Gradle Guides (https://guides.gradle.org)

Building Spring Boot 2 Applications with Gradle

Contents

This guide shows how to build a new Gradle project for Spring Boot 2.0. First we show some noteworthy features of Spring Boot and its Gradle plugin. Next we'll setup the Gradle project, apply the Spring Boot plugin, use the Gradle BOM support to define the dependencies and create an example project to show an integration with Gradle Build Scans (https://gradle.com/build-scans).

Noteworthy Spring Boot 2 features

As Spring Boot uses the Spring Framework 5.x, the minimum Java version was bumped to 8 with support for Java 9. With this release Spring also includes support for Kotlin 1.2.x.

In addition to that, it now fully supports Reactive Spring with which you are able to build reactive applications. The whole autoconfiguration mechanism provided by Spring Boot has been enriched as well with several new reactive versions of for example MongoDB, Redis and others.

The Spring Boot Gradle plugin went through a major overhaul with the following improvements:

- To build executable jars and wars, the `bootRepackage` task has been replaced with `bootJar` and `bootWar` respectively.
- The plugin itself does not automatically apply the Spring Dependency Management plugin anymore. Instead it does react to the Spring Dependency Management plugin being applied and configured with the `spring-boot-dependencies` BOM (bill of materials. We will go into more detail about the BOM support later in this post.)

What you'll need

- About 13 minutes
- A text editor or IDE
- The Java Development Kit (JDK), version 1.8 or higher
- A Gradle distribution (https://gradle.org/install), version 4.6 or better

Initializing the Gradle project

First we need to initialize the Gradle project. For that we use Gradle's `init` task which creates a template project with an empty build file. The generated project includes the Gradle wrapper out of the box such that you can easily share the project with users that do not have Gradle locally installed. It also adds the default source directories, test dependencies and JCenter as default dependency repository. Please have a look at its documentation (https://docs.gradle.org/current/userguide/build_init_plugin.html) to read more about the `init` task.

First we need to create the sample project folder in our home directory and initialize the project:

```
$ mkdir ~/gradle-spring-boot-project
$ cd ~/gradle-spring-boot-project
$ gradle init  --type java-application
```

The generated project has the following structure:

```
gradle-spring-boot-project
├── build.gradle
├── gradle
│    └── wrapper
│        ├── gradle-wrapper.jar
│        └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    ├── main
    │   └── java
    │       └── App.java
    └── test
        └── java
            └── AppTest.java
```

Next we need to apply the Spring Boot plugin and define the dependencies.

Applying the Spring Boot plugin and configuring the dependencies

Spring provides a standalone Spring Boot Gradle plugin which adds some tasks and configurations to ease the work with Spring Boot based projects. To start off we first need to apply the plugin. For that open the `build.gradle` file and adapt the `plugin` block such that it looks like the following snippet:

build.gradle

GROOVY

```groovy
plugins {
    id 'java'
    id 'com.gradle.build-scan' version '2.0.2'
    id 'org.springframework.boot' version '2.0.5.RELEASE'
    id 'io.spring.dependency-management' version '1.0.7.RELEASE'
}
```

We also apply Gradle's `java` and build scan plugin, with which we are able to create a build scan. We will cover what build scans are later in this guide.

Next we need to add the dependencies needed to compile and run our example as we are not using Spring's dependency management plugin. For that we use the Gradle's BOM support and load the Spring Boot BOM file to be able to resolve all required dependencies with the proper version.

If you'd like to read more about Gradle's BOM support please visit this page (https://docs.gradle.org/current/userguide/managing_transitive_dependencies.html#sec:bom_import) .

To define the dependencies adapt the `dependencies` block as shown below. This snippet will add the Spring Boot BOM file as first dependency with the specified Spring Boot version. The other dependencies do not need to have a specific version as these are implicitly defined in the BOM file.

build.gradle

GROOVY

```groovy
dependencies {
    implementation 'org.springframework.boot:spring-boot-dependencies:2.0.5.RELEASE'
    implementation 'org.springframework.boot:spring-boot-starter-web'

    testImplementation 'org.springframework.boot:spring-boot-starter-test'

    components {
        withModule('org.springframework:spring-beans') {
            allVariants {
                withDependencyConstraints {
                    // Need to patch constraints because snakeyaml is an optional dependency
                    it.findAll { it.name == 'snakeyaml' }.each { it.version { strictly '1.19' } }
                }
            }
        }
    }
}
```

To comply with the Spring Boot BOM the `components` block is needed to strictly use the `snakeyaml` dependency with version `1.19` as the `spring-beans` dependency has version `1.20` as transitive dependency.

If using a Gradle version before 5.0, we need to enable this by adding the following line to the `settings.gradle` file in the root of the project:

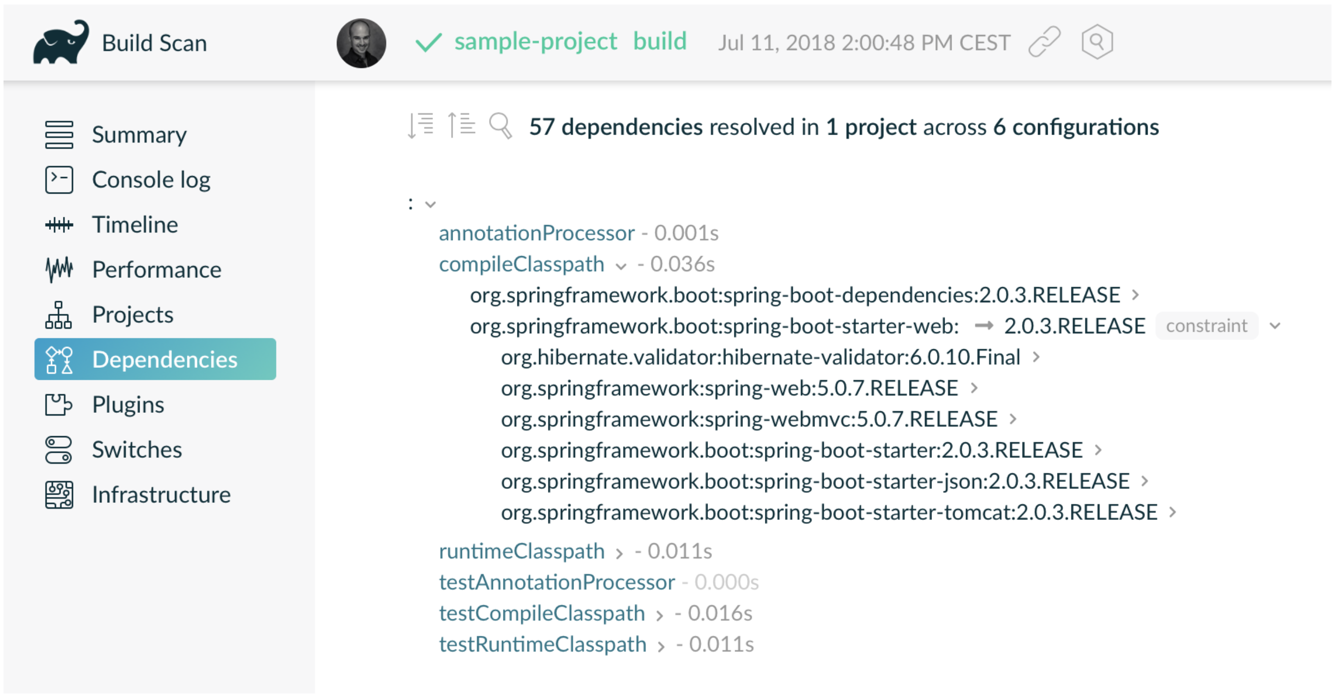
settings.gradle

GROOVY

```groovy
enableFeaturePreview('IMPROVED_POM_SUPPORT')
```

If you would like to explore the versions of the dependencies used, which transitive dependencies are included or see where you have conflicts you can find this information in a build scan (https://gradle.com/build-scans).

The following screenshot shows an example of the dependencies section of the build scan:



57 dependencies resolved in 1 project across 6 configurations

```
:
  annotationProcessor - 0.001s
  compileClasspath - 0.036s
    org.springframework.boot:spring-boot-dependencies:2.0.3.RELEASE
    org.springframework.boot:spring-boot-starter-web: → 2.0.3.RELEASE constraint
      org.hibernate.validator:hibernate-validator:6.0.10.Final
      org.springframework:spring-web:5.0.7.RELEASE
      org.springframework:spring-webmvc:5.0.7.RELEASE
      org.springframework.boot:spring-boot-starter:2.0.3.RELEASE
      org.springframework.boot:spring-boot-starter-json:2.0.3.RELEASE
      org.springframework.boot:spring-boot-starter-tomcat:2.0.3.RELEASE
  runtimeClasspath - 0.011s
  testAnnotationProcessor - 0.000s
  testCompileClasspath - 0.016s
  testRuntimeClasspath - 0.011s
```

You can also explore the above build scan here (https://gradle.com/s/ofnoymriygxtw).

To enable this functionality you need to add the following block to your `build.gradle` file. This will always publish a build scan after each build and always accept the license agreement (https://gradle.com/legal/terms-of-service).

build.gradle

GROOVY

```groovy
buildScan {

    // always accept the terms of service
    termsOfServiceUrl = 'https://gradle.com/terms-of-service'
    termsOfServiceAgree = 'yes'

    // always publish a build scan
    publishAlways()
}
```

Creating a "Hello Gradle" sample application

For the example application we create a simple "Hello Gradle" application. First we need to move the `App` and `AppTest` classes to a `hello` package to facilitate Spring's component scan. For that create the `src/main/java/hello` and `src/test/java/hello` directories, move the respective classes to the folders.

Next adapt the `App` class located in the `src/main/java/hello` folder and replace its content with the following:

App.java

```java
package hello;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class App {

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

}
```

HelloGradleController.java

```java
package hello;

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

@RestController("/")
public class HelloGradleController {

    @GetMapping
    public String helloGradle() {
        return "Hello Gradle!";
    }

}
```

In the above snippets we create a new Spring Boot application and a `HelloGradleController` which returns `Hello Gradle!` when a `GET` request is processed on the root path of the application.

To test this functionality we need to create an integration test. For that adapt the `AppTest` class located in the `src/test/java/hello` folder and replace its content with the following:

AppTest.java

JAVA
```java
package hello;

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.test.context.junit4.SpringRunner;
import org.springframework.test.web.servlet.MockMvc;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@RunWith(SpringRunner.class)
@SpringBootTest(classes = App.class)
@AutoConfigureMockMvc
public class AppTest {

    @Autowired
    private MockMvc mvc;

    @Test
    public void helloGradle() throws Exception {
        mvc.perform(get("/"))
            .andExpect(status().isOk())
            .andExpect(content().string("Hello Gradle!"));
    }

}
```

The `helloGradle` test method spins up the `App` Spring Boot application and asserts the returned content when doing a `GET` request on the root path.

As a last step we need to define the main class name for the Spring Boot jar file. For that we need to define the `mainClassName` attribute on the `bootJar` configuration closure. Add the following snippet to your `build.gradle` and then we are ready to run the Spring Boot application.

build.gradle

GROOVY
```groovy
bootJar {
    mainClassName = 'hello.App'
}
```

Building and running the Spring Boot application

To build the executable jar you can execute the following command:

```
./gradlew bootJar
```

The executable jar is located in the `build/libs` directory and you can run it by executing the following command:

```
java -jar build/libs/gradle-spring-boot-project.jar
```

Another way to run the application is by executing the following Gradle command:

```
./gradlew bootRun
```

This command will run the Spring Boot application on the default port `8080` directly. After a successful startup you can open your browser and access http://localhost:8080 and you should see the `Hello Gradle!` message in the browser window.

Migrate from an existing Spring Boot 1.5 project

If you already have an existing 1.5.x Spring Boot project and want to migrate to the newer 2.x version you can follow this guide
(https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.0-Migration-Guide#spring-boot-gradle-plugin)
. Please read the upgrade notes carefully to successfully upgrade to the newest Spring Boot Gradle plugin.

Next Steps

Now that you know the basics of the new Spring Boot Gradle plugin, you can read its documentation (https://docs.spring.io/spring-boot/docs/2.0.3.RELEASE/gradle-plugin/reference/html) for further details.

Please also have a look at Gradle Enterprise (https://gradle.com) if you are interested in build scans and even more metrics and tools for your builds on premise.

Help improve this guide

Have feedback or a question? Found a typo? Like all Gradle guides, help is just a GitHub issue away. Please add an issue or pull request to gradle-guides/building-spring-boot-2-projects-with-gradle (https://github.com/gradle-guides/building-spring-boot-2-projects-with-gradle/) and we'll get back to you.

Docs
User Manual (https://docs.gradle.org/current/userguide/userguide.html)
DSL Reference (https://docs.gradle.org/current/dsl/)
Release Notes (https://docs.gradle.org/current/release-notes.html)
Javadoc (https://docs.gradle.org/current/javadoc/)

News
Blog (https://blog.gradle.org/)
Newsletter (https://newsletter.gradle.com/)
Twitter (https://twitter.com/gradle)

Products
Build Scans (https://gradle.com/build-scans)
Build Cache (https://gradle.com/build-cache)
Enterprise Docs (https://gradle.com/enterprise/resources)

Get Help
Forums (https://discuss.gradle.org/c/help-discuss)
GitHub (https://github.com/gradle/)
Training (https://gradle.org/training/)
Services (https://gradle.org/services/)

Stay `UP-TO-DATE` on new features and news

By entering your email, you agree to our Terms (https://gradle.org/terms/) and Privacy Policy (https://gradle.org/privacy/), including receipt of emails. You can unsubscribe at any time.

Email address Subscribe

Careers (https://gradle.com/careers) | Privacy (https://gradle.org/privacy) | Terms of Service (https://gradle.org/terms) | Contact (https://gradle.org/contact/)