



Red Hat Fuse 7.13

Getting Started with Fuse on Spring Boot

Get started quickly with Red Hat Fuse on Spring Boot

Red Hat Fuse 7.13 Getting Started with Fuse on Spring Boot

Get started quickly with Red Hat Fuse on Spring Boot

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Get started with Fuse on Spring Boot.

Table of Contents

PREFACE	3
MAKING OPEN SOURCE MORE INCLUSIVE	4
CHAPTER 1. GETTING STARTED WITH FUSE ON SPRING BOOT	5
1.1. ABOUT FUSE ON SPRING BOOT	5
1.2. GENERATING YOUR BOOSTER PROJECT	5
1.3. BUILDING YOUR BOOSTER PROJECT	6
CHAPTER 2. SETTING UP MAVEN LOCALLY	9
2.1. PREPARING TO SET UP MAVEN	9
2.2. ADDING RED HAT REPOSITORIES TO MAVEN	9
2.3. USING LOCAL MAVEN REPOSITORIES	11
2.4. SETTING MAVEN MIRROR USING ENVIRONMENTAL VARIABLES OR SYSTEM PROPERTIES	11
2.4.1. About Maven mirror	11
2.4.2. Adding Maven mirror to settings.xml	12
2.4.3. Setting Maven mirror using environmental variable or system property	12
2.4.4. Using Maven options to specify Maven mirror url	12
2.5. ABOUT MAVEN ARTIFACTS AND COORDINATES	12

PREFACE

To get started with Fuse, you need to download and install the files for your Spring Boot container. The information and instructions here guide you in installing, developing, and building your first Fuse application.

- [Chapter 1, *Getting started with Fuse on Spring Boot*](#)
- [Chapter 2, *Setting up Maven locally*](#)

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our [CTO Chris Wright's message](#).

CHAPTER 1. GETTING STARTED WITH FUSE ON SPRING BOOT

To develop Fuse applications on Spring Boot, get started by generating and building a Fuse sample booster project that runs on Spring Boot. The following topics provide details:

- [Section 1.1, “About Fuse on Spring Boot”](#)
- [Section 1.2, “Generating your booster project”](#)
- [Section 1.3, “Building your booster project”](#)

1.1. ABOUT FUSE ON SPRING BOOT

[Spring Boot](#) is an evolution of the well-known Spring container. A distinctive quality of the Spring Boot container is that container functionality is divided up into small chunks, which can be deployed independently. This quality enables you to deploy a container with a small footprint, specialized for a particular kind of service, and this ability happens to be exactly what you need to fit the paradigm of a *microservices architecture*.

Distinctive features of this container technology are:

- Particularly suited to running on a scalable cloud platform (Kubernetes and OpenShift).
- Small footprint (ideal for microservices architecture).
- Optimized for *convention over configuration*.
- No application server required. You can run a Spring Boot application Jar directly in a JVM.

1.2. GENERATING YOUR BOOSTER PROJECT

Fuse booster projects exist to help developers get started with running standalone applications. The instructions provided here guide you through generating one of those booster projects, the Circuit Breaker booster. This exercise demonstrates useful components of the Fuse on Spring Boot.

The [Netflix/Hystrix](#) circuit breaker enables distributed applications to handle interruptions to network connectivity and temporary unavailability of backend services. The basic idea of the circuit breaker pattern is that the loss of a dependent service is detected automatically and an alternative behavior can be programmed, in case the backend service is temporarily unavailable.

The Fuse circuit breaker booster consists of two related services:

- A **name** service, the backend service that returns a name to greet.
- A **greetings** service, the frontend service that invokes the **name** service to get a name and then returns the string, **Hello, NAME**.

In this booster demonstration, the Hystrix circuit breaker is inserted between the **greetings** service and the **name** service. If the backend **name** service becomes unavailable, the **greetings** service can fall back to an alternative behavior and respond to the client immediately, instead of being blocked while it waits for the **name** service to restart.

Prerequisites

- You must have access to the [Red Hat Developer Platform](#).
- You must have a supported version of the Java Developer Kit (JDK). See the [Supported Configurations](#) page for details.
- You must have installed and configured [Apache Maven 3.3.x](#) or later as described in [Setting up Maven locally](#)).

Procedure

1. Navigate to <https://developers.redhat.com/launch>.
2. Click **START**.
The launcher wizard prompts you to log in to your Red Hat account.
3. Click the **Log in or register** button and then log in.
4. On the **Launcher** page, click the **Deploy an Example Application** button.
5. On the **Create Example Application** page, type the name, **fuse-circuit-breaker**, in the **Create Example Application as** field.
6. Click **Select an Example**.
7. In the **Example** dialog, select the **Circuit Breaker** option. A **Select a Runtime** dropdown menu appears.
 - a. From the **Select a Runtime** dropdown, select **Fuse**.
 - b. From the version dropdown menu, select **7.13 (Red Hat Fuse)** (do not select the **2.21.2 (Community)** version).
 - c. Click **Save**.
8. On the **Create Example Application** page, click **Download**.
9. When you see the **Your Application is Ready** dialog, click **Download.zip**. Your browser downloads the generated booster project (packaged as a ZIP file).
10. Use an archive utility to extract the generated project to a convenient location on your local file system.

1.3. BUILDING YOUR BOOSTER PROJECT

These instructions guide you through building the Circuit Breaker booster with Fuse on Spring Boot.

Prerequisites

- You must have generated and downloaded your booster project via the [Red Hat Developer Portal](#).
- You must have a supported version of the Java Developer Kit (JDK). See the [Supported Configurations](#) page for details.
- You must have installed and configured [Apache Maven 3.3.x](#) or later as described in [Setting up Maven locally](#)).

Procedure

1. Open a shell prompt and build the project from the command line, using Maven:

```
cd fuse-circuit-breaker
```

```
mvn clean package
```

After Maven builds the project, it displays a **Build Success** message.

2. Open a new shell prompt and start the name service, as follows:

```
cd name-service
```

```
mvn spring-boot:run -DskipTests -Dspring-boot.run.arguments="--server.port=8081"
```

As Spring Boot starts up, you should see output similar to the following:

```
...
2019-05-06 20:19:59.401 INFO 9553 --- [      main] o.a.camel.spring.SpringCamelContext
: Route: route1 started and consuming from: servlet:/name?httpMethodRestrict=GET
2019-05-06 20:19:59.402 INFO 9553 --- [      main] o.a.camel.spring.SpringCamelContext
: Total 1 routes, of which 1 are started
2019-05-06 20:19:59.403 INFO 9553 --- [      main] o.a.camel.spring.SpringCamelContext
: Apache Camel 2.21.0.fuse-730078-redhat-00001 (CamelContext: camel-1) started in 0.287
seconds
2019-05-06 20:19:59.406 INFO 9553 --- [      main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-05-06 20:19:59.473 INFO 9553 --- [      main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8081 (http)
2019-05-06 20:19:59.479 INFO 9553 --- [      main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 5.485 seconds (JVM
running for 9.841)
```

3. Open a new shell prompt and start the greetings service, as follows:

```
cd greetings-service
```

```
mvn spring-boot:run -DskipTests
```

As Spring Boot starts up, you should see output similar to the following:

```
...
2019-05-06 20:22:19.051 INFO 9729 --- [      main] o.a.c.c.s.CamelHttpTransportServlet
: Initialized CamelHttpTransportServlet[name=CamelServlet, contextPath=]
2019-05-06 20:22:19.115 INFO 9729 --- [      main]
b.c.e.u.UndertowEmbeddedServletContainer : Undertow started on port(s) 8080 (http)
2019-05-06 20:22:19.123 INFO 9729 --- [      main]
com.redhat.fuse.boosters.cb.Application  : Started Application in 7.68 seconds (JVM
running for 12.66)
```

The greetings service exposes a REST endpoint at the <http://localhost:8080/camel/greetings> URL.

- Invoke the REST endpoint by either opening the URL in a web browser or by opening another shell prompt and typing the following **curl** command:

```
curl http://localhost:8080/camel/greetings
```

Here is the response:

```
{"greetings":"Hello, Jacopo"}
```

- To demonstrate the circuit breaker functionality provided by Camel Hystrix, kill the backend name service by typing **Ctrl-C** in the shell prompt window where the name service is running. Now that the name service is unavailable, the circuit breaker kicks in to prevent the greetings service from hanging when it is invoked.
- Invoke the greetings REST endpoint by either opening <http://localhost:8080/camel/greetings> in a web browser or by typing the following **curl** command in another shell prompt window:

```
curl http://localhost:8080/camel/greetings
```

Here is the response:

```
{"greetings":"Hello, default fallback"}
```

In the window where the greetings service is running, the log shows the following sequence of messages:

```
2019-05-06 20:24:16.952 INFO 9729 --- [-CamelHystrix-2] route2           : Try
to call name Service
2019-05-06 20:24:16.956 INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector    : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.956 INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector    : Retrying request
2019-05-06 20:24:16.957 INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector    : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.957 INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector    : Retrying request
2019-05-06 20:24:16.957 INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector    : I/O exception (java.net.ConnectException) caught
when processing request: Connection refused (Connection refused)
2019-05-06 20:24:16.957 INFO 9729 --- [-CamelHystrix-2]
o.a.c.httpclient.HttpMethodDirector    : Retrying request
2019-05-06 20:24:16.964 INFO 9729 --- [-CamelHystrix-2] route2           : We
are falling back!!!!
```

- For more information about this example, open the **Circuit Breaker - Red Hat Fuse** page at <http://localhost:8080/> (while the **greetings-service** is running). This page includes a link to the Hystrix dashboard that monitors the state of the circuit breaker.

CHAPTER 2. SETTING UP MAVEN LOCALLY

Typical Fuse application development uses Maven to build and manage projects.

The following topics describe how to set up Maven locally:

- [Section 2.1, “Preparing to set up Maven”](#)
- [Section 2.2, “Adding Red Hat repositories to Maven”](#)
- [Section 2.3, “Using local Maven repositories”](#)
- [Section 2.4, “Setting Maven mirror using environmental variables or system properties”](#)
- [Section 2.5, “About Maven artifacts and coordinates”](#)

2.1. PREPARING TO SET UP MAVEN

Maven is a free, open source, build tool from Apache. Typically, you use Maven to build Fuse applications.

Procedure

1. Download the latest version of Maven from the [Maven download page](#).
2. Ensure that your system is connected to the Internet.
While building a project, the default behavior is that Maven searches external repositories and downloads the required artifacts. Maven looks for repositories that are accessible over the Internet.

You can change this behavior so that Maven searches only repositories that are on a local network. That is, Maven can run in an offline mode. In offline mode, Maven looks for artifacts in its local repository. See [Section 2.3, “Using local Maven repositories”](#).

2.2. ADDING RED HAT REPOSITORIES TO MAVEN

To access artifacts that are in Red Hat Maven repositories, you need to add those repositories to Maven’s **settings.xml** file. Maven looks for the **settings.xml** file in the **.m2** directory of the user’s home directory. If there is not a user specified **settings.xml** file, Maven uses the system-level **settings.xml** file at **M2_HOME/conf/settings.xml**.

Prerequisite

You know the location of the **settings.xml** file in which you want to add the Red Hat repositories.

Procedure

In the **settings.xml** file, add **repository** elements for the Red Hat repositories as shown in this example:

```
<?xml version="1.0"?>
<settings>

  <profiles>
    <profile>
      <id>extra-repos</id>
```

```
<activation>
  <activeByDefault>true</activeByDefault>
</activation>
<repositories>
  <repository>
    <id>redhat-ga-repository</id>
    <url>https://maven.repository.redhat.com/ga</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>redhat-ea-repository</id>
    <url>https://maven.repository.redhat.com/earlyaccess/all</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>jboss-public</id>
    <name>JBoss Public Repository Group</name>
    <url>https://repository.jboss.org/nexus/content/groups/public/</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>redhat-ga-repository</id>
    <url>https://maven.repository.redhat.com/ga</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </pluginRepository>
  <pluginRepository>
    <id>redhat-ea-repository</id>
    <url>https://maven.repository.redhat.com/earlyaccess/all</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </pluginRepository>
  <pluginRepository>
    <id>jboss-public</id>
    <name>JBoss Public Repository Group</name>
    <url>https://repository.jboss.org/nexus/content/groups/public</url>
  </pluginRepository>
</pluginRepositories>
```

```

    </pluginRepositories>
  </profile>
</profiles>

<activeProfiles>
  <activeProfile>extra-repos</activeProfile>
</activeProfiles>

</settings>

```

2.3. USING LOCAL MAVEN REPOSITORIES

If you are running a container without an Internet connection, and you need to deploy an application that has dependencies that are not available offline, you can use the Maven dependency plug-in to download the application's dependencies into a Maven offline repository. You can then distribute this customized Maven offline repository to machines that do not have an Internet connection.

Procedure

1. In the project directory that contains the **pom.xml** file, download a repository for a Maven project by running a command such as the following:

```

mvn org.apache.maven.plugins:maven-dependency-plugin:3.1.0:go-offline -
Dmaven.repo.local=/tmp/my-project

```

In this example, Maven dependencies and plug-ins that are required to build the project are downloaded to the **/tmp/my-project** directory.

2. Distribute this customized Maven offline repository internally to any machines that do not have an Internet connection.

2.4. SETTING MAVEN MIRROR USING ENVIRONMENTAL VARIABLES OR SYSTEM PROPERTIES

When running the applications you need access to the artifacts that are in the Red Hat Maven repositories. These repositories are added to Maven's **settings.xml** file. Maven checks the following locations for **settings.xml** file:

- looks for the specified url
- if not found looks for **\${user.home}/.m2/settings.xml**
- if not found looks for **\${maven.home}/conf/settings.xml**
- if not found looks for **\${M2_HOME}/conf/settings.xml**
- if no location is found, empty **org.apache.maven.settings.Settings** instance is created.

2.4.1. About Maven mirror

Maven uses a set of remote repositories to access the artifacts, which are currently not available in local repository. The list of repositories almost always contains Maven Central repository, but for Red Hat Fuse, it also contains Maven Red Hat repositories. In some cases where it is not possible or allowed to

access different remote repositories, you can use a mechanism of Maven mirrors. A mirror replaces a particular repository URL with a different one, so all HTTP traffic when remote artifacts are being searched for can be directed to a single URL.

2.4.2. Adding Maven mirror to settings.xml

To set the Maven mirror, add the following section to Maven's **settings.xml**:

```
<mirror>
  <id>all</id>
  <mirrorOf>*</mirrorOf>
  <url>http://host:port/path</url>
</mirror>
```

No mirror is used if the above section is not found in the **settings.xml** file. To specify a global mirror without providing the XML configuration, you can use either system property or environmental variables.

2.4.3. Setting Maven mirror using environmental variable or system property

To set the Maven mirror using either environmental variable or system property, you can add:

- Environmental variable called **MAVEN_MIRROR_URL** to **bin/setenv** file
- System property called **mavenMirrorUrl** to **etc/system.properties** file

2.4.4. Using Maven options to specify Maven mirror url

To use an alternate Maven mirror url, other than the one specified by environmental variables or system property, use the following maven options when running the application:

- **-DmavenMirrorUrl=mirrorId::mirrorUrl**
for example, **-DmavenMirrorUrl=my-mirror::http://mirror.net/repository**
- **-DmavenMirrorUrl=mirrorUrl**
for example, **-DmavenMirrorUrl=http://mirror.net/repository**. In this example, the `<id>` of the `<mirror>` is just a mirror.

2.5. ABOUT MAVEN ARTIFACTS AND COORDINATES

In the Maven build system, the basic building block is an *artifact*. After a build, the output of an artifact is typically an archive, such as a JAR or WAR file.

A key aspect of Maven is the ability to locate artifacts and manage the dependencies between them. A *Maven coordinate* is a set of values that identifies the location of a particular artifact. A basic coordinate has three values in the following form:

groupId:artifactId:version

Sometimes Maven augments a basic coordinate with a *packaging* value or with both a *packaging* value and a *classifier* value. A Maven coordinate can have any one of the following forms:

```
groupId:artifactId:version
groupId:artifactId:packaging:version
groupId:artifactId:packaging:classifier:version
```


Here are descriptions of the values:

groupId

Defines a scope for the name of the artifact. You would typically use all or part of a package name as a group ID. For example, **org.fusesource.example**.

artifactId

Defines the artifact name relative to the group ID.

version

Specifies the artifact's version. A version number can have up to four parts: **n.n.n.n**, where the last part of the version number can contain non-numeric characters. For example, the last part of **1.0-SNAPSHOT** is the alphanumeric substring, **0-SNAPSHOT**.

packaging

Defines the packaged entity that is produced when you build the project. For OSGi projects, the packaging is **bundle**. The default value is **jar**.

classifier

Enables you to distinguish between artifacts that were built from the same POM, but have different content.

Elements in an artifact's POM file define the artifact's group ID, artifact ID, packaging, and version, as shown here:

```
<project ... >
...
<groupId>org.fusesource.example</groupId>
<artifactId>bundle-demo</artifactId>
<packaging>bundle</packaging>
<version>1.0-SNAPSHOT</version>
...
</project>
```

To define a dependency on the preceding artifact, you would add the following **dependency** element to a POM file:

```
<project ... >
...
<dependencies>
<dependency>
<groupId>org.fusesource.example</groupId>
<artifactId>bundle-demo</artifactId>
<version>1.0-SNAPSHOT</version>
</dependency>
</dependencies>
...
</project>
```



NOTE

It is not necessary to specify the **bundle** package type in the preceding dependency, because a bundle is just a particular kind of JAR file and **jar** is the default Maven package type. If you do need to specify the packaging type explicitly in a dependency, however, you can use the **type** element.

