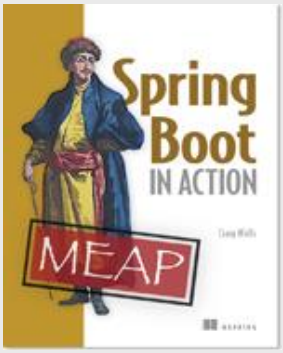

Initializing a Spring Boot project with Spring Initializr

By Craig Walls

The Spring Initializr is ultimately a web application that can generate a Spring Boot project structure for you. In this article, we take a look at it.

Sometimes the hardest part of any project is getting started. You need to set up a directory structure for various project artifacts, create a build file, and populate the build file with dependencies. While the Spring Boot CLI removes much of this setup work, if you favor a more traditional Java project structure, then you'll want to look at the Spring Initializr.

The Spring Initializr is ultimately a web application that can generate a Spring Boot project structure for you. It doesn't generate any application code, but it will give you a basic project structure and either a Maven or a Gradle build specification to build your code with. All you need to do is write the application code.

Spring Initializr can be used several ways, including:

- A web-based interface
- Via Spring ToolSuite
- Using the Spring Boot CLI

We'll look at how to use each of these interfaces to the Initializr, starting with the web-based interface.

USING SPRING INITIALIZR'S WEB INTERFACE

The most straightforward way to use the Spring Initializr is to point your web browser to http://start.spring.io. You should see a form similar to the one in figure 1.

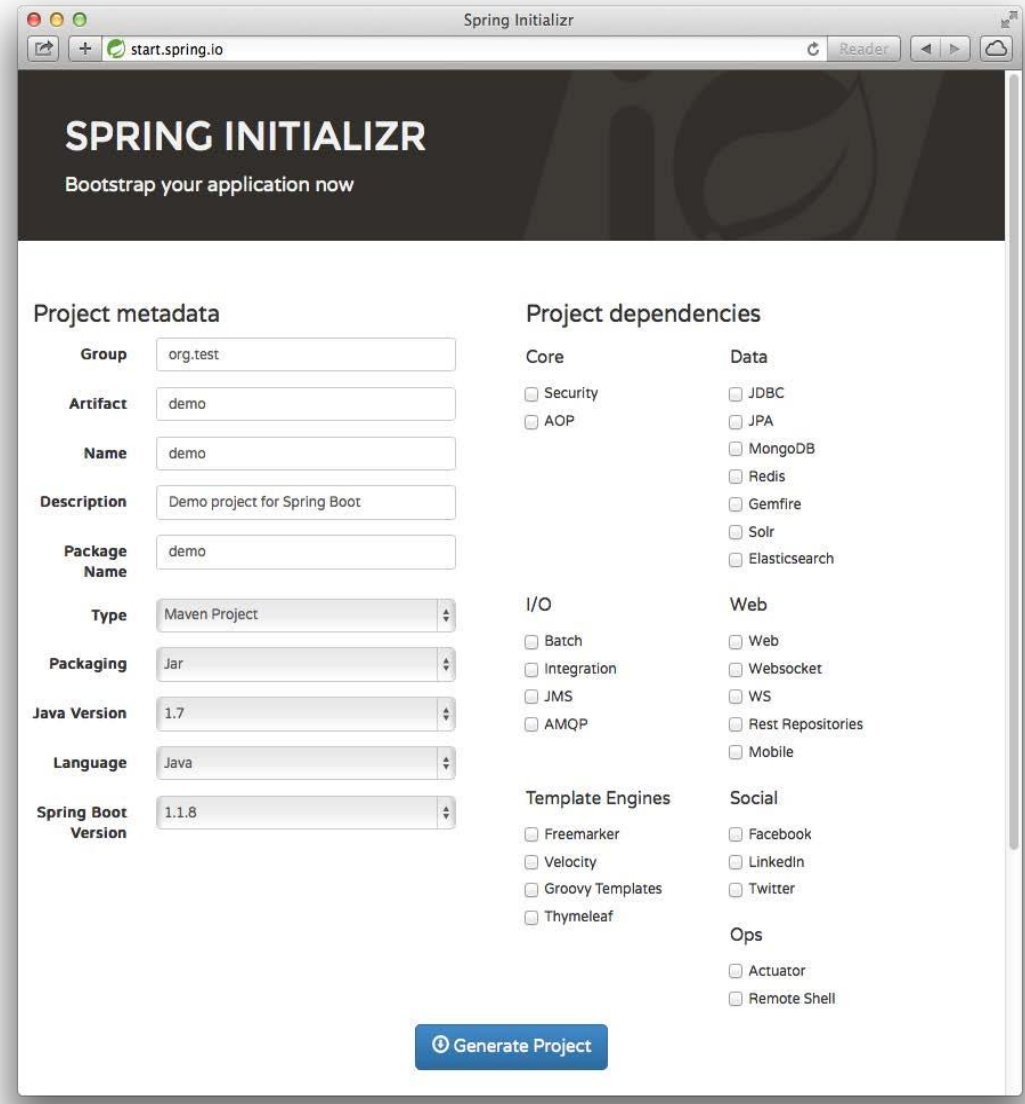

Figure 1 Spring Initializr is a web application that generates empty Spring projects as starting points for development.

On the left side of the Initializr form you're asked for some basic information about your project. This includes information such as the project name, the base package name, whether you want to build the project with Maven or Gradle, and which version of Java to build against. If you're familiar with Maven or Gradle, you probably recognize most of these fields as project metadata that goes into a Maven pom.xml file or a Gradle build.gradle file.

On the right side of the form you're asked to specify project dependencies. More accurately, this part of the form is asking you to specify the kind of functionality you intend to develop into your application. For example, if you're building a web application, you'll want to select the "Web" checkbox. Likewise, if your application will be backed by a relational database accessed with JPA, then you'll want to select the "JPA" checkbox.

If you've glanced at Appendix A, then you'll recognize that these checkboxes correspond to Spring Boot Starter dependencies. In fact, by selecting the "Web" checkbox, you're telling the Initializr to add Spring Boot's web starter as a dependency to the project's build file.

Once you've filled in the form and made your dependency selections, click the "Generate Project" button to have Spring Initializr generate a project for you. The project it generates will be presented to you as a zip file (whose name is determined by the value in the "artifact" field) that is downloaded by your browser. The contents of the zip file will vary slightly, depending on the choices you made before clicking "Generate Project." In any event, the zip file contains a bare-bones project to get you started with developing an application with Spring Boot.

For example, suppose that you were to specify the following to Spring Initializr:

- Artifact : myapp
- Package Name : myapp
- Type : Gradle Project
- Dependencies : Web and JPA

After clicking "Generate Project," you'll be given a zip file named myapp.zip. After unzipping it, you'll have a project structure similar to what is shown in figure 2.

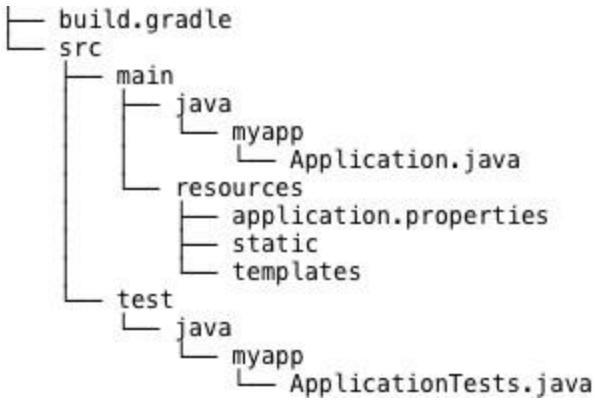

Figure 2 Initializr-created projects provide a minimal foundation on which to build Spring Boot applications.

As you can see, there's very little code in this project. Aside from a couple of empty directories, it also includes:

- build.gradle : A Gradle build specification. Had you chosen a Maven project, this would be replaced with pom.xml.
- Application.java : A class with a `main()` method to bootstrap the application.
- ApplicationTests.java : An empty JUnit test class instrumented to load a Spring
- application context using Spring Boot auto-configuration.
- application.properties : An empty properties file for you to add configuration properties as you see fit.

Even the empty directories have significance in a Spring Boot application. The static directory is where you can put any static content (JavaScript, stylesheets, images, etc) to be served from the web application. And, as you'll see later, you can put templates that render model data in the templates directory.

You'll probably import the Initializr-created project into your IDE of choice. But if Spring ToolSuite is your IDE of choice, then you can create the project directly in the IDE. Let's have a look at Spring ToolSuite's support for creating Spring Boot projects.

CREATING SPRING BOOT PROJECTS IN SPRING TOOLSUITE

Spring ToolSuite has long been a fantastic IDE for developing Spring applications. Now it also integrates with the Spring Initializr making it a great way to get started with Spring Boot.

NOTE Spring ToolSuite is a distribution of the Eclipse IDE that is outfitted with several features to aid with Spring development. You can download Spring ToolSuite from http://spring.io/tools/sts.

To create a new Spring Boot application in Spring ToolSuite, select the NewSpring Starter Project menu item from the File menu. When you do, Spring ToolSuite will present you with a dialog similar to the one shown in figure 3.

Figure 3 Spring ToolSuite integrates with Spring Initializr to create and directly import Spring Boot projects

into the IDE.

As you can see, this dialog asks for the same information as the web-based Spring Initializr. In fact, the data you provide here will be fed to Spring Initializr to create a project zip file, just as with the web-based form.

If you'd like to specify where in the filesystem to create the project or whether to add it to a specific working set within the IDE, click the "Next >" button. You'll be presented with a second dialog like the one shown in figure 4.

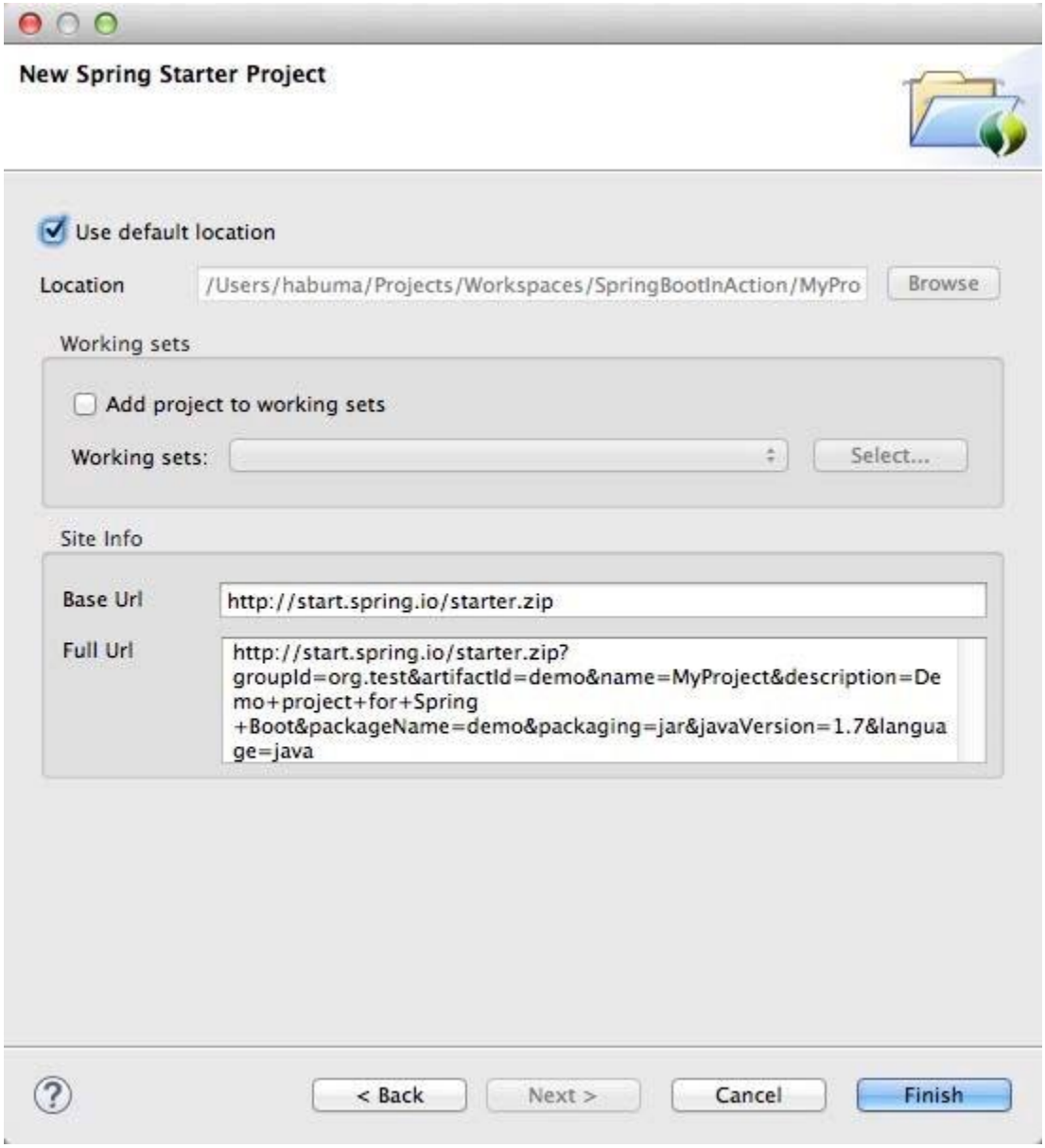

Figure 4 The second page of the Spring Starter Project dialog offers you a chance to specify where the project is created.

The "Location" field specifies where the project will reside on the filesystem. If you take advantage of Eclipse's working sets to organize your projects, you can have the project added to a specific working set by checking the "Add project to working sets" checkbox and selecting a working set.

Clicking the "Finish" button kicks off the project generation and import process. It's important to understand that Spring ToolSuite's Spring Starter Project dialog delegates to the Spring Initializr at http://start.spring.io to produce the project. Therefore, you must be connected to the internet in order for it to work.

Once the project has been imported into your workspace, you're ready to start developing your application. As you develop the application, you'll find that Spring ToolSuite has a few more Spring Boot specific tricks up its sleeves. For instance, you can run your application with an embedded server by selecting Run AsSpring Boot Application from the Run menu.

USING THE INITIALIZR FROM THE SPRING BOOT CLI

The Spring Boot CLI is a great way to develop Spring application by just writing code. However, the Spring Boot CLI also has a few commands that can help you use the Initializr to kick-start development on a more traditional Java project.

Starting with version 1.2.0, the Spring Boot CLI includes an `init` command that acts as a client interface to the Initializr. The simplest use of the `init` command is to create a baseline Spring Boot project:

```
$ spring init
```

After contacting the Initializr web application, the `init` command will conclude by downloading a demo.zip file. If you unzip this project, you'll find a typical project structure with a Maven `pom.xml` build specification. The Maven build specification is minimal, with only baseline starter dependencies for Spring Boot and testing. You'll probably want a little more than that, though.

Let's say you want to start out by building a web application that uses JPA for data persistence and is secured with Spring Security. You can specify those initial dependencies with either `--dependencies -d` or :

```
$ spring init -dweb,jpa,security
```

This will give you a demo.zip containing the same project structure as before, but with Spring Boot's web, JPA, and security starters expressed as dependencies in `pom.xml`. Note that it's important to not type a space between `-d` and the dependencies. Failing to do so will result in the ZIP file being downloaded with the name `web,jpa,security`.

Now let's say that you'd rather build this project with Gradle. No problem. Just specify Gradle as the build type with the `--build` parameter:

```
$ spring init -dweb,jpa,security --build gradle
```

By default, the build specification for both Maven and Gradle builds will produce an executable JAR file. If you'd rather produce a WAR file, you can specify so with the `--packaging -p` or parameter:

```
$ spring init -dweb,jpa,security --build gradle -p war
```

So far, the ways we've used the `init` command has resulted in a ZIP file being downloaded. If you'd like for the CLI to crack open that ZIP file for you, you can specify a directory for the project to be extracted to:

```
$ spring init -dweb,jpa,security --build gradle -p war myapp/
```

The slash (/) at the end indicates that you want the project to be extracted to the `myapp` directory. If you leave the slash off, then `init` will download the ZIP file and name it `myapp`.

Optionally, if you want the CLI to extract the generated project into the current directory, you can use either the `--extract` paramter or the `-x` parameter:

```
$ spring init -dweb,jpa,security --build gradle -p jar -x
```

The `init` command has several other parameters, including parameters for building a Groovy-based project, specifying the Java version to compile with, and selecting a version of Spring Boot to build against. You can discover all of the parameters by using the `help` command:

```
$ spring help init
```

You can also find what choices are available for those parameters by using the `--list -l` or parameter with the `init` command:

```
$ spring init -l
```

Whether you use Initializr's web-based interface, create your projects from Spring ToolSuite, or leverage the Spring Boot CLI to initialize a project, projects created using the Spring Boot Initializr have a familiar project layout, not unlike other Java projects you may have developed before.