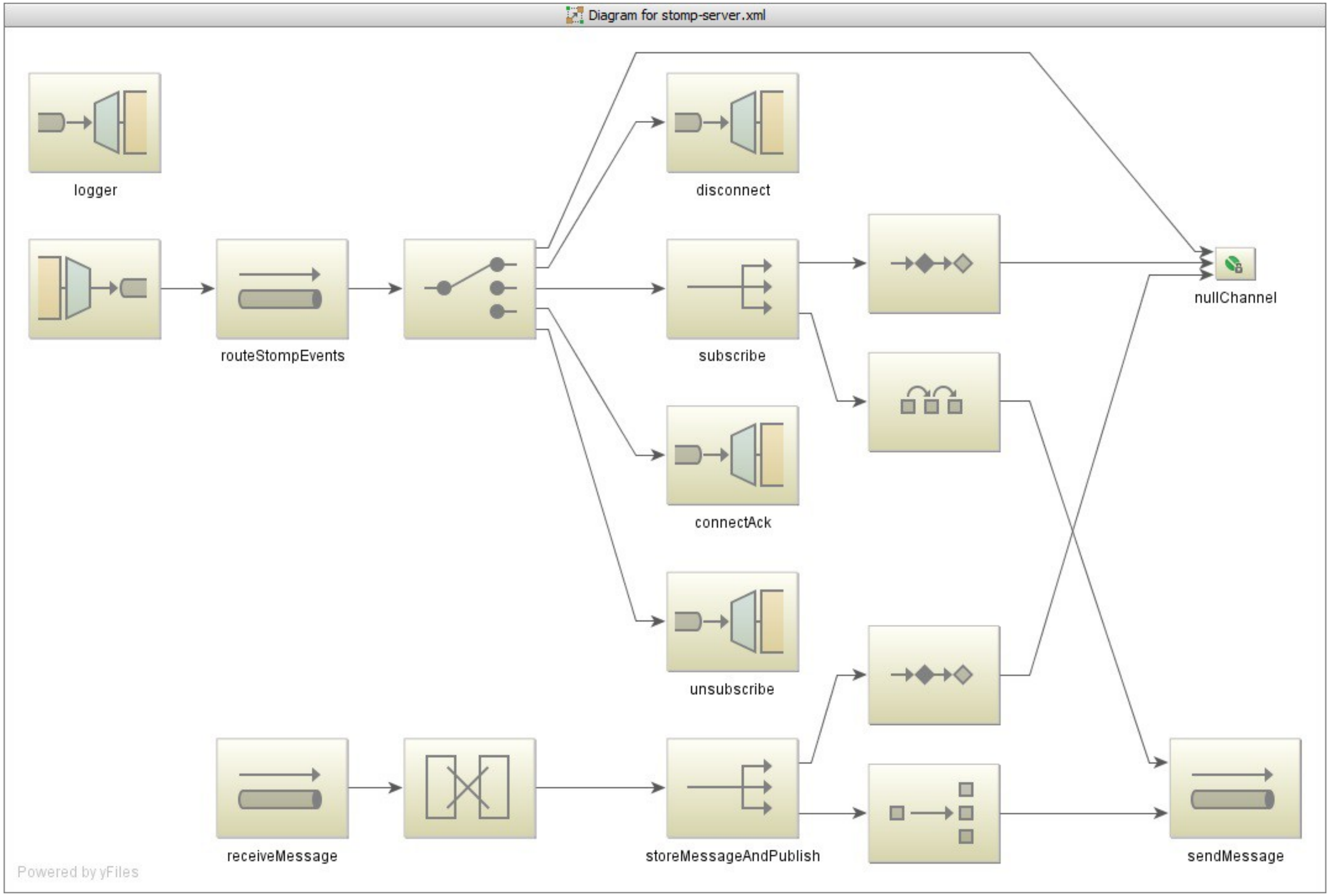


Spring Integration

- ***Spring Integration*** uses the same configuration model from spring framework: ***XML, Java Config*** or ***Annotations***
- To use ***Spring Integration*** in your Spring Boot application add the ***spring-boot-starter-integration*** dependency
- In your ***@Configuration*** class use the ***@EnableIntegration***; this annotation registers many infrastructure components like:
  - ***errorChannel***, ***LoggingHandler***, ***taskScheduler***, ***jsonPath*** and more.
  - adds several ***BeanFactoryPostProcessor*** and ***BeanPostProcessor*** beans to enhance the integration environment.
  - adds annotations processors to parse Messaging Annotations.

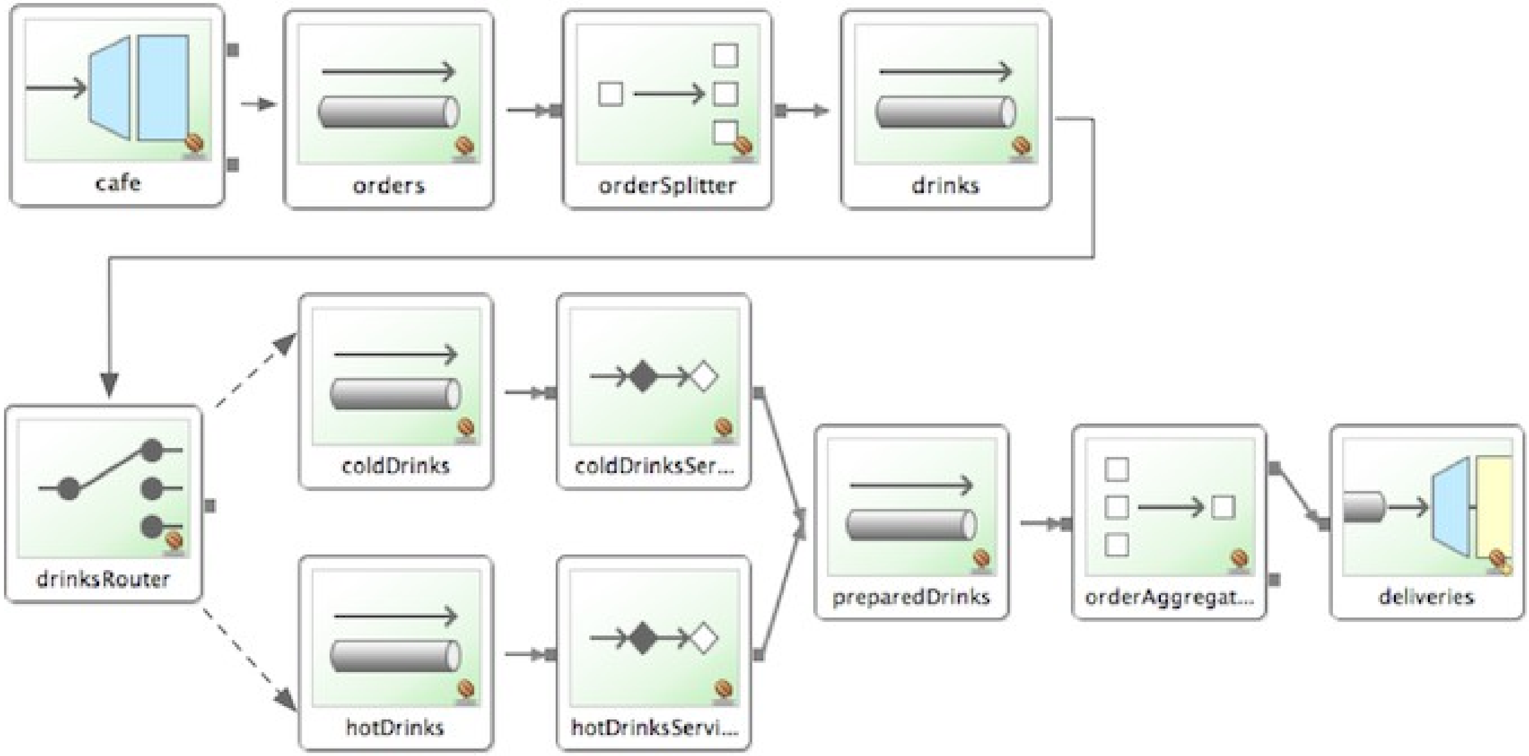
Spring Integration

IntelliJ - XML



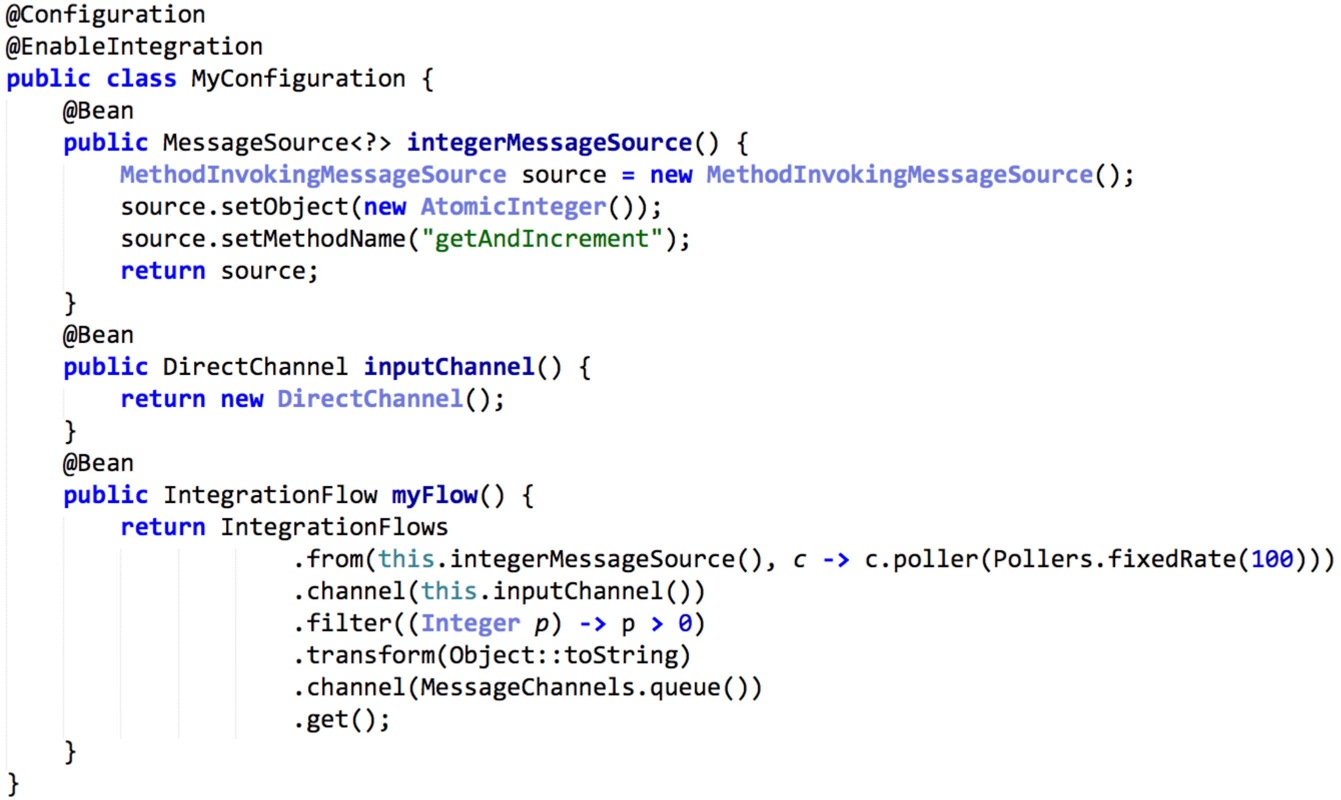
Spring Integration

STS - XML



Spring Integration

- ***Spring Integration*** offers a ***DSL*** extension that provides a set of convenient **Builders** and a **fluent API** to configure ***Spring Integration*** message from spring ***@Configuration*** classes



```
@Configuration
@EnableIntegration
public class MyConfiguration {
    @Bean
    public MessageSource<?> integerMessageSource() {
        MethodInvokingMessageSource source = new MethodInvokingMessageSource();
        source.setObject(new AtomicInteger());
        source.setMethodName("getAndIncrement");
        return source;
    }
    @Bean
    public DirectChannel inputChannel() {
        return new DirectChannel();
    }
    @Bean
    public IntegrationFlow myFlow() {
        return IntegrationFlows
                    .from(this.integerMessageSource(), c -> c.poller(Pollers.fixedRate(100)))
                    .channel(this.inputChannel())
                    .filter((Integer p) -> p > 0)
                    .transform(Object::toString)
                    .channel(MessageChannels.queue())
                    .get();
    }
}
```

Agenda

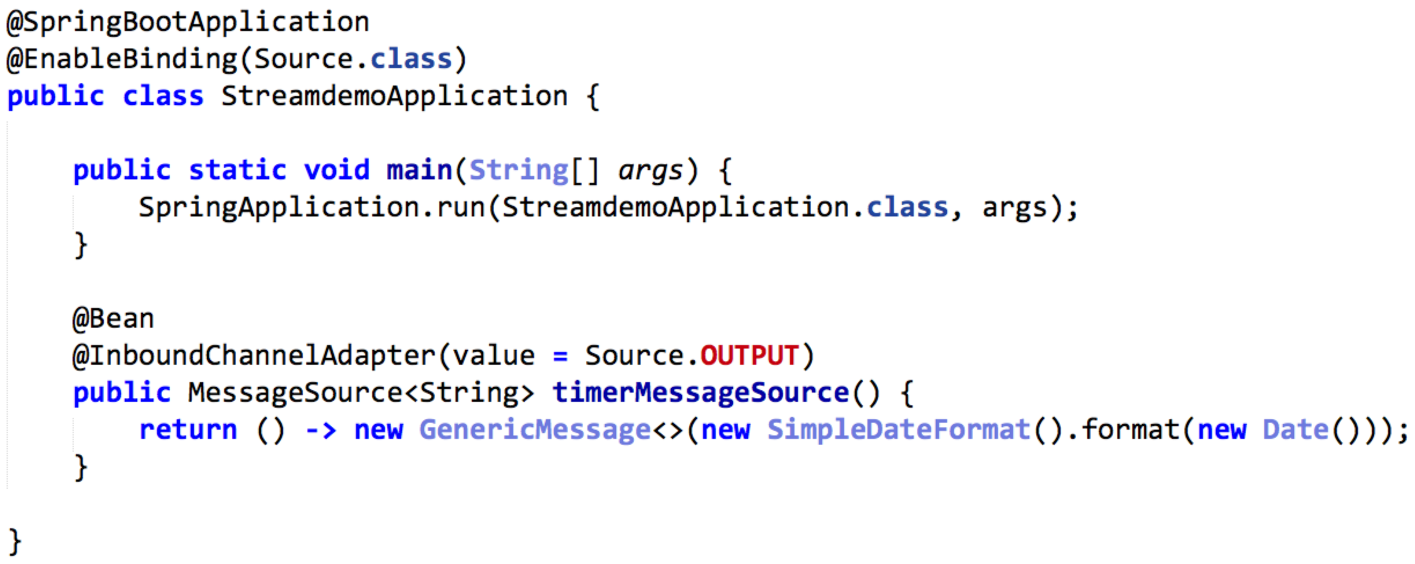
- Spring Integration
- Spring Boot Cloud Stream

Spring Boot Cloud Stream

- ***Spring Cloud Stream*** is a framework for building ***Message-Driven Microservices***.
- ***Spring Cloud Stream*** builds upon ***Spring Boot*** to create ***DevOps*** friendly microservice applications and ***Spring Integration*** to provide connectivity to message brokers.
- ***Spring Cloud Stream*** provides an opinionated configuration of message brokers, introducing the concepts of persistent pub/sub semantics, consumer groups and partitions across several middleware vendors.

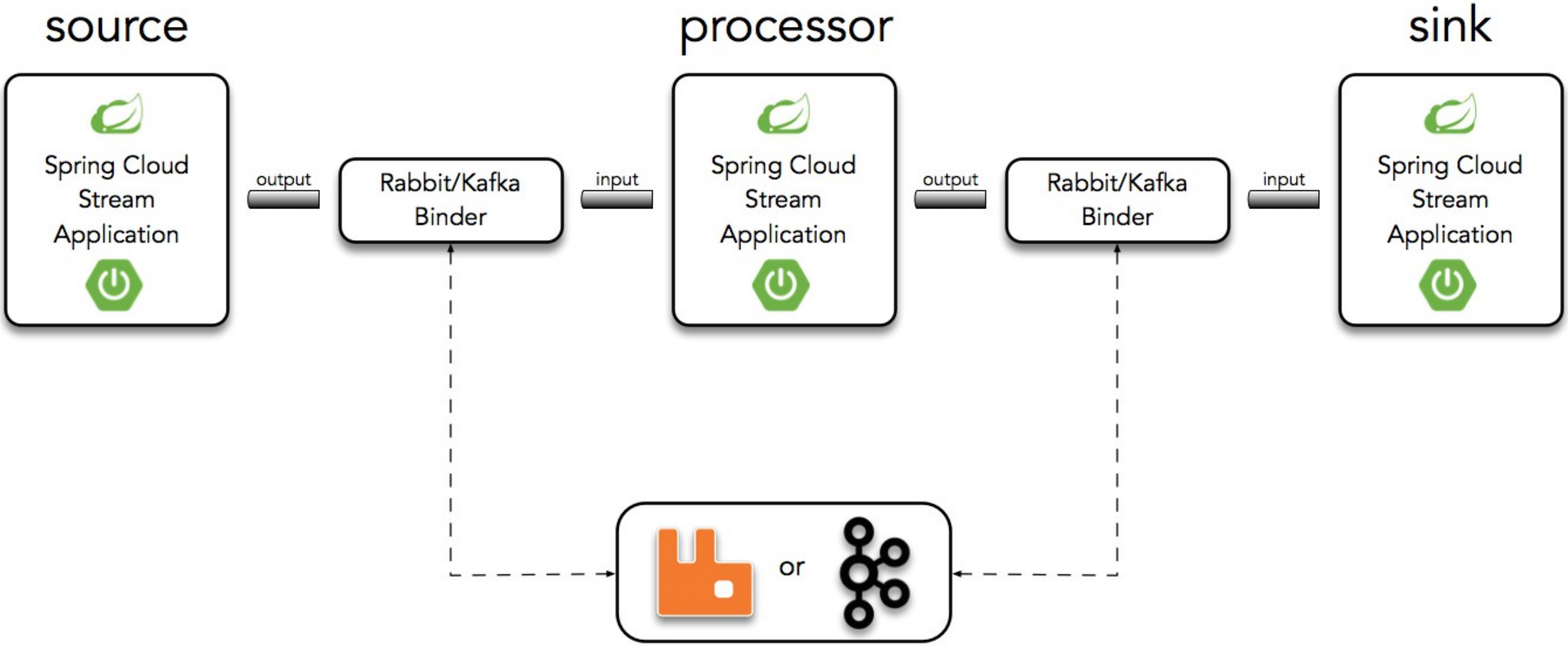
Spring Boot Cloud Stream

- To use ***spring cloud stream*** in your application, add a ***<dependencyManagement/>*** tag and the ***spring-cloud-stream*** dependency.
- By adding ***@EnableBinding*** to your main application, you get immediate connectivity to a message broker and by adding ***@StreamListener*** to a method, you will receive events for stream processing.



```java
@SpringBootApplication
@EnableBinding(Source.class)
public class StreamdemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(StreamdemoApplication.class, args);
    }

    @Bean
    @InboundChannelAdapter(value = Source.OUTPUT)
    public MessageSource<String> timerMessageSource() {
        return () -> new GenericMessage<>(new SimpleDateFormat().format(new Date()));
    }

}
```

Spring Boot Cloud Stream


source
processor
sink
Spring Cloud Stream Application
output
Rabbit/Kafka Binder
input
Spring Cloud Stream Application
output
Rabbit/Kafka Binder
input
Spring Cloud Stream Application
or

Spring Boot Cloud Stream

- ***Spring Cloud Stream Application Starters*** are standalone executable applications that communicate over messaging middleware such as Apache Kafka and RabbitMQ.

- These applications can run independently on variety of runtime platforms including: ***Cloud Foundry***, **Apache Yarn**, **Apache Mesos**, **Kubernetes**, **Docker**, or even on your laptop

- Features:
  - Run standalone as ***Spring Boot*** applications
  - Compose microservice as streaming pipelines in ***Spring Cloud Data Flow***
  - Consume microservice applications as **Maven** or **Docker** artifacts
  - Override configuration parameters via command-line, environment variables, or YAML file
  - Provide infrastructure to test the applications in isolation

https://start-scs.cfapps.io/

Lab

Spring Boot with Spring Integration and Spring Cloud Stream

Summary

- ***Spring Integration***, extends the Spring programming model to support the well-known Enterprise Integration Patterns.
  - ***Spring Integration*** enables lightweight messaging within Spring-based applications and supports integration with external systems via declarative adapters
- ***Cloud Stream***, is a framework for building message-driven microservice applications.
  - Uses ***Spring Integration*** to provide connectivity to message brokers

Pivotal

A NEW PLATFORM FOR A NEW ERA

Microservices with Spring Boot

Spring Boot Developer

Deploying Microservices to Pivotal Cloud Foundry

Agenda

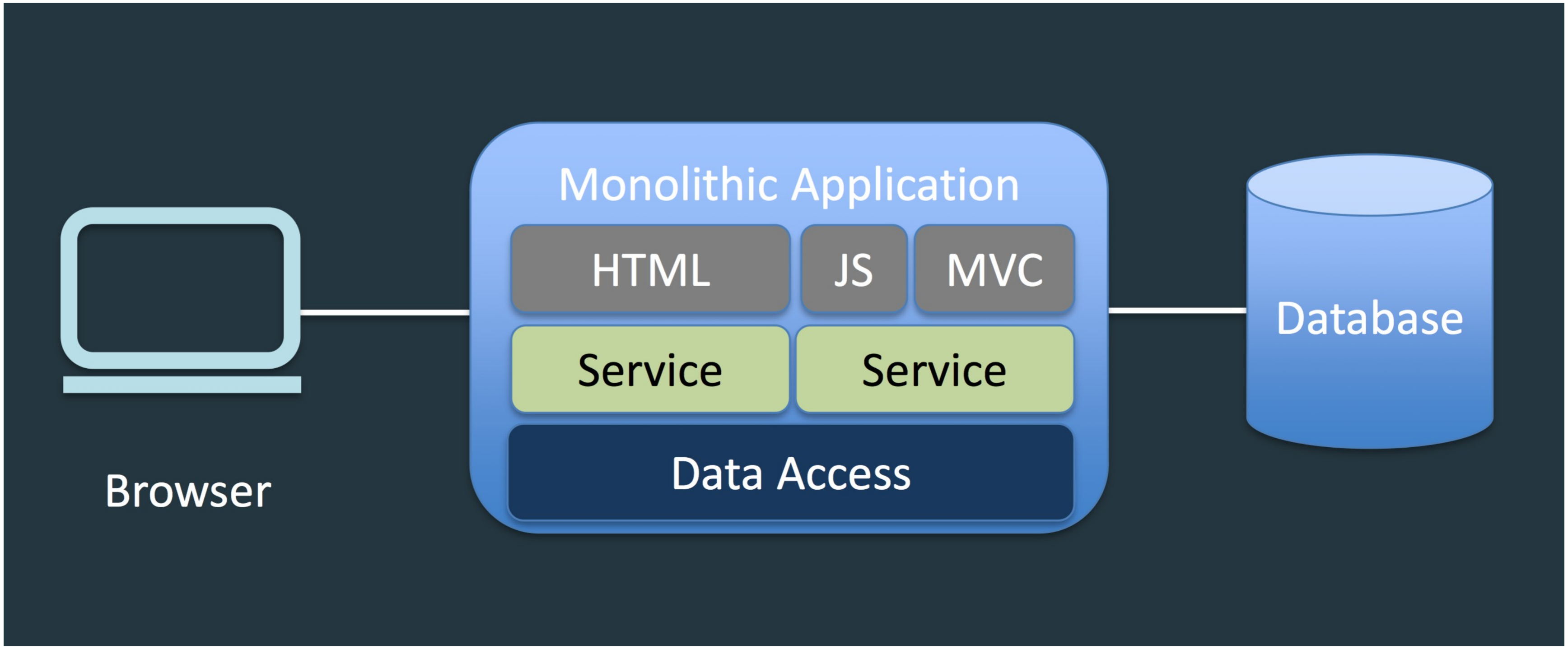
- Microservices
- Cloud Foundry
- Spring Boot in the Cloud

Microservices

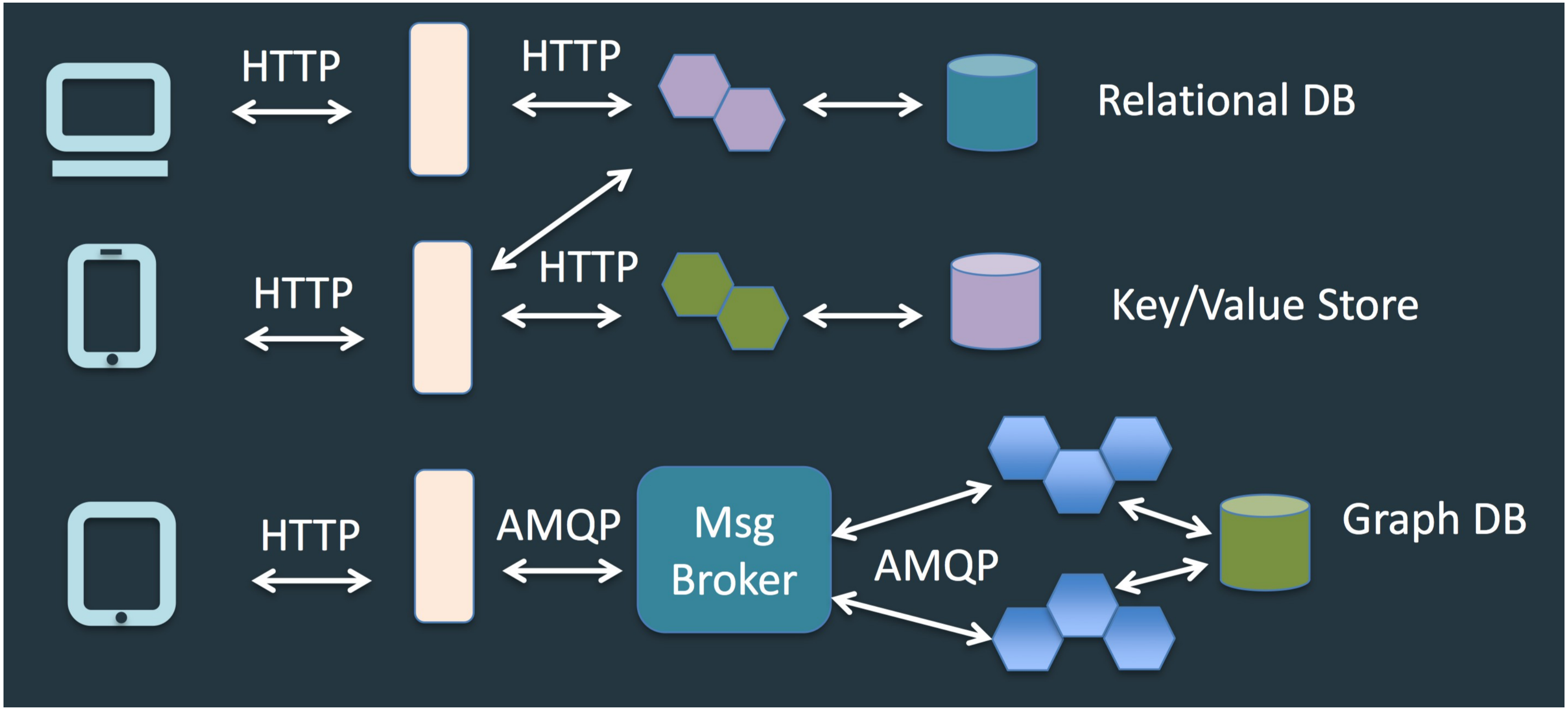
- Microservices is not a new word
  - term coined in 2005 by Dr. Peter Rodgers
  - back then called: micro web services, based on SOAP

Microservices are loosely-couple services with bounded-contexts that perform a single well-defined function

Microservices: Monolith



Microservices



Microservices

Benefits of Microservices:

- Smaller code base, easy to maintain
- Easy to scale, independent deployment
- Technology diversity
- Fault isolation
  - component failure unlikely to bring down the whole system
- Better support for parallel teams

Microservices

Microservices Features:

- API (contracts) interaction only
  - Loosely coupled
  - RESTful APIs
- Bounded-Context / **D**omain-**D**riven-**D**esign
  - Single view of data
- Polyglot persistence and development
- Easy to scale, independent deployment

Microservices

Tradeoffs

- Monolith:
  - Easier to build at first
  - More complex to enhance and maintain
- Microservices:
  - Harder to build at first
  - Simpler to extend, enhance and maintain
  - Scaling out (more processes) easier
  - Many more moving parts to manage

Agenda

- Microservices
- Cloud Foundry
- Spring Boot in the Cloud

Cloud Foundry

Why a platform?

- Deploying distributed systems is complicated
  - Security, Resilience, Redundancy, Load-Balancing

- A platform provides the necessary tools:
  - Natural fit for deploying a microservices-based system
  - Applications instances are the unit of deployment
  - Can be started, stopped and restarted independently on-demand
  - Provide dynamic load-balancing, scaling and routing

Agenda

- Microservices
- Cloud Foundry
- Spring Boot in the Cloud

Spring Boot in the Cloud

Spring Boot can easily be deploy to cloud foundry, just by executing:

cf push my-spring-boot-app.jar

Once deployed multiple microservices issues that now arise:

- How do they find each other?
- How do we decide with instance to use?
- What happens if a microservice is not responding?
- How do we control access?
- How do they communicate?

Lab

Deploying Microservices to Pivotal Cloud Foundry

Summary

- Microservices / Cloud Foundry
- Monolith vs. Microservices
- Tradeoffs
- Cloud foundry
- Spring Boot in the Cloud

Pivotal

A NEW PLATFORM FOR A NEW ERA

Extending Spring Boot

Spring Boot Developer

Custom spring-boot-starter and auto-configuration

Agenda

- Custom Spring Boot Starter
  - auto-configuration
- Spring Boot CLI
  - Custom plugin

Custom Spring Boot Starter

- ***auto-configuration*** can be associated to a "***starter***" that provides the ***auto-configuration*** code as well as the typical libraries that you would use with it

- Under the hood, ***auto-configuration*** is implemented with standard ***@Configuration*** classes

- Additional ***@Conditional*** annotations are used to constrain when the ***auto-configuration*** should apply

- Usually ***auto-configuration*** classes use ***@ConditionalOnClass*** and ***@ConditionalOnMissingBean*** annotations.

Custom Spring Boot Starter

- ***Spring Boot*** checks for the presence of a ***META-INF/spring.factories*** file within your published jar

- You can use the ***@AutoConfigureAfter*** or ***@AutoConfigureBefore*** annotations if your configuration needs to be applied in a specific order

Custom Spring Boot Starter

- A custom ***Spring Boot Starter*** may contain the following components:
  - An ***autoconfigure*** module that contains the ***auto-configuration*** code.
  - The ***starter*** module that provides a dependency to the ***autoconfigure*** module as well as the library and any additional

- Naming:
  - **DO NOT** start your module names with ***spring-boot***

Lab

Custom Spring Boot Starter

Agenda

- Custom Spring Boot Starter
  - auto-configuration
- Spring Boot CLI
  - custom plugin

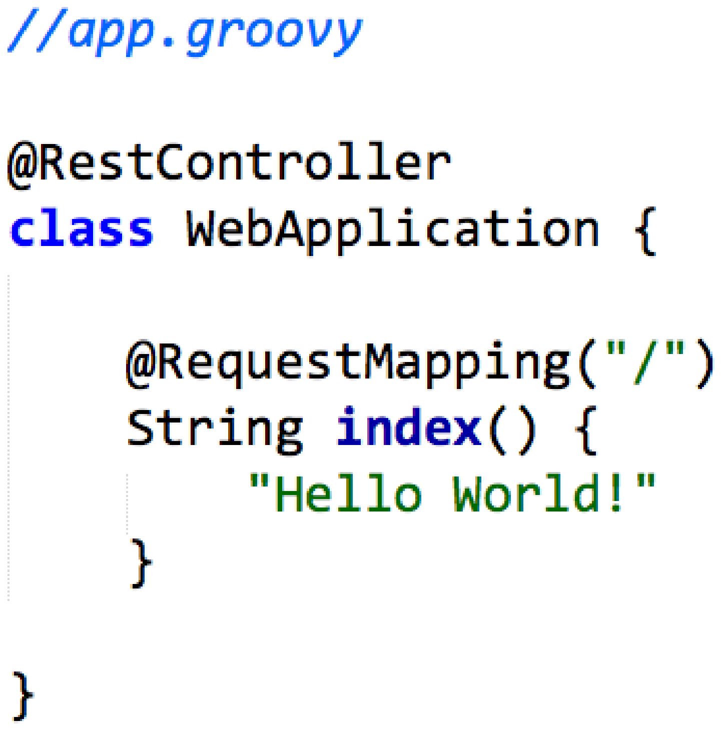
Spring Boot CLI

- The ***Spring Boot CLI*** is a command line tool that can be used if you want to quickly develop with ***Spring***.

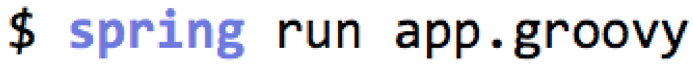
- It allows you to run ***Groovy*** scripts, which means that you have a familiar Java-like syntax, without so much boilerplate code.

- You can also bootstrap a new project or write your own command for it.

Spring Boot CLI



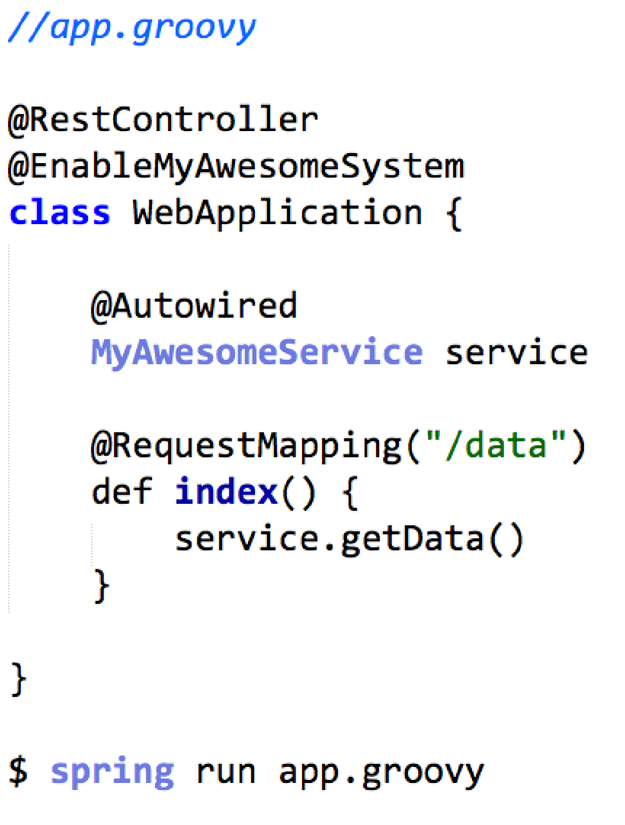
```
//app.groovy

@RestController
class WebApplication {

    @RequestMapping("/")
    String index() {
        "Hello World!"
    }

}
```



```
$ spring run app.groovy
```

Custom Spring Boot CLI plugin

- You can create an custom plugin to create prototypes of your own apps



```
//app.groovy

@RestController
@EnableMyAwesomeSystem
class WebApplication {

    @Autowired
    MyAwesomeService service

    @RequestMapping("/data")
    def index() {
        service.getData()
    }

}

$ spring run app.groovy
```

Custom Spring Boot CLI plugin

- To create a custom Spring Boot CLI plugin is necessary:
  - Include the ***spring-boot-cli*** dependency with ***version*** and ***scope provided***.
  - Add the ***META-INF/services*** files and declare the classes needed:
    - ***org.springframework.boot.cli.command.CommandFactory***
    - ***org.springframework.boot.cli.compiler.CompilerAutoConfiguration***
    - ***org.springframework.boot.cli.compiler.SpringBootAstTransformation***

Custom Spring Boot CLI plugin

- Create (if needed) the command factory classes by implementing the ***CommandFactory*** interface and declare them in the ***org.springframework.boot.cli.command.CommandFactory*** file.

- Create the necessary ***auto-configuration*** classes by extending from ***CompilerAutoConfiguration*** and declare them in the ***org.springframework.boot.cli.compiler.CompilerAutoConfiguration*** file

- Create (if needed) the necessary **BOM AST** transformation classes by extending from ***GenericBomAstTransformation*** class and declare them in the ***org.springframework.boot.cli.compiler.SpringBootAstTransformation*** file

Summary

- Extending Spring Boot
  - Custom spring-boot-starter:
    - auto-configuration - @Conditional
  - Spring Boot CLI, custom extension

Pivotal

A NEW PLATFORM FOR A NEW ERA

Finishing Up

Course Completed

What's next?

What's Next

- Congratulations, we've finished the course

- What to do next?
  - Certification
  - Other courses
  - Resources
  - Evaluation

- Check-out optional sections on ...

Certification

Pivotal CERTIFIED Spring Professional

- Computer-based exam
  - 50 multiple-choice questions
  - 90 minutes
  - Passing score: 76% (38 questions answered successfully)

- Preparation
  - Review all the slides
  - Redo the labs

Certification: Questions

Typical question

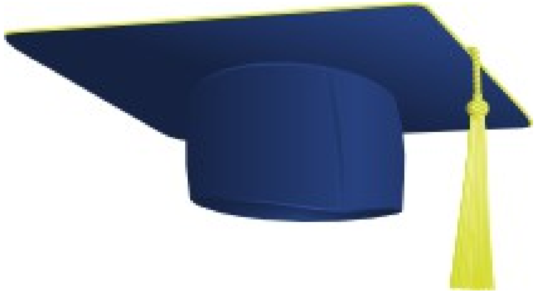
- Statements
  - a. An application context holds Spring beans
  - b. An application context manages bean scope
  - c. Spring provides many types of application context
- Pick the correct response:
  1. Only a. is correct
  2. Both a. and c. are correct
  3. All are correct
  4. None are correct

Certification: Logistics

Voucher is valid for 3 months – *do it soon!*

- Where?
  - Online at PSI (Innovative Exams)
    - https://www.examslocal.com
- How?
  - You should receive a certification voucher by email
  - Register/sign-in and book an exam using the voucher
    - http://it.psionline.com/exam-faqs/pivotal-faq
  - Take the test from *any* location
- For more information, email
  - education@pivotal.io

Other courses



- Many courses available
  - Core Spring
  - Web Applications with Spring
  - Enterprise Spring
  - Spring Boot
  - Spring Cloud Services
  - Pivotal Cloud Foundry
  - Gemfire, Rabbit MQ …
- More details here:
  - http://www.pivotal.io/training

Core Spring

- Four day course covering
  - Application configuration using
    Java Configuration, XML and/or Annotations
  - How Spring works internally and makes use of Aspect Oriented Programming
  - Data persistence using JDBC and JPA
  - Declarative Transaction Management
  - Introduction to web-applications and Spring MVC
  - Building RESTful Servers
  - Spring Boot, Spring Cloud and Microservices

Spring Web

- 4-day workshop

- Making the most of Spring in the web layer
  - Spring MVC
  - Spring Web Flow
  - REST using MVC and AJAX
  - Security of Web applications
  - Performance testing

- Spring Web Application Developer certification

Enterprise Spring

Pivotal CERTIFIED Spring Enterprise Integration Specialist

- Building loosely coupled event-driven architectures
  - Separate processing, communications & integration
- 4 day course covering
  - Tasks, Scheduling and Concurrency
  - Advanced transaction management
  - REST Web Services with Spring MVC
  - Spring Batch
  - Spring Integration
  - Data Ingestion, Transformation and Extractions

Spring Boot Developer

- 2 day workshop
  - Introduction to Spring Boot
  - Building Web and REST Applications
  - Integrating Data Management
  - Using Actuators, Health Monitoring
  - Security and OAuth2
  - Messaging using RabbitMQ
  - Deployment

Spring Cloud Services

Microservices With Spring

- 2 day course
  - Introduction to Spring Boot
    - Underpins all Spring Cloud projects
  - Pushing Applications to a PaaS
    - Using Pivotal Cloud Foundry
  - What are Microservices?
    - Architecting a microservices solution
  - Cloud infrastructure services and Netflix OSS
    - Service Configuration
    - Service Registration
    - Load-balancing and fault tollerence

Cloud Foundry Developer

- 3 day course covering
  - Application deployment to Cloud Foundry
    - Typically these are Web and/or REST applications
    - Deployment using cf tool or an IDE
  - Cloud Foundry Concepts
    - Logging, Continuous Integration, Monitoring
    - Accessing and defining Services
    - Using and customizing Buildpacks
  - Design considerations: “12 Factor”
    - JVM application specifics, using Spring Cloud

Formerly: Developing Applications with Cloud Foundry

Cloud Foundry Administrator

- 3 day course covering
  - Administration
    - Deploying Cloud Foundry to vSphere or AWS
    - Configuring and Managing Cloud Foundry
    - Working with BOSH
  - Application deployment to Cloud Foundry
    - Includes the basic topics from the Developer course
    - Logging, Continuous Integration, Monitoring
    - Design considerations: “12 Factor” Applications

Broader course than Developer with administrator emphasis

Pivotal Support Offerings

- Global organization provides 24x7 support
  - How to Register: http://tinyurl.com/piv-support

- Premium and Developer support offerings:
  - http://www.pivotal.io/support/offerings
  - http://www.pivotal.io/support/oss
  - Both Pivotal App Suite *and* Open Source products

- Support Portal: https://support.pivotal.io
  - Community forums, Knowledge Base, Product documents

Pivotal Consulting

- Custom consulting engagement?
  - Contact us to arrange it
    - http://www.pivotal.io/contact/spring-support
    - Even if you don't have a support contract!

- Pivotal Labs
  - Agile development experts
  - Assist with design, development and product management
    - http://www.pivotal.io/agile
    - http://pivotallabs.com

Resources

- The Spring reference documentation
  - http://spring.io/docs
  - Already 800+ pages!
- The official technical blog
  - http://spring.io/blog
- Stack Overflow – Active Spring Forums
  - http://stackoverflow.com

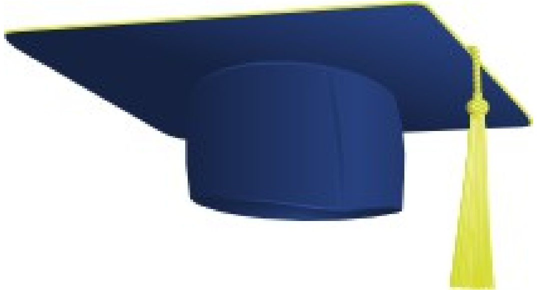
Resources (2)

- You can register issues on our Jira repository
  - https://jira.spring.io

- The source code is available here
  - https://github.com/spring-projects/spring-framework

- Follow Spring development
  - https://fisheye.springsource.org/browse/

Thank You!



- We hope you enjoyed the course
- Please fill out the evaluation form
  - Americas: http://tinyurl.com/usa-eval
  - EMEA: http://tinyurl.com/emea-eval
  - Asia-Pac: http://tinyurl.com/apj-eval
- Once you've done, login to *Pivotal Academy*
  - You can download your Attendance Certificate

***If** your course is registered at Pivotal Academy*

Don't forget the optional sections