GETTING STARTED

Building an Application with Spring Boot

This guide provides a sampling of how Spring Boot helps you accelerate and facilitate application development. As you read more Spring Getting Started guides, you will see more use cases for Spring Boot. It is meant to give you a quick taste of Spring Boot. If you want to create your own Spring Boot-based project, visit Spring Initializr, fill in your project details, pick your options, and you can download either a Maven build file, or a bundled up project as a zip file.

What you'll build

You'll build a simple web application with Spring Boot and add some useful services to it.

What you'll need

- About 15 minutes
- A favorite text editor or IDE
- JDK 1.8 or later
- Gradle 4+ or Maven 3.2+
- You can also import the code straight into your IDE:
  - Spring Tool Suite (STS)
  - IntelliJ IDEA

How to complete this guide

Like most Spring Getting Started guides, you can start from scratch and complete each step, or you can bypass basic setup steps that are already familiar to you. Either way, you end up with working code.

To **start from scratch**, move on to Build with Gradle.

To **skip the basics**, do the following:

- Download and unzip the source repository for this guide, or clone it using Git: `git clone https://github.com/spring-guides/gs-spring-boot.git`

- cd into `gs-spring-boot/initial`
- Jump ahead to [initial].

When you’re finished, you can check your results against the code in `gs-spring-boot/complete`.

Build with Gradle

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with Gradle and Maven is included here. If you’re not familiar with either, refer to Building Java Projects with Gradle or Building Java Projects with Maven.

Create the directory structure

In a project directory of your choosing, create the following subdirectory structure; for example, with `mkdir -p src/main/java/hello` on *nix systems:

```
└── src
    └── main
        └── java
            └── hello
```

Create a Gradle build file

Below is the initial Gradle build file.

`build.gradle`

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:2.0.5.RELEASE")
    }
```

```
}


apply plugin: 'java'

apply plugin: 'eclipse'

apply plugin: 'idea'

apply plugin: 'org.springframework.boot'

apply plugin: 'io.spring.dependency-management'


bootJar {

    baseName = 'gs-spring-boot'

    version =  '0.1.0'

}


repositories {

    mavenCentral()

}


sourceCompatibility = 1.8

targetCompatibility = 1.8


dependencies {

    compile("org.springframework.boot:spring-boot-starter-web")
    testCompile("junit:junit")

}
```

The Spring Boot gradle plugin provides many convenient features:

- It collects all the jars on the classpath and builds a single, runnable "über-jar", which makes it more convenient to execute and transport your service.
- It searches for the `public static void main()` method to flag as a runnable class.
- It provides a built-in dependency resolver that sets the version number to match Spring Boot dependencies. You can override any version you wish, but it will default to Boot's chosen set of versions.

Build with Maven

First you set up a basic build script. You can use any build system you like when building apps with Spring, but the code you need to work with Maven is included here. If you're not familiar with Maven, refer to Building Java Projects with Maven.

Create the directory structure

In a project directory of your choosing, create the following subdirectory structure; for example, with `mkdir -p src/main/java/hello` on *nix systems:

```
└── src

    └── main

        └── java

            └── hello
```

`pom.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>org.springframework</groupId>

    <artifactId>gs-spring-boot</artifactId>

    <version>0.1.0</version>
```

```xml
    <parent>

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

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

        <version>2.0.5.RELEASE</version>

    </parent>



    <dependencies>

        <dependency>

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

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

        </dependency>

    </dependencies>



    <properties>

        <java.version>1.8</java.version>

    </properties>





    <build>

        <plugins>

            <plugin>

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

            </plugin>

        </plugins>

    </build>



</project>
```

The Spring Boot Maven plugin provides many convenient features:

- It collects all the jars on the classpath and builds a single, runnable "über-jar", which makes it more convenient to execute and transport your service.
- It searches for the `public static void main()` method to flag as a runnable class.
- It provides a built-in dependency resolver that sets the version number to match Spring Boot dependencies. You can override any version you wish, but it will default to Boot’s chosen set of versions.

Build with your IDE

- Read how to import this guide straight into Spring Tool Suite.
- Read how to work with this guide in IntelliJ IDEA.

Working a Getting Started guide with STS

This guide walks you through using Spring Tool Suite (STS) to build one of the Getting Started guides.

What you'll build

You'll pick a Spring guide and import it into Spring Tool Suite. Then you can read the guide, work on the code, and run the project.

What you'll need

- About 15 minutes
- Spring Tool Suite (STS)
- JDK 8 or later

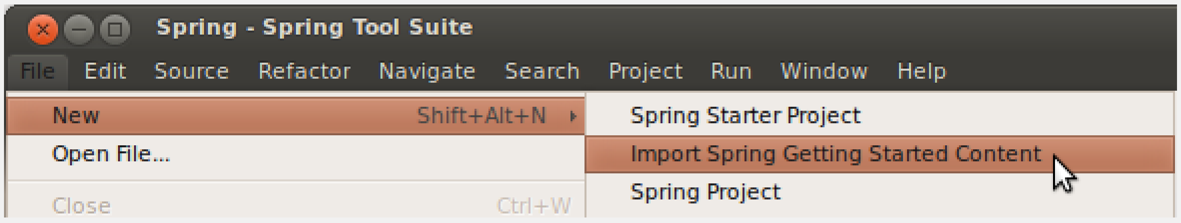
Installing STS

If you don't have STS installed yet, visit the link up above. From there, you can download a copy for your platform. To install it simply unpack the downloaded archive.

When you're done, go ahead and launch STS.

Importing a Getting Started guide

With STS up and running, open the **Import Spring Getting Started Content** wizard from the **File** menu.



A pop-up wizard will offer you the chance to search and pick any of the published guides from the Spring website. You can either skim the list, or enter search words to instantly filter the options.]

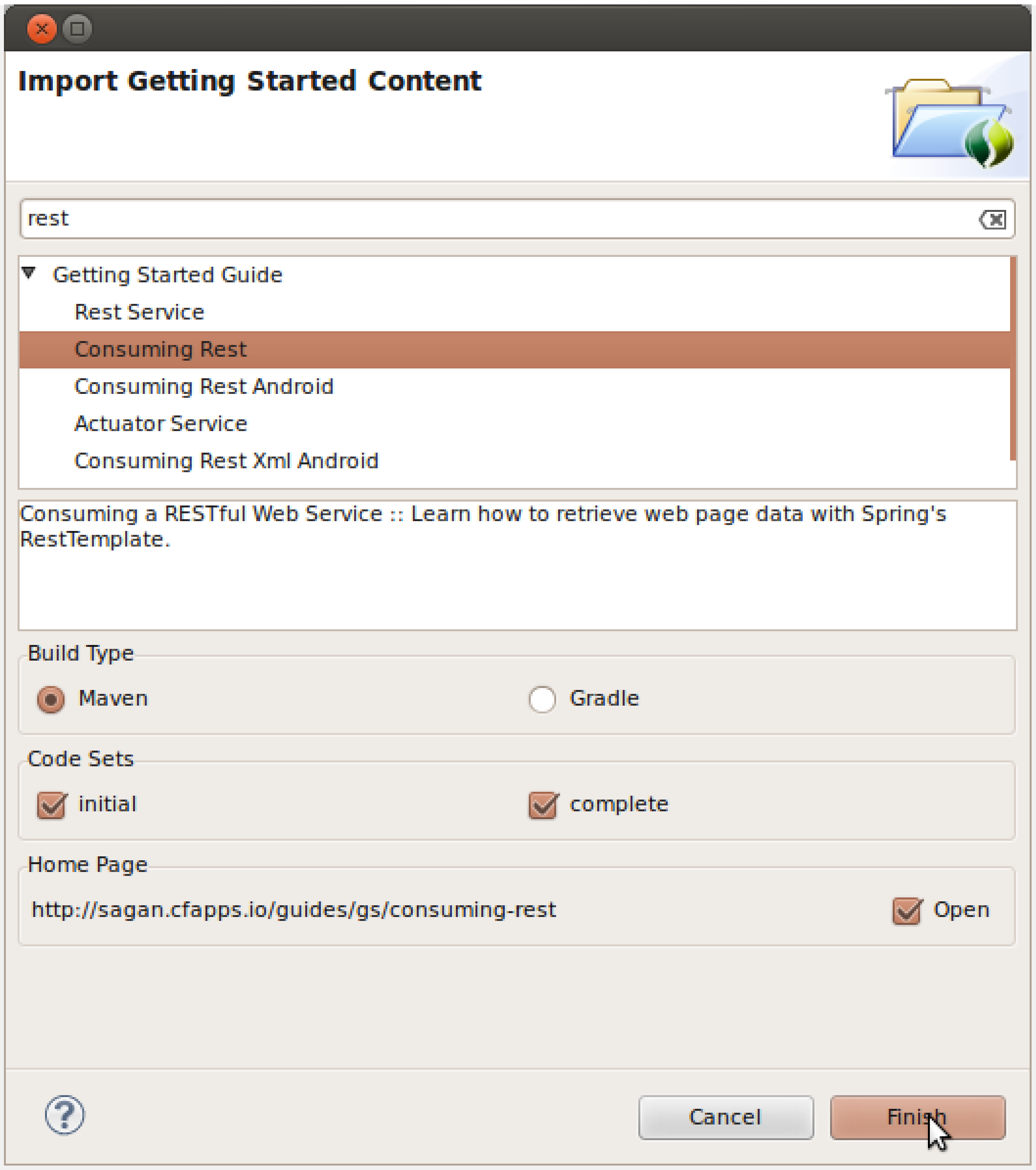
The criteria is applied to both the title and the description when offering instant search results. Wildcards are supported.

You can pick either Maven or Gradle as the build system to use.

You can also decide whether to grab the **initial** code set, **complete** code set, or both. For most projects, the **initial** code set is an empty project, making it possible for you to copy-and-paste your way through a guide. The **complete** code set is all the code from the guide already entered. If you grab both, you can compare your work against the guide's and see the differences.

Finally, you can have STS open a browser tab to the guide on the website. This will let you work through a guide without having to leave STS.

For the purpose of this guide, enter **rest** into the instant search box. Then pick Consuming Rest. Pick **Maven** for building, and **initial** and **complete** code sets. Also opt to open the web page as shown below:



Import Getting Started Content

rest

Getting Started Guide
- Rest Service
- Consuming Rest
- Consuming Rest Android
- Actuator Service
- Consuming Rest Xml Android

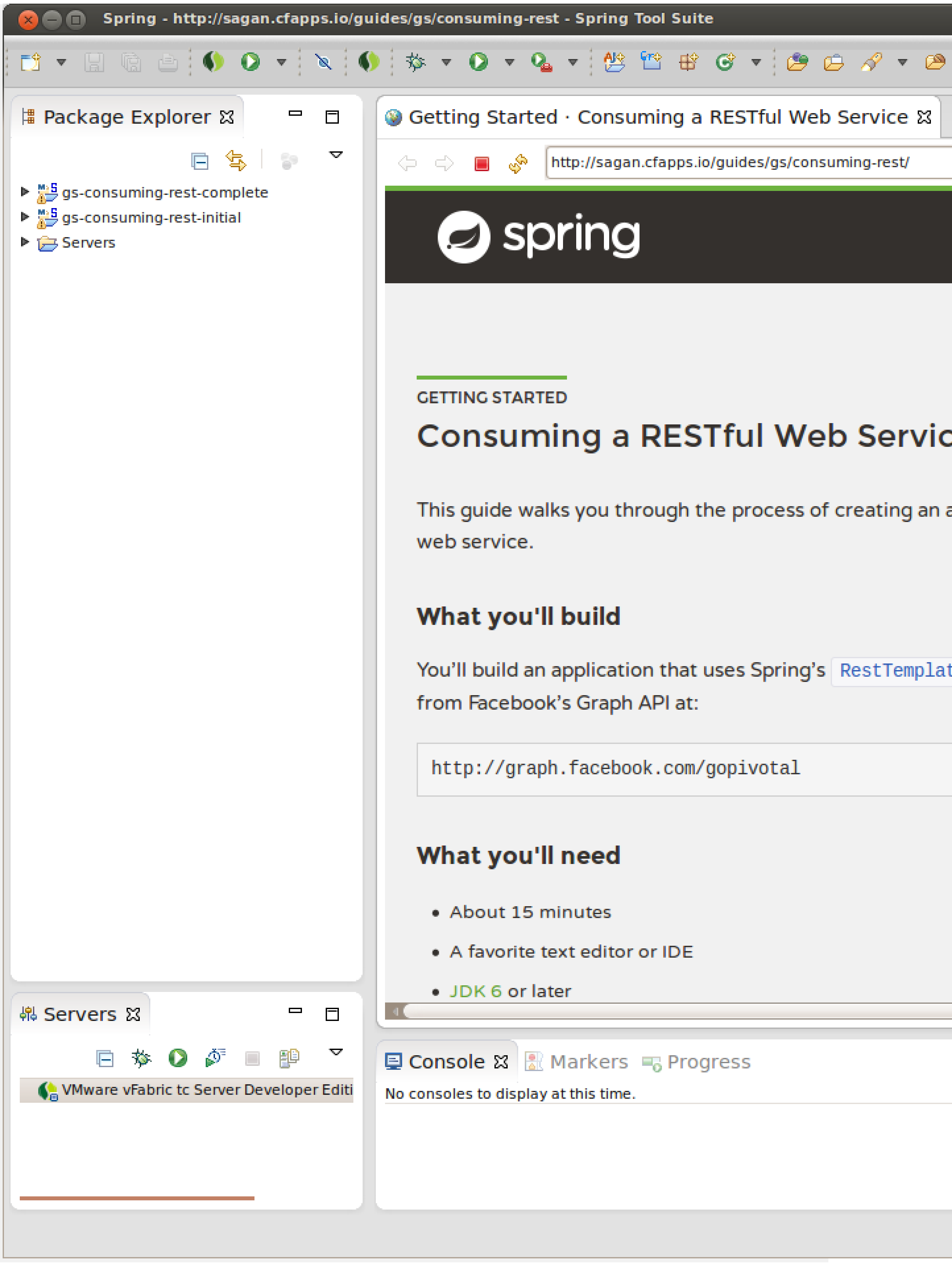
Consuming a RESTful Web Service :: Learn how to retrieve web page data with Spring's RestTemplate.

Build Type: [✓ Maven] Gradle

Code Sets: [✓ initial] [✓ complete]

Home Page: http://sagan.cfapps.io/guides/gs/consuming-rest [✓ Open]

STS will create two new projects in your workspace, import the Consuming Rest code base (both initial and complete), and open a browser tab inside STS as shown below:


Spring - http://sagan.cfapps.io/guides/gs/consuming-rest - Spring Tool Suite
Package Explorer
gs-consuming-rest-complete
gs-consuming-rest-initial
Servers
Getting Started · Consuming a RESTful Web Service
http://sagan.cfapps.io/guides/gs/consuming-rest/
spring
Servers
VMware vFabric tc Server Developer Editi
Console
Markers
Progress

GETTING STARTED

Consuming a RESTful Web Servic

This guide walks you through the process of creating an a
web service.

What you'll build

You'll build an application that uses Spring's `RestTemplat`
from Facebook's Graph API at:

```
http://graph.facebook.com/gopivotal
```

What you'll need

- About 15 minutes
- A favorite text editor or IDE
- JDK 6 or later

No consoles to display at this time.

From here, you can walk through the guide and navigate to the code files.

Summary

Congratulations! You have setup Spring Tool Suite, imported the Consuming Rest getting started guide, and opened a browser tab to walk through it.

See Also

The following guides may also be helpful:

- Working a Getting Started guide with IntelliJ IDEA

Want to write a new guide or contribute to an existing one? Check out our contribution guidelines.

All guides are released with an ASLv2 license for the code, and an Attribution, NoDerivatives creative commons license for the writing.

Working a Getting Started guide with IntelliJ IDEA

This guide walks you through using IntelliJ IDEA to build one of the Getting Started guides.

What you'll build

You'll pick a Spring guide and import it into IntelliJ IDEA. Then you can read the guide, work on the code, and run the project.

What you'll need

- About 15 minutes
- IntelliJ IDEA
- JDK 6 or later

Installing IntelliJ IDEA

If you don’t have IntelliJ IDEA (Ultimate Edition) installed yet, visit the link up above. From there, you can download a copy for your platform. To install it simply unpack the downloaded archive.

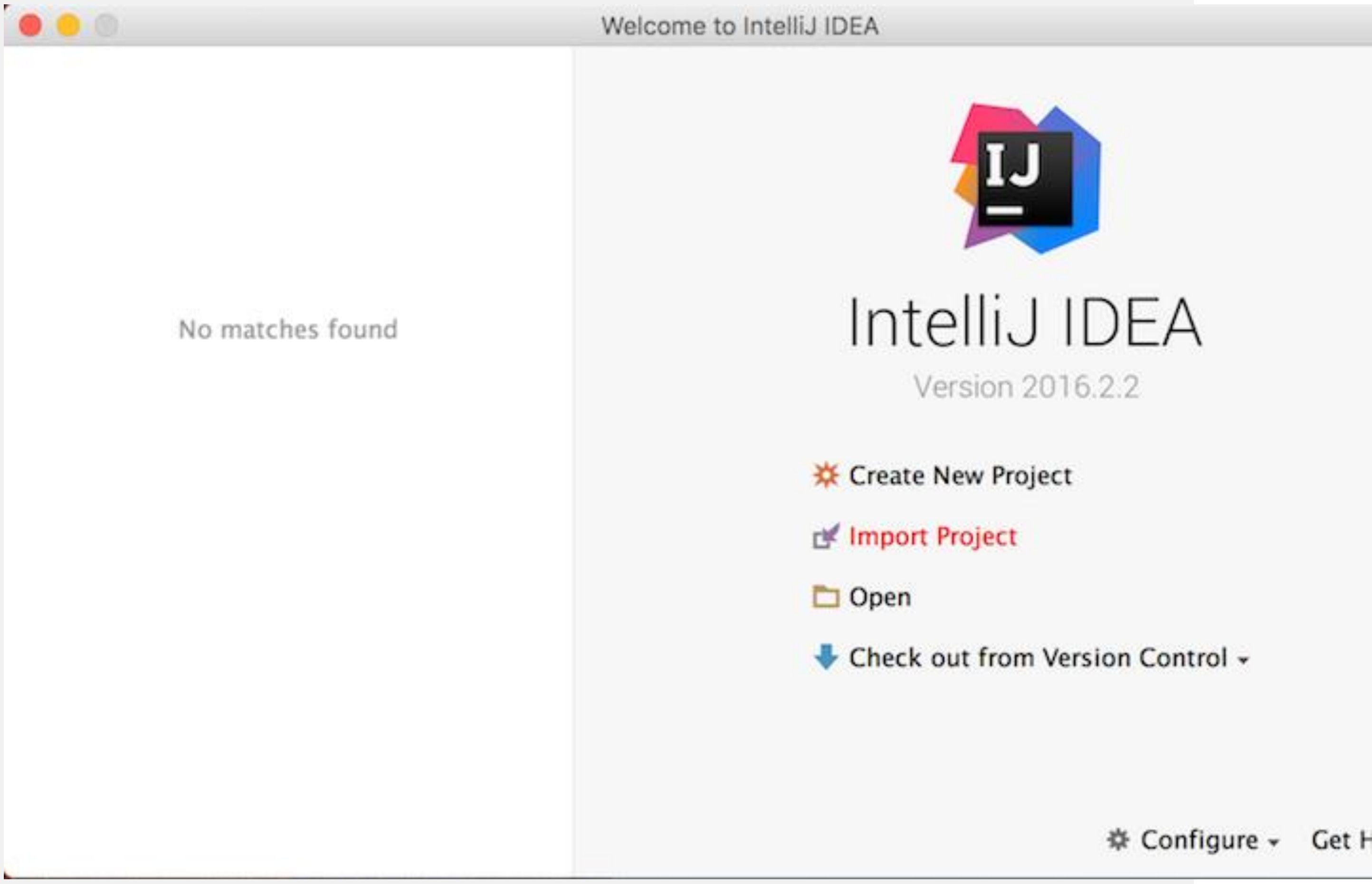
When you’re done, go ahead and launch IntelliJ IDEA.

Importing a Getting Started guide

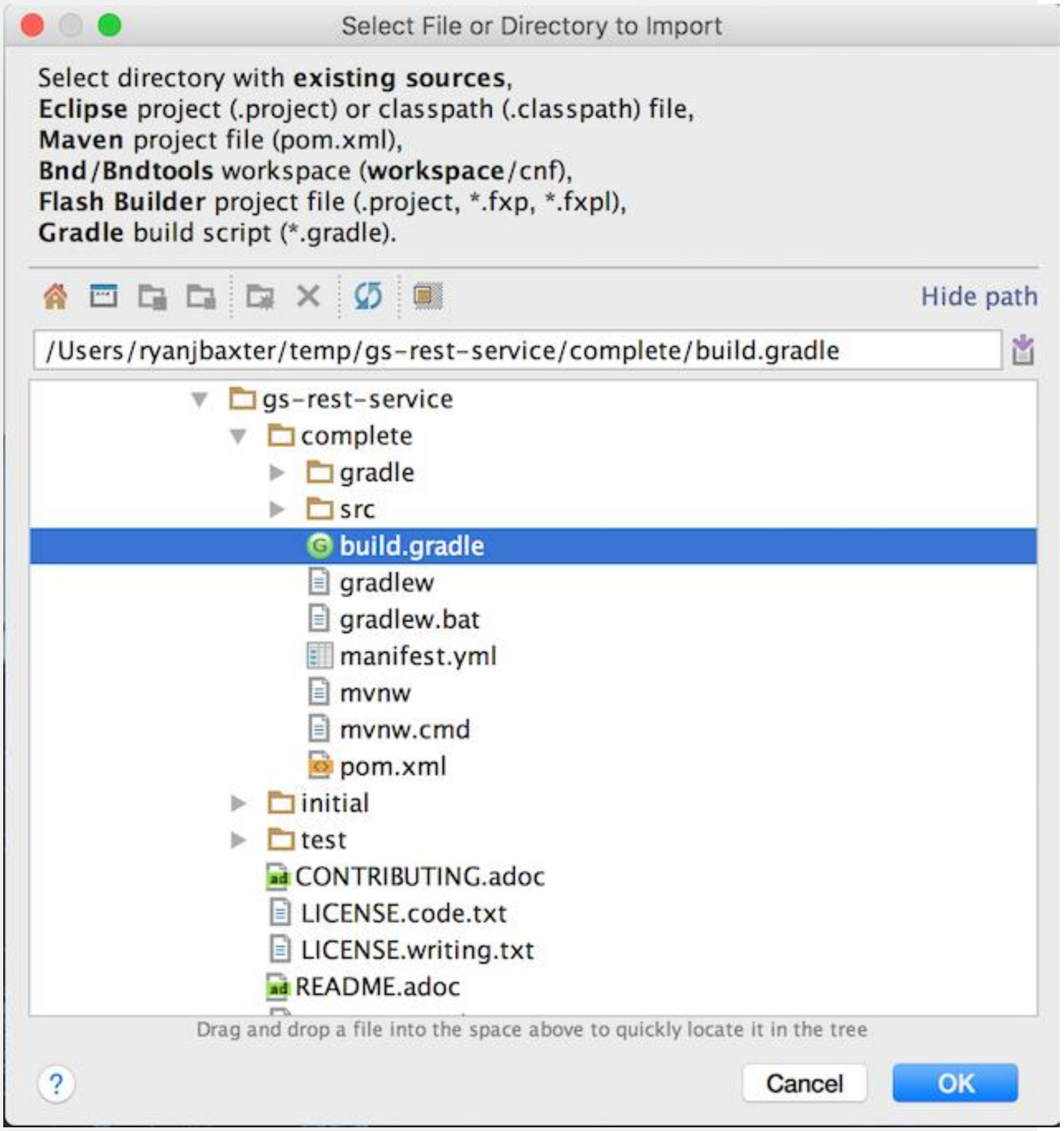
To import an existing project you need some code, so clone or copy one of the Getting Started guides, e.g. the REST Service guide:

```
$ git clone https://github.com/spring-guides/gs-rest-service.git
```

With IntelliJ IDEA up and running, click **Import Project** on the **Welcome Screen**, or **File | Open** on the main menu:



In the pop-up dialog make sure to select
either Maven's **pom.xml** or Gradle's **build.gradle** file under the **complete** folder:



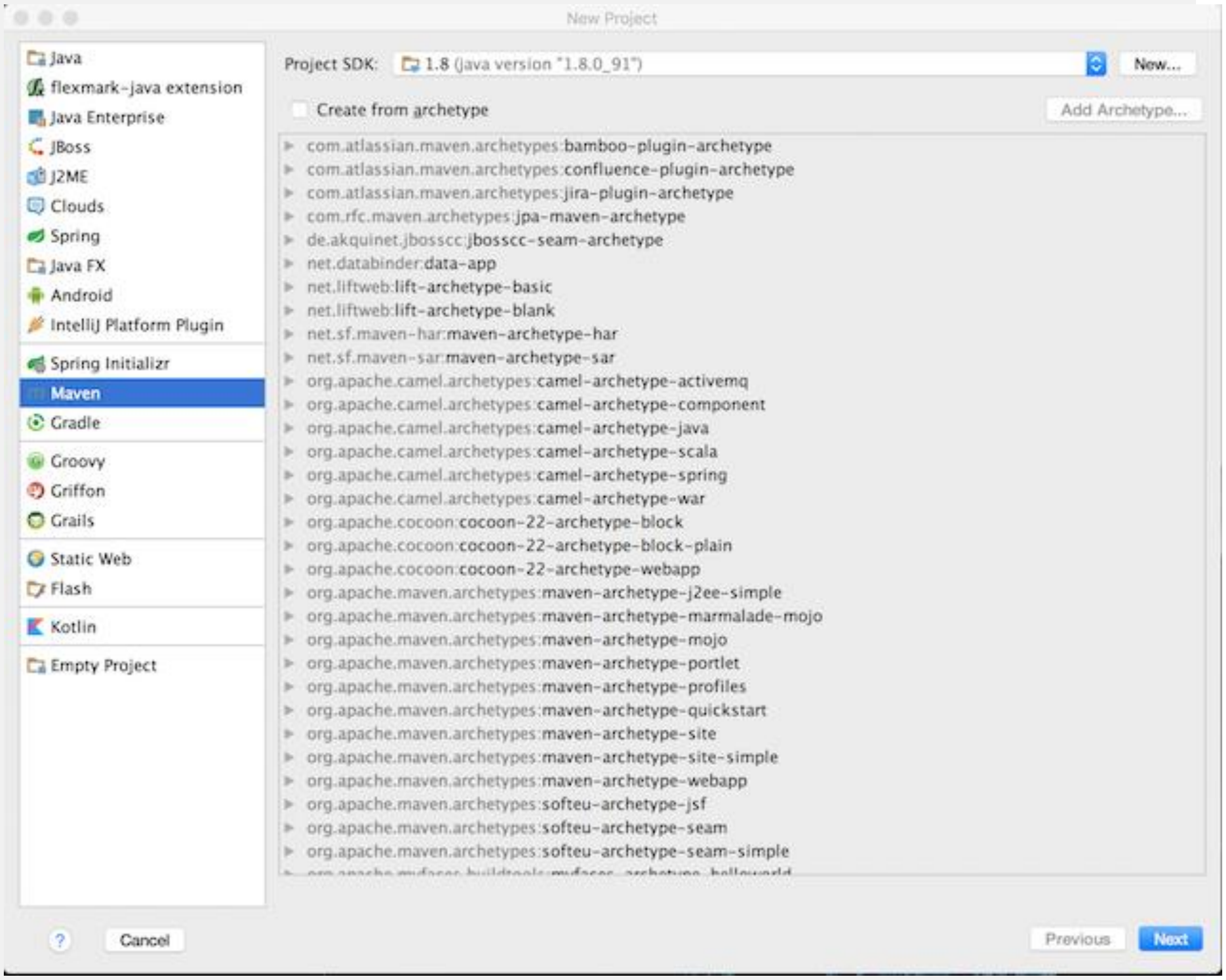
Select directory with **existing sources**,
Eclipse project (.project) or classpath (.classpath) file,
Maven project file (pom.xml),
Bnd/Bndtools workspace (**workspace**/cnf),
Flash Builder project file (.project, *.fxp, *.fxpl),
Gradle build script (*.gradle).

/Users/ryanjbaxter/temp/gs-rest-service/complete/build.gradle

IntelliJ IDEA will create a project with all the code from the guide ready to run.

Creating a Project from Scratch

In case you'd like to start with an empty project and copy-and-paste your way through the guide, create a new **Maven** or **Gradle** project in the **Project Wizard**:


New Project
Java
flexmark-java extension
Java Enterprise
JBoss
J2ME
Clouds
Spring
Java FX
Android
IntelliJ Platform Plugin
Spring Initializr
Maven
Gradle
Groovy
Griffon
Grails
Static Web
Flash
Kotlin
Empty Project
New...
Create from archetype
Add Archetype...
com.atlassian.maven.archetypes:bamboo-plugin-archetype
com.atlassian.maven.archetypes:confluence-plugin-archetype
com.atlassian.maven.archetypes:jira-plugin-archetype
com.rfc.maven.archetypes:jpa-maven-archetype
de.akquinet.jbosscc:jbosscc-seam-archetype
net.databinder:data-app
net.liftweb:lift-archetype-basic
net.liftweb:lift-archetype-blank
net.sf.maven-har:maven-archetype-har
net.sf.maven-sar:maven-archetype-sar
org.apache.camel.archetypes:camel-archetype-activemq
org.apache.camel.archetypes:camel-archetype-component
org.apache.camel.archetypes:camel-archetype-java
org.apache.camel.archetypes:camel-archetype-scala
org.apache.camel.archetypes:camel-archetype-spring
org.apache.camel.archetypes:camel-archetype-war
org.apache.cocoon:cocoon-22-archetype-block
org.apache.cocoon:cocoon-22-archetype-block-plain
org.apache.cocoon:cocoon-22-archetype-webapp
org.apache.maven.archetypes:maven-archetype-j2ee-simple
org.apache.maven.archetypes:maven-archetype-marmalade-mojo
org.apache.maven.archetypes:maven-archetype-mojo
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
org.apache.maven.archetypes:maven-archetype-site
org.apache.maven.archetypes:maven-archetype-site-simple
org.apache.maven.archetypes:maven-archetype-webapp
org.apache.maven.archetypes:softeu-archetype-jsf
org.apache.maven.archetypes:softeu-archetype-seam
org.apache.maven.archetypes:softeu-archetype-seam-simple
?
Cancel
Previous
Next

Project SDK: 1.8 (java version "1.8.0_91")

See Also

The following guide may also be helpful:

- Working a Getting Started guide with STS

Learn what you can do with Spring Boot

Spring Boot offers a fast way to build applications. It looks at your classpath and at beans you have configured, makes reasonable assumptions about what you're missing, and adds it. With Spring Boot you can focus more on business features and less on infrastructure.

For example:

- Got Spring MVC? There are several specific beans you almost always need, and Spring Boot adds them automatically. A Spring MVC app also needs a servlet container, so Spring Boot automatically configures embedded Tomcat.
- Got Jetty? If so, you probably do NOT want Tomcat, but instead embedded Jetty. Spring Boot handles that for you.
- Got Thymeleaf? There are a few beans that must always be added to your application context; Spring Boot adds them for you.

These are just a few examples of the automatic configuration Spring Boot provides. At the same time, Spring Boot doesn't get in your way. For example, if Thymeleaf is on your path, Spring Boot adds a `SpringTemplateEngine` to your application context automatically. But if you define your own `SpringTemplateEngine` with your own settings, then Spring Boot won't add one. This leaves you in control with little effort on your part.

Spring Boot doesn't generate code or make edits to your files. Instead, when you start up your application, Spring Boot dynamically wires up beans and settings and applies them to your application context.

Create a simple web application

Now you can create a web controller for a simple web application.

`src/main/java/hello/HelloController.java`

```
package hello;



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

import org.springframework.web.bind.annotation.RequestMapping;
```

```

@RestController

public class HelloController {



    @RequestMapping("/")

    public String index() {

        return "Greetings from Spring Boot!";

    }



}
```

The class is flagged as a `@RestController`, meaning it’s ready for use by Spring MVC to handle web requests. `@RequestMapping` maps `/` to the `index()` method. When invoked from a browser or using curl on the command line, the method returns pure text. That’s because `@RestController` combines `@Controller` and `@ResponseBody`, two annotations that results in web requests returning data rather than a view.

Create an Application class

Here you create an `Application` class with the components:

`src/main/java/hello/Application.java`

```
package hello;



import java.util.Arrays;



import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.Bean;



@SpringBootApplication

public class Application {



    public static void main(String[] args) {

        SpringApplication.run(Application.class, args);

    }



    @Bean

    public CommandLineRunner commandLineRunner(ApplicationContext ctx) {

        return args -> {



            System.out.println("Let's inspect the beans provided by Spring
Boot:");



            String[] beanNames = ctx.getBeanDefinitionNames();

            Arrays.sort(beanNames);

            for (String beanName : beanNames) {
                System.out.println(beanName);

            }



        };

    }



}
```

`@SpringBootApplication` is a convenience annotation that adds all of the following:

- `@Configuration` tags the class as a source of bean definitions for the application context.
- `@EnableAutoConfiguration` tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.
- Normally you would add `@EnableWebMvc` for a Spring MVC app, but Spring Boot adds it automatically when it sees **spring-webmvc** on the classpath. This flags the application as a web application and activates key behaviors such as setting up a `DispatcherServlet`.
- `@ComponentScan` tells Spring to look for other components, configurations, and services in the `hello` package, allowing it to find the controllers.

The `main()` method uses Spring Boot's `SpringApplication.run()` method to launch an application. Did you notice that there wasn't a single line of XML? No **web.xml** file either. This web application is 100% pure Java and you didn't have to deal with configuring any plumbing or infrastructure.

There is also a `CommandLineRunner` method marked as a `@Bean` and this runs on start up. It retrieves all the beans that were created either by your app or were automatically added thanks to Spring Boot. It sorts them and prints them out.

Run the application

To run the application, execute:

```
./gradlew build && java -jar build/libs/gs-spring-boot-0.1.0.jar
```

If you are using Maven, execute:

```
mvn package && java -jar target/gs-spring-boot-0.1.0.jar
```

You should see some output like this:

```
Let's inspect the beans provided by Spring Boot:

application

beanNameHandlerMapping

defaultServletHandlerMapping

dispatcherServlet

embeddedServletContainerCustomizerBeanPostProcessor

handlerExceptionResolver

helloController

httpRequestHandlerAdapter

messageSource

mvcContentNegotiationManager

mvcConversionService

mvcValidator

org.springframework.boot.autoconfigure.MessageSourceAutoConfiguration

org.springframework.boot.autoconfigure.PropertyPlaceholderAutoConfiguration

org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration

org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration$
DispatcherServletConfiguration
```

```
org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration$
EmbeddedTomcat

org.springframework.boot.autoconfigure.web.ServerPropertiesAutoConfiguration

org.springframework.boot.context.embedded.properties.ServerProperties

org.springframework.context.annotation.ConfigurationClassPostProcessor.enhancedConfig
urationProcessor

org.springframework.context.annotation.ConfigurationClassPostProcessor.importAwarePro
cessor

org.springframework.context.annotation.internalAutowiredAnnotationProcessor

org.springframework.context.annotation.internalCommonAnnotationProcessor

org.springframework.context.annotation.internalConfigurationAnnotationProcessor

org.springframework.context.annotation.internalRequiredAnnotationProcessor

org.springframework.web.servlet.config.annotation.DelegatingWebMvcConfiguration

propertySourcesBinder

propertySourcesPlaceholderConfigurer

requestMappingHandlerAdapter

requestMappingHandlerMapping

resourceHandlerMapping

simpleControllerHandlerAdapter

tomcatEmbeddedServletContainerFactory

viewControllerHandlerMapping
```

You can clearly see **org.springframework.boot.autoconfigure** beans. There is also a `tomcatEmbeddedServletContainerFactory`.

Check out the service.

```
$ curl localhost:8080

Greetings from Spring Boot!
```

Add Unit Tests

You will want to add a test for the endpoint you added, and Spring Test already provides some machinery for that, and it's easy to include in your project.

Add this to your build file's list of dependencies:

```
    testCompile("org.springframework.boot:spring-boot-starter-test")
```

If you are using Maven, add this to your list of dependencies:

```
        <dependency>

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

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

            <scope>test</scope>

        </dependency>
```

Now write a simple unit test that mocks the servlet request and response through your endpoint:

`src/test/java/hello/HelloControllerTest.java`

```
package hello;



import static org.hamcrest.Matchers.equalTo;

import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
```

```
import static
org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;



import org.junit.Test;

import org.junit.runner.RunWith;

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

import
org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;



@RunWith(SpringRunner.class)

@SpringBootTest

@AutoConfigureMockMvc

public class HelloControllerTest {



    @Autowired

    private MockMvc mvc;



    @Test
    public void getHello() throws Exception {

mvc.perform(MockMvcRequestBuilders.get("/").accept(MediaType.APPLICATION_JSON))

                .andExpect(status().isOk())

                .andExpect(content().string(equalTo("Greetings from Spring
Boot!")));

    }

}
```

The `MockMvc` comes from Spring Test and allows you, via a set of convenient builder classes, to send HTTP requests into the `DispatcherServlet` and make assertions about the result. Note the use of the `@AutoConfigureMockMvc` together with `@SpringBootTest` to inject a `MockMvc` instance. Having used `@SpringBootTest` we are asking for the whole application context to be created. An alternative would be to ask Spring Boot to create only the web layers of the context using the `@WebMvcTest`. Spring Boot automatically tries to locate the main application class of your application in either case, but you can override it, or narrow it down, if you want to build something different.

As well as mocking the HTTP request cycle we can also use Spring Boot to write a very simple full-stack integration test. For example, instead of (or as well as) the mock test above we could do this:

`src/test/java/hello/HelloControllerIT.java`

```
package hello;



import static org.hamcrest.Matchers.*;

import static org.junit.Assert.*;



import java.net.URL;
import org.junit.Before;

import org.junit.Test;

import org.junit.runner.RunWith;

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.boot.test.web.client.TestRestTemplate;

import org.springframework.boot.web.server.LocalServerPort;

import org.springframework.http.ResponseEntity;

import org.springframework.test.context.junit4.SpringRunner;


@RunWith(SpringRunner.class)

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)

public class HelloControllerIT {


    @LocalServerPort

    private int port;


    private URL base;


    @Autowired

    private TestRestTemplate template;
    @Before
    public void setUp() throws Exception {
        this.base = new URL("http://localhost:" + port + "/");
    }


    @Test
    public void getHello() throws Exception {
        ResponseEntity<String> response = template.getForEntity(base.toString(),
                String.class);
        assertThat(response.getBody(), equalTo("Greetings from Spring Boot!"));
    }
}
```

The embedded server is started up on a random port by virtue of the `webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT` and the actual port is discovered at runtime with the `@LocalServerPort`.

Add production-grade services

If you are building a web site for your business, you probably need to add some management services. Spring Boot provides several out of the box with its actuator module, such as health, audits, beans, and more.

Add this to your build file's list of dependencies:

```
    compile("org.springframework.boot:spring-boot-starter-actuator")
```

If you are using Maven, add this to your list of dependencies:

```
        <dependency>

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

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

        </dependency>
```

Then restart the app:

```
./gradlew build && java -jar build/libs/gs-spring-boot-0.1.0.jar
```

If you are using Maven, execute:

```
mvn package && java -jar target/gs-spring-boot-0.1.0.jar
```

You will see a new set of RESTful end points added to the application. These are management services provided by Spring Boot.

```
2018-03-17 15:42:20.088  ... : Mapped "{[/error],produces=[text/html]}" onto public
org.s...

2018-03-17 15:42:20.089  ... : Mapped "{[/error]}" onto public
org.springframework.http.R...

2018-03-17 15:42:20.121  ... : Mapped URL path [/webjars/**] onto handler of type
[class ...

2018-03-17 15:42:20.121  ... : Mapped URL path [/**] onto handler of type [class
org.spri...

2018-03-17 15:42:20.157  ... : Mapped URL path [/**/favicon.ico] onto handler of type
[cl...

2018-03-17 15:42:20.488  ... : Mapped
"{[/actuator/health],methods=[GET],produces=[application/vnd...

2018-03-17 15:42:20.490  ... : Mapped
"{[/actuator/info],methods=[GET],produces=[application/vnd.s...

2018-03-17 15:42:20.491  ... : Mapped
"{[/actuator],methods=[GET],produces=[application/vnd.spring...
```

They include: errors, actuator/health, actuator/info, actuator.

There is also a `/actuator/shutdown` endpoint, but it's only visible by default via JMX. To enable it as an HTTP endpoint, add `management.endpoints.shutdown.enabled=true` to your `application.properties` file.

It's easy to check the health of the app.

```
$ curl localhost:8080/actuator/health

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

You can try to invoke shutdown through curl.

```
$ curl -X POST localhost:8080/actuator/shutdown

{"timestamp":1401820343710,"error":"Method Not
Allowed","status":405,"message":"Request method 'POST' not supported"}
```

Because we didn't enable it, the request is blocked by the virtue of not existing.

For more details about each of these REST points and how you can tune their settings with an `application.properties` file, you can read detailed docs about the endpoints.

View Spring Boot's starters

You have seen some of Spring Boot's "starters". You can see them all here in source code.

JAR support and Groovy support

The last example showed how Spring Boot makes it easy to wire beans you may not be aware that you need. And it showed how to turn on convenient management services.

But Spring Boot does yet more. It supports not only traditional WAR file deployments, but also makes it easy to put together executable JARs thanks to Spring Boot's loader module. The various guides demonstrate this dual support through the `spring-boot-gradle-plugin` and `spring-boot-maven-plugin`.

On top of that, Spring Boot also has Groovy support, allowing you to build Spring MVC web apps with as little as a single file.

Create a new file called **app.groovy** and put the following code in it:

```
@RestController

class ThisWillActuallyRun {



    @RequestMapping("/")

    String home() {

        return "Hello World!"

    }



}
```

It doesn't matter where the file is. You can even fit an application that small inside a single tweet!

Next, install Spring Boot's CLI.

Run it as follows:

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

This assumes you shut down the previous application, to avoid a port collision.

From a different terminal window:

```
$ curl localhost:8080

Hello World!
```

Spring Boot does this by dynamically adding key annotations to your code and using Groovy Grape to pull down libraries needed to make the app run.

Summary

Congratulations! You built a simple web application with Spring Boot and learned how it can ramp up your development pace. You also turned on some handy production services. This is only a small sampling of what Spring Boot can do. Checkout Spring Boot's online docs if you want to dig deeper.

See Also

The following guides may also be helpful:

- Securing a Web Application
- Serving Web Content with Spring MVC

Want to write a new guide or contribute to an existing one? Check out our contribution guidelines.

All guides are released with an ASLv2 license for the code, and an Attribution, NoDerivatives creative commons license for the writing.