



Red Hat Data Grid 8.1

Data Grid Spring Boot Starter

Use Data Grid with your Spring Boot project

Red Hat Data Grid 8.1 Data Grid Spring Boot Starter

Use Data Grid with your Spring Boot project

Legal Notice

Copyright © 2021 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

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

CHAPTER 1. RED HAT DATA GRID	3
1.1. DATA GRID DOCUMENTATION	3
1.2. DATA GRID DOWNLOADS	3
1.3. MAKING OPEN SOURCE MORE INCLUSIVE	3
CHAPTER 2. SETTING UP YOUR PROJECT	4
2.1. ENFORCING DATA GRID VERSIONS	4
2.2. ADDING DEPENDENCIES FOR USAGE MODES	4
CHAPTER 3. RUNNING IN EMBEDDED MODE	6
3.1. ADDING THE EMBEDDED CACHEMANAGER BEAN	6
3.2. CACHE MANAGER CONFIGURATION BEANS	6
3.3. ENABLING SPRING CACHE SUPPORT	7
CHAPTER 4. RUNNING IN SERVER MODE	8
4.1. SETTING UP THE REMOTE CACHEMANAGER	8
4.2. CONFIGURING MARSHALLING	8
4.3. CACHE MANAGER CONFIGURATION BEANS	9
4.4. ENABLING SPRING CACHE SUPPORT	9
4.5. EXPOSING DATA GRID STATISTICS	10
CHAPTER 5. USING SPRING SESSION	11
5.1. ENABLING SPRING SESSION SUPPORT	11
CHAPTER 6. APPLICATION PROPERTIES	12

CHAPTER 1. 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.

1.1. DATA GRID DOCUMENTATION

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

- [Data Grid 8.1 Documentation](#)
- [Data Grid 8.1 Component Details](#)
- [Supported Configurations for Data Grid 8.1](#)
- [Data Grid 8 Feature Support](#)
- [Data Grid Deprecated Features and Functionality](#)

1.2. 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.

1.3. 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 2. SETTING UP YOUR PROJECT

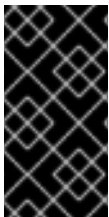
Add dependencies for the Data Grid Spring Boot Starter to your project.

2.1. 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.

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>2.3.6.Final-redhat-00001</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.

2.2. 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>2.3.6.Final-redhat-00001</version>  
</dependency>
```

Remote Client/Server Mode

```
<dependency>  
  <groupId>org.infinispan</groupId>  
  <artifactId>infinispan-spring-boot-starter-remote</artifactId>  
  <version>2.3.6.Final-redhat-00001</version>  
</dependency>
```

CHAPTER 3. RUNNING IN EMBEDDED MODE

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

3.1. ADDING THE EMBEDDED CACHEMANAGER 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"));
```

3.2. 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);
}

```

3.3. 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](#).

CHAPTER 4. RUNNING IN SERVER MODE

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

4.1. SETTING UP THE REMOTECACHEMANAGER

1. Provide the location for the Data Grid server so the starter can create the **RemoteCacheManager** bean.
The starter first tries to find the server location in **hotrod-client.properties** and then from **application.properties**.
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;
}
```

Hot Rod client properties

Specify client configuration in **hotrod-client.properties** on your classpath, for example:

```
# List Infinispan or Data Grid servers by IP address or hostname at port 11222.
infinispan.client.hotrod.server_list=127.0.0.1:6667
```

For more information, see [org.infinispan.client.hotrod.configuration](https://www.infinispan.org/documentation/8.1.x/html/InfinispanClientHotRodConfiguration.html).

Application properties

Configure your project with **application.properties**. See [Application Properties](#) for more information.

4.2. CONFIGURING MARSHALLING

Configure Data Grid servers to use Java serialization to marshall objects.

By default Data Grid server uses a ProtoStream serialization library as the default marshaller. However, the ProtoStream marshaller is not supported for Spring integration. For this reason you should use the Java Serialization Marshaller.

- Specify the following properties in your **application.properties**:

```
infinispan.remote.marshaller=org.infinispan.commons.marshall.JavaSerializationMarshaller
1
infinispan.remote.java-serial-whitelist=your_marshalled_beans_package.* 2
```

- 1 Use the Java Serialization Marshaller.
- 2 Adds your classes to the serialization whitelist so Data Grid marshalls your objects. You can specify a comma-separated list of fully qualified class names or a regular expression to match classes.

4.3. 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.

4.4. 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](#).

4.5. 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:

```
@Autowired
CacheMetricsRegistrar cacheMetricsRegistrar;

@Autowired
CacheManager cacheManager;
...

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

CHAPTER 5. USING SPRING SESSION

5.1. 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**

CHAPTER 6. 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 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.serverList =

# 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 whitelist.
#infinispan.remote.java-serial-whitelist=
```