Data Grid Documentation
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

1. RED HAT DATA GRID ................................................................. 2
   1.1. Data Grid Documentation ............................................. 2
   1.2. Data Grid Downloads .................................................... 2

2. SETTING UP YOUR PROJECT ..................................................... 2
   2.1. Enforcing Data Grid Versions ...................................... 2
   2.2. Adding Dependencies for Usage Modes ....................... 3

3. RUNNING IN EMBEDDED MODE .................................................. 3
   3.1. Adding the EmbeddedCacheManager Bean .................. 3
   3.2. Cache Manager Configuration Beans ....................... 4
   3.3. Enabling Spring Cache Support ................................. 5

4. RUNNING IN SERVER MODE ....................................................... 5
   4.1. Setting Up the RemoteCacheManager ....................... 5
   4.2. Configuring Marshalling .......................................... 6
   4.3. Cache Manager Configuration Beans .................... 6
   4.4. Enabling Spring Cache Support ........................... 7
   4.5. Exposing Data Grid Statistics .............................. 7

5. USING SPRING SESSION .......................................................... 8
   5.1. Enabling Spring Session Support ............................ 8

6. APPLICATION PROPERTIES ....................................................... 9
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 8.0 Documentation
- Data Grid 8.0 Component Details
- Supported Configurations for Data Grid 8.0

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.

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:

```xml
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-bom</artifactId>
    </dependency>
  </dependencies>
</dependencyManagement>
```
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

```xml
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-spring-boot-starter-embedded</artifactId>
  <version>2.2.4.Final-redhat-00001</version>
</dependency>
```

### Remote Client/Server Mode

```xml
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-spring-boot-starter-remote</artifactId>
  <version>2.2.4.Final-redhat-00001</version>
</dependency>
```

3. RUNNING IN EMBEDDED MODE

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

3.1. 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:

   ```java
   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:

   ```java
   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**

```java
@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:

```java
@Configuration
public class CacheConfiguration {
    @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

```java
@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.

### 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:
Hot Rod client properties

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

```bash
# 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`.

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`:

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

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

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

Configuration Bean

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

InfinispanRemoteCacheCustomizer Bean

```java
@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:

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
    <version>${version.spring.boot}</version>
</dependency>
```
You must then activate statistics for the appropriate cache instances, either programatically or declaratively.

**Programmatically**

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

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

**Declaratively**

```xml
<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:

```java
@Autowire
CacheMetricsRegistrar cacheMetricsRegistrar;

@Autowire
CacheManager cacheManager;

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

## 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:
Configure your project with `application.properties` or `application.yaml`.

```Properties
# 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=