Red Hat AMQ 7.3

Using the AMQ Spring Boot Starter

For Use with AMQ Clients 2.4

Last Updated: 2019-06-18

Red Hat AMQ 7.3 Using the AMQ Spring Boot Starter

For Use with AMQ Clients 2.4

Legal Notice

Abstract

This guide describes how to install and configure the library, run hands-on examples, and use your client with other AMQ components.

Table of Contents

CHAPTER 1. OVERVIEW

AMQ Spring Boot Starter is an adapter for creating Spring-based applications that use AMQ messaging. It provides a Spring Boot starter module that allows you to build standalone Spring applications. The starter uses the AMQ JMS client to communicate using the AMQP 1.0 protocol.

AMQ Spring Boot Starter is part of AMQ Clients, a suite of messaging libraries supporting multiple languages and platforms. For an overview of the clients, see AMQ Clients Overview. For information about this release, see AMQ Clients 2.4 Release Notes .

AMQ Spring Boot Starter is based on the AMQP 1.0 JMS Spring Boot project.

1.1. KEY FEATURES

- Quickly build standalone Spring applications with built-in messaging
- Automatic configuration of JMS resources
- Configurable pooling of JMS connections and sessions

1.2. SUPPORTED STANDARDS AND PROTOCOLS

- Version 2.1 of the Spring Boot API
- Version 2.0 of the Java Message Service API
- Version 1.0 of the Advanced Message Queueing Protocol (AMQP)

1.3. SUPPORTED CONFIGURATIONS

AMQ Spring Boot Starter supports the following OS and language versions:

- Red Hat Enterprise Linux 6, 7, and 8 with the following JDKs:
  - OpenJDK 8 and 11
  - Oracle JDK 8
  - IBM JDK 8
- HP-UX 11i with HP-UX JVM 8
- IBM AIX 7.1 with IBM JDK 8
- Oracle Solaris 10 and 11 with Oracle JDK 8
- Microsoft Windows 10 Pro with Oracle JDK 8
- Microsoft Windows Server 2012 R2 and 2016 with Oracle JDK 8

AMQ Spring Boot Starter is supported in combination with the following AMQ components and versions:

- All versions of AMQ Broker

- All versions of AMQ Interconnect
- All versions of AMQ Online
- A-MQ 6 versions 6.2.1 and higher

For more information, see Red Hat AMQ Supported Configurations .

CHAPTER 2. INSTALLATION

This chapter guides you through the steps to install AMQ Spring Boot Starter in your environment.

Using the Red Hat Maven repository

The library uses Apache Maven as its build tool. You can configure your Maven environment to download the library from the Red Hat Maven repository.

Procedure

1. Add the Red Hat repository to your Maven settings or POM file. For example configuration files, see Section B.1, “Using the online repository” .

```
<repository>
  <id>red-hat-ga</id>
  <url>https://maven.repository.redhat.com/ga</url>
</repository>
```

2. Add the library dependency to your POM file.

```
<dependency>
  <groupId>org.amqphub.spring</groupId>
  <artifactId>amqp-10-jms-spring-boot-starter</artifactId>
  <version>2.1.3.redhat-00004</version>
</dependency>
```

The library is now available in your Maven project.

CHAPTER 3. GETTING STARTED

This chapter guides you through a simple exercise to help you get started using AMQ Spring Boot Starter.

3.1. PREREQUISITES

To build the example, Maven must be configured to use the Red Hat repository.

3.2. PREPARING THE BROKER

The example programs require a running broker with a queue named **example**. Follow these steps to define the queue and start the broker:

Procedure

1. Install the broker.
2. Create a broker instance. Enable anonymous access.
3. Start the broker instance and check the console for any critical errors logged during startup.

```
$ <broker-instance-dir>/bin/artemis run
...
14:43:20,158 INFO  [org.apache.activemq.artemis.integration.bootstrap] AMQ101000: Starting ActiveMQ Artemis Server
...
15:01:39,686 INFO  [org.apache.activemq.artemis.core.server] AMQ221020: Started Acceptor at 0.0.0.0:5672 for protocols [AMQP]
...
15:01:39,691 INFO  [org.apache.activemq.artemis.core.server] AMQ221007: Server is now live
```

4. Use the **artemis queue** command to create a queue called **example**.

```
<broker-instance-dir>/bin/artemis queue create --name example --auto-create-address --anycast
```

You are prompted to answer a series of questions. For yes or no questions, type **N**. Otherwise, press Enter to accept the default value.

3.3. HELLO WORLD

This example sends a greeting to a queue called **example** and then receives it back.

Example: Sending and receiving "Hello World!" - HelloWorld.java

```
package net.example;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.jms.annotation.EnableJms;
import org.springframework.jms.annotation.JmsListener;
import org.springframework.jms.core.JmsTemplate;

@EnableJms
@SpringBootApplication
public class HelloWorld implements CommandLineRunner {
    @Autowired
    private JmsTemplate jmsTemplate;

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

    @Override
    public void run(String... strings) throws Exception {
        sendMessage("Hello World!");
    }

    public void sendMessage(String text) {
        System.out.println(String.format("Sending '%s'", text));
        this.jmsTemplate.convertAndSend("example", text);
    }

    @JmsListener(destination = "example")
    public void receiveMessage(String text) {
        System.out.println(String.format("Received '%s'", text));
    }
}
```

Running the example

To compile and run the example program, use the following procedure:

Procedure

1. Create a new project directory. This is referred to as **<project-dir>** in the steps that follow.
2. Copy the example listing to the following location:

```
<project-dir>/src/main/java/net/example/HelloWorld.java
```

3. Use a text editor to create a new **<project-dir>/pom.xml** file. Add the following XML to it:

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>net.example</groupId>
  <artifactId>example</artifactId>
  <version>1.0.0-SNAPSHOT</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.5.RELEASE</version>
  </parent>
```

```
  <dependencies>
    <dependency>
      <groupId>org.amqphub.spring</groupId>
      <artifactId>amqp-10-jms-spring-boot-starter</artifactId>
      <version>2.1.3.redhat-00004</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

4. Change to the project directory and use the **mvn** command to compile the program.

```
mvn clean package
```

5. Use the **java** command to run the program.

```
java -jar target/example-1.0.0-SNAPSHOT.jar
```

Running the Hello World example results in the following console output:

```
$ java -jar target/example-1.0.0-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.5.RELEASE)

[...]
2018-11-05 16:21:09.849  INFO 14805 --- [main] net.example.HelloWorld: Started
HelloWorld in 1.074 seconds (JVM running for 1.38)
Sending 'Hello World!'
Received 'Hello World!'
```

3.4. ADDITIONAL EXAMPLES

More example programs are available from the AMQP 1.0 JMS Spring Boot project and the Spring project.

CHAPTER 4. CONFIGURATION

The following options can be used in an **application-properties** file to configure your Spring Boot application.

4.1. CONNECTION OPTIONS

These options determine how AMQ Spring Boot Starter establishes connections to remote AMQP peers. The starter uses AMQ JMS to communicate over the network. For more information, see Using the AMQ JMS Client.

amqphub.amqp10jms.remoteUrl

The connection URI that the AMQ JMS client uses to establish new connections.

Connection URI format

```
amqp[s]://host:port[?option=value[&option2=value...]]
```

For more information, see Connection URIs in Using the AMQ JMS Client.

amqphub.amqp10jms.username

The username used to authenticate the connection.

amqphub.amqp10jms.password

The password used to authenticate the connection.

amqphub.amqp10jms.clientId

The client ID applied to the connection.

amqphub.amqp10jms.receiveLocalOnly

If enabled, calls to **receive** with a timeout argument check the consumer's local message buffer only. Otherwise, the remote peer is checked as well to ensure there are no messages available. It is disabled by default.

amqphub.amqp10jms.receiveNoWaitLocalOnly

If enabled, calls to **receiveNoWait** check the consumer's local message buffer only. Otherwise, the remote peer is checked as well to ensure there are no messages available. It is disabled by default.

4.2. POOLING OPTIONS

These options determine how AMQ Spring Boot Starter caches JMS connections and sessions. The starter uses AMQ JMS Pool for its pooling. For more information, see Using the AMQ JMS Pool Library .

amqphub.amqp10jms.pool.enabled

Controls whether pooling is enabled. It is disabled by default.

amqphub.amqp10jms.pool.maxConnections

The maximum number of connections for a single pool. The default is 1.

amqphub.amqp10jms.pool.maxSessionsPerConnection

The maximum number of sessions for each connection. The default is 500. A negative value removes any limit.
If the limit is exceeded, **createSession()** either blocks or throws an exception, depending on configuration.

amqphub.amqp10jms.pool.blockIfSessionPoolIsFull

If enabled, calls to **createSession()** block until a session becomes available in the pool. It is enabled by default.
If disabled, calls to **createSession()** throw an **IllegalStateException** if no session is available.

amqphub.amqp10jms.pool.blockIfSessionPoolIsFullTimeout

The time in milliseconds before a blocked call to **createSession()** throws an **IllegalStateException**. The default is -1, meaning the call blocks forever.

amqphub.amqp10jms.pool.connectionIdleTimeout

The time in milliseconds before a connection not currently on loan can be evicted from the pool. The default is 30 seconds. A value of 0 disables the timeout.

amqphub.amqp10jms.pool.connectionCheckInterval

The time in milliseconds between periodic checks for expired connections. The default is 0, meaning the check is disabled.

amqphub.amqp10jms.pool.useAnonymousProducers

If enabled, use a single anonymous JMS **MessageProducer** for all calls to **createProducer()**. It is enabled by default.
In rare cases, this behavior is undesirable. If disabled, every call to **createProducer()** results in a new **MessageProducer** instance.

amqphub.amqp10jms.pool.explicitProducerCacheSize

When not using anonymous producers, the JMS **Session** can be configured to cache a certain number of **MessageProducer** objects with explicit destinations. As new producers are created that do not match the cached producers, the oldest entry in the cache is evicted.

amqphub.amqp10jms.pool.useProviderJMSContext

If enabled, use the **JMSContext** classes of the underlying JMS provider. It is disabled by default.
In normal operation, the pool uses its own generic **JMSContext** implementation to wrap connections from the pool instead of using the provider implementation. The generic implementation might have limitations the provider implementation does not. However, when enabled, connections from the **JMSContext** API are not managed by the pool.

APPENDIX A. USING YOUR SUBSCRIPTION

AMQ is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

Accessing your account

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

Activating a subscription

1. Go to access.redhat.com.
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

Downloading ZIP and TAR files

To access ZIP or TAR files, use the customer portal to find the relevant files for download. If you are using RPM packages, this step is not required.

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at access.redhat.com/downloads.
2. Locate the **Red Hat AMQ** entries in the **JBOSS INTEGRATION AND AUTOMATION** category.
3. Select the desired AMQ product. The **Software Downloads** page opens.
4. Click the **Download** link for your component.

Registering your system for packages

To install RPM packages on Red Hat Enterprise Linux, your system must be registered. If you are using ZIP or TAR files, this step is not required.

1. Go to access.redhat.com.
2. Navigate to **Registration Assistant**.
3. Select your OS version and continue to the next page.
4. Use the listed command in your system terminal to complete the registration.

To learn more see How to Register and Subscribe a System to the Red Hat Customer Portal .

APPENDIX B. USING RED HAT MAVEN REPOSITORIES

This section describes how to use Red Hat-provided Maven repositories in your software.

B.1. USING THE ONLINE REPOSITORY

Red Hat maintains a central Maven repository for use with your Maven-based projects. For more information, see the repository welcome page.

There are two ways to configure Maven to use the Red Hat repository:

- Add the repository to your Maven settings
- Add the repository to your POM file

Adding the repository to your Maven settings

This method of configuration applies to all Maven projects owned by your user, as long as your POM file does not override the repository configuration and the included profile is enabled.

Procedure

1. Locate the Maven **settings.xml** file. It is usually inside the **.m2** directory in the user home directory. If the file does not exist, use a text editor to create it.
   On Linux or UNIX:

   ```
   /home/<username>/.m2/settings.xml
   ```

   On Windows:

   ```
   C:\Users\<username>\.m2\settings.xml
   ```

2. Add a new profile containing the Red Hat repository to the **profiles** element of the **settings.xml** file, as in the following example:

   Example: A Maven settings.xml file containing the Red Hat repository

   ```
   <settings>
     <profiles>
       <profile>
         <id>red-hat</id>
         <repositories>
           <repository>
             <id>red-hat-ga</id>
             <url>https://maven.repository.redhat.com/ga</url>
           </repository>
         </repositories>
         <pluginRepositories>
           <pluginRepository>
             <id>red-hat-ga</id>
             <url>https://maven.repository.redhat.com/ga</url>
             <releases>
               <enabled>true</enabled>
             </releases>
             <snapshots>
   ```

```
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
 </profile>
</profiles>
<activeProfiles>
 <activeProfile>red-hat</activeProfile>
</activeProfiles>
</settings>
```

For more information about Maven configuration, see the Maven settings reference.

Adding the repository to your POM file

To configure a repository directly in your project, add a new entry to the **repositories** element of your POM file, as in the following example:

Example: A Maven pom.xml file containing the Red Hat repository

```
<project>
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.example</groupId>
 <artifactId>example-app</artifactId>
 <version>1.0.0</version>

 <repositories>
  <repository>
   <id>red-hat-ga</id>
   <url>https://maven.repository.redhat.com/ga</url>
  </repository>
 </repositories>
</project>
```

For more information about POM file configuration, see the Maven POM reference.

B.2. USING A LOCAL REPOSITORY

Red Hat provides file-based Maven repositories for some of its components. These are delivered as downloadable archives that you can extract to your local filesystem.

To configure Maven to use a locally extracted repository, apply the following XML in your Maven settings or POM file:

```
<repository>
 <id>red-hat-local</id>
 <url>${repository-url}</url>
</repository>
```

${repository-url} must be a file URL containing the local filesystem path of the extracted repository.

Table B.1. Example URLs for local Maven repositories

Operating system	Filesystem path	URL
Linux or UNIX	**/home/alice/maven-repository**	**file:/home/alice/maven-repository**
Windows	**C:\repos\red-hat**	**file:C:\repos\red-hat**

Revised on 2019-06-18 17:15:44 UTC