Red Hat Data Grid 8.4

Data Grid Spring Boot Starter

Use Data Grid with your Spring Boot project

Last Updated: 2024-04-19

Red Hat Data Grid 8.4 Data Grid Spring Boot Starter

Use Data Grid with your Spring Boot project

Legal Notice

Abstract

Quickly get your Spring Boot project up and running with a set of managed transitive dependencies that include everything your Spring Boot project needs to seamlessly interact with Data Grid. Note that while the Data Grid Spring Boot starter gives you a convenient way to get started with Spring Boot it is optional. To use Data Grid with Spring Boot you can simply add the dependencies you want.

Table of Contents

RED HAT DATA GRID

Data Grid is a high-performance, distributed in-memory data store.

Schemaless data structure

Flexibility to store different objects as key-value pairs.

Grid-based data storage

Designed to distribute and replicate data across clusters.

Elastic scaling

Dynamically adjust the number of nodes to meet demand without service disruption.

Data interoperability

Store, retrieve, and query data in the grid from different endpoints.

DATA GRID DOCUMENTATION

Documentation for Data Grid is available on the Red Hat customer portal.

- Data Grid 8.4 Documentation
- Data Grid 8.4 Component Details
- Supported Configurations for Data Grid 8.4
- Data Grid 8 Feature Support
- Data Grid Deprecated Features and Functionality

DATA GRID DOWNLOADS

Access the Data Grid Software Downloads on the Red Hat customer portal.

NOTE

You must have a Red Hat account to access and download Data Grid software.

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. SETTING UP THE SPRING BOOT STARTER

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

IMPORTANT

Data Grid supports Spring Boot version 2.x and version 3. Be aware that Spring Boot version 3 requires Java 17.

The examples in this document include artifacts for the latest version of Spring Boot. If you want to use Spring Boot 2.x use:

- **infinispan-spring-boot-starter-embedded**
- **infinispan-spring-boot-starter-remote**

1.1. ENFORCING DATA GRID VERSIONS

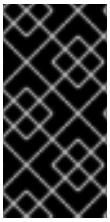
This starter uses a high-level API to ensure compatibility between major versions of Data Grid. However you can enforce a specific version of Data Grid with the **infinispan-bom** module.

Procedure

- Add **infinispan-bom** to your **pom.xml** file before the starter dependencies.

```
<properties>
  <version.infinispan>14.0.21.Final-redhat-00001</version.infinispan>
</properties>

<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-boot3-starter</artifactId>
      </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.

Additional resources

- Spring and Spring Boot code tutorials
- Spring Boot Runtime Guide, Using the Spring Boot BOM to manage dependency versions

1.2. ADDING DEPENDENCIES FOR USAGE MODES

Data Grid provides different dependencies for embedded caches and remote caches.

Procedure

- Add one of the following to your **pom.xml** file:

Embedded caches

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-spring-boot3-starter-embedded</artifactId>
</dependency>
```

Remote caches

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-spring-boot3-starter-remote</artifactId>
</dependency>
```

CHAPTER 2. USING EMBEDDED CACHES

Embed Data Grid caches directly in your project for in-memory data storage.

2.1. ADDING THE EMBEDDEDCACHEMANAGER BEAN

Configure your application to use embedded caches.

Procedure

1. Add **infinispan-spring-boot3-starter-embedded** to your project's classpath to enable Embedded mode.
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 caches directly within your application, as in the following 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 can 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

With both embedded and remote caches, Data Grid provides an implementation of Spring Cache that you can enable.

Procedure

- Add the **@EnableCaching** annotation to your application.

If the Data Grid starter detects the:

- **EmbeddedCacheManager** bean, it instantiates a new **SpringEmbeddedCacheManager**.

- **RemoteCacheManager** bean, it instantiates a new **SpringRemoteCacheManager**.

Reference

Spring Cache Reference

CHAPTER 3. USING REMOTE CACHES

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

3.1. SETTING UP THE REMOTECACHEMANAGER

Configure your application to use remote caches on Data Grid clusters.

1. Provide the addresses where Data Grid Server listens for client connections so the starter can create the **RemoteCacheManager** bean.
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;
}
```

3.1.1. Properties Files

You can specify properties in either **hotrod-client.properties** or **application.properties**.

Properties can be in both properties files but the starter applies the configuration in **hotrod-client.properties** first, which means that file takes priority over **application.properties**.

hotrod-client.properties
Properties in this file take the format of **infinispan.client.hotrod.***, for example:

```
# List Data Grid servers by IP address or hostname at port localhost:11222.
infinispan.client.hotrod.server_list=127.0.0.1:11222
```

- Hot Rod client configuration API

application.properties
Properties in this file take the format of **infinispan.remote.***, for example:

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

Additional resources

- Application Properties

3.2. CONFIGURING MARSHALLING

Configure Data Grid to marshall Java objects into binary format so they can be transferred over the wire or stored to disk.

By default Data Grid uses a Java Serialization marshaller, which requires you to add your classes to an allow list. As an alternative you can use ProtoStream, which requires you to annotate your classes and generate a **SerializationContextInitializer** for custom Java objects.

Procedure

1. Open **hotrod-client.properties** or **application.properties** for editing.
2. Do one of the following:
   - Use ProtoStream as the marshaller.

     ```
     infinispan.client.hotrod.marshaller=org.infinispan.commons.marshall.ProtoStreamMarshalle
     ```

     ```
     infinispan.remote.marshaller=org.infinispan.commons.marshall.ProtoStreamMarshaller
     ```

   - Add your classes to the serialization allow list if you use Java Serialization. You can specify a comma-separated list of fully qualified class names or a regular expression to match classes.

     ```
     infinispan.client.hotrod.java_serial_allowlist=your_marshalled_beans_package.*
     ```

     ```
     infinispan.remote.java-serial-allowlist=your_marshalled_beans_package.*
     ```

3. Save and close your properties file.

Additional resources

- Cache Encoding and Marshalling

3.3. 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.

3.4. ENABLING SPRING CACHE SUPPORT

With both embedded and remote caches, Data Grid provides an implementation of Spring Cache that you can enable.

Procedure

- Add the **@EnableCaching** annotation to your application.

If the Data Grid starter detects the:

- **EmbeddedCacheManager** bean, it instantiates a new **SpringEmbeddedCacheManager**.
- **RemoteCacheManager** bean, it instantiates a new **SpringRemoteCacheManager**.

Reference

Spring Cache Reference

3.5. EXPOSING DATA GRID STATISTICS

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

Procedure

1. 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>
```

2. 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 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"));
```

CHAPTER 4. USING SPRING SESSION

4.1. ENABLING SPRING SESSION SUPPORT

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

Procedure

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**

NOTE

Data Grid does not provide a default cache. To use Spring Session, you must first create a Data Grid cache.

CHAPTER 5. APPLICATION PROPERTIES

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

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

# Enables embedded 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 embedded 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 server connections.
# Values are true (default) or false.
#infinispan.remote.enabled =

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

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

# Sets a timeout value for initializing connections with servers.
#infinispan.remote.connectTimeout =

# Sets the maximum number of attempts to connect to servers.
#infinispan.remote.maxRetries =

# Specifies the marshaller to use.
#infinispan.remote.marshaller =

# Adds your classes to the serialization allow list.
#infinispan.remote.java-serial-allowlist=
```