Red Hat JBoss Data Grid 7.2

Migration Guide

For Use with JBoss Data Grid 7.2

Last Updated: 2019-06-10

Red Hat JBoss Data Grid 7.2 Migration Guide

For Use with JBoss Data Grid 7.2

Legal Notice

Abstract

This guide presents information about migrating to Red Hat JBoss Data Grid 7.2.

Table of Contents

CHAPTER 1. INTRODUCTION

1.1. RED HAT JBOSS DATA GRID

Red Hat JBoss Data Grid is a distributed in-memory data grid, which provides the following capabilities:

- Schemaless key-value store – JBoss Data Grid is a NoSQL database that provides the flexibility to store different objects without a fixed data model.
- Grid-based data storage – JBoss Data Grid is designed to easily replicate data across multiple nodes.
- Elastic scaling – Adding and removing nodes is simple and non-disruptive.
- Multiple access protocols – It is easy to access the data grid using REST, Memcached, Hot Rod, or simple map-like API.

1.2. ABOUT THE MIGRATION GUIDE

The purpose of this guide is to document the changes that are required to successfully run and deploy Red Hat JBoss Data Grid applications on subsequent versions of Red Hat JBoss Data Grid. It provides information on how to resolve deployment and runtime problems and to how prevent changes in application behavior. This is the first step in moving to the new platform. Once the application is successfully deployed and running, plans can be made to upgrade individual components to use the new functions and features of Red Hat JBoss Data Grid 7.2.

CHAPTER 2. MIGRATING TO JBOSS DATA GRID 7.2

2.1. MIGRATING TO JBOSS DATA GRID 7.2

When migrating to JBoss Data Grid 7.2 from JBoss Data Grid 6.6 or earlier, first follow the steps in Migrating to JBoss Data Grid 7.0 and then perform the steps in this section.

2.1.1. Product Changes

Review changes to JBoss Data Grid that impact functionality or configuration between releases.

2.1.1.1. Update Configuration for Partition Handling

In releases of JBoss Data Grid earlier than 7.2, the configuration for network partition handling was as follows:

```
<partition-handling enabled="true|false">
```

As of JBoss Data Grid 7.2 the following configuration takes effect:

```
<partition-handling when-split="ALLOW_READ_WRITES" merge-policy="NONE"/>
```

Before migrating to JBoss Data Grid 7.2, you should review the documentation to understand the different partition handling strategies and merge policies that JBoss Data Grid can use to resolve conflicts.

See the following sections of the Administration and Configuration Guide for more information:

- Handling Network Partitions (Split Brain)
- Merge Policies
- Configuration of Partition Handling Between Releases

2.1.1.2. Eviction Strategy Deprecation

JBoss Data Grid 7.2 improves the data container by deprecating the **eviction** configuration and adding a **memory** configuration. You should review these changes before migrating from an earlier release. For more information, see Set Up JVM Memory Management in the Administration and Configuration Guide.

CHAPTER 3. MIGRATING TO JBOSS DATA GRID 7.1

3.1. MIGRATING TO JBOSS DATA GRID 7.1

When migrating to JBoss Data Grid 7.1 from JBoss Data Grid 6.6 or earlier, first follow the steps in Migrating to JBoss Data Grid 7.0 and then perform the steps in this section.

3.2. PRODUCT CHANGES

3.2.1. Remote Client-Server Configuration Changes

The configuration files for remote client-server mode have been updated in JBoss Data Grid 7.1, and there have been numerous changes to the included configuration files. It is strongly recommended to use the base configurations included in JBoss Data Grid 7.1, and then apply any modifications to these base files. The new configurations result in increased performance, and provide a baseline for further tunings.

In addition, it is no longer required to define a **mode** for clustered caches. Starting in JBoss Data Grid 7.1 the **mode** defaults to **SYNC**.

CHAPTER 4. MIGRATING TO JBOSS DATA GRID 7.0

4.1. PRODUCT CHANGES

4.1.1. Library Mode Configuration Update

The XML schema for Library mode configurations has been changed to be more consistent with the remote client-server schema. This results in several changes that must be made to the configuration file.

To assist in these configurations a tool is provided in the library distribution. The following steps discuss using this tool to automatically convert existing infinispan configuration files to the latest supported schema.

Procedure: Windows Instructions for Using the Library Mode Configuration Converter

1. Open a command prompt.
2. Navigate to the *bin* directory of the library distribution:

```
cd C:\path\to\jboss-datagrid-${jdg-version}-library\bin\
```

3. If any custom classes were referenced in the original configuration then the .jar files containing these classes must be placed in the *C:\path\to\jboss-datagrid-${jdg-version}-library* directory. If no custom classes were referenced then this step may be skipped.
4. Execute the *config-converter.bat* utility, passing in the name of the existing *infinispan_6.0.xml* configuration file and the destination as arguments:

```
config-converter C:\path\to\infinispan_6.0.xml C:\path\to\infinispan-new.xml
```

5. Ensure that *infinispan-new.xml* has been created with the new Library mode configurations. This file may now be used in Library mode applications with JBoss Data Grid 7.0.

Procedure: Linux Instructions for Using the Library Mode Configuration Converter.

1. Open a console to execute the following commands.
2. Navigate to the *bin/* directory of the library distribution:

```
cd /path/to/jboss-datagrid-${jdg-version}-library/bin/
```

3. If any custom classes were referenced in the original configuration then the .jar files containing these classes must be placed in the */path/to/jboss-datagrid-${jdg-version}-library/* directory. If no custom classes were referenced then this step may be skipped.
4. Execute the *config-converter.sh* utility, passing in the name of the existing *infinispan_6.0.xml* configuration file and the destination as arguments:

```
./config-converter.sh /path/to/infinispan_6.0.xml /path/to/infinispan-new.xml
```

5. Ensure that *infinispan-new.xml* has been created with the new Library mode configurations. This file may now be used in Library mode applications with JBoss Data Grid 7.0.

NOTE

If no destination file is specified, such as *infinispan-new.xml* in the above examples, the utility will instead display the converted xml to the console.

4.1.2. CLI Commands

The following CLI commands have been changed for JBoss Data Grid 7.0:

- **clear** - This command has been replaced by **clearcache**.
- **help $COMMAND** - Instead of executing **help** per command, each command now has a built in **--help** flag that is used instead.

4.1.3. Server Managment

The default JBoss Data Grid server management port has been changed to **9990**.

4.1.4. JGroups ENCRYPT Protocol Deprecated

The JGroups **ENCRYPT** protocol has been deprecated, and should be replaced with either the **SYM_ENCRYPT** or **ASYM_ENCRYPT** protocols. Both of these protocols must be placed directly under **NAKACK2**. Configuration details are provided in the *Administration and Configuration Guide* ; however, example configurations for each protocol are shown below:

SYM_ENCRYPT Configuration Example

```
<SYM_ENCRYPT sym_algorithm="AES"
        encrypt_entire_message="true"
        keystore_name="defaultStore.keystore"
        store_password="changeit"
        alias="myKey"/>
```

ASYM_ENCRYPT Configuration Example

```
<ASYM_ENCRYPT encrypt_entire_message="true"
         sym_keylength="128"
         sym_algorithm="AES/ECB/PKCS5Padding"
         asym_keylength="512"
         asym_algorithm="RSA"/>
```

4.1.5. JBoss EAP Maven Repository Dependencies

Red Hat JBoss Data Grid 7.0 and later depends on the JBoss EAP Maven repository. In addition to the artifacts in the JBoss Data Grid Maven repository, you should also install the artifacts from the JBoss EAP 7.0.0 Maven repository.

4.2. APPLICATION CHANGES

4.2.1. NotifyingFuture replaced by CompletableFuture

The **NotifyingFuture** interface has been removed in JBoss Data Grid 7.0 and replaced with **CompletableFuture**. This class was replaced due to **CompletableFuture** being included as a standard

class in Java 8, allowing for additional features such as chaining, implementing constructs such as conditionals, and timeouts to be utilized without introducing additional custom code.

Due to this change any applications utilizing **NotifyingFuture** will need to be replaced with **CompletableFuture** to function correctly with JBoss Data Grid 7.0.

4.2.2. JBoss Modules CDI Dependency

JBoss Modules CDI Dependencies

The package containing the CDI classes has changed. All of these classes have moved from **org.infinispan.cdi** to **org.infinispan.cdi.embedded**, and any applications utilizing CDI should have their imports updated to use the new packages.

Configuration of CDI/JCache

CDI Extensions have been split into two separate types of classes:

1. CDI Classes that begin with **Injected** before the classname, such as **org.infinispan.jcache.annotation.InjectedCacheResultInterceptor**. These classes are intended for use in managed environments, such as JBoss EAP.

2. CDI classes without **Injected** before the classname, such as **org.infinispan.jcache.annotation.CacheResultInterceptor** - This class should be used in standalone environments.

4.2.3. Spring 3 support removed

Spring 3 support has been removed in JBoss Data Grid 7.0. Any applications will need to be rewritten to use Spring 4.

4.2.4. RemoteCache Deprecated Methods

The following methods have been deprecated in JBoss Data Grid 7.0:

- **remoteCache.getVersioned** - This method has been replaced by **remoteCache.getWithMetadata(Object)**.

- **remoteCache.getBulk, getBulk(size)** - Both of these methods have proven expensive for large data sets. The **remoteCache.retrieveEntries*** methods offer a lazy, pull-style, approach which retrieves bulk data more efficiently.

4.2.5. Interceptors Deprecated

Support for custom interceptors is being deprecated in JBoss Data Grid 7.0. A new method of executing custom interceptors is expected to be introduced in JBoss Data Grid 7.1. In addition, the interceptor stack is part of JBoss Data Grid's internal API, and is subject to change from release to release. Due to this it is not recommended to use custom interceptors directly from your application.

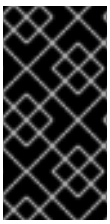
4.2.6. Map/Reduce replaced by Distributed Streams

Map/Reduce has been removed in JBoss Data Grid 7.0.0. This feature has been replaced by Distributed Streams, which were shown to have better performance.

4.2.7. Hibernate Search

4.2.7.1. Hibernate Search

The underlying Hibernate Search modules have been upgraded from 4.6 to 5.6, and as such there have been significant changes to this component. The following sections walk through all updates necessary to use the new version.



IMPORTANT

As the Hibernate Search modules have been updated, it is necessary to delete all existing indexes and rebuild the index, as indexes produced by previous versions are not compatible with the new version in use by JBoss Data Grid 7.0. Instructions for rebuilding the index are found in the *Developer Guide* .

4.2.7.2. Hibernate Search Class Name and Package Updates

Several classes have been relocated to a different package than their previous package; the following table includes the original class name and package along with the new location.

Table 4.1. Hibernate Search Class Package Updates

JDG 6.x Location	JDG 7.0 Location
org.hibernate.search.engine.impl.SearchMappingBuilder	org.hibernate.search.engine.spi.SearchMappingBuilder
org.hibernate.search.Environment	org.hibernate.search.cfg.Environment
org.hibernate.search.FullTextFilter	org.hibernate.search.filter.FullTextFilter
org.hibernate.search.indexes.impl.DirectoryBasedIndexManager	org.hibernate.search.indexes.spi.DirectoryBasedIndexManager
org.hibernate.search.infinispan.impl.InfinispanDirectoryProvider	org.hibernate.search.infinispan.spi.InfinispanDirectoryProvider
org.hibernate.search.ProjectionConstants	org.hibernate.search.engine.ProjectionConstants
org.hibernate.search.SearchException	org.hibernate.search.exception.SearchException
org.hibernate.search.spi.MassIndexerFactory	org.hibernate.search.batchindexing.spi.MassIndexerFactory
org.hibernate.search.spi.SearchFactoryBuilder	org.hibernate.search.spi.SearchIntegratorBuilder
org.hibernate.search.spi.SearchFactoryIntegrator	org.hibernate.search.spi.SearchIntegrator
org.hibernate.search.Version	org.hibernate.search.engine.Version
org.apache.lucene.queryParser.QueryParser	org.apache.lucene.queryparser.classic.QueryParser

Table 4.2. Hibernate Search Class Name Updates

JDG 6.x Name	JDG 7.0 Name
SpatialMode.GRID	SpatialMode.HASH

4.2.7.3. Hibernate Search Additions

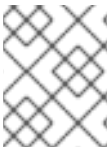
The following additions have been made to the Hibernate Search modules:

- A new method, **getShardIdentifiersForDeletion()**, has been added to **org.hibernate.search.store.ShardIdentifierProvider**. Any existing implementations which were not derived from **ShardIdentifierProviderTemplate** should be updated.
- The Faceting engine has been significantly updated; however, a **@Facet** or **@Facets** annotation must be used on any fields intended for faceting.

4.2.7.4. Hibernate Search Lucene Updates

Number and Date Index Format

Numbers and dates are now indexed as numeric fields by default. Properties of type **Date**, **Calendar**, **int**, **long**, **float**, **double**, and their corresponding wrappers, are no longer indexed as strings; instead, they use Lucene's appropriate numeric encoding. If any query previously targeted a string encoded field, but is not encoded numerically, it will need to be updated. Numeric fields must be searched with a **NumericRangeQuery**; however, if the Search query DSL is in use it should generate the correct query. In addition, any fields targeted by faceting should now be string encoded.



NOTE

Date and **Calendar** instances are encoded as a **long** value, representing the number of milliseconds since **January 1, 1970, 00:00:00 GMT**.

Any **id** fields are an exception to this rule. Even when these fields are represented by a numeric type they will continue to be indexed as a string keyword by default.

The changes mean the use of **@NumericField** is now obsolete, unless a custom precision for the numeric encoding is explicitly specified.

Previous, string based, index format may be used by specifying a string encoding field bridge. For integers this is **org.hibernate.search.bridge.builtin.IntegerBridge**; other publicly available field bridges may be found in the **org.hibernate.search.bridge.builtin** package.

For dates and calendars the indexing format may be changed via the **EncodingType** enum; for example, **@DateBridge(encoding=EncodingType.STRING)** or **@CalendarBridge(encoding=EncodingType.STRING)**.

Id fields of embedded relations are no longer indexed by default

When using an **@IndexedEmbedded** annotation to include fields from a related entity, previously the **id** of the related entity was included. These fields are no longer included by default, but may be enabled by using the **includeEmbeddedObjectId** attriubte of the **@IndexedEmbedded** annotation, as seen below:

```
@IndexedEmbedded(includeEmbeddedObjectId=true)
```

Sorting Query Results

The **SortField** API has been modified to use **SortField.Type.${CLASSNAME}** instead of the previous **SortField.${CLASSNAME}**. This behavior is highlighted below:

```
// Previous API call
fulltextQuery.setSort( new Sort( new SortField( "title", SortField.STRING ) ) );

// New API call
fulltextQuery.setSort( new Sort( new SortField( "title", SortField.Type.STRING ) ) );
```

Any applications not precisely matching the Lucene field encoding for sorting options in Lucene 4 would have worked without error; however, the new version of Lucene 5 now correctly catches these errors and will report a runtime exception. The expected field encoding must be matched according to its type. A field indexed as a **String** requires a single term, even after analysis, a numeric field encoded as an Integer must match **SortField.Type.INT**, a numeric field encoded as a Long must be sorted with **SortField.Type.LONG**, and so forth.

Faceting on Numeric Fields

Similar to Sorting Query Results, each target field must match an appropriate kind of Faceting Query. For instance, a Numeric Facet Query must match a field that was indexed exclusively via the matching Numeric type.

Null value tokens for Numeric Fields

When using **@Field(indexNullAs=)** to encode a null marker value in the index, the type of marker must be compatible with all other values which are indexed in the same field. Previously a string keyword, such as **null**, could be encoded with numeric fields; however, this is no longer allowed. A number must now be used to represent the null value, such as **-1**.

Lucene Classes Moved

Many of the **Analyzer**, **TokenFilter**, and **CharFilter** classes have been moved, and it is recommended to examine Lucene's API documentation to find the correct replacement.

Similarly, many of the Solr utility classes, such as **TokenizerFactory** or **TokenFilterFactory** were moved into Apache Lucene. Any applications using these classes should also be updated using the Lucene's API documentation.

Due to this refactoring the Solr dependencies are now incorporated within Lucene, and no longer need to be specified; any applications that previously had Solr dependencies defined should remove them and only use the Lucene dependencies.

4.2.7.5. Hibernate Search Deprecated Classes

The following list are deprecated classes within the Hibernate Search modules:

- **org.hibernate.search.annotations.Key** - there is no replacement for this class; however, it is no longer required for users to provide a **Key** object.
- **ContainedInMapping#numericField()** - Use **field().numericField()** instead.