Red Hat Data Grid 7.3

Data Grid Spring Boot Starter

Build Spring Boot Applications with Data Grid

Last Updated: 2020-12-02

Red Hat Data Grid 7.3 Data Grid Spring Boot Starter

Build Spring Boot Applications with Data Grid

Legal Notice

Abstract

Quickly get your Spring Boot project up and running with Data Grid.

Table of Contents

1. SETTING UP YOUR PROJECT 2
1.1. Spring Boot Starter Versions 2
1.2. Enforcing Data Grid Versions 2
1.3. Adding Dependencies for Usage Modes 3

2. RUNNING IN EMBEDDED MODE 3
2.1. Adding the EmbeddedCacheManager Bean 3
2.2. Cache Manager Configuration Beans 4
2.3. Enabling Spring Cache Support 5

3. RUNNING IN SERVER MODE 5
3.1. Setting Up the RemoteCacheManager 5
3.2. Cache Manager Configuration Beans 5
3.3. Enabling Spring Cache Support 6
3.4. Exposing Data Grid Statistics 6

4. USING SPRING SESSION 7
4.1. Enabling Spring Session Support 7

5. APPLICATION PROPERTIES 8

The Data Grid starter provides a set of managed transitive dependencies that include everything your Spring Boot project needs to seamlessly interact with Data Grid.

TIP
The Data Grid Spring Boot starter gives you a convenient way to get started with Spring Boot but is optional. To use Data Grid with Spring Boot you can simply add the dependencies you want.

1. SETTING UP YOUR PROJECT

Add dependencies for the Data Grid Spring Boot Starter to your project.

1.1. Spring Boot Starter Versions

Data Grid supports Spring Boot 1.5.x and 2.x. For Spring Boot 1.5.x, use the Spring 4 dependencies. For Spring Boot 2.x, use the Spring 5 dependencies.

- Spring Boot 1.5.x: Replace **${version.infinispan.starter}** with **1.0.7.Final-redhat-00019**.
- Spring Boot 2.x: Replace **${version.infinispan.starter}** with **2.1.10.Final-redhat-00005**.

1.2. Enforcing Data Grid Versions

This starter uses a high-level API to ensure compatibility between major versions of Data Grid. However you can enforce a specific version of Data Grid with the **infinispan-bom** module.

Add **infinispan-bom** to your **pom.xml** file before the starter dependencies, as follows:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-bom</artifactId>
      <version>${version.infinispan}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>${version.spring.boot}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-spring-boot-starter</artifactId>
      <version>${version.infinispan.starter}</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

IMPORTANT

The Data Grid Spring Boot starter uses different Spring Boot versions to other projects such as Red Hat OpenShift Application Runtimes. If you want to use a specific Spring Boot version for compatibility with other projects, you must add the correct dependency to your project.

1.3. Adding Dependencies for Usage Modes

Data Grid provides different dependencies for each usage mode. Add one of the following to your **pom.xml** file:

Embedded Mode

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-spring-boot-starter-embedded</artifactId>
  <version>${version.infinispan.starter}</version>
</dependency>
```

Remote Client/Server Mode

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-spring-boot-starter-remote</artifactId>
  <version>${version.infinispan.starter}</version>
</dependency>
```

2. RUNNING IN EMBEDDED MODE

Embed the Data Grid library in your project for in-memory data storage.

2.1. Adding the EmbeddedCacheManager Bean

1. Add **infinispan-spring-boot-starter-embedded** to your project's classpath to enable Embedded mode.
   This starter operates in Remote Client/Server mode with **infinispan-spring-boot-starter-remote** on the classpath by default.

2. Use the Spring **@Autowired** annotation to include an **EmbeddedCacheManager** bean in your Java configuration classes, as in the following example:

```
private final EmbeddedCacheManager cacheManager;

@Autowired
public YourClassName(EmbeddedCacheManager cacheManager) {
   this.cacheManager = cacheManager;
}
```

   You are now ready to use Data Grid in Embedded Mode. Here is a simple example:

```
cacheManager.getCache("testCache").put("testKey", "testValue");
System.out.println("Received value from cache: " +
cacheManager.getCache("testCache").get("testKey"));
```

2.2. Cache Manager Configuration Beans

You customize the cache manager with the following configuration beans:

- **InfinispanGlobalConfigurer**
- **InfinispanCacheConfigurer**
- **Configuration**
- **InfinispanConfigurationCustomizer**
- **InfinispanGlobalConfigurationCustomizer**

NOTE

You can create one **InfinispanGlobalConfigurer** bean only. However you can create multiple configurations with the other beans.

InfinispanCacheConfigurer Bean

```
@Bean
public InfinispanCacheConfigurer cacheConfigurer() {
 return manager -> {
  final Configuration ispnConfig = new ConfigurationBuilder()
                    .clustering()
                    .cacheMode(CacheMode.LOCAL)
                    .build();

  manager.defineConfiguration("local-sync-config", ispnConfig);
 };
}
```

Configuration Bean

Link the bean name to the cache that it configures, as follows:

```
@Bean(name = "small-cache")
public org.infinispan.configuration.cache.Configuration smallCache() {
    return new ConfigurationBuilder()
        .read(baseCache)
        .memory().size(1000L)
        .memory().evictionType(EvictionType.COUNT)
        .build();
}

@Bean(name = "large-cache")
public org.infinispan.configuration.cache.Configuration largeCache() {
    return new ConfigurationBuilder()
        .read(baseCache)
        .memory().size(2000L)
        .build();
}
```

Customizer Beans

```
@Bean
public InfinispanGlobalConfigurationCustomizer globalCustomizer() {
  return builder -> builder.transport().clusterName(CLUSTER_NAME);
}

@Bean
public InfinispanConfigurationCustomizer configurationCustomizer() {
  return builder -> builder.memory().evictionType(EvictionType.COUNT);
}
```

2.3. Enabling Spring Cache Support

Add the **@EnableCaching** annotation to your application to enable Spring Cache support.

When this starter detects the **EmbeddedCacheManager** bean, it instantiates a new **SpringEmbeddedCacheManager**, which provides an implementation of Spring Cache.

3. RUNNING IN SERVER MODE

Store and retrieve data from remote Data Grid clusters using Hot Rod, a custom TCP binary wire protocol.

3.1. Setting Up the RemoteCacheManager

1. Provide the location for the Data Grid server so the starter can create the **RemoteCacheManager** bean.
   This starter first attempts to locate the server from the **hotrod-client.properties** file on the classpath. If not found, the starter then attempts to locate the server from your **application.properties** file.

   - **hotrod-client.properties**:

     ```
     infinispan.client.hotrod.server_list=127.0.0.1:11222
     ```

   - **application.properties**:

     ```
     infinispan.remote.server-list=127.0.0.1:11222
     ```

2. Use the Spring **@Autowired** annotation to include your own custom cache manager class in your application:

   ```
   private final RemoteCacheManager cacheManager;

   @Autowired
   public YourClassName(RemoteCacheManager cacheManager) {
      this.cacheManager = cacheManager;
   }
   ```

3.2. Cache Manager Configuration Beans

Customize the cache manager with the following configuration beans:

- **InfinispanRemoteConfigurer**

- **Configuration**
- **InfinispanRemoteCacheCustomizer**

NOTE

You can create one **InfinispanRemoteConfigurer** bean only. However you can create multiple configurations with the other beans.

InfinispanRemoteConfigurer Bean

```
@Bean
public InfinispanRemoteConfigurer infinispanRemoteConfigurer() {
   return () -> new ConfigurationBuilder()
      .addServer()
      .host("127.0.0.1")
      .port(12345)
      .build();
}
```

Configuration Bean

```
@Bean
public org.infinispan.client.hotrod.configuration.Configuration customConfiguration() {
   new ConfigurationBuilder()
      .addServer()
      .host("127.0.0.1")
      .port(12345)
      .build();
}
```

InfinispanRemoteCacheCustomizer Bean

```
@Bean
public InfinispanRemoteCacheCustomizer customizer() {
   return b -> b.tcpKeepAlive(false);
}
```

TIP

Use the **@Ordered** annotation to apply customizers in a specific order.

3.3. Enabling Spring Cache Support

Add the **@EnableCaching** annotation to your application to enable Spring Cache support.

When the Data Grid starter detects the **RemoteCacheManager** bean, it instantiates a new **SpringRemoteCacheManager**, which provides an implementation of Spring Cache.

3.4. Exposing Data Grid Statistics

Data Grid supports the Spring Boot Actuator to expose cache statistics as metrics.

To use the Actuator, add the following to your **pom.xml** file:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
  <version>${version.spring.boot}</version>
 </dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>${version.spring.boot}</version>
</dependency>
```

You must then activate statistics for the appropriate cache instances, either programmatically or declaratively.

Programmatically

```
@Bean
public InfinispanCacheConfigurer cacheConfigurer() {
  return cacheManager -> {
    final org.infinispan.configuration.cache.Configuration config =
        new ConfigurationBuilder()
            .jmxStatistics().enable()
            .build();

    cacheManager.defineConfiguration("my-cache", config);
  };
}
```

Declaratively

```
<local-cache name="my-cache" statistics="true"/>
```

The Spring Boot Actuator registry binds cache instances when your application starts. If you create caches dynamically, you should use the **CacheMetricsRegistrar** bean to bind caches to the Actuator registry, as follows:

```
@Autowire
CacheMetricsRegistrar cacheMetricsRegistrar;

@Autowire
CacheManager cacheManager;
...

cacheMetricsRegistrar.bindCacheToRegistry(cacheManager.getCache("my-cache"));
```

4. USING SPRING SESSION

4.1. Enabling Spring Session Support

Data Grid Spring Session support is built on **SpringRemoteCacheManager** and **SpringEmbeddedCacheManager**. This starter produces those beans by default.

To use Spring Session in your project, do the following:

1. Add this starter to your project.
2. Add Spring Session to the classpath.
3. Add the following annotations to your configuration:
    - **@EnableCaching**
    - **@EnableInfinispanRemoteHttpSession**
    - **@EnableInfinispanEmbeddedHttpSession**

5. APPLICATION PROPERTIES

Configure your project with **application.properties** or **application.yaml**.

```
# List Infinispan or Data Grid servers by IP address or hostname at port 11222.
infinispan.remote.server-list=127.0.0.1:11222

#
# Embedded Properties - Uncomment properties to use them.
#

# Enables Infinispan or Data Grid capabilities in your application.
# Values are true (default) or false.
#infinispan.embedded.enabled =

# Sets the Spring state machine ID.
#infinispan.embedded.machineId =

# Sets the name of the Infinispan or Data Grid cluster.
#infinispan.embedded.clusterName =

# Specifies a XML configuration file that takes priority over the global
# configuration bean or any configuration customizer.
#infinispan.embedded.configXml =

#
# Server Properties - Uncomment properties to use them.
#

# Specifies a custom filename for Hot Rod client properties.
#infinispan.remote.clientProperties =

# Enables remote Infinispan or Data Grid servers.
# Values are true (default) or false.
#infinispan.remote.enabled =

# Defines a comma-separated list of Infinispan or Data Grid servers
# in this format: `host1[:port],host2[:port]`.
#infinispan.remote.serverList =
```

```
# Sets a timeout value, in milliseconds, for socket connections.
#infinispan.remote.socketTimeout =

# Sets a timeout value for initializing connections with Infinispan or
# Data Grid servers.
#infinispan.remote.connectTimeout =

# Sets the maximum number of attempts to connect to Infinispan or
# Data Grid servers.
#infinispan.remote.maxRetries =
```