Red Hat JBoss Data Grid 6.2

Administration and Configuration Guide

For use with Red Hat JBoss Data Grid 6.2.1

Last Updated: 2017-11-21

Red Hat JBoss Data Grid 6.2 Administration and Configuration Guide

For use with Red Hat JBoss Data Grid 6.2.1

Misha Husnain Ali
Red Hat Engineering Content Services
mhusnain@redhat.com

Gemma Sheldon
Red Hat Engineering Content Services
gsheldon@redhat.com

Mandar Joshi
Red Hat Engineering Content Services
majoshi@redhat.com

Rakesh Ghatvisave
Red Hat Engineering Content Services
rghatvis@redhat.com

Legal Notice

Abstract

This guide presents information about the administration and configuration of Red Hat JBoss Data Grid 6.2.1

Table of Contents

CHAPTER 1. SETTING UP RED HAT JBOSS DATA GRID

1.1. PREREQUISITES

The only prerequisites to set up Red Hat JBoss Data Grid is a Java Virtual Machine and that the most recent supported version of the product is installed on your system.

Report a bug

1.2. STEPS TO SET UP RED HAT JBOSS DATA GRID

The following steps outline the necessary (and optional, where stated) steps for a first time basic configuration of Red Hat JBoss Data Grid. It is recommended that the steps are followed in the order specified and not skipped unless they are identified as optional steps.

Procedure 1.1. Set Up JBoss Data Grid

1. **Set Up the Cache Manager**
   The first step in a JBoss Data Grid configuration is a cache manager. Cache managers can retrieve cache instances and create cache instances quickly and easily using previously specified configuration templates. For details about setting up a cache manager, see Part I, “Set Up a Cache Manager”

2. **Set Up JVM Memory Management**
   An important step in configuring your JBoss Data Grid is to set up memory management for your Java Virtual Machine (JVM). JBoss Data Grid offers features such as eviction and expiration to help manage the JVM memory.

   a. **Set Up Eviction**
      Use eviction to specify the logic used to remove entries from the in-memory cache implementation based on how often they are used. JBoss Data Grid offers different eviction strategies for finer control over entry eviction in your data grid. Eviction strategies and instructions to configure them are available in Chapter 3, *Set Up Eviction*.

   b. **Set Up Expiration**
      To set upper limits to an entry's time in the cache, attach expiration information to each entry. Use expiration to set up the maximum period an entry is allowed to remain in the cache and how long the retrieved entry can remain idle before being removed from the cache. For details, see Chapter 4, *Set Up Expiration*

3. **Monitor Your Cache**
   JBoss Data Grid uses logging via JBoss Logging to help users monitor their caches.

   a. **Set Up Logging**
      It is not mandatory to set up logging for your JBoss Data Grid, but it is highly recommended. JBoss Data Grid uses JBoss Logging, which allows the user to easily set up automated logging for operations in the data grid. Logs can subsequently be used to troubleshoot errors and identify the cause of an unexpected failure. For details, see Chapter 5, *Set Up Logging*

4. **Set Up Cache Modes**
   Cache modes are used to specify whether a cache is local (simple, in-memory cache) or a clustered cache (replicates state changes over a small subset of nodes). Additionally, if a cache is clustered, either replication, distribution or invalidation mode must be applied to

determine how the changes propagate across the subset of nodes. For details, see Part IV, "Set Up Cache Modes"

5. **Set Up Locking for the Cache**
   When replication or distribution is in effect, copies of entries are accessible across multiple nodes. As a result, copies of the data can be accessed or modified concurrently by different threads. To maintain consistency for all copies across nodes, configure locking. For details, see Part V, "Set Up Locking for the Cache" and Chapter 12, *Set Up Isolation Levels*

6. **Set Up and Configure a Cache Store**
   JBoss Data Grid offers the passivation feature (or cache writing strategies if passivation is turned off) to temporarily store entries removed from memory in a persistent, external cache store. To set up passivation or a cache writing strategy, you must first set up a cache store.

   a. **Set Up a Cache Store**
      The cache store serves as a connection to the persistent store. Cache stores are primarily used to fetch entries from the persistent store and to push changes back to the persistent store. For details, see Part VI, "Set Up and Configure a Cache Store"

   b. **Set Up Passivation**
      Passivation stores entries evicted from memory in a cache store. This feature allows entries to remain available despite not being present in memory and prevents potentially expensive write operations to the persistent cache. For details, see Part VII, "Set Up Passivation"

   c. **Set Up a Cache Writing Strategy**
      If passivation is disabled, every attempt to write to the cache results in writing to the cache store. This is the default Write-Through cache writing strategy. Set the cache writing strategy to determine whether these cache store writes occur synchronously or asynchronously. For details, see Part VIII, "Set Up Cache Writing"

7. **Monitor Caches and Cache Managers**
   JBoss Data Grid includes two primary tools to monitor the cache and cache managers once the data grid is up and running.

   a. **Set Up JMX**
      JMX is the standard statistics and management tool used for JBoss Data Grid. Depending on the use case, JMX can be configured at a cache level or a cache manager level or both. For details, see Chapter 19, *Set Up Java Management Extensions (JMX)*

   b. **Set Up Red Hat JBoss Operations Network (JON)**
      Red Hat JBoss Operations Network (JON) is the second monitoring solution available for JBoss Data Grid. JBoss Operations Network (JON) offers a graphical interface to monitor runtime parameters and statistics for caches and cache managers. For details, see Chapter 20, *Set Up JBoss Operations Network (JON)*

8. **Introduce Topology Information**
   Optionally, introduce topology information to your data grid to specify where specific types of information or objects in your data grid are located. Server hinting is one of the ways to introduce topology information in JBoss Data Grid.

   a. **Set Up Server Hinting**
      When set up, server hinting provides high availability by ensuring that the original and backup copies of data are not stored on the same physical server, rack or data center. This is optional in cases such as a replicated cache, where all data is backed up on all servers, racks and data centers. For details, see Chapter 26, *High Availability Using Server Hinting*

The subsequent chapters detail each of these steps towards setting up a standard JBoss Data Grid configuration.

Report a bug

PART I. SET UP A CACHE MANAGER

CHAPTER 2. CACHE MANAGERS

A Cache Manager is the primary mechanism to retrieve a cache instance in Red Hat JBoss Data Grid, and is a starting point for using the cache.

In JBoss Data Grid, a cache manager is useful because:

- it creates multiple cache instances on demand using a provided standard.
- it retrieves existing cache instanced (i.e. caches that have already been created).

Report a bug

2.1. TYPES OF CACHE MANAGERS

Red Hat JBoss Data Grid offers the following Cache Managers:

- **EmbeddedCacheManager** is a cache manager that runs within the Java Virtual Machine (JVM) used by the client. Currently, JBoss Data Grid offers only the **DefaultCacheManager** implementation of the **EmbeddedCacheManager** interface.
- **RemoteCacheManager** is used to access remote caches. When started, the **RemoteCacheManager** instantiates connections to the Hot Rod server (or multiple Hot Rod servers). It then manages the persistent **TCP** connections while it runs. As a result, **RemoteCacheManager** is resource-intensive. The recommended approach is to have a single **RemoteCacheManager** instance for each Java Virtual Machine (JVM).

Report a bug

2.2. CREATING CACHEMANAGERS

2.2.1. Create a New RemoteCacheManager

Use the following configuration to configure a new **RemoteCacheManager**:

```
import org.infinispan.client.hotrod.configuration.Configuration;
import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;

Configuration conf = new
ConfigurationBuilder().addServer().host("localhost").port(11222).build();
RemoteCacheManager manager = new RemoteCacheManager(conf);
RemoteCache defaultCache = manager.getCache();
```

Configuration Explanation

An explanation of each line of the provided configuration is as follows:

1. Use the **ConfigurationBuilder()** method to configure a new builder. The ***.addServer()*** property adds a remote server, specified via the ***.host(<hostname|ip>)*** and ***.port(<port>)*** properties.

   ```
   Configuration conf = new
   ConfigurationBuilder().addServer().host(<hostname|ip>).port(<port>).
   build();
   ```

2. Create a new **`RemoteCacheManager`** using the supplied configuration.

```
RemoteCacheManager manager = new RemoteCacheManager(conf);
```

3. Retrieve the default cache from the remote server.

```
RemoteCache defaultCache = manager.getCache();
```

Report a bug

2.2.2. Create a New Embedded Cache Manager

Use the following instructions to create a new EmbeddedCacheManager without using CDI:

Procedure 2.1. Create a New Embedded Cache Manager

1. Create a configuration XML file. For example, create the **`my-config.file.xml`** file on the classpath (in the **`resources/`** folder) and add the configuration information in this file.

2. Use the following programmatic configuration to create a cache manager using the configuration file:

```
EmbeddedCacheManager manager = new DefaultCacheManager("my-config-
file.xml");
Cache defaultCache = manager.getCache();
```

The outlined procedure creates a new EmbeddedCacheManager using the basic configuration specified.

Report a bug

2.2.3. Create a New Embedded Cache Manager Using CDI

Use the following steps to create a new EmbeddedCacheManager instance using CDI:

Procedure 2.2. Use CDI to Create a New EmbeddedCacheManager

1. Specify a default configuration:

```
public class Config {
    @Produces
    public Configuration defaultEmbeddedConfiguration () {
        return new ConfigurationBuilder()
                        .eviction()
                                .strategy(EvictionStrategy.LRU)
                                .maxEntries(100)
                        .build();
    }
}
```

2. Create a clustered or a non-clustered cache.

3. Invoke the method to create an EmbeddedCacheManager.

```
...
@Inject
EmbeddedCacheManager cacheManager;
...
```

Report a bug

2.3. MULTIPLE CACHE MANAGERS

Cache managers are an entry point to the cache and Red Hat JBoss Data Grid allows users to create multiple cache managers. Each cache manager is configured with a different global configuration, which includes settings for things like JMX, executors and clustering.

Report a bug

2.3.1. Create Multiple Caches with a Single Cache Manager

Red Hat JBoss Data Grid allows using the same cache manager to create multiple caches, each with a different cache mode (synchronous and asynchronous cache modes).

Report a bug

2.3.2. Using Multiple Cache Managers

Red Hat JBoss Data Grid allows multiple cache managers to be used. In most cases, such as with replication and networking components, cache instances share internal components and a single cache manager is sufficient.

However, if multiple caches are required to have different network characteristics, for example if one cache uses the **`TCP`** protocol and the other uses the **`UDP`** protocol, multiple cache managers must be used.

Report a bug

2.3.3. Create Multiple Cache Managers

Red Hat JBoss Data Grid allows users to create multiple cache managers of various types by repeating the procedure used to create the first cache manager (and adjusting the contents of the configuration file, if required).

To use the declarative API to create multiple new cache managers, copy the contents of the **`infinispan.xml`** file to a new configuration file. Edit the new file for the desired configuration and then use the new file for a new cache manager.

Report a bug

PART II. SET UP JVM MEMORY MANAGEMENT

CHAPTER 3. SET UP EVICTION

3.1. ABOUT EVICTION

Eviction is the process of removing entries from memory to prevent running out of memory. Entries that are evicted from memory remain in cache stores and the rest of the cluster to prevent permanent data loss.

Red Hat JBoss Data Grid executes eviction tasks by utilizing user threads which are already interacting with the data container. JBoss Data Grid uses a separate thread to prune expired cache entries from the cache.

Eviction occurs individually on a per node basis, rather than occurring as a cluster-wide operation. Each node uses an eviction thread to analyze the contents of its in-memory container to determine which entries require eviction. The free memory in the Java Virtual Machine (JVM) is not a consideration during the eviction analysis, even as a threshold to initialize entry eviction.

In JBoss Data Grid, eviction provides a mechanism to efficiently remove entries from the in-memory representation of a cache and push them to a persistent store. This ensures that the memory can always accommodate new entries as they are fetched and that evicted entries are preserved in the cluster instead of lost.

Additionally, eviction strategies can be used as required for your configuration to set up which entries are evicted and when eviction occurs.

See Also:

- Section 4.3, “Eviction and Expiration Comparison”

Report a bug

3.2. EVICTION STRATEGIES

Each eviction strategy has specific benefits and use cases, as outlined below:

Table 3.1. Eviction Strategies

Strategy Name	Operations	Details
`EvictionStrategy.NONE`	No eviction occurs.	-
`EvictionStrategy.LRU`	Least Recently Used eviction strategy. This strategy evicts entries that have not been used for the longest period. This ensures that entries that are reused periodically remain in memory.	

Strategy Name	Operations	Details
`EvictionStrategy.UNORDERED`	Unordered eviction strategy. This strategy evicts entries without any ordered algorithm and may therefore evict entries that are required later. However, this strategy saves resources because no algorithm related calculations are required before eviction.	This strategy is recommended for testing purposes and not for a real work implementation.
`EvictionStrategy.LIRS`	Low Inter-Reference Recency Set eviction strategy.	LIRS is Red Hat JBoss Data Grid's default eviction algorithm because it suits a large variety of production use cases.

Report a bug

3.2.1. LRU Eviction Algorithm Limitations

In the Least Recently Used (LRU) eviction algorithm, the least recently used entry is evicted first. The entry that has not been accessed the longest gets evicted first from the cache. However, LRU eviction algorithm sometimes does not perform optimally in cases of weak access locality. The weak access locality is a technical term used for entries which are put in the cache and not accessed for a long time and entries to be accessed soonest are replaced. In such cases, problems such as the following can appear:

- Single use access entries are not replaced in time.
- Entries that are accessed first are unnecessarily replaced.

Report a bug

3.3. USING EVICTION

In Red Hat JBoss Data Grid, eviction is disabled by default. If an empty `<eviction />` element is used to enable eviction without any strategy or maximum entries settings, the following default values are automatically implemented:

- Strategy: If no eviction strategy is specified, ***`EvictionStrategy.NONE`*** is assumed as a default.
- max-entries/maxEntries: If no value is specified, the ***`max-entries`***/maxEntries value is set to **`-1`**, which allows unlimited entries.

Report a bug

3.3.1. Initialize Eviction

To initialize eviction, set the eviction element's ***`max-entries`*** attributes value to a number greater than zero. Adjust the value set for ***`max-entries`*** to discover the optimal value for your configuration. It is important to remember that if too large a value is set for ***`max-entries`***, Red Hat JBoss Data Grid runs out of memory.

The following procedure outlines the steps to initialize eviction in JBoss Data Grid:

Procedure 3.1. Initialize Eviction

1. **Add the Eviction Tag**
   Add the <eviction> tag to your project's <cache> tags as follows:

   ```
   <eviction />
   ```

2. **Set the Eviction Strategy**
   Set the ***strategy*** value to set the eviction strategy employed. Possible values are **LRU**, **UNORDERED** and **LIRS** (or **NONE** if no eviction is required). The following is an example of this step:

   ```
   <eviction strategy="LRU" />
   ```

3. **Set the Maximum Entries**
   Set the maximum number of entries allowed in memory. The default value is **-1** for unlimited entries.

   a. In Library mode, set the ***maxEntries*** parameter as follows:

      ```
      <eviction strategy="LRU" maxEntries="200" />
      ```

   b. In Remote Client Server mode, set the ***max-entries*** as follows:

      ```
      <eviction strategy="LRU" max-entries="200" />
      ```

Result

Eviction is configured for the target cache.

Report a bug

3.3.2. Eviction Configuration Examples

Configure eviction in Red Hat JBoss Data Grid using the configuration bean or the XML file. Eviction configuration is done on a per-cache basis.

- A sample XML configuration for Library mode is as follows:

  ```
  <eviction strategy="LRU" maxEntries="2000"/>
  ```

- A sample XML configuration for Remote Client Server Mode is as follows:

  ```
  <eviction strategy="LRU" max-entries="20"/>
  ```

- A sample programmatic configuration for Library Mode is as follows:

  ```
  Configuration c = new
  ```

```
ConfigurationBuilder().eviction().strategy(EvictionStrategy.LRU)
               .maxEntries(2000)
               .build();
```

NOTE

JBoss Data Grid's Library mode uses the ***maxEntries*** parameter while Remote Client-Server mode uses the ***max-entries*** parameter to configure eviction.

Report a bug

3.3.3. Eviction Configuration Troubleshooting

In Red Hat JBoss Data Grid, the size of a cache can be larger than the value specified for the ***max-entries*** parameter of the **`configuration`** element. This is because although the ***max-entries*** value can be configured to a value that is not a power of two, the underlying algorithm will alter the value to **V**, where **V** is the closest power of two value that is larger than the ***max-entries*** value. Eviction algorithms are in place to ensure that the size of the cache container will never exceed the value **V**.

Report a bug

3.3.4. Eviction and Passivation

To ensure that a single copy of an entry remains, either in memory or in a cache store, use passivation in conjunction with eviction.

The primary reason to use passivation instead of a normal cache store is that updating entries require less resources when passivation is in use. This is because passivation does not require an update to the cache store.

Report a bug

CHAPTER 4. SET UP EXPIRATION

4.1. ABOUT EXPIRATION

Red Hat JBoss Data Grid uses expiration to attach one or both of the following values to an entry:

- A lifespan value.
- A maximum idle time value.

Expiration can be specified on a per-entry or per-cache basis and the per-entry configuration overrides per-cache configurations. If expiration is not configured at the cache level, cache entries are created immortal (i.e. they will never expire) as a default. Conversely, if expiration is configured at the cache level, the expiration defaults apply to all entries which do not explicitly specify a ***`lifespan`*** or ***`maxIdle`*** value.

Expired entries, unlike evicted entries, are removed globally, which removes them from memory, cache stores and the cluster.

Expiration automates the removal of entries that have not been used for a specified period of time from the memory. Expiration and eviction are different because:

- expiration removes entries based on the period they have been in memory. Expiration only removes entries when the life span period concludes or when an entry has been idle longer than the specified idle time.
- eviction removes entries based on how recently (and often) they are used. Eviction only removes entries when too many entries are present in the memory. If a cache store has been configured, evicted entries are persisted in the cache store.

Report a bug

4.2. EXPIRATION OPERATIONS

Expiration in Red Hat JBoss Data Grid allows you to set a life span or maximum idle time value for each key/value pair stored in the cache.

The life span or maximum idle time can be set to apply cache-wide or defined for each key/value pair using the cache API. The life span (***`lifespan`***) or maximum idle time (***`maxIdle`*** in Library Mode and ***`max-idle`*** in Remote Client-Server Mode) defined for an individual key/value pair overrides the cache-wide default for the entry in question.

Report a bug

4.3. EVICTION AND EXPIRATION COMPARISON

Expiration is a top-level construct in Red Hat JBoss Data Grid, and is represented in the global configuration, as well as the cache API.

Eviction is limited to the cache instance it is used in, whilst expiration is cluster-wide. Expiration life spans (***`lifespan`***) and idle time (***`maxIdle`*** in Library Mode and ***`max-idle`*** in Remote Client-Server Mode) values are replicated alongside each cache entry.

Report a bug

4.4. CACHE ENTRY EXPIRATION NOTIFICATIONS

Red Hat JBoss Data Grid does not guarantee that an eviction occurs immediately upon timeout. Instead, a number of mechanisms are used in collaboration to ensure efficient eviction. An expired entry is removed from the cache when either:

- A user thread requests an entry and discovers that the entry has expired.
- An entry is passivated/overflowed to disk and is discovered to have expired.
- The eviction maintenance thread discovers that an entry it has found is expired.

Report a bug

4.5. CONFIGURE EXPIRATION

In Red Hat JBoss Data Grid, expiration is configured in a manner similar to eviction.

Procedure 4.1. Configure Expiration

1. **Add the Expiration Tag**
   Add the <expiration> tag to your project's <cache> tags as follows:

   ```
   <expiration />
   ```

2. **Set the Expiration Lifespan**
   Set the ***lifespan*** value to set the period of time (in milliseconds) an entry can remain in memory. The following is an example of this step:

   ```
   <expiration lifespan="1000" />
   ```

3. **Set the Maximum Idle Time**
   Set the time that entries are allowed to remain idle (unused) after which they are removed (in milliseconds). The default value is `-1` for unlimited time.

   a. In Library mode, set the ***maxIdle*** parameter as follows:

   ```
   <expiration lifespan="1000" maxIdle="1000" />
   ```

   - In Remote Client Server mode, set the ***max-idle*** as follows:

     ```
     <expiration lifespan="1000" max-idle="1000" />
     ```

Result

Expiration is now configured for the cache implementation.

Report a bug

4.6. MORTAL AND IMMORTAL DATA

In addition to storing entities, Red Hat JBoss Data Grid allows you to attach mortality information to data. For example, using the standard `put(key, value)` creates an entry that will never expire, called an immortal entry. Alternatively, an entry created using **`put(key, value, lifespan, timeunit)`** is a mortal entry that has a specified fixed life span, after which it expires.

In addition to the ***`lifespan`*** parameter, JBoss Data Grid also provides a ***`maxIdle`*** parameter used to determine expiration. The ***`maxIdle`*** and ***`lifespan`*** parameters can be used in various combinations to set the life span of an entry.

As a default, newly created entries do not have a life span or maximum idle time value set. Without these two values, a data entry will never expire and is therefore known as immortal data.

Entry mortality (or its expiration values) can be set by setting the life span and maximum idle time values for the entry. After being set, these values must be persisted in the cache stores to ensure that they survive eviction and passivation.

Report a bug

4.7. TROUBLESHOOTING EXPIRATION

If expiration does not appear to be working, it may be due to an entry being marked for expiration but not being removed.

Multiple-cache operations such as **`put()`** are passed a life span value as a parameter. This value defines the interval after which the entry must expire. In cases where eviction is not configured and the life span interval expires, it can appear as if Red Hat JBoss Data Grid has not removed the entry. For example, when viewing JMX statistics, such as the **`number of entries`**, you may see an out of date count, or the persistent store associated with JBoss Data Grid may still contain this entry. Behind the scenes, JBoss Data Grid has marked it as an expired entry, but has not removed it. Removal of such entries happens in one of two ways:

- Any attempt to use **`get()`** or **`containsKey()`** for the expired entry, causes JBoss Data Grid to detect the entry as an expired one and remove it.
- Enabling the eviction feature causes the eviction thread to periodically detect and purge expired entries.

Report a bug

PART III. MONITOR YOUR CACHE

CHAPTER 5. SET UP LOGGING

5.1. ABOUT LOGGING

Red Hat JBoss Data Grid provides highly configurable logging facilities for both its own internal use and for use by deployed applications. The logging subsystem is based on JBoss LogManager and it supports several third party application logging frameworks in addition to JBoss Logging.

The logging subsystem is configured using a system of log categories and log handlers. Log categories define what messages to capture, and log handlers define how to deal with those messages (write to disk, send to console, etc).

After a JBoss Data Grid cache is configured with operations such as eviction and expiration, logging tracks relevant activity (including errors or failures).

When set up correctly, logging provides a detailed account at what occurred in the environment and when. Logging also helps track activity that occurred just before a crash or problem in the environment. This information is useful when troubleshooting or when attempting to identify the source of a crash or error.

Report a bug

5.2. SUPPORTED APPLICATION LOGGING FRAMEWORKS

Red Hat JBoss LogManager supports the following logging frameworks:

- JBoss Logging, which is included with Red Hat JBoss Data Grid 6.
- Apache Commons Logging
- Simple Logging Facade for Java (SLF4J)
- Apache log4j
- Java SE Logging (java.util.logging)

Report a bug

5.2.1. About JBoss Logging

JBoss Logging is the application logging framework that is included in
JBoss Enterprise Application Platform 6. As a result of this inclusion, Red Hat JBoss Data Grid 6 also uses JBoss Logging.

JBoss Logging provides an easy way to add logging to an application. Add code to the application that uses the framework to send log messages in a defined format. When the application is deployed to an application server, these messages can be captured by the server and displayed and/or written to file according to the server's configuration.

Report a bug

5.2.2. JBoss Logging Features

JBoss Logging includes the following features:

- Provides an innovative, easy to use *typed* logger.
- Full support for internationalization and localization. Translators work with message bundles in properties files while developers can work with interfaces and annotations.
- Build-time tooling to generate typed loggers for production, and runtime generation of typed loggers for development.

Report a bug

5.3. BOOT LOGGING

The boot log is the record of events that occur while the server is starting up (or booting). Red Hat JBoss Data Grid also includes a server log, which includes log entries generated after the server concludes the boot process.

Report a bug

5.3.1. Configure Boot Logging

Edit the **`logging.properties`** file to configure the boot log. This file is a standard Java properties file and can be edited in a text editor. Each line in the file has the format of **`property=value`**.

In Red Hat JBoss Data Grid, the **`logging.properties`** file is available in the **`$JDG_HOME/standalone/configuration`** folder.

Report a bug

5.3.2. Default Log File Locations

The following table provides a list of log files in Red Hat JBoss Data Grid and their locations:

Table 5.1. Default Log File Locations

Log File	Location	Description
`boot.log`	**`$JDG_HOME/standalone/log/`**	The Server Boot Log. Contains log messages related to the start up of the server.
`server.log`	**`$JDG_HOME/standalone/log/`**	The Server Log. Contains all log messages once the server has launched.

Report a bug

5.4. LOGGING ATTRIBUTES

5.4.1. About Log Levels

Log levels are an ordered set of enumerated values that indicate the nature and severity of a log message. The level of a given log message is specified by the developer using the appropriate methods of their chosen logging framework to send the message.

Red Hat JBoss Data Grid supports all the log levels used by the supported application logging frameworks. The six most commonly used log levels are (ordered by lowest to highest severity):

1. **TRACE**
2. **DEBUG**
3. **INFO**
4. **WARN**
5. **ERROR**
6. **FATAL**

Log levels are used by log categories and handlers to limit the messages they are responsible for. Each log level has an assigned numeric value which indicates its order relative to other log levels. Log categories and handlers are assigned a log level and they only process log messages of that numeric value or higher. For example a log handler with the level of **WARN** will only record messages of the levels **WARN**, **ERROR** and **FATAL**.

Report a bug

5.4.2. Supported Log Levels

The following table lists log levels that are supported in Red Hat JBoss Data Grid. Each entry includes the log level, its value and description. The log level values indicate each log level's relative value to other log levels. Additionally, log levels in different frameworks may be named differently, but have a log value consistent to the provided list.

Table 5.2. Supported Log Levels

Log Level	Value	Description
FINEST	300	-
FINER	400	-
TRACE	400	Used for messages that provide detailed information about the running state of an application. **TRACE** level log messages are captured when the server runs with the **TRACE** level enabled.

Log Level	Value	Description
DEBUG	500	Used for messages that indicate the progress of individual requests or activities of an application. **DEBUG** level log messages are captured when the server runs with the **DEBUG** level enabled.
FINE	500	-
CONFIG	700	-
INFO	800	Used for messages that indicate the overall progress of the application. Used for application start up, shut down and other major lifecycle events.
WARN	900	Used to indicate a situation that is not in error but is not considered ideal. Indicates circumstances that can lead to errors in the future.
WARNING	900	-
ERROR	1000	Used to indicate an error that has occurred that could prevent the current activity or request from completing but will not prevent the application from running.
SEVERE	1000	-
FATAL	1100	Used to indicate events that could cause critical service failure and application shutdown and possibly cause JBoss Data Grid to shut down.

Report a bug

5.4.3. About Log Categories

Log categories define a set of log messages to capture and one or more log handlers which will process the messages.

The log messages to capture are defined by their Java package of origin and log level. Messages from classes in that package and of that log level or higher (with greater or equal numeric value) are captured by the log category and sent to the specified log handlers. As an example, the **`WARNING`** log level results in log values of **`900`**, **`1000`** and **`1100`** are captured.

Log categories can optionally use the log handlers of the root logger instead of their own handlers.

Report a bug

5.4.4. About the Root Logger

The root logger captures all log messages sent to the server (of a specified level) that are not captured by a log category. These messages are then sent to one or more log handlers.

By default the root logger is configured to use a console and a periodic log handler. The periodic log handler is configured to write to the file **`server.log`**. This file is sometimes referred to as the server log.

Report a bug

5.4.5. About Log Handlers

Log handlers define how captured log messages are recorded by Red Hat JBoss Data Grid. The six types of log handlers configurable in JBoss Data Grid are:

- **`Console`**
- **`File`**
- **`Periodic`**
- **`Size`**
- **`Async`**
- **`Custom`**

Log handlers direct specified log objects to a variety of outputs (including the console or specified log files). Some log handlers used in JBoss Data Grid are wrapper log handlers, used to direct other log handlers' behavior.

Log handlers are used to direct log outputs to specific files for easier sorting or to write logs for specific intervals of time. They are primarily useful to specify the kind of logs required and where they are stored or displayed or the logging behavior in JBoss Data Grid.

Report a bug

5.4.6. Log Handler Types

The following table lists the different types of log handlers available in Red Hat JBoss Data Grid:

Table 5.3. Log Handler Types

Log Handler Type	Description	Use Case

Log Handler Type	Description	Use Case
Console	Console log handlers write log messages to either the host operating system’s standard out (`stdout`) or standard error (`stderr`) stream. These messages are displayed when JBoss Data Grid is run from a command line prompt.	The Console log handler is preferred when JBoss Data Grid is administered using the command line. In such a case, the messages from a Console log handler are not saved unless the operating system is configured to capture the standard out or standard error stream.
File	File log handlers are the simplest log handlers. Their primary use is to write log messages to a specified file.	File log handlers are most useful if the requirement is to store all log entries according to the time in one place.
Periodic	Periodic file handlers write log messages to a named file until a specified period of time has elapsed. Once the time period has elapsed, the specified time stamp is appended to the file name. The handler then continues to write into the newly created log file with the original name.	The Periodic file handler can be used to accumulate log messages on a weekly, daily, hourly or other basis depending on the requirements of the environment.
Size	Size log handlers write log messages to a named file until the file reaches a specified size. When the file reaches a specified size, it is renamed with a numeric prefix and the handler continues to write into a newly created log file with the original name. Each size log handler must specify the maximum number of files to be kept in this fashion.	The Size handler is best suited to an environment where the log file size must be consistent.
Async	Async log handlers are wrapper log handlers that provide asynchronous behavior for one or more other log handlers. These are useful for log handlers that have high latency or other performance problems such as writing a log file to a network file system.	The Async log handlers are best suited to an environment where high latency is a problem or when writing to a network file system.

Log Handler Type	Description	Use Case
Custom	Custom log handlers enable to you to configure new types of log handlers that have been implemented. A custom handler must be implemented as a Java class that extends `java.util.logging.Handler` and be contained in a module.	Custom log handlers create customized log handler types and are recommended for advanced users.

Report a bug

5.4.7. Selecting Log Handlers

The following are the most common uses for each of the log handler types available for Red Hat JBoss Data Grid:

- The **`Console`** log handler is preferred when JBoss Data Grid is administered using the command line. In such a case, errors and log messages appear on the console window and are not saved unless separately configured to do so.
- The **`File`** log handler is used to direct log entries into a specified file. This simplicity is useful if the requirement is to store all log entries according to the time in one place.
- The **`Periodic`** log handler is similar to the **`File`** handler but creates files according to the specified period. As an example, this handler can be used to accumulate log messages on a weekly, daily, hourly or other basis depending on the requirements of the environment.
- The **`Size`** log handler also writes log messages to a specified file, but only while the log file size is within a specified limit. Once the file size reaches the specified limit, log files are written to a new log file. This handler is best suited to an environment where the log file size must be consistent.
- The **`Async`** log handler is a wrapper that forces other log handlers to operate asynchronously. This is best suited to an environment where high latency is a problem or when writing to a network file system.
- The **`Custom`** log handler creates new, customized types of log handlers. This is an advanced log handler.

Report a bug

5.4.8. About Log Formatters

A log formatter is the configuration property of a log handler. The log formatter defines the appearance of log messages that originate from the relevant log handler. The log formatter is a string that uses the same syntax as the **`java.util.Formatter`** class.

See http://docs.oracle.com/javase/6/docs/api/java/util/Formatter.html for more information.

Report a bug

5.5. LOGGING SAMPLE CONFIGURATIONS

5.5.1. Sample XML Configuration for the Root Logger

The following procedure demonstrates a sample configuration for the root logger.

Procedure 5.1. Configure the Root Logger

1. **Set the *`level`* Property**
   The ***`level`*** property sets the maximum level of log message that the root logger records.

   ```
   <subsystem xmlns="urn:jboss:domain:logging:1.2">
      <root-logger>
         <level name="INFO"/>
   ```

2. **List *`handlers`***
   `handlers` is a list of log handlers that are used by the root logger.

   ```
   <subsystem xmlns="urn:jboss:domain:logging:1.2">
        <root-logger>
           <level name="INFO"/>
           <handlers>
              <handler name="CONSOLE"/>
              <handler name="FILE"/>
           </handlers>
        </root-logger>
     </subsystem>
   ```

Report a bug

5.5.2. Sample XML Configuration for a Log Category

The following procedure demonstrates a sample configuration for a log category.

Procedure 5.2. Configure a Log Category

1. **Define the Category**
   Use the ***`category`*** property to specify the log category from which log messages will be captured.

   The ***`use-parent-handlers`*** is set to **`"true"`** by default. When set to **`"true"`**, this category will use the log handlers of the root logger in addition to any other assigned handlers.

   ```
   <subsystem xmlns="urn:jboss:domain:logging:1.2">
      <logger category="com.company.accounts.rec" use-parent-
   handlers="true">
   ```

2. **Set the *`level`* property**
   Use the ***`level`*** property to set the maximum level of log message that the log category records.

   ```
   <subsystem xmlns="urn:jboss:domain:logging:1.2">
   ```

```
   <logger category="com.company.accounts.rec" use-parent-
handlers="true">
      <level name="WARN"/>
```

3. **List *handlers***
   handlers is a list of log handlers.

```
<subsystem xmlns="urn:jboss:domain:logging:1.2">
   <logger category="com.company.accounts.rec" use-parent-
handlers="true">
      <level name="WARN"/>
      <handlers>
         <handler name="accounts-rec"/>
      </handlers>
   </logger>
</subsystem>
```

Report a bug

5.5.3. Sample XML Configuration for a Console Log Handler

The following procedure demonstrates a sample configuration for a console log handler.

Procedure 5.3. Configure the Console Log Handler

1. **Add the Log Handler Identifier Information**
   The ***name*** property sets the unique identifier for this log handler.

   When ***autoflush*** is set to **"true"** the log messages will be sent to the handler's target immediately upon request.

```
<subsystem xmlns="urn:jboss:domain:logging:1.2">
   <console-handler name="CONSOLE" autoflush="true">
```

2. **Set the *level* Property**
   The ***level*** property sets the maximum level of log messages recorded.

```
<subsystem xmlns="urn:jboss:domain:logging:1.2">
   <console-handler name="CONSOLE" autoflush="true">
      <level name="INFO"/>
```

3. **Set the *encoding* Output**
   Use ***encoding*** to set the character encoding scheme to be used for the output.

```
<subsystem xmlns="urn:jboss:domain:logging:1.2">
   <console-handler name="CONSOLE" autoflush="true">
      <level name="INFO"/>
      <encoding value="UTF-8"/>
```

4. **Define the *target* Value**

The ***target*** property defines the system output stream where the output of the log handler goes. This can be **System.err** for the system error stream, or **System.out** for the standard out stream.

```
<subsystem xmlns="urn:jboss:domain:logging:1.2">
   <console-handler name="CONSOLE" autoflush="true">
      <level name="INFO"/>
      <encoding value="UTF-8"/>
      <target value="System.out"/>
```

5. **Define the *filter-spec* Property**
   The ***filter-spec*** property is an expression value that defines a filter. The example provided defines a filter that does not match a pattern: **not(match("JBAS.*"))**.

```
<subsystem xmlns="urn:jboss:domain:logging:1.2">
   <console-handler name="CONSOLE" autoflush="true">
      <level name="INFO"/>
      <encoding value="UTF-8"/>
      <target value="System.out"/>
      <filter-spec value="not(match(&quot;JBAS.*&quot;))"/>
```

6. **Specify the *formatter***
   Use ***formatter*** to list the log formatter used by the log handler.

```
<subsystem xmlns="urn:jboss:domain:logging:1.2">
   <console-handler name="CONSOLE" autoflush="true">
      <level name="INFO"/>
      <encoding value="UTF-8"/>
      <target value="System.out"/>
      <filter-spec value="not(match(&quot;JBAS.*&quot;))"/>
      <formatter>
         <pattern-formatter pattern="%K{level}%d{HH:mm:ss,SSS} %-5p
[%c] (%t) %s%E%n"/>
      </formatter>
   </console-handler>
</subsystem>
```

Report a bug

5.5.4. Sample XML Configuration for a File Log Handler

The following procedure demonstrates a sample configuration for a file log handler.

Procedure 5.4. Configure the File Log Handler

1. **Add the File Log Handler Identifier Information**
   The ***name*** property sets the unique identifier for this log handler.

   When ***autoflush*** is set to **"true"** the log messages will be sent to the handler's target immediately upon request.

```
<file-handler name="accounts-rec-trail" autoflush="true">
```

2. **Set the *level* Property**
   The ***level*** property sets the maximum level of log message that the root logger records.

   ```
   <file-handler name="accounts-rec-trail" autoflush="true">
       <level name="INFO"/>
   ```

3. **Set the *encoding* Output**
   Use ***encoding*** to set the character encoding scheme to be used for the output.

   ```
   <file-handler name="accounts-rec-trail" autoflush="true">
       <level name="INFO"/>
       <encoding value="UTF-8"/>
   ```

4. **Set the *file* Object**
   The ***file*** object represents the file where the output of this log handler is written to. It has two configuration properties: ***relative-to*** and ***path***.

   The ***relative-to*** property is the directory where the log file is written to. JBoss Enterprise Application Platform 6 file path variables can be specified here. The **jboss.server.log.dir** variable points to the **log/** directory of the server.

   The ***path*** property is the name of the file where the log messages will be written. It is a relative path name that is appended to the value of the ***relative-to*** property to determine the complete path.

   ```
   <file-handler name="accounts-rec-trail" autoflush="true">
       <level name="INFO"/>
       <encoding value="UTF-8"/>
       <file relative-to="jboss.server.log.dir" path="accounts-rec-
   trail.log"/>
   ```

5. **Specify the *formatter***
   Use ***formatter*** to list the log formatter used by the log handler.

   ```
   <file-handler name="accounts-rec-trail" autoflush="true">
       <level name="INFO"/>
       <encoding value="UTF-8"/>
       <file relative-to="jboss.server.log.dir" path="accounts-rec-
   trail.log"/>
       <formatter>
           <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
   %s%E%n"/>
       </formatter>
   ```

6. **Set the *append* Property**
   When the ***append*** property is set to **"true"**, all messages written by this handler will be appended to an existing file. If set to **"false"** a new file will be created each time the application server launches. Changes to ***append*** require a server reboot to take effect.

   ```
   <file-handler name="accounts-rec-trail" autoflush="true">
       <level name="INFO"/>
       <encoding value="UTF-8"/>
   ```

```
    <file relative-to="jboss.server.log.dir" path="accounts-rec-
trail.log"/>
    <formatter>
        <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
    </formatter>
    <append value="true"/>
</file-handler>
```

Report a bug

5.5.5. Sample XML Configuration for a Periodic Log Handler

The following procedure demonstrates a sample configuration for a periodic log handler.

Procedure 5.5. Configure the Periodic Log Handler

1. **Add the Periodic Log Handler Identifier Information**
   The ***name*** property sets the unique identifier for this log handler.

   When ***autoflush*** is set to **"true"** the log messages will be sent to the handler's target immediately upon request.

   ```
   <periodic-rotating-file-handler name="FILE" autoflush="true">
   ```

2. **Set the *level* Property**
   The ***level*** property sets the maximum level of log message that the root logger records.

   ```
   <periodic-rotating-file-handler name="FILE" autoflush="true">
      <level name="INFO"/>
   ```

3. **Set the *encoding* Output**
   Use ***encoding*** to set the character encoding scheme to be used for the output.

   ```
   <periodic-rotating-file-handler name="FILE" autoflush="true">
      <level name="INFO"/>
      <encoding value="UTF-8"/>
   ```

4. **Specify the *formatter***
   Use ***formatter*** to list the log formatter used by the log handler.

   ```
   <periodic-rotating-file-handler name="FILE" autoflush="true">
      <level name="INFO"/>
      <encoding value="UTF-8"/>
      <formatter>
         <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
   %s%E%n"/>
      </formatter>
   ```

5. **Set the *file* Object**
   The ***file*** object represents the file where the output of this log handler is written to. It has two configuration properties: ***relative-to*** and ***path***.

The ***relative-to*** property is the directory where the log file is written to. JBoss Enterprise Application Platform 6 file path variables can be specified here. The **jboss.server.log.dir** variable points to the **log/** directory of the server.

The ***path*** property is the name of the file where the log messages will be written. It is a relative path name that is appended to the value of the ***relative-to*** property to determine the complete path.

```
<periodic-rotating-file-handler name="FILE" autoflush="true">
   <level name="INFO"/>
   <encoding value="UTF-8"/>
   <formatter>
      <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
   </formatter>
   <file relative-to="jboss.server.log.dir" path="server.log"/>
```

6. **Set the *suffix* Value**
   The ***suffix*** is appended to the filename of the rotated logs and is used to determine the frequency of rotation. The format of the ***suffix*** is a dot (.) followed by a date string, which is parsable by the **java.text.SimpleDateFormat** class. The log is rotated on the basis of the smallest time unit defined by the ***suffix***. For example, **yyyy-MM-dd** will result in daily log rotation. See http://docs.oracle.com/javase/6/docs/api/index.html?java/text/SimpleDateFormat.html

```
<periodic-rotating-file-handler name="FILE" autoflush="true">
   <level name="INFO"/>
   <encoding value="UTF-8"/>
   <formatter>
      <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
   </formatter>
   <file relative-to="jboss.server.log.dir" path="server.log"/>
   <suffix value=".yyyy-MM-dd"/>
```

7. **Set the *append* Property**
   When the ***append*** property is set to **"true"**, all messages written by this handler will be appended to an existing file. If set to **"false"** a new file will be created each time the application server launches. Changes to ***append*** require a server reboot to take effect.

```
<periodic-rotating-file-handler name="FILE" autoflush="true">
   <level name="INFO"/>
   <encoding value="UTF-8"/>
   <formatter>
      <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
   </formatter>
   <file relative-to="jboss.server.log.dir" path="server.log"/>
   <suffix value=".yyyy-MM-dd"/>
   <append value="true"/>
</periodic-rotating-file-handler>
```

Report a bug

5.5.6. Sample XML Configuration for a Size Log Handler

The following procedure demonstrates a sample configuration for a size log handler.

Procedure 5.6. Configure the Size Log Handler

1. **Add the Size Log Handler Identifier Information**
   The ***name*** property sets the unique identifier for this log handler.

   When ***autoflush*** is set to **`"true"`** the log messages will be sent to the handler's target immediately upon request.

   ```
   <size-rotating-file-handler name="accounts_debug" autoflush="false">
   ```

2. **Set the *`level`* Property**
   The ***level*** property sets the maximum level of log message that the root logger records.

   ```
   <size-rotating-file-handler name="accounts_debug" autoflush="false">
      <level name="DEBUG"/>
   ```

3. **Set the *encoding* Output**
   Use ***encoding*** to set the character encoding scheme to be used for the output.

   ```
   <size-rotating-file-handler name="accounts_debug" autoflush="false">
      <level name="DEBUG"/>
      <encoding value="UTF-8"/>
   ```

4. **Set the *`file`* Object**
   The ***file*** object represents the file where the output of this log handler is written to. It has two configuration properties: ***relative-to*** and ***path***.

   The ***relative-to*** property is the directory where the log file is written to. JBoss Enterprise Application Platform 6 file path variables can be specified here. The **`jboss.server.log.dir`** variable points to the **`log/`** directory of the server.

   The ***path*** property is the name of the file where the log messages will be written. It is a relative path name that is appended to the value of the ***relative-to*** property to determine the complete path.

   ```
   <size-rotating-file-handler name="accounts_debug" autoflush="false">
      <level name="DEBUG"/>
      <encoding value="UTF-8"/>
      <file relative-to="jboss.server.log.dir" path="accounts-
   debug.log"/>
   ```

5. **Specify the *`rotate-size`* Value**
   The maximum size that the log file can reach before it is rotated. A single character appended to the number indicates the size units: **`b`** for bytes, **`k`** for kilobytes, **`m`** for megabytes, **`g`** for gigabytes. For example: **`50m`** for 50 megabytes.

   ```
   <size-rotating-file-handler name="accounts_debug" autoflush="false">
      <level name="DEBUG"/>
   ```

```
   <encoding value="UTF-8"/>
   <file relative-to="jboss.server.log.dir" path="accounts-
debug.log"/>
   <rotate-size value="500k"/>
```

6. **Set the *max-backup-index* Number**

   The maximum number of rotated logs that are kept. When this number is reached, the oldest log is reused.

```
<size-rotating-file-handler name="accounts_debug" autoflush="false">
   <level name="DEBUG"/>
   <encoding value="UTF-8"/>
   <file relative-to="jboss.server.log.dir" path="accounts-
debug.log"/>
   <rotate-size value="500k"/>
   <max-backup-index value="5"/>
```

7. **Specify the *formatter***

   Use ***formatter*** to list the log formatter used by the log handler.

```
<size-rotating-file-handler name="accounts_debug" autoflush="false">
   <level name="DEBUG"/>
   <encoding value="UTF-8"/>
   <file relative-to="jboss.server.log.dir" path="accounts-
debug.log"/>
   <rotate-size value="500k"/>
   <max-backup-index value="5"/>
   <formatter>
        <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
    </formatter>
```

8. **Set the *append* Property**

   When the ***append*** property is set to **"true"**, all messages written by this handler will be appended to an existing file. If set to **"false"** a new file will be created each time the application server launches. Changes to ***append*** require a server reboot to take effect.

```
<size-rotating-file-handler name="accounts_debug" autoflush="false">
   <level name="DEBUG"/>
   <encoding value="UTF-8"/>
   <file relative-to="jboss.server.log.dir" path="accounts-
debug.log"/>
   <rotate-size value="500k"/>
   <max-backup-index value="5"/>
   <formatter>
        <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
    </formatter>
   <append value="true"/>
</size-rotating-file-handler>
```

Report a bug

5.5.7. Sample XML Configuration for a Async Log Handler

The following procedure demonstrates a sample configuration for an async log handler

Procedure 5.7. Configure the Async Log Handler

1. **Add the Async Log Handler Identifier Information**
   The ***name*** property sets the unique identifier for this log handler.

   ```
   <async-handler name="Async_NFS_handlers">
   ```

2. **Set the *`level`* Property**
   The ***`level`*** property sets the maximum level of log message that the root logger records.

   ```
   <async-handler name="Async_NFS_handlers">
      <level name="INFO"/>
   ```

3. **Define the *`queue-length`***
   The ***`queue-length`*** defines the maximum number of log messages that will be held by this handler while waiting for sub-handlers to respond.

   ```
   <async-handler name="Async_NFS_handlers">
      <level name="INFO"/>
      <queue-length value="512"/>
   ```

4. **Set Overflow Response**
   The ***`overflow-action`*** defines how this handler responds when its queue length is exceeded. This can be set to **BLOCK** or **`DISCARD`**. **BLOCK** makes the logging application wait until there is available space in the queue. This is the same behavior as an non-async log handler. **`DISCARD`** allows the logging application to continue but the log message is deleted.

   ```
   <async-handler name="Async_NFS_handlers">
      <level name="INFO"/>
      <queue-length value="512"/>
      <overflow-action value="block"/>
   ```

5. **List *`subhandlers`***
   The ***`subhandlers`*** list is the list of log handlers to which this async handler passes its log messages.

   ```
   <async-handler name="Async_NFS_handlers">
      <level name="INFO"/>
      <queue-length value="512"/>
      <overflow-action value="block"/>
      <subhandlers>
         <handler name="FILE"/>
         <handler name="accounts-record"/>
      </subhandlers>
   </async-handler>
   ```

Report a bug

PART IV. SET UP CACHE MODES

CHAPTER 6. CACHE MODES

Red Hat JBoss Data Grid provides two modes:

- Local mode is the only non-clustered cache mode offered in JBoss Data Grid. In local mode, JBoss Data Grid operates as a simple single-node in-memory data cache. Local mode is most effective when scalability and failover are not required and provides high performance in comparison with clustered modes.
- Clustered mode replicates state changes to a small subset of nodes. The subset size is sufficient for fault tolerance purposes but not large enough to hinder scalability. Before attempting to use clustered mode, it is important to first configure JGroups for a clustered configuration. For details about configuring JGroups, see Section 24.3, “Configure JGroups (Library Mode)”

Report a bug

6.1. ABOUT CACHE CONTAINERS

Cache containers are used in Red Hat JBoss Data Grid's Remote Client-Server mode as a starting point for a cache. The **`cache-container`** element acts as a parent of one or more (local or clustered) caches. To add clustered caches to the container, transport must be defined.

The following procedure demonstrates a sample cache container configuration:

Procedure 6.1. How to Configure the Cache Container

1. **Specify the Cache Container**
   The **`cache-container`** element specifies information about the cache container using the following parameters:

   ```
   <subsystem xmlns="urn:infinispan:server:core:6.0"
       default-cache-container="default">
   ```

   a. **Set the Cache Container Name**
      The ***`name`*** parameter defines the name of the cache container.

      ```
      <subsystem xmlns="urn:infinispan:server:core:6.0"
          default-cache-container="default">
       <cache-container name="default" />
      ```

   b. **Specify the Default Cache**
      The ***`default-cache`*** parameter defines the name of the default cache used with the cache container.

      ```
      <subsystem xmlns="urn:infinispan:server:core:6.0"
          default-cache-container="default">
       <cache-container name="default"
          default-cache="default" />
      ```

   c. **Enable/Disable Statistics**
      The ***`statistics`*** attribute is optional and is **`true`** by default. Statistics are useful in monitoring JBoss Data Grid via JMX or JBoss Operations Network, however they

adversely affect performance. Disable this attribute by setting it to **`false`** if it is not required.

```
<subsystem xmlns="urn:infinispan:server:core:6.0"
    default-cache-container="default">
 <cache-container name="default"
    default-cache="default"
    statistics="true"/>
```

d. **Define the Listener Executor**
The ***`listener-executor`*** defines the executor used for asynchronous cache listener notifications.

```
<subsystem xmlns="urn:infinispan:server:core:6.0"
    default-cache-container="default">
 <cache-container name="default"
    default-cache="default"
    statistics="true"
    listener-executor="infinispan-listener" />
```

e. **Set the Cache Container Start Mode**
The ***`start`*** parameter indicates when the cache container starts, i.e. whether it will start lazily when requested or "eagerly" when the server starts up. Valid values for this parameter are **EAGER** and **LAZY**.

```
<subsystem xmlns="urn:infinispan:server:core:6.0"
    default-cache-container="default">
 <cache-container name="default"
    default-cache="default"
    statistics="true"
    listener-executor="infinispan-listener"
    start="EAGER">
```

2. **Per-cache Statistics**
If ***`statistics`*** are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the ***`statistics`*** attribute to **`false`**.

```
<subsystem xmlns="urn:infinispan:server:core:6.0"
    default-cache-container="default">
 <cache-container name="default"
    default-cache="default"
    statistics="true"
    listener-executor="infinispan-listener"
    start="EAGER">
  <local-cache name="default"
     statistics="true">
   ...
  </local-cache>
 </cache-container>
</subsystem>
```

Report a bug

6.2. LOCAL MODE

Using Red Hat JBoss Data Grid's local mode instead of a map provides a number of benefits.

Caches offer features that are unmatched by simple maps, such as:

- Write-through and write-behind caching to persist data.
- Entry eviction to prevent the Java Virtual Machine (JVM) running out of memory.
- Support for entries that expire after a defined period.

JBoss Data Grid is built around a high performance, read-based data container that uses techniques such as optimistic and pessimistic locking to manage lock acquisitions.

JBoss Data Grid also uses compare-and-swap and other lock-free algorithms, resulting in high throughput multi-CPU or multi-core environments. Additionally, JBoss Data Grid's Cache API extends the JDK's **`ConcurrentMap`**, resulting in a simple migration process from a map to JBoss Data Grid.

Report a bug

6.2.1. Configure Local Mode (Remote Client-Server Mode)

A local cache can be added to any cache container. The following example demonstrates how to add the **`local-cache`** element.

Procedure 6.2. The `local-cache` Element

The **`local-cache`** element specifies information about the local cache used with the cache container using the following parameters:

1. **Add the Local Cache Name**
   The ***`name`*** parameter specifies the name of the local cache to use.

```
<cache-container name="local"
                default-cache="default"
                statistics="true">
        <local-cache name="default" >
```

2. **Set the Cache Container Start Mode**
   The ***`start`*** parameter indicates when the cache container starts, i.e. whether it will start lazily when requested or "eagerly" when the server starts up. Valid values for this parameter are **EAGER** and **LAZY**.

```
<cache-container name="local"
                default-cache="default"
                statistics="true">
        <local-cache name="default"
                       start="EAGER" >
```

3. **Configure Batching**
   The ***`batching`*** parameter specifies whether batching is enabled for the local cache.

```
<cache-container name="local"
```

```
                    default-cache="default"
                    statistics="true">
          <local-cache name="default"
                            start="EAGER"
                            batching="false" >
```

4. **Per-cache Statistics**
   If ***statistics*** are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the ***statistics*** attribute to **false**.

```
<cache-container name="local"
                    default-cache="default"
                    statistics="true">
          <local-cache name="default"
                            start="EAGER"
                            batching="false"
                            statistics="true">
```

5. **Specify Indexing Type**
   The ***indexing*** parameter specifies the type of indexing used for the local cache. Valid values for this parameter are **NONE**, **LOCAL** and **ALL**.

```
<cache-container name="local"
                    default-cache="default"
                    statistics="true">
          <local-cache name="default"
                            start="EAGER"
                            batching="false"
                            statistics="true">
                    <indexing index="NONE">
                   <property
name="default.directory_provider">ram</property>
                    </indexing>
          </local-cache>
```

Alternatively, create a **DefaultCacheManager** with the "no-argument" constructor. Both of these methods create a local default cache.

Local and clustered caches are able to coexist in the same cache container, however where the container is without a **<transport/>** it can only contain local caches. The container used in the example can only contain local caches as it does not have a **<transport/>**.

The cache interface extends the **ConcurrentMap** and is compatible with multiple cache systems.

Report a bug

6.2.2. Configure Local Mode (Library Mode)

In Red Hat JBoss Data Grid's Library mode, setting a cache's ***mode*** parameter to **local** equals not specifying a clustering mode at all. In the case of the latter, the cache defaults to local mode, even if its cache manager defines a transport.

Set the cluster mode to local as follows:

```
<clustering mode="local" />
```

Report a bug

6.3. CLUSTERED MODES

Red Hat JBoss Data Grid offers the following clustered modes:

- Replication Mode replicates any entry that is added across all cache instances in the cluster.
- Invalidation Mode does not share any data, but signals remote caches to initiate the removal of invalid entries.
- Distribution Mode stores each entry on a subset of nodes instead of on all nodes in the cluster.

The clustered modes can be further configured to use synchronous or asynchronous transport for network communications.

Report a bug

6.3.1. Asynchronous and Synchronous Operations

When a clustered mode (such as invalidation, replication or distribution) is used, data is propagated to other nodes in either a synchronous or asynchronous manner.

If synchronous mode is used, the sender waits for responses from receivers before allowing the thread to continue, whereas asynchronous mode transmits data but does not wait for responses from other nodes in the cluster to continue operations.

Asynchronous mode prioritizes speed over consistency, which is ideal for use cases such as HTTP session replications with sticky sessions enabled. Such a session (or data for other use cases) is always accessed on the same cluster node, unless this node fails.

Report a bug

6.3.2. Cache Mode Troubleshooting

6.3.2.1. Invalid Data in ReadExternal

If invalid data is passed to **`readExternal`**, it can be because when using **`Cache.putAsync()`**, starting serialization can cause your object to be modified, causing the datastream passed to **`readExternal`** to be corrupted. This can be resolved if access to the object is synchronized.

Report a bug

6.3.2.2. About Asynchronous Communications

In Red Hat JBoss Data Grid, the local, distributed and replicated modes are represented by the **`local-cache`**, **`distributed-cache`** and **`replicated-cache`** elements respectively. Each of these elements contains a *mode* property, the value of which can be set to **`SYNC`** for synchronous or **`ASYNC`** for asynchronous communications.

An example of this configuration is as follows:

```
<replicated-cache name="default"
                  start="EAGER"
                  mode="SYNC"
                  batching="false"
                  statistics="true">
                 ...
</replicated-cache>
```

NOTE

This configuration is valid for both JBoss Data Grid's usage modes (Library mode and Remote Client-Server mode).

Report a bug

6.3.2.3. Cluster Physical Address Retrieval

How can the physical addresses of the cluster be retrieved?

The physical address can be retrieved using an instance method call. For example: **`AdvancedCache.getRpcManager().getTransport().getPhysicalAddresses()`**.

Report a bug

6.4. STATE TRANSFER

State transfer occurs automatically in Red Hat JBoss Data Grid whenever a node joins or leaves the cluster.

The new node receives the cache state from the existing nodes when it joins the cluster in both distribution and replication mode. State Transfer also occurs to nodes in redistributing the state after a node leaves the cluster in distribution mode.

The State transfer can occur regardless of whether the cache is in-memory state or persistent state.

- In replication mode, the node joining the cluster receives a copy of the data currently on the other nodes in the cache. This occurs when the existing nodes push a part of the current cache state.
- In distribution mode, each node contains a slice of the entire key space, which is determined through consistent hashing. When a new node joins the cluster it receives a slice of the key space that has been taken from each of the existing nodes. State transfer results in the new node receiving a slice of the key space and the existing nodes shedding a portion of the data they were previously responsible for.

Report a bug

6.4.1. Non-Blocking State Transfer

Non-Blocking State Transfer in Red Hat JBoss Data Grid aims to minimize the time in which a cluster or node is unable to respond due to a state transfer in progress.

In JBoss Data Grid, Non-Blocking State Transfer

- allows state transfer to occur without a drop in the performance of the cluster. However, if a drop in performance does occur during the state transfer it will not throw an exception, and will allow processes to continue.
- does not add a mechanism for resolving data conflicts after a merge, however it ensures it is feasible to add one in the future.

Report a bug

6.4.2. Suppress State Transfer via JMX

State transfer can be suppressed using JMX in order to bring down and relaunch a cluster for maintenance. This operation permits a more efficient cluster shutdown and startup, and removes the risk of Out Of Memory errors when bringing down a grid.

When a new node joins the cluster and rebalancing is suspended, the **`getCache()`** call will timeout after **`stateTransfer.timeout`** expires unless rebalancing is re-enabled or **`stateTransfer.awaitInitialTransfer`**is set to **`false`**.

Disabling state transfer and rebalancing can be used for partial cluster shutdown or restart, however there is the possibility that data may be lost in a partial cluster shutdown due to state transfer being disabled.

Report a bug

6.4.3. The rebalancingEnabled Attribute

Suppressing rebalancing can only be triggered via the **`rebalancingEnabled`** JMX attribute, and requires no specific configuration.

The **`rebalancingEnabled`** attribute can be modified for the entire cluster from the **`LocalTopologyManager`** JMX Mbean on any node. This attribute is **`true`** by default, and is configurable programmatically.

Servers such as Hot Rod attempt to start all caches declared in the configuration during startup. If rebalancing is disabled, the cache will fail to start. Therefore, it is mandatory to use the following setting in a server environment:

```
<await-initial-transfer="false"/>
```

Report a bug

CHAPTER 7. SET UP DISTRIBUTION MODE

7.1. ABOUT DISTRIBUTION MODE

When enabled, Red Hat JBoss Data Grid's distribution mode stores each entry on a subset of the nodes in the grid instead of replicating each entry on every node. Typically, each entry is stored on more than one node for redundancy and fault tolerance.

As a result of storing entries on selected nodes across the cluster, distribution mode provides improved scalability compared to other clustered modes.

A cache using distribution mode can transparently locate keys across a cluster using the consistent hash algorithm.

Report a bug

7.2. DISTRIBUTION MODE'S CONSISTENT HASH ALGORITHM

Distribution mode uses a consistent hash algorithm to select a node from the cluster to store entries upon. The consistent hash algorithm is configured with the number of copies of each cache entry to be maintained within the cluster.

The number of copies set for each data item requires balancing performance and fault tolerance. Creating too many copies of the entry can impair performance and too few copies can result in data loss in case of node failure.

Report a bug

7.3. LOCATING ENTRIES IN DISTRIBUTION MODE

The consistent hash algorithm used in Red Hat JBoss Data Grid's distribution mode can locate entries deterministically, without multicasting a request or maintaining expensive metadata.

A **PUT** operation can result in as many remote calls as specified by the ***num_copies*** parameter, while a **GET** operation executed on any node in the cluster results in a single remote call. In the background, the **GET** operation results in the same number of remote calls as a **PUT** operation (specifically the value of the ***num_copies*** parameter), but these occur in parallel and the returned entry is passed to the caller as soon as one returns.

Report a bug

7.4. RETURN VALUES IN DISTRIBUTION MODE

In Red Hat JBoss Data Grid's distribution mode, a synchronous request is used to retrieve the previous return value if it cannot be found locally. A synchronous request is used for this task irrespective of whether distribution mode is using asynchronous or synchronous processes.

Report a bug

7.5. CONFIGURE DISTRIBUTION MODE (REMOTE CLIENT-SERVER MODE)

Distribution mode is a clustered mode in Red Hat JBoss Data Grid. Distribution mode can be added to any cache container using the following procedure:

Procedure 7.1. The `distributed-cache` Element

The **`distributed-cache`** element configures settings for the distributed cache using the following parameters:

1. **Add the Cache Name**
   The **name** parameter provides a unique identifier for the cache.

   ```
   <cache-container name="clustered"
      default-cache="default"
      statistics="true">
    <transport executor="infinispan-transport" lock-timeout="60000"/>
     <distributed-cache name="default" />
   ```

2. **Set the Clustered Cache Start Mode**
   The **mode** parameter sets the clustered cache mode. Valid values are **SYNC** (synchronous) and **ASYNC** (asynchronous).

   ```
   <cache-container name="clustered"
      default-cache="default"
      statistics="true">
    <transport executor="infinispan-transport" lock-timeout="60000"/>
    <distributed-cache name="default"
         mode="SYNC" />
   ```

3. **Specify Number of Segments**
   The (optional) **`segments`** parameter specifies the number of hash space segments per cluster. The recommended value for this parameter is ten multiplied by the cluster size and the default value is **`80`**.

   ```
   <cache-container name="clustered"
      default-cache="default"
      statistics="true">
    <transport executor="infinispan-transport" lock-timeout="60000"/>
    <distributed-cache name="default"
         mode="SYNC"
         segments="20" />
   ```

4. **Set the Cache Start Mode**
   The **`start`** parameter specifies whether the cache starts when the server starts up or when it is requested or deployed.

   ```
   <cache-container name="clustered"
      default-cache="default"
      statistics="true">
    <transport executor="infinispan-transport" lock-timeout="60000"/>
    <distributed-cache name="default"
         mode="SYNC"
   ```

```
        segments="20"
        start="EAGER"/>
```

5. **Per-cache Statistics**
   If ***statistics*** are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the ***statistics*** attribute to **`false`**.

```
<cache-container name="clustered"
   default-cache="default"
   statistics="true">
 <transport executor="infinispan-transport" lock-timeout="60000"/>
 <distributed-cache name="default"
      mode="SYNC"
      segments="20"
      start="EAGER"
      statistics="true">
   ...
 </distributed-cache>
</cache-container>
```

IMPORTANT

JGroups must be appropriately configured for clustered mode before attempting to load this configuration.

For details about the **`cache-container`**, **`locking`**, and **`transaction`** elements, see the appropriate chapter.

Report a bug

7.6. CONFIGURE DISTRIBUTION MODE (LIBRARY MODE)

The following procedure shows a distributed cache configuration in Red Hat JBoss Data Grid's Library mode.

Procedure 7.2. Distributed Cache Configuration

1. **Set the Clustered Mode**
   The **`clustering`** element's ***mode*** parameter's value determines the clustering mode selected for the cache.

```
<clustering mode="dist">
```

2. **Specify the Remote Call Timeout**
   The **sync** element's ***replTimeout*** parameter specifies the maximum time period in milliseconds for an acknowledgment after a remote call. If the time period ends without any acknowledgment, an exception is thrown.

```
<clustering mode="dist">
        <sync replTimeout="${TIME}" />
```

3. **Define State Transfer Settings**
   The `stateTransfer` element specifies how state is transferred when a node leaves or joins the cluster. It uses the following parameters:

   a. **Specify State Transfer Batch Size**
      The ***chunkSize*** parameter specifies the size of cache entry state batches to be transferred. If this value is greater than `0`, the value set is the size of chunks sent. If the value is less than `0`, all states are transferred at the same time.

```
<clustering mode="dist">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}" />
```

   b. **Set *fetchInMemoryState* Parameter**
      The ***fetchInMemoryState*** parameter when set to `true`, requests state information from neighboring caches on start up. This impacts the start up time for the cache.

```
<clustering mode="dist">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}" />
```

   c. **Define the *awaitInitialTransfer* Parameter**
      The ***awaitInitialTransfer*** parameter causes the first call to method **`CacheManager.getCache()`** on the joiner node to block and wait until the joining is complete and the cache has finished receiving state from neighboring caches (if ***fetchInMemoryState*** is enabled). This option applies to distributed and replicated caches only and is enabled by default.

```
<clustering mode="dist">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}" />
```

   d. **Set *timeout* Value**
      The ***timeout*** parameter specifies the maximum time (in milliseconds) the cache waits for responses from neighboring caches with the requested states. If no response is received within the the ***timeout*** period, the start up process aborts and an exception is thrown.

```
<clustering mode="dist">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
```

4. **Specify Transport Configuration**
   The `transport` element defines the transport configuration for the cache as follows:

   a. **Specify the Cluster Name**
      The ***clusterName*** parameter specifies the name of the cluster. Nodes can only connect to clusters that share the same name.

```
<clustering mode="dist">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
        <transport clusterName="${NAME}" />
```

b. **Set the *distributedSyncTimeout* Value**

The ***distributedSyncTimeout*** parameter specifies the time to wait to acquire a lock on the distributed lock. This distributed lock ensures that a single cache can transfer state or rehash state at a time.

```
<clustering mode="dist">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
        <transport clusterName="${NAME}"
                   distributedSyncTimeout="${TIME}" />
```

c. **Set the Network Transport**

The ***transportClass*** parameter specifies a class that represents a network transport for the cache.

```
<clustering mode="dist">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
        <transport clusterName="${NAME}"
                   distributedSyncTimeout="${TIME}"
                   transportClass="${CLASS}" />
</clustering>
```

Report a bug

7.7. SYNCHRONOUS AND ASYNCHRONOUS DISTRIBUTION

Distribution mode only supports synchronous communication. To elicit meaningful return values from certain public API methods, it is essential to use synchronized communication when using distribution mode.

Example 7.1. Communication Mode example

For example, with three caches in a cluster, cache **A**, **B** and **C**, and a key **K** that maps cache **A** to **B**. Perform an operation on cluster **C** that requires a return value, for example **`Cache.remove(K)`**. To execute successfully, the operation must first synchronously forward the call to both cache **A** and **B**, and then wait for a result returned from either cache **A** or **B**. If asynchronous communication was used, the usefulness of the returned values cannot be guaranteed, despite the operation behaving as expected.

Report a bug

7.8. GET AND PUT USAGE IN DISTRIBUTION MODE

In distribution mode, the cache performs a remote **`GET`** command before a write command. This occurs because certain methods (for example, **`Cache.put()`**) return the previous value associated with the specified key according to the **`java.util.Map`** contract. When this is performed on an instance that does not own the key and the entry is not found in the L1 cache, the only reliable way to elicit this return value is to perform a remote **`GET`** before the **`PUT`**.

The **`GET`** operation that occurs before the **`PUT`** operation is always synchronous, whether the cache is synchronous or asynchronous, because Red Hat JBoss Data Grid must wait for the return value.

Report a bug

7.8.1. Distributed GET and PUT Operation Resource Usage

In distribution mode, the cache may execute a **`GET`** operation before executing the desired **`PUT`** operation.

This operation is very expensive in terms of resources. Despite operating in an synchronous manner, a remote **`GET`** operation does not wait for all responses, which would result in wasted resources. The **`GET`** process accepts the first valid response received, which allows its performance to be unrelated to cluster size.

Use the ***`Flag.SKIP_REMOTE_LOOKUP`*** flag for a per-invocation setting if return values are not required for your implementation.

Such actions do not impair cache operations and the accurate functioning of all public methods, but do break the **`java.util.Map`** interface contract. The contract breaks because unreliable and inaccurate return values are provided to certain methods. As a result, ensure that these return values are not used for any important purpose on your configuration.

Report a bug

CHAPTER 8. SET UP REPLICATION MODE

8.1. ABOUT REPLICATION MODE

Red Hat JBoss Data Grid's replication mode is a simple clustered mode. Cache instances automatically discover neighboring instances on other Java Virtual Machines (JVM) on the same network and subsequently form a cluster with the discovered instances. Any entry added to a cache instance is replicated across all cache instances in the cluster and can be retrieved locally from any cluster cache instance.

In JBoss Data Grid's replication mode, return values are locally available before the replication occurs.

Report a bug

8.2. OPTIMIZED REPLICATION MODE USAGE

Replication mode is used for state sharing across a cluster. However, the cluster performance is optimal only when the target cluster contains less than ten servers.

In larger clusters, the fact that a large number of replication messages must be transmitted results in reduced performance.

Red Hat JBoss Data Grid can be configured to use UDP multicast, which improves performance to a limited degree for larger clusters.

Report a bug

8.3. CONFIGURE REPLICATION MODE (REMOTE CLIENT-SERVER MODE)

Replication mode is a clustered cache mode in Red Hat JBoss Data Grid. Replication mode can be added to any cache container using the following procedure.

Procedure 8.1. The *replicated-cache* Element

The **replicated-cache** element configures settings for the distributed cache using the following parameters:

1. **Add the Cache Name**
   The **name** parameter provides a unique identifier for the cache.

   ```
   <cache-container name="local"
      default-cache="default"
      statistics="true">
    <replicated-cache name="default">
   ```

2. **Set the Clustered Cache Start Mode**
   The **mode** parameter sets the clustered cache mode. Valid values are **SYNC** (synchronous) and **ASYNC** (asynchronous).

   ```
   <cache-container name="local"
      default-cache="default"
      statistics="true">
   ```

```
 <replicated-cache name="default"
     mode="SYNC">
```

3. **Set the Cache Start Mode**
   The **start** parameter specifies whether the cache starts when the server starts up or when it is requested or deployed.

```
<cache-container name="local"
   default-cache="default"
   statistics="true">
 <replicated-cache name="default"
    mode="SYNC"
    start="EAGER">
```

4. **Per-cache Statistics**
   If ***statistics*** are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the ***statistics*** attribute to **false**.

```
<cache-container name="local"
   default-cache="default"
   statistics="true">
 <replicated-cache name="default"
    mode="SYNC"
    start="EAGER"
    statistics="true">
        ...
 </replicated-cache>
</cache-container>
```

5. **Set Up Transactions**
   The **transaction** element sets up the transaction mode for the replicated cache.

```
<cache-container name="local"
   default-cache="default"
   statistics="true">
 <replicated-cache name="default"
    mode="SYNC"
    start="EAGER"
    statistics="true">
  <transaction mode="NONE" />
 </replicated-cache>
</cache-container>
```

IMPORTANT

JGroups must be appropriately configured for clustered mode before attempting to load this configuration.

For details about the **cache-container**, **locking**, and **transaction** elements, see the appropriate chapter.

Report a bug

8.4. CONFIGURE REPLICATION MODE (LIBRARY MODE)

The following procedure shows a replication mode configuration in Red Hat JBoss Data Grid's Library mode.

Procedure 8.2. Replication Mode Configuration

1. **Set the Clustered Mode**
   The `clustering` element's *mode* parameter's value determines the clustering mode selected for the cache.

   ```
   <clustering mode="repl">
   ```

2. **Specify the Remote Call Timeout**
   The `sync` element's *replTimeout* parameter specifies the maximum time period in milliseconds for an acknowledgment after a remote call. If the time period ends without any acknowledgment, an exception is thrown.

   ```
   <clustering mode="repl">
           <sync replTimeout="${TIME}" />
   ```

3. **Define State Transfer Settings**
   The `stateTransfer` element specifies how state is transferred when a node leaves or joins the cluster. It uses the following parameters:

   a. **Specify State Transfer Batch Size**
      The *chunkSize* parameter specifies the size of cache entry state batches to be transferred. If this value is greater than `0`, the value set is the size of chunks sent. If the value is less than `0`, all states are transferred at the same time.

      ```
      <clustering mode="repl">
              <sync replTimeout="${TIME}" />
              <stateTransfer chunkSize="${SIZE}" />
      ```

   b. **Set *fetchInMemoryState* Parameter**
      The *fetchInMemoryState* parameter when set to `true`, requests state information from neighboring caches on start up. This impacts the start up time for the cache.

      ```
      <clustering mode="repl">
              <sync replTimeout="${TIME}" />
              <stateTransfer chunkSize="${SIZE}"
                             fetchInMemoryState="{true/false}" />
      ```

   c. **Define the *awaitInitialTransfer* Parameter**
      The *awaitInitialTransfer* parameter causes the first call to method `CacheManager.getCache()` on the joiner node to block and wait until the joining is complete and the cache has finished receiving state from neighboring caches (if *fetchInMemoryState* is enabled). This option applies to distributed and replicated caches only and is enabled by default.

      ```
      <clustering mode="repl">
              <sync replTimeout="${TIME}" />
      ```

```
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}" />
```

d. **Set the *timeout* Value**

   The ***timeout*** parameter specifies the maximum time (in milliseconds) the cache waits for responses from neighboring caches with the requested states. If no response is received within the the ***timeout*** period, the start up process aborts and an exception is thrown.

```
<clustering mode="repl">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
```

4. **Specify Transport Configuration**

   The **transport** element defines the transport configuration for the cache as follows:

   a. **Specify the Cluster Name**

      The ***clusterName*** parameter specifies the name of the cluster. Nodes can only connect to clusters that share the same name.

```
<clustering mode="repl">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
        <transport clusterName="${NAME}" />
```

   b. **Set the *distributedSyncTimeout* Value**

      The ***distributedSyncTimeout*** parameter specifies the time to wait to acquire a lock on the distributed lock. This distributed lock ensures that a single cache can transfer state or rehash state at a time.

```
<clustering mode="repl">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
        <transport clusterName="${NAME}"
                   distributedSyncTimeout="${TIME}" />
```

   c. **Set the Network Transport**

      The ***transportClass*** parameter specifies a class that represents a network transport for the cache.

```
<clustering mode="repl">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
```

```
                        awaitInitialTransfer="{true/false}"
                        timeout="${TIME}" />
        <transport clusterName="${NAME}"
                   distributedSyncTimeout="${TIME}"
                   transportClass="${CLASS}" />
</clustering>
```

Report a bug

8.5. SYNCHRONOUS AND ASYNCHRONOUS REPLICATION

Replication mode can be synchronous or asynchronous depending on the problem being addressed.

- Synchronous replication blocks a thread or caller (for example on a **`put()`** operation) until the modifications are replicated across all nodes in the cluster. By waiting for acknowledgments, synchronous replication ensures that all replications are successfully applied before the operation is concluded.
- Asynchronous replication operates significantly faster than synchronous replication because it does not need to wait for responses from nodes. Asynchronous replication performs the replication in the background and the call returns immediately. Errors that occur during asynchronous replication are written to a log. As a result, a transaction can be successfully completed despite the fact that replication of the transaction may not have succeeded on all the cache instances in the cluster.

Report a bug

8.5.1. Troubleshooting Asynchronous Replication Behavior

In some instances, a cache configured for asynchronous replication or distribution may wait for responses, which is synchronous behavior. This occurs because caches behave synchronously when both state transfers and asynchronous modes are configured. This synchronous behavior is a prerequisite for state transfer to operate as expected.

Use one of the following to remedy this problem:

- Disable state transfer and use a **`ClusteredCacheLoader`** to lazily look up remote state as and when needed.
- Enable state transfer and ***REPL_SYNC***. Use the Asynchronous API (for example, the **`cache.putAsync(k, v)`**) to activate 'fire-and-forget' capabilities.
- Enable state transfer and ***REPL_ASYNC***. All RPCs end up becoming synchronous, but client threads will not be held up if a replication queue is enabled (which is recommended for asynchronous mode).

Report a bug

8.6. THE REPLICATION QUEUE

In replication mode, Red Hat JBoss Data Grid uses a replication queue to replicate changes across nodes based on the following:

- Previously set intervals.

- The queue size exceeding the number of elements.
- A combination of previously set intervals and the queue size exceeding the number of elements.

The replication queue ensures that during replication, cache operations are transmitted in batches instead of individually. As a result, a lower number of replication messages are transmitted and fewer envelopes are used, resulting in improved JBoss Data Grid performance.

A disadvantage of using the replication queue is that the queue is periodically flushed based on the time or the queue size. Such flushing operations delay the realization of replication, distribution, or invalidation operations across cluster nodes. When the replication queue is disabled, the data is directly transmitted and therefore the data arrives at the cluster nodes faster.

A replication queue is used in conjunction with asynchronous mode.

Report a bug

8.6.1. Replication Queue Usage

When using the replication queue, do one of the following:

- Disable asynchronous marshalling; or
- Set the ***max-threads*** count value to **1** for the **transport executor**. The **transport executor** is defined in **standalone.xml** as follows:

  ```
  <transport executor="infinispan-transport"/>
  ```

To implement either of these solutions, the replication queue must be in use in asynchronous mode. Asynchronous mode can be set, along with the queue timeout (***queue-flush-interval***, value is in milliseconds) and queue size (***queue-size***) as follows:

```
<replicated-cache name="asyncCache"
                  start="EAGER"
                  mode="ASYNC"
                  batching="false"
                  indexing="NONE"
                  statistics="true"
                  queue-size="1000"
                  queue-flush-interval="500">
                ...
</replicated-cache>
```

The replication queue allows requests to return to the client faster, therefore using the replication queue together with asynchronous marshalling does not present any significant advantages.

Report a bug

8.7. ABOUT REPLICATION GUARANTEES

In a clustered cache, the user can receive synchronous replication guarantees as well as the parallelism associated with asynchronous replication. Red Hat JBoss Data Grid provides an asynchronous API for this purpose.

The asynchronous methods used in the API return Futures, which can be queried. The queries block the thread until a confirmation is received about the success of any network calls used.

Report a bug

8.8. REPLICATION TRAFFIC ON INTERNAL NETWORKS

Some cloud providers charge less for traffic over internal **`IP`** addresses than for traffic over public **`IP`** addresses, or do not charge at all for internal network traffic (for example, GoGrid). To take advantage of lower rates, you can configure Red Hat JBoss Data Grid to transfer replication traffic using the internal network. With such a configuration, it is difficult to know the internal **`IP`** address you are assigned. JBoss Data Grid uses JGroups interfaces to solve this problem.

Report a bug

CHAPTER 9. SET UP INVALIDATION MODE

9.1. ABOUT INVALIDATION MODE

Invalidation is a clustered mode that does not share any data, but instead removes potentially obsolete data from remote caches. Using this cache mode requires another, more permanent store for the data such as a database.

Red Hat JBoss Data Grid, in such a situation, is used as an optimization for a system that performs many read operations and prevents database usage each time a state is needed.

When invalidation mode is in use, data changes in a cache prompts other caches in the cluster to evict their outdated data from memory.

Report a bug

9.2. CONFIGURE INVALIDATION MODE (REMOTE CLIENT-SERVER MODE)

Invalidation mode is a clustered mode in Red Hat JBoss Data Grid. Invalidation mode can be added to any cache container using the following procedure:

Procedure 9.1. The `invalidation-cache` Element

The **`invalidation-cache`** element configures settings for the distributed cache using the following parameters:

1. **Add the Cache Name**
   The **name** parameter provides a unique identifier for the cache.

```
<cache-container name="local"
       default-cache="default"
       statistics="true">
 <invalidation-cache name="default">
```

2. **Set the Clustered Cache Start Mode**
   The **mode** parameter sets the clustered cache mode. Valid values are **SYNC** (synchronous) and **ASYNC** (asynchronous).

```
<cache-container name="local"
       default-cache="default"
       statistics="true">
 <invalidation-cache name="default"
      mode="ASYNC">
```

3. **Set the Cache Start Mode**
   The **`start`** parameter specifies whether the cache starts when the server starts up or when it is requested or deployed.

```
<cache-container name="local"
       default-cache="default"
       statistics="true">
```

```
 <invalidation-cache name="default"
       mode="ASYNC"
       start="EAGER">
```

4. **Per-cache Statistics**
   If ***statistics*** are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the ***statistics*** attribute to **false**.

```
<cache-container name="local"
        default-cache="default"
        statistics="true">
 <invalidation-cache name="default"
       mode="ASYNC"
       start="EAGER"
       statistics="true">
   ...
 </invalidation-cache>
</cache-container>
```

IMPORTANT

JGroups must be appropriately configured for clustered mode before attempting to load this configuration.

For details about the **cache-container**, **locking**, and **transaction** elements, see the appropriate chapter.

Report a bug

9.3. CONFIGURE INVALIDATION MODE (LIBRARY MODE)

The following procedure shows an invalidation mode cache configuration in Red Hat JBoss Data Grid's Library mode.

Procedure 9.2. Invalidation Mode Configuration

1. **Set the Clustered Mode**
   The **clustering** element's ***mode*** parameter's value determines the clustering mode selected for the cache.

```
<clustering mode="inv">
```

2. **Specify the Remote Call Timeout**
   The **sync** element's ***replTimeout*** parameter specifies the maximum time period in milliseconds for an acknowledgment after a remote call. If the time period ends without any acknowledgment, an exception is thrown.

```
<clustering mode="inv">
        <sync replTimeout="${TIME}" />
```

3. **Define State Transfer Settings**

The **stateTransfer** element specifies how state is transferred when a node leaves or joins the cluster. It uses the following parameters:

a. **Specify State Transfer Batch Size**
   The ***chunkSize*** parameter specifies the size of cache entry state batches to be transferred. If this value is greater than **0**, the value set is the size of chunks sent. If the value is less than **0**, all states are transferred at the same time.

```
<clustering mode="inv">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}" />
```

b. **Set *fetchInMemoryState* Parameter**
   The ***fetchInMemoryState*** parameter when set to **true**, requests state information from neighboring caches on start up. This impacts the start up time for the cache.

```
<clustering mode="inv">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}" />
```

c. **Define the *awaitInitialTransfer* Parameter**
   The ***awaitInitialTransfer*** parameter causes the first call to method **CacheManager.getCache()** on the joiner node to block and wait until the joining is complete and the cache has finished receiving state from neighboring caches (if ***fetchInMemoryState*** is enabled). This option applies to distributed and replicated caches only and is enabled by default.

```
<clustering mode="inv">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}" />
```

d. **Set *timeout* Value**
   The ***timeout*** parameter specifies the maximum time (in milliseconds) the cache waits for responses from neighboring caches with the requested states. If no response is received within the the ***timeout*** period, the start up process aborts and an exception is thrown.

```
<clustering mode="inv">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
```

4. **Specify Transport Configuration**
   The **transport** element defines the transport configuration for the cache as follows:

a. **Specify the Cluster Name**
   The ***clusterName*** parameter specifies the name of the cluster. Nodes can only connect to clusters that share the same name.

```
<clustering mode="inv">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
        <transport clusterName="${NAME}" />
```

b. **Set the *`distributedSyncTimeout`* Value**

The ***`distributedSyncTimeout`*** parameter specifies the time to wait to acquire a lock on the distributed lock. This distributed lock ensures that a single cache can transfer state or rehash state at a time.

```
<clustering mode="inv">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
        <transport clusterName="${NAME}"
                   distributedSyncTimeout="${TIME}" />
```

c. **Set the Network Transport**

The ***`transportClass`*** parameter specifies a class that represents a network transport for the cache.

```
<clustering mode="inv">
        <sync replTimeout="${TIME}" />
        <stateTransfer chunkSize="${SIZE}"
                       fetchInMemoryState="{true/false}"
                       awaitInitialTransfer="{true/false}"
                       timeout="${TIME}" />
        <transport clusterName="${NAME}"
                   distributedSyncTimeout="${TIME}"
                   transportClass="${CLASS}" />
</clustering>
```

Report a bug

9.4. SYNCHRONOUS/ASYNCHRONOUS INVALIDATION

In Red Hat JBoss Data Grid's Library mode, invalidation operates either asynchronously or synchronously.

- Synchronous invalidation blocks the thread until all caches in the cluster have received invalidation messages and evicted the obsolete data.
- Asynchronous invalidation operates in a fire-and-forget mode that allows invalidation messages to be broadcast without blocking a thread to wait for responses.

Report a bug

9.5. THE L1 CACHE AND INVALIDATION

An invalidation message is generated each time a key is updated. This message is multicast to each node that contains data that corresponds to current L1 cache entries. The invalidation message ensures that each of these nodes marks the relevant entry as invalidated.

Report a bug

PART V. SET UP LOCKING FOR THE CACHE

CHAPTER 10. LOCKING

Red Hat JBoss Data Grid provides locking mechanisms to prevent dirty reads (where a transaction reads an outdated value before another transaction has applied changes to it) and non-repeatable reads.

Report a bug

10.1. CONFIGURE LOCKING (REMOTE CLIENT-SERVER MODE)

In Remote Client-Server mode, locking is configured using the **`locking`** element within the cache tags (for example, **`invalidation-cache`**, **`distributed-cache`**, **`replicated-cache`** or **`local-cache`**).

The following is a sample procedure of a basic locking configuration for a default cache in Red Hat JBoss Data Grid's Remote Client-Server mode.

Procedure 10.1. Configure Locking (Remote Client-Server Mode)

1. **Define the Isolation Level**
   The ***`isolation`*** parameter defines the isolation level used for the local cache. Valid values for this parameter are **`REPEATABLE_READ`** and **`READ_COMMITTED`**.

   ```
   <distributed-cache>
    <locking isolation="REPEATABLE_READ" />
   ```

2. **Set the *`acquire-timeout`* Parameter**
   The ***`acquire-timeout`*** parameter specifies the number of milliseconds after which lock acquisition will time out.

   ```
   <distributed-cache>
    <locking isolation="REPEATABLE_READ"
            acquire-timeout="30000" />
   ```

3. **Set Number of Lock Stripes**
   The ***`concurrency-level`*** parameter defines the number of lock stripes used by the LockManager.

   ```
   <distributed-cache>
    <locking isolation="REPEATABLE_READ"
            acquire-timeout="30000"
            concurrency-level="1000" />
   ```

4. **Set Lock Striping**
   The ***`striping`*** parameter specifies whether lock striping will be used for the local cache.

   ```
   <distributed-cache>
    <locking isolation="REPEATABLE_READ"
            acquire-timeout="30000"
            concurrency-level="1000"
   ```

```
            striping="false" />
            ...
</distributed-cache>
```

Report a bug

10.2. CONFIGURE LOCKING (LIBRARY MODE)

For Library mode, the **`locking`** element and its parameters are set within the optional **`configuration`** element on a per cache basis. For example, for the default cache, the **`configuration`** element occurs within the **`default`** element and for each named cache, it occurs within the **namedCache** element. The following is an example of this configuration:

Procedure 10.2. Configure Locking (Library Mode)

1. **Set the Concurrency Level**
   The ***`concurrencyLevel`*** parameter specifies the concurrency level for the lock container. Set this value according to the number of concurrent threads interacting with the data grid.

   ```
   <infinispan>
    ...
    <default>
     <locking concurrencyLevel="${VALUE}" />
   ```

2. **Specify the Cache Isolation Level**
   The ***`isolationLevel`*** parameter specifies the cache's isolation level. Valid isolation levels are **`READ_COMMITTED`** and **`REPEATABLE_READ`**. For details about isolation levels, see Section 12.1, “About Isolation Levels”

   ```
   <infinispan>
    ...
    <default>
     <locking concurrencyLevel="${VALUE}"
       isolationLevel="${LEVEL}" />
   ```

3. **Set the Lock Acquisition Timeout**
   The ***`lockAcquisitionTimeout`*** parameter specifies time (in milliseconds) after which a lock acquisition attempt times out.

   ```
   <infinispan>
    ...
    <default>
     <locking concurrencyLevel="${VALUE}"
      isolationLevel="${LEVEL}"
      lockAcquisitionTimeout="${TIME}" />
   ```

4. **Configure Lock Striping**
   The ***`useLockStriping`*** parameter specifies whether a pool of shared locks are maintained for all entries that require locks. If set to **FALSE**, locks are created for each entry in the cache. For details, see Section 11.1, “About Lock Striping”

   ```
   <infinispan>
   ```

```
...
<default>
 <locking concurrencyLevel="${VALUE}"
  isolationLevel="${LEVEL}"
  lockAcquisitionTimeout="${TIME}"
  useLockStriping="${TRUE/FALSE}" />
```

- **Set *`writeSkewCheck`* Parameter**
  The ***`writeSkewCheck`*** parameter is only valid if the ***`isolationLevel`*** is set to **REPEATABLE_READ**. If this parameter is set to **FALSE**, a disparity between a working entry and the underlying entry at write time results in the working entry overwriting the underlying entry. If the parameter is set to **TRUE**, such conflicts (namely write skews) throw an exception.

  ```
  <infinispan>
   ...
   <default>
    <locking concurrencyLevel="${VALUE}"
     isolationLevel="${LEVEL}"
     lockAcquisitionTimeout="${TIME}"
     useLockStriping="${TRUE/FALSE}"
     writeSkewCheck="${TRUE/FALSE}" />
  ```

Report a bug

10.3. LOCKING TYPES

10.3.1. About Optimistic Locking

Optimistic locking allows multiple transactions to complete simultaneously by deferring lock acquisition to the transaction prepare time.

Optimistic mode assumes that multiple transactions can complete without conflict. It is ideal where there is little contention between multiple transactions running concurrently, as transactions can commit without waiting for other transaction locks to clear. With ***`writeSkewCheck`*** enabled, transactions in optimistic locking mode roll back if one or more conflicting modifications are made to the data before the transaction completes.

Report a bug

10.3.2. About Pessimistic Locking

Pessimistic locking is also known as eager locking.

Pessimistic locking prevents more than one transaction being written to a key by enforcing cluster-wide locks on each write operation. Locks are only released once the transaction is completed either through committing or being rolled back.

Pessimistic mode is used where a high contention on keys is occurring, resulting in inefficiencies and unexpected roll back operations.

Report a bug

10.3.3. Pessimistic Locking Types

Red Hat JBoss Data Grid includes explicit pessimistic locking and implicit pessimistic locking:

- Explicit Pessimistic Locking, which uses the JBoss Data Grid Lock API to allow cache users to explicitly lock cache keys for the duration of a transaction. The Lock call attempts to obtain locks on specified cache keys across all nodes in a cluster. This attempt either fails or succeeds for all specified cache keys. All locks are released during the commit or rollback phase.

- Implicit Pessimistic Locking ensures that cache keys are locked in the background as they are accessed for modification operations. Using Implicit Pessimistic Locking causes JBoss Data Grid to check and ensure that cache keys are locked locally for each modification operation. Discovering unlocked cache keys causes JBoss Data Grid to request a cluster-wide lock to acquire a lock on the unlocked cache key.

Report a bug

10.3.4. Explicit Pessimistic Locking Example

The following is an example of explicit pessimistic locking that depicts a transaction that runs on one of the cache nodes:

Procedure 10.3. Transaction with Explicit Pessimistic Locking

1. When the line **`cache.lock(K)`** executes, a cluster-wide lock is acquired on **`K`**.

   ```
   tx.begin()
   cache.lock(K)
   ```

2. When the line **`cache.put(K,V5)`** executes, it guarantees success.

   ```
   tx.begin()
   cache.lock(K)
   cache.put(K,V5)
   ```

3. When the line **`tx.commit()`** executes, the locks held for this process are released.

   ```
   tx.begin()
   cache.lock(K)
   cache.put(K,V5)
   tx.commit()
   ```

Report a bug

10.3.5. Implicit Pessimistic Locking Example

An example of implicit pessimistic locking using a transaction that runs on one of the cache nodes is as follows:

Procedure 10.4. Transaction with Implicit Pessimistic locking

1. When the line **`cache.put(K,V)`** executes, a cluster-wide lock is acquired on **`K`**.

```
tx.begin()
cache.put(K,V)
```

2. When the line **`cache.put(K2,V2)`** executes, a cluster-wide lock is acquired on **`K2`**.

```
tx.begin()
cache.put(K,V)
cache.put(K2,V2)
```

3. When the line **`cache.put(K,V5)`** executes, the lock acquisition is non operational because a cluster-wide lock for **`K`** has been previously acquired. The **`put`** operation will still occur.

```
tx.begin()
cache.put(K,V)
cache.put(K2,V2)
cache.put(K,V5)
```

4. When the line **`tx.commit()`** executes, all locks held for this transaction are released.

```
tx.begin()
cache.put(K,V)
cache.put(K2,V2)
cache.put(K,V5)
tx.commit()
```

Report a bug

10.3.6. Configure Locking Mode (Remote Client-Server Mode)

To configure a locking mode in Red Hat JBoss Data Grid's Remote Client-Server mode, use the ***`locking`*** parameter within the **`transaction`** element as follows:

```
<transaction locking="OPTIMISTIC/PESSIMISTIC" />
```

Report a bug

10.3.7. Configure Locking Mode (Library Mode)

In Red Hat JBoss Data Grid's Library mode, the locking mode is set within the **`transaction`** element as follows:

```
<transaction transactionManagerLookupClass="
{TransactionManagerLookupClass}"
      transactionMode="{TRANSACTIONAL,NON_TRANSACTIONAL}"
      lockingMode="{OPTIMISTIC,PESSIMISTIC}"
      useSynchronization="true">
</transaction>
```

Set the ***`lockingMode`*** value to **`OPTIMISTIC`** or **`PESSIMISTIC`** to configure the locking mode used for the transactional cache.

Report a bug

10.4. LOCKING OPERATIONS

10.4.1. About the LockManager

The `LockManager` component is responsible for locking an entry before a write process initiates. The `LockManager` uses a `LockContainer` to locate, hold and create locks. The two types of `LockContainers` generally used in such implementations are available. The first type offers support for lock striping while the second type supports one lock per entry.

See Also:

- Chapter 11, *Set Up Lock Striping*

Report a bug

10.4.2. About Lock Acquisition

Red Hat JBoss Data Grid acquires remote locks lazily by default. The node running a transaction locally acquires the lock while other cluster nodes attempt to lock cache keys that are involved in a two phase prepare/commit phase. JBoss Data Grid can lock cache keys in a pessimistic manner either explicitly or implicitly.

Report a bug

10.4.3. About Concurrency Levels

Concurrency refers to the number of threads simultaneously interacting with the data grid. In Red Hat JBoss Data Grid, concurrency levels refer to the number of concurrent threads used within a lock container.

In JBoss Data Grid, concurrency levels determine the size of each striped lock container. Additionally, concurrency levels tune all related JDK `ConcurrentHashMap` based collections, such as those internal to `DataContainers`.

Report a bug

CHAPTER 11. SET UP LOCK STRIPING

11.1. ABOUT LOCK STRIPING

Lock Striping allocates locks from a shared collection of (fixed size) locks in the cache. Lock allocation is based on the hash code for each entry's key. Lock Striping provides a highly scalable locking mechanism with fixed overhead. However, this is at the cost of potentially unrelated entries being blocked by the same lock.

Lock Striping is disabled as a default in Red Hat JBoss Data Grid. If lock striping remains disabled, a new lock is created for each entry. This alternative approach can provide greater concurrent throughput, but also results in additional memory usage, garbage collection churn, and other disadvantages.

Report a bug

11.2. CONFIGURE LOCK STRIPING (REMOTE CLIENT-SERVER MODE)

Lock striping in Red Hat JBoss Data Grid's Remote Client-Server mode is enabled using the ***`striping`*** element to **`true`**.

For example:

```
<locking isolation="REPEATABLE_READ"
  acquire-timeout="20000"
  concurrency-level="500"
  striping="true" />
```

Report a bug

11.3. CONFIGURE LOCK STRIPING (LIBRARY MODE)

Lock striping is disabled by default in Red Hat JBoss Data Grid. Configure lock striping in JBoss Data Grid's Library mode using the ***`useLockStriping`*** parameter as follows:

```
<infinispan>
 ...
 <default>

   <locking concurrencyLevel="${VALUE}"
     isolationLevel="${LEVEL}"
     lockAcquisitionTimeout="${TIME}"
     useLockStriping="${TRUE/FALSE}"
     writeSkewCheck="${TRUE/FALSE}" />
   ...

 </default>
</infinispan>
```

The ***`useLockStriping`*** parameter specifies whether a pool of shared locks are maintained for all entries that require locks. If set to **FALSE**, locks are created for each entry in the cache. If set to **TRUE**, lock striping is enabled and shared locks are used as required from the pool.

The ***concurrencyLevel*** is used to specify the size of the shared lock collection use when lock striping is enabled.

The ***isolationLevel*** parameter specifies the cache's isolation level. Valid isolation levels are **READ_COMMITTED** and **REPEATABLE_READ**.

The ***lockAcquisitionTimeout*** parameter specifies time (in milliseconds) after which a lock acquisition attempt times out.

The ***writeSkewCheck*** check determines if a modification to the entry from a different transaction should roll back the transaction. Write skew set to true requires ***isolation_level*** set to **REPEATABLE_READ**. The default value for ***writeSkewCheck*** and ***isolation_level*** are **FALSE** and **READ_COMMITTED** respectively.

Report a bug

CHAPTER 12. SET UP ISOLATION LEVELS

12.1. ABOUT ISOLATION LEVELS

Isolation levels determine when readers can view a concurrent write. ***READ_COMMITTED*** and ***REPEATABLE_READ*** are the two isolation modes offered in Red Hat JBoss Data Grid.

- **READ_COMMITTED**. This is the default isolation level because it is applicable to a wide variety of requirements.
- **REPEATABLE_READ**. This can be configured using the `locking` configuration element.

For isolation mode configuration examples in JBoss Data Grid, see the lock striping configuration samples:

- See Section 11.2, "Configure Lock Striping (Remote Client-Server Mode)" for a Remote Client-Server mode configuration sample.
- See Section 11.3, "Configure Lock Striping (Library Mode)" for a Library mode configuration sample.

Report a bug

12.2. ABOUT READ_COMMITTED

READ_COMMITTED is one of two isolation modes available in Red Hat JBoss Data Grid.

In JBoss Data Grid's ***READ_COMMITTED*** mode, write operations are made to copies of data rather than the data itself. A write operation blocks other data from being written, however writes do not block read operations. As a result, both ***READ_COMMITTED*** and ***REPEATABLE_READ*** modes permit read operations at any time, regardless of when write operations occur.

In ***READ_COMMITTED*** mode multiple reads of the same key within a transaction can return different results due to write operations modifying data between reads. This phenomenon is known as non-repeatable reads and is avoided in ***REPEATABLE_READ*** mode.

Report a bug

12.3. ABOUT REPEATABLE_READ

REPEATABLE_READ is one of two isolation modes available in Red Hat JBoss Data Grid.

Traditionally, ***REPEATABLE_READ*** does not allow write operations while read operations are in progress, nor does it allow read operations when write operations occur. This prevents the "non-repeatable read" phenomenon, which occurs when a single transaction has two read operations on the same row but the retrieved values differ (possibly due to a write operating modifying the value between the two read operations).

JBoss Data Grid's ***REPEATABLE_READ*** isolation mode preserves the value of a row before a modification occurs. As a result, the "non-repeatable read" phenomenon is avoided because a second read operation on the same row retrieves the preserved value rather than the new modified value. As a result, the two values retrieved by the two read operations will always match, even if a write operation occurs between the two reads.

Report a bug

PART VI. SET UP AND CONFIGURE A CACHE STORE

CHAPTER 13. CACHE STORES

The cache store connects Red Hat JBoss Data Grid to the persistent data store. The cache store is used to:

- fetch data from the data store when a copy is not in the cache.
- push modifications made to the data in cache back to the data store.

Caches that share the same cache manager can have different cache store configurations, as cache stores are associated with individual caches.

Report a bug

13.1. FILE SYSTEM BASED CACHE STORES

Red Hat JBoss Data Grid includes one file system based cache store: the **`SingleFileCacheStore`**.

The **`SingleFileCacheStore`** is a simple, file system based implementation and a replacement to the older file system based cache store: the **`FileCacheStore`**.

`SingleFileCacheStore` stores all key/value pairs and their corresponding metadata information in a single file. To speed up data location, it also keeps all keys and the positions of their values and metadata in memory. Hence, using the single file cache store slightly increases the memory required, depending on the key size and the amount of keys stored. Hence **`SingleFileCacheStore`** is not recommended for use cases where the keys are too big.

To reduce memory consumption, the size of the cache store can be set to a fixed number of entries to store in the file. However, this works only when Infinispan is used as a cache. When Infinispan used this way, data which is not present in Infinispan can be recomputed or re-retrieved from the authoritative data store and stored in Infinispan cache. The reason for this limitation is because once the maximum number of entries is reached, older data in the cache store is removed, so if Infinispan was used as an authoritative data store, it would lead to data loss which is undesirable in this use case

Due to its limitations, **`SingleFileCacheStore`** can be used in a limited capacity in production environments. It can not be used on shared file system (such as NFS and Windows shares) due to a lack of proper file locking, resulting in data corruption. Furthermore, file systems are not inherently transactional, resulting in file writing failures during the commit phase if the cache is used in a transactional context.

Report a bug

13.1.1. Single File Store Configuration (Remote Client-Server Mode)

The following is an example of a Single File Store configuration for Red Hat JBoss Data Grid's Remote Client-Server mode:

Procedure 13.1. Configure the Single File Store

1. **Add the Cache Name**
   The ***name*** parameter of the **`local-cache`** attribute is used to specify a name for the cache.

   ```
   <local-cache name="default">
   ```

2. **Per-cache Statistics**
   If ***statistics*** are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the ***statistics*** attribute to **false**.

   ```
   <local-cache name="default" statistics="true">
   ```

3. **Configure the `file-store` Element**
   The **file-store** element specifies configuration information for the single file store.

   The ***name*** parameter of the **file-store** element is used to specify a name for the file store.

   ```
   <local-cache name="default" statistics="true">
       <file-store name="myFileStore" />
   ```

4. **Set the passivation Parameter**
   The ***passivation*** parameter determines whether entries in the cache are passivated (**true**) or if the cache store retains a copy of the contents in memory (**false**).

   ```
   <local-cache name="default" statistics="true">
       <file-store name="myFileStore"
                   passivation="true" />
   ```

5. **Set the *purge* Parameter**
   The ***purge*** parameter specifies whether or not the cache store is purged when it is started. Valid values for this parameter are **true** and **false**.

   ```
   <local-cache name="default" statistics="true">
       <file-store name="myFileStore"
                   passivation="true"
                   purge="true" />
   ```

6. **Set the *shared* Parameter**
   The ***shared*** parameter is used when multiple cache instances share a cache store. This parameter can be set to prevent multiple cache instances writing the same modification multiple times. Valid values for this parameter are **true** and **false**. However, the ***shared*** parameter is not recommended for **file-store**.

   ```
   <local-cache name="default" statistics="true">
       <file-store name="myFileStore"
                   passivation="true"
                   purge="true"
                   shared="false" />
   ```

7. **Specify the Directory Path Within the *relative-to* Parameter**
   The ***relative-to*** property is the directory where the **file-store** stores the data. It is used to define a named path.

   The ***path*** property is the name of the file where the data is stored. It is a relative path name that is appended to the value of the ***relative-to*** property to determine the complete path.

   ```
   <local-cache name="default" statistics="true">
   ```

```
    <file-store name="myFileStore"
                passivation="true"
                purge="true"
                shared="false"
                relative-to="{PATH}"
                path="{DIRECTORY}" />
</local-cache>
```

8. **Specify the maximum number of entries**
   The ***maxEntries*** parameter provides maximum number of entries allowed. The default value is -1 for unlimited entries.

```
<local-cache name="default" statistics="true">
    <file-store name="myFileStore"
                passivation="true"
                purge="true"
                shared="false"
                relative-to="{PATH}"
                path="{DIRECTORY}"
                max-entries="10000"/>
</local-cache>
```

9. **Set the fetch-state Parameter**
   The ***fetch-state*** parameter when set to true fetches the persistent state when joining a cluster. If multiple cache stores are chained, only one of them can have this property enabled. Persistent state transfer with a shared cache store does not make sense, as the same persistent store that provides the data will just end up receiving it. Therefore, if a shared cache store is used, the cache does not allow a persistent state transfer even if a cache store has this property set to true. It is recommended to set this property to true only in a clustered environment. The default value for this parameter is false.

```
<local-cache name="default" statistics="true">
    <file-store name="myFileStore"
                passivation="true"
                purge="true"
                shared="false"
                relative-to="{PATH}"
                path="{DIRECTORY}"
                max-entries="10000"
                fetch-state="true"/>
</local-cache>
```

10. **Set the preload Parameter**
    The ***preload*** parameter when set to true, loads the data stored in the cache store into memory when the cache starts. However, setting this parameter to true affects the performance as the startup time is increased. The default value for this parameter is false.

```
<local-cache name="default" statistics="true">
    <file-store name="myFileStore"
                passivation="true"
                purge="true"
                shared="false"
                relative-to="{PATH}"
                path="{DIRECTORY}"
                max-entries="10000"
                fetch-state="true"
                preload="false"/>
</local-cache>
```

11. **Set the singleton Parameter**
    The ***singleton*** parameter enables a singleton store cache store. SingletonStore is a delegating cache store used when only one instance in a cluster can interact with the underlying store. However, ***singleton*** parameter is not recommended for **file-store**.

```
<local-cache name="default" statistics="true">
    <file-store name="myFileStore"
                passivation="true"
                purge="true"
                shared="false"
                relative-to="{PATH}"
                path="{DIRECTORY}"
                max-entries="10000"
                fetch-state="true"
                preload="false"
                singleton="true"/>
</local-cache>
```

Report a bug

13.1.2. Single File Store Configuration (Library Mode)

In Red Hat JBoss Grid's Library mode, configure a Single File Cache Store as follows:.

Procedure 13.2. Configuring the Single File Store in Library Mode

The singleFile element is used to configure the Single File Cache Store. Configure the following in **infinispan.xml**.

1. Add the name value to the ***namedCache*** element. The following is an example of this step:

```
<namedCache name="writeThroughToFile">
```

2. In the **persistence** element, set the ***passivation*** parameter to **false**. Possible values are true and false.

```
<namedCache name="writeThroughToFile">
      <persistence passivation="false" />
```

3. Set up a single file configuration using the **singleFile** element:

   - ***fetchPersistentState*** - If set to **true**, the persistent state is fetched when joining a cluster. If multiple cache stores are chained, only one cache store can have this property set to **true**. The default for this value is **false**.

   - The ***ignoreModifications*** parameter determines whether operations that modify the cache (e.g. put, remove, clear, store, etc.) do not affect the cache store. As a result, the cache store can become out of sync with the cache.

- The ***purgeOnStartup*** parameter specifies whether the cache store is purged when initially started.

- The ***shared*** parameter is set to **true** when multiple cache instances share a cache store, which prevents multiple cache instances writing the same modification individually. The default for this attribute is **false**. However, the ***shared*** parameter is not recommended for **file-store**.

- The ***preload*** parameter sets whether the cache store data is pre-loaded into memory and becomes immediately accessible after starting up. The disadvantage of setting this to true is that the start up time increases. The default value for this attribute is **false**.

- The ***location*** parameter points to the location of file store.

- The ***maxEntries*** parameter provides maximum number of entries allowed. The default value is -1 for unlimited entries.

- The ***maxKeysInMemory*** parameter is used to speed up data lookup. The single file store keeps an index of keys and their positions in the file using the ***maxKeysInMemory*** parameter. The default value for this parameter is -1.

```
<namedCache name="writeThroughToFile">
      <persistence passivation="false">
         <singleFile fetchPersistentState="true"
                     ignoreModifications="false"
                     purgeOnStartup="false"
                     shared="false"
                     preload="false"
                     location="/tmp/Another-FileCacheStore-
Location"
                     maxEntries="100"
                     maxKeysInMemory="100">
         </singleFile>
      </persistence>
 </namedCache>
```

4. Add the **async** element to configure the asynchronous settings:

- The ***enabled*** parameter determines whether the file store is asynchronous.

- The ***threadPoolSize*** parameter specifies the number of threads that concurrently apply modifications to the store. The default value for this parameter is **5**.

- The ***flushLockTimeout*** parameter specifies the time to acquire the lock which guards the state to be flushed to the cache store periodically. The default value for this parameter is **1**.

- The ***modificationQueueSize*** parameter specifies the size of the modification queue for the asynchronous store. If updates are made at a rate that is faster than the underlying cache store can process this queue, then the asynchronous store behaves like a synchronous store for that period, blocking until the queue can accept more elements. The default value for this parameter is **1024** elements.

- The ***shutdownTimeout*** parameter specifies the time to stop the cache store. Default value for this parameter is **25** seconds.

```
<namedCache name="writeThroughToFile">
      <persistence passivation="false">
         <singleFile fetchPersistentState="true"
                     ignoreModifications="false"
                     purgeOnStartup="false"
                     shared="false"
                     preload="false"
                     location="/tmp/Another-FileCacheStore-
Location"
                     maxEntries="100"
                     maxKeysInMemory="100">
            <async enabled="true"
                threadPoolSize="500"
                flushLockTimeout="1"
            modificationQueueSize="1024"
            shutdownTimeout="25000"/>
        </singleFile>
      </persistence>
 </namedCache>
```

Report a bug

13.1.3. Migrating data from FileCacheStore to SingleFileCacheStore

Red Hat JBoss Data Grid 6.2 stores data in a different format than previous versions of JBoss Data Grid. As a result, the newer version of JBoss Data Grid cannot read data stored by older versions. Use rolling upgrades to upgrade persisted data from the format used by the old JBoss Data Grid to the new format. Additionally, the newer version of JBoss Data Grid also stores persistence configuration information in a different location.

Rolling upgrades is the process by which a JBoss Data Grid installation is upgraded without a service shutdown. In Library mode, it refers to a node installation where JBoss Data Grid is running in Library mode. For JBoss Data Grid servers, it refers to the server side components. The upgrade can be due to either hardware or software change such as upgrading JBoss Data Grid.

Rolling upgrades are available in JBoss Data Grid's Library and Remote Client-Server modes.

Report a bug

13.2. REMOTE CACHE STORES

The **`RemoteCacheStore`** is an implementation of the cache loader that stores data in a remote Red Hat JBoss Data Grid cluster. The **`RemoteCacheStore`** uses the Hot Rod client-server architecture to communicate with the remote cluster.

For remote cache stores, Hot Rod provides load balancing, fault tolerance and the ability to fine tune the connection between the **`RemoteCacheStore`** and the cluster.

Report a bug

13.2.1. Remote Cache Store Configuration (Remote Client-Server Mode)

The following is a sample remote cache store configuration for Red Hat JBoss Data Grid's Remote Client-Server mode.

Procedure 13.3. Configure the Remote Cache Store

The parameters of the **remote-store** element define the following information:

1. The ***cache*** parameter defines the name for the remote cache. If left undefined, the default cache is used instead.

   ```
   <remote-store cache="default">
   ```

2. The ***socket-timeout*** parameter sets whether the value defined in ***SO_TIMEOUT*** (in milliseconds) applies to remote Hot Rod servers on the specified timeout. A timeout value of **0** indicates an infinite timeout.

   ```
   <remote-store cache="default"
                 socket-timeout="60000">
   ```

3. The ***tcp-no-delay*** sets whether **TCP_NODELAY** applies on socket connections to remote Hot Rod servers.

   ```
   <remote-store cache="default"
                 socket-timeout="60000"
                 tcp-no-delay="true">
   ```

4. The ***hotrod-wrapping*** sets whether a wrapper is required for Hot Rod on the remote store.

   ```
   <remote-store cache="default"
                 socket-timeout="60000"
                 tcp-no-delay="true"
                 hotrod-wrapping="true">
   ```

5. The single parameter for the **remote-server** element is as follows:

   a. The ***outbound-socket-binding*** parameter sets the outbound socket binding for the remote server.

   ```
   <remote-store cache="default"
                 socket-timeout="60000"
                 tcp-no-delay="true"
                 hotrod-wrapping="true">
    <remote-server outbound-socket-binding="remote-store-hotrod-server"
   />
   </remote-store>
   ```

Report a bug

13.2.2. Remote Cache Store Configuration (Library Mode)

Procedure 13.4. Remote Cache Store Configuration

The following is a sample remote cache store configuration for Red Hat JBoss Data Grid's Library mode.

1. **Configure the Persistence Element**
   Create a **persistence** element, with the ***passivation*** set to **false**.

```
<persistence passivation="false" />
```

2. Create a **remoteStore** element within the **persistence** element to configure the attributes for the Remote Cache Store.

```
<persistence passivation="false">
 <remoteStore xmlns="urn:infinispan:config:remote:6.0"
       fetchPersistentState="false"
       shared="true"
       preload="false"
       ignoreModifications="false"
       purgeOnStartup="false"
       tcpNoDelay="true"
       pingOnStartup="true"
       keySizeEstimate="62"
       valueSizeEstimate="512"
       forceReturnValues="false">
 </remoteStore>
</persistence>
```

   - Add the ***fetchPersistentState*** attribute. If set to **true**, the persistent state is fetched when the remote cache joins the cluster. If multiple cache stores are chained, only one cache store can have this property set to **true** . The default for this value is **false**.
   - Add the ***shared*** attribute. This is set to **true** when multiple cache instances share a cache store, which prevents multiple cache instances writing the same modification individually. The default for this attribute is **false**.
   - Add the ***preload*** attribute. When set to **true**, the cache store data is pre-loaded into memory and becomes immediately accessible after starting up. The disadvantage of setting this to **true** is that the start up time increases. The default value for this attribute is **false**.
   - Add the ***ignoreModifications*** attribute. When set to true, this attribute prevents cache modification operations such as put, remove, clear, store, etc. from affecting the cache store. As a result, the cache store can become out of sync with the cache. The default value for this attribute is **false**.
   - Add the ***purgeOnStartup*** attribute. If set to **true**, the cache store is purged during the start up process. The default value for this attribute is **false**.
   - Add the ***tcpNoDelay*** attribute. If set to **true**, this triggers the **TCP *NODELAY*** stack. The default value for this attribute is **true**.
   - Add the ***pingOnStartup*** attribute. If set to **true**, a ping request is sent to a back end server to fetch the cluster topology. The default value for this attribute is **true**.
   - Add the ***keySizeEstimate*** attribute. This value is the class name of the driver user to connect to the database. The default value for this attribute is **64**.

- Add the ***valueSizeEstimate*** attribute. This value is the size of the byte buffers when serializing and deserializing values. The default value for this attribute is **512**.

- Add the ***forceReturnValues*** attribute. This attribute sets whether ***FORCE_RETURN_VALUE*** is enabled for all calls. The default value for this attribute is **false**.

3. Create a **servers** element within the **remoteStore** element to set up the server information. Add a **server** element within the general **servers** element to add the information for a single server.

```
<persistence passivation="false">
 <remoteStore xmlns="urn:infinispan:config:remote:6.0"
       fetchPersistentState="false"
       shared="true"
       preload="false"
       ignoreModifications="false"
       purgeOnStartup="false"
       tcpNoDelay="true"
       pingOnStartup="true"
       keySizeEstimate="62"
       valueSizeEstimate="512"
       forceReturnValues="false">
  <servers>
   <server host="127.0.0.1"
    port="19711"/>
  </servers>
 </remoteStore>
</persistence>
```

- Add the ***host*** attribute to configure the host address.

- Add the ***port*** attribute to configure the port used by the Remote Cache Store.

4. Create a **connectionPool** element to the **remoteStore** element.

```
<persistence passivation="false">
 <remoteStore xmlns="urn:infinispan:config:remote:6.0"
       fetchPersistentState="false"
       shared="true"
       preload="false"
       ignoreModifications="false"
       purgeOnStartup="false"
       tcpNoDelay="true"
       pingOnStartup="true"
       keySizeEstimate="62"
       valueSizeEstimate="512"
       forceReturnValues="false">
  <servers>
   <server host="127.0.0.1"
    port="19711"/>
  </servers>
  <connectionPool maxActive="99"
    maxIdle="97"
```

```
        maxTotal="98" />
   </remoteStore>
  </persistence>
```

- Add a ***maxActive*** attribute. This indicates the maximum number of active connections for each server at a time. The default value for this attribute is **-1** which indicates an infinite number of active connections.

- Add a ***maxIdle*** attribute. This indicates the maximum number of idle connections for each server at a time. The default value for this attribute is **-1** which indicates an infinite number of idle connections.

- Add a ***maxTotal*** attribute. This indicates the maximum number of persistent connections within the combined set of servers. The default setting for this attribute is **-1** which indicates an infinite number of connections.

Report a bug

13.2.3. Define the Outbound Socket for the Remote Cache Store

The Hot Rod server used by the remote cache store is defined using the **outbound-socket-binding** element in a **standalone.xml** file.

An example of this configuration in the **standalone.xml** file is as follows:

```
<server>
    ...
    <socket-binding-group name="standard-sockets"
         default-interface="public"
         port-offset="${jboss.socket.binding.port-offset:0}">
        ...
        <outbound-socket-binding name="remote-store-hotrod-server">
            <remote-destination host="remote-host"
               port="11222"/>
        </outbound-socket-binding>
    </socket-binding-group>
</server>
```

Report a bug

13.3. CUSTOM CACHE STORES

Custom cache stores are a customized implementation of Red Hat JBoss Data Grid cache stores.

IMPORTANT

Custom Cache Stores are not supported in Red Hat JBoss Data Grid.

Report a bug

13.3.1. Custom Cache Store Configuration (Remote Client-Server Mode)

The following is a sample configuration for a custom cache store in Red Hat JBoss Data Grid's Remote Client-Server mode:

```
<local-cache name="default"
 statistics="true">
     <store class="my.package.CustomCacheStore">
         <properties>
             <property name="customStoreProperty" value="10" />
         </properties>
     </store>
</local-cache>
```

IMPORTANT

To allow JBoss Data Grid to locate the defined class, create a module using the module of another (relevant) cache store as a template and add it to the **`org.jboss.as.clustering.infinispan`** module dependencies.

IMPORTANT

Custom Cache Stores are not supported in JBoss Data Grid.

Report a bug

13.3.2. Custom Cache Store Configuration (Library Mode)

The following is a sample configuration for a custom cache store in Red Hat JBoss Data Grid's Library mode:

```
<persistence>
 <store class="org.infinispan.custom.CustomCacheStore"
        preload="true"
        shared="true">
  <properties>
   <property name="customStoreProperty"
      value="10" />
  </properties>
            </store>
</persistence>
```

IMPORTANT

Custom Cache Stores are not supported in JBoss Data Grid.

Report a bug

13.4. ABOUT ASYNCHRONOUS CACHE STORE MODIFICATIONS

In Red Hat JBoss Data Grid, modifications to asynchronous cache stores are coalesced or aggregated for the interval that is being applied by the modification thread. This ensures that multiple modifications for the same key are detected, only the last state of the key is applied. This improves efficiency by reducing the number of calls to the cache store.

Report a bug

13.5. LEVELDB CACHE STORE

LevelDB is a key-value storage engine that provides an ordered mapping from string keys to string values.

The LevelDB Cache Store uses two filesystem directories. Each directory is configured for a LevelDB database. One directory stores the non-expired data and the second directory stores the expired keys before a purge.

Report a bug

13.5.1. Configuring LevelDB Cache Store

Procedure 13.5. To configure LevelDB Cache Store:

1. Obtain the **leveldbjni-all-1.7.jar** file from https://github.com/fusesource/leveldbjni and copy it to the **modules/system/layers/base/org/fusesource/leveldbjni-all/main** directory.

2. Modify the **module.xml** file to include JAR files in the resources:

```
<module xmlns="urn:jboss:module:1.1"
name="org.fusesource.leveldbjni-all">
   <properties>
      <property name="jboss.api" value="private"/>
   </properties>

   <resources>
      <resource-root path="leveldbjni-all-1.7.jar"/>
        <!-- Insert resources here -->
   </resources>

   <dependencies>
   </dependencies>
</module>
```

3. Add the following to a cache definition in **standalone.xml** to configure the database:

```
<leveldb-store path="/path/to/leveldb/data"
    passivation="false"
    purge="false" >
    <expiration path="/path/to/leveldb/expires/data" />
</leveldb-store>
```

NOTE

The LevelDB library and JNI connector are not part of the JBoss Data Grid 6.2 distribution. To use the tested version, add the file **leveldbjni-all-1.7.jar** from https://github.com/fusesource/leveldbjni to your JBoss Data Grid deployment.

Report a bug

13.5.2. LevelDB Cache Store Programmatic Configuration

The following is a sample programmatic configuration of LevelDB Cache Store.

Procedure 13.6. LevelDB Cache Store programmatic configuration

1. **Create a New Configuration Builder**
   Use the ***ConfigurationBuilder*** to create a new configuration object.

   ```
   Configuration cacheConfig = new ConfigurationBuilder().persistence()
   ```

2. **Add the LevelDBStoreConfigurationBuilder Store**
   Add the store to the ***LevelDBCacheStoreConfigurationBuilder*** instance to build its configuration.

   ```
   Configuration cacheConfig = new ConfigurationBuilder().persistence()
                     .addStore(LevelDBStoreConfigurationBuilder.class)
   ```

3. **Set Up Location**
   Set the LevelDB Cache Store location path. The specified path stores the primary cache store data. The directory is automatically created if it does not exist.

   ```
   Configuration cacheConfig = new ConfigurationBuilder().persistence()
                     .addStore(LevelDBStoreConfigurationBuilder.class)
                     .location("/tmp/leveldb/data")
   ```

4. **Set Up Expired Location**
   Specify the location for expired data using the ***expiredLocation*** parameter for the LevelDB Store. The specified path stores expired data before it is purged. The directory is automatically created if it does not exist.

   ```
   Configuration cacheConfig = new ConfigurationBuilder().persistence()
                     .addStore(LevelDBStoreConfigurationBuilder.class)
                     .location("/tmp/leveldb/data")
                     .expiredLocation("/tmp/leveldb/expired").build();
   ```

NOTE

Programmatic configurations can only be used with Red Hat JBoss Data Grid Library mode.

Report a bug

13.5.3. LevelDB Cache Store Sample XML Configuration (Library Mode)

The following is a sample XML configuration of LevelDB Cache Store.

Procedure 13.7. LevelDB Cache Store Sample XML Configuration

1. **Add the namedCache Element**
   Specify the LevelDB Cache Store's name using the ***name*** parameter in the ***namedCache*** element as follows:

   ```
   <namedCache name="vehicleCache">
   ```

2. **Add the persistence Element**
   Specify the value for ***passivation*** parameter in the ***persistence*** element as follows. Possible values are true and false.

   ```
   <namedCache name="vehicleCache">
       <persistence passivation="false">
   ```

3. **Add the leveldbStore Element**
   Specify the location to store the primary cache store date using the ***location*** parameter in the ***leveldbStore*** element as follows. The directory is automatically created if it does not exist. The following is an example of this step:

   ```
   <namedCache name="vehicleCache">
         <persistence passivation="false">
             <leveldbStore location="/path/to/leveldb/data" />
   ```

4. **Set the Expired Value**
   Specify the location for expired data using the ***expiredLocation*** parameter as follows. The directory stores expired data before it is purged. The directory is automatically created if it does not exist.

   ```
   <namedCache name="vehicleCache">
         <persistence passivation="false">
             <leveldbStore location="/path/to/leveldb/data"
                        expiredLocation="/path/to/expired/data" />
   ```

5. **Set the Shared Parameter**
   Specify the value for ***shared*** parameter in the ***leveldbStore*** element as follows. Possible values are true and false.

   ```
   <namedCache name="vehicleCache">
         <persistence passivation="false">
             <leveldbStore location="/path/to/leveldb/data"
                        expiredLocation="/path/to/expired/data"
   shared="true" />
   ```

6. **Set the Preload Parameter**
   Specify the value for ***preload*** parameter in the ***leveldbStore*** element as follows. Possible values are true and false.

   ```
   <namedCache name="vehicleCache">
         <persistence passivation="false">
             <leveldbStore location="/path/to/leveldb/data"
                        expiredLocation="/path/to/expired/data"
   ```

```
shared="true" preload="true"/>
      </persistence>
   </namedCache>
```

Report a bug

CHAPTER 14. DATASOURCE MANAGEMENT

14.1. ABOUT JDBC

The JDBC API is the standard that defines how databases are accessed by Java applications. An application configures a datasource that references a JDBC driver. Application code can then be written against the driver, rather than the database. The driver converts the code to the database language. This means that if the correct driver is installed, an application can be used with any supported database.

The JDBC 4.0 specification is defined here: http://jcp.org/en/jsr/detail?id=221.

To get started with JDBC and datasources, see the section about JDBC Drivers.

Report a bug

14.2. TYPES OF DATASOURCES

The two general types of resources are referred to as `datasources` and `XA datasources`.

Non-XA datasources are used for applications which do not use transactions, or applications which use transactions with a single database.

XA datasources are used by applications whose transactions are distributed across multiple databases. XA datasources introduce additional overhead.

You specify the type of your datasource when you create it in the Management Console or Management CLI.

Report a bug

14.3. USING JDBC DRIVERS

14.3.1. Install a JDBC Driver as a Core Module

Prerequisites

Before performing this task, you need to meet the following prerequisites:

- Download the JDBC driver from your database vendor. JDBC driver download locations are listed here: Section 14.3.2, "JDBC Driver Download Locations" .
- Extract the archive.

Procedure 14.1. Install a JDBC Driver as a Core Module

1. Create a file path structure under the *EAP_HOME*`/modules/` directory. For example, for a MySQL JDBC driver, create a directory structure as follows: *EAP_HOME*`/modules/com/mysql/main/`.
2. Copy the JDBC driver JAR into the `main/` subdirectory.
3. In the `main/` subdirectory, create a `module.xml` file. The following is an example of a `module.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.15.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

4. Start the Server.

5. Start the Management CLI.

6. Run the following CLI command to add the JDBC driver module as a driver:

```
/subsystem=datasources/jdbc-driver=DRIVER_NAME:add(driver-
name=DRIVER_NAME,driver-module-name=MODULE_NAME,driver-xa-
datasource-class-name=XA_DATASOURCE_CLASS_NAME)
```

Example 14.1. Example CLI Command

```
/subsystem=datasources/jdbc-driver=mysql:add(driver-
name=mysql,driver-module-name=com.mysql,driver-xa-datasource-
class-name=com.mysql.jdbc.jdbc2.optional.MysqlXADataSource)
```

Result

The JDBC driver is now installed and set up as a core module, and is available to be referenced by application datasources.

Report a bug

14.3.2. JDBC Driver Download Locations

The following table gives the standard download locations for JDBC drivers of common databases used with JBoss EAP 6. These links point to third-party websites which are not controlled or actively monitored by Red Hat. For the most up-to-date drivers for your database, check your database vendor's documentation and website.

Table 14.1. JDBC driver download locations

Vendor	Download Location
MySQL	http://www.mysql.com/products/connector/
PostgreSQL	http://jdbc.postgresql.org/
Oracle	http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html

Vendor	Download Location
IBM	http://www-306.ibm.com/software/data/db2/java/
Sybase	http://www.sybase.com/products/allproductsa-z/softwaredeveloperkit/jconnect
Microsoft	http://msdn.microsoft.com/data/jdbc/

Report a bug

14.3.3. Access Vendor Specific Classes

Summary

This topic covers the steps required to use the JDBC specific classes. This is necessary when an application uses a vendor specific functionality that is not part of the JDBC API.

WARNING

This is advanced usage. Only applications that need functionality not found in the JDBC API should implement this procedure.

IMPORTANT

This process is required when using the reauthentication mechanism, and accessing vendor specific classes.

IMPORTANT

Follow the vendor specific API guidelines closely, as the connection is being controlled by the IronJacamar container.

Prerequisites

- Section 14.3.1, “Install a JDBC Driver as a Core Module” .

Procedure 14.2. Add a Dependency to the Application

- ◦ **Configure the `MANIFEST.MF` file**

  a. Open the application's `META-INF/MANIFEST.MF` file in a text editor.

  b. Add a dependency for the JDBC module and save the file.

```
Dependencies: MODULE_NAME
```

Example 14.2. Example Dependency

```
Dependencies: com.mysql
```

- a. **Create a `jboss-deployment-structure.xml` file**
  Create a file called **`jboss-deployment-structure.xml`** in the **`META-INF/`** or **`WEB-INF`** folder of the application.

Example 14.3. Example `jboss-deployment-structure.xml` file

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="com.mysql" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

Example 14.4. Access the Vendor Specific API

The example below accesses the MySQL API.

```
import java.sql.Connection;
import org.jboss.jca.adapters.jdbc.WrappedConnection;

  Connection c = ds.getConnection();
  WrappedConnection wc = (WrappedConnection)c;
  com.mysql.jdbc.Connection mc = wc.getUnderlyingConnection();
```

Report a bug

14.4. DATASOURCE CONFIGURATION

14.4.1. Datasource Parameters

Table 14.2. Datasource parameters common to non-XA and XA datasources

Parameter	Description
jndi-name	The unique JNDI name for the datasource.
pool-name	The name of the management pool for the datasource.
enabled	Whether or not the datasource is enabled.

Parameter	Description
use-java-context	Whether to bind the datasource to global JNDI.
spy	Enable `spy` functionality on the JDBC layer. This logs all JDBC traffic to the datasource. Note that the logging category `jboss.jdbc.spy` must also be set to the log level **DEBUG** in the logging subsystem.
use-ccm	Enable the cached connection manager.
new-connection-sql	A SQL statement which executes when the connection is added to the connection pool.
transaction-isolation	One of the following: • TRANSACTION_READ_UNCOMMITTED • TRANSACTION_READ_COMMITTED • TRANSACTION_REPEATABLE_READ • TRANSACTION_SERIALIZABLE • TRANSACTION_NONE
url-delimiter	The delimiter for URLs in a connection-url for High Availability (HA) clustered databases.
url-selector-strategy-class-name	A class that implements interface `org.jboss.jca.adapters.jdbc.URLSelectorStrategy`.
security	Contains child elements which are security settings. See Table 14.7, "Security parameters".
validation	Contains child elements which are validation settings. See Table 14.8, "Validation parameters".
timeout	Contains child elements which are timeout settings. See Table 14.9, "Timeout parameters".
statement	Contains child elements which are statement settings. See Table 14.10, "Statement parameters".

Table 14.3. Non-XA datasource parameters

Parameter	Description
jta	Enable JTA integration for non-XA datasources. Does not apply to XA datasources.

Parameter	Description
connection-url	The JDBC driver connection URL.
driver-class	The fully-qualified name of the JDBC driver class.
connection-property	Arbitrary connection properties passed to the method `Driver.connect(url,props)`. Each connection-property specifies a string name/value pair. The property name comes from the name, and the value comes from the element content.
pool	Contains child elements which are pooling settings. See Table 14.5, "Pool parameters common to non-XA and XA datasources".

Table 14.4. XA datasource parameters

Parameter	Description
xa-datasource-property	A property to assign to implementation class `XADataSource`. Specified by *name=value*. If a setter method exists, in the format `set`***`Name`***, the property is set by calling a setter method in the format of `set`***`Name`***`(`***`value`***`)`.
xa-datasource-class	The fully-qualified name of the implementation class `javax.sql.XADataSource`.
driver	A unique reference to the classloader module which contains the JDBC driver. The accepted format is *driverName#majorVersion.minorVersion*.
xa-pool	Contains child elements which are pooling settings. See Table 14.5, "Pool parameters common to non-XA and XA datasources" and Table 14.6, "XA pool parameters".
recovery	Contains child elements which are recovery settings. See Table 14.11, "Recovery parameters".

Table 14.5. Pool parameters common to non-XA and XA datasources

Parameter	Description
min-pool-size	The minimum number of connections a pool holds.
max-pool-size	The maximum number of connections a pool can hold.

Parameter	Description
prefill	Whether to try to prefill the connection pool. An empty element denotes a `true` value. The default is `false`.
use-strict-min	Whether the pool-size is strict. Defaults to `false`.
flush-strategy	Whether the pool is flushed in the case of an error. Valid values are: • FailingConnectionOnly • IdleConnections • EntirePool The default is `FailingConnectionOnly`.
allow-multiple-users	Specifies if multiple users will access the datasource through the getConnection(user, password) method, and whether the internal pool type accounts for this behavior.

Table 14.6. XA pool parameters

Parameter	Description
is-same-rm-override	Whether the `javax.transaction.xa.XAResource.isSameRM(XAResource)` class returns `true` or `false`.
interleaving	Whether to enable interleaving for XA connection factories.
no-tx-separate-pools	Whether to create separate sub-pools for each context. This is required for Oracle datasources, which do not allow XA connections to be used both inside and outside of a JTA transaction. Using this option will cause your total pool size to be twice `max-pool-size`, because two actual pools will be created.
pad-xid	Whether to pad the Xid.
wrap-xa-resource	Whether to wrap the XAResource in an `org.jboss.tm.XAResourceWrapper` instance.

Table 14.7. Security parameters

Parameter	Description
user-name	The username to use to create a new connection.
password	The password to use to create a new connection.
security-domain	Contains the name of a JAAS security-manager which handles authentication. This name correlates to the application-policy/name attribute of the JAAS login configuration.
reauth-plugin	Defines a reauthentication plug-in to use to reauthenticate physical connections.

Table 14.8. Validation parameters

Parameter	Description
valid-connection-checker	An implementation of interface **`org.jboss.jca.adaptors.jdbc.ValidConnectionChecker`** which provides a **`SQLException.isValidConnection(Connection e)`** method to validate a connection. An exception means the connection is destroyed. This overrides the parameter **`check-valid-connection-sql`** if it is present.
check-valid-connection-sql	An SQL statement to check validity of a pool connection. This may be called when a managed connection is taken from a pool for use.
validate-on-match	Indicates whether connection level validation is performed when a connection factory attempts to match a managed connection for a given set. Specifying "true" for **`validate-on-match`** is typically not done in conjunction with specifying "true" for **`background-validation`**. **`Validate-on-match`** is needed when a client must have a connection validated prior to use. This parameter is false by default.
background-validation	Specifies that connections are validated on a background thread. Background validation is a performance optimization when not used with **`validate-on-match`**. If **`validate-on-match`** is true, using **`background-validation`** could result in redundant checks. Background validation does leave open the opportunity for a bad connection to be given to the client for use (a connection goes bad between the time of the validation scan and prior to being handed to the client), so the client application must account for this possibility.

Parameter	Description
background-validation-millis	The amount of time, in milliseconds, that background validation runs.
use-fast-fail	If true, fail a connection allocation on the first attempt, if the connection is invalid. Defaults to `false`.
stale-connection-checker	An instance of `org.jboss.jca.adapters.jdbc.StaleConnectionChecker` which provides a Boolean `isStaleConnection(SQLException e)` method. If this method returns `true`, the exception is wrapped in an `org.jboss.jca.adapters.jdbc.StaleConnectionException`, which is a subclass of `SQLException`.
exception-sorter	An instance of `org.jboss.jca.adapters.jdbc.ExceptionSorter` which provides a Boolean `isExceptionFatal(SQLException e)` method. This method validates whether an exception is broadcast to all instances of `javax.resource.spi.ConnectionEventListener` as a `connectionErrorOccurred` message.

Table 14.9. Timeout parameters

Parameter	Description
use-try-lock	Uses `tryLock()` instead of `lock()`. This attempts to obtain the lock for the configured number of seconds, before timing out, rather than failing immediately if the lock is unavailable. Defaults to `60` seconds. As an example, to set a timeout of 5 minutes, set `<use-try-lock>300</use-try-lock>`.
blocking-timeout-millis	The maximum time, in milliseconds, to block while waiting for a connection. After this time is exceeded, an exception is thrown. This blocks only while waiting for a permit for a connection, and does not throw an exception if creating a new connection takes a long time. Defaults to 30000, which is 30 seconds.
idle-timeout-minutes	The maximum time, in minutes, before an idle connection is closed. The actual maximum time depends upon the idleRemover scan time, which is half of the smallest `idle-timeout-minutes` of any pool.

Parameter	Description
set-tx-query-timeout	Whether to set the query timeout based on the time remaining until transaction timeout. Any configured query timeout is used if no transaction exists. Defaults to `false`.
query-timeout	Timeout for queries, in seconds. The default is no timeout.
allocation-retry	The number of times to retry allocating a connection before throwing an exception. The default is `0`, so an exception is thrown upon the first failure.
allocation-retry-wait-millis	How long, in milliseconds, to wait before retrying to allocate a connection. The default is 5000, which is 5 seconds.
xa-resource-timeout	If non-zero, this value is passed to method `XAResource.setTransactionTimeout`.

Table 14.10. Statement parameters

Parameter	Description
track-statements	Whether to check for unclosed statements when a connection is returned to a pool and a statement is returned to the prepared statement cache. If false, statements are not tracked. **Valid values** • `true`: statements and result sets are tracked, and a warning is issued if they are not closed. • `false`: neither statements or result sets are tracked. • `nowarn`: statements are tracked but no warning is issued. This is the default.
prepared-statement-cache-size	The number of prepared statements per connection, in a Least Recently Used (LRU) cache.
share-prepared-statements	Whether asking for the same statement twice without closing it uses the same underlying prepared statement. The default is `false`.

Table 14.11. Recovery parameters

Parameter	Description
recover-credential	A username/password pair or security domain to use for recovery.
recover-plugin	An implementation of the `org.jboss.jca.core.spi.recoveryRecoveryPlugin` class, to be used for recovery.

Report a bug

14.4.2. Datasource Connection URLs

Table 14.12. Datasource Connection URLs

Datasource	Connection URL
PostgreSQL	jdbc:postgresql://*SERVER_NAME*:*PORT*/*DATABASE_NAME*
MySQL	jdbc:mysql://*SERVER_NAME*:*PORT*/*DATABASE_NAME*
Oracle	jdbc:oracle:thin:@*ORACLE_HOST*:*PORT*:*ORACLE_SID*
IBM DB2	jdbc:db2://*SERVER_NAME*:*PORT*/*DATABASE_NAME*
Microsoft SQLServer	jdbc:microsoft:sqlserver://*SERVER_NAME*:*PORT*;DatabaseName=*DATABASE_NAME*

Report a bug

14.4.3. Datasource Extensions

Datasource deployments can use several extensions in the JDBC resource adapter to improve the connection validation, and check whether an exception should reestablish the connection. Those extensions are:

Table 14.13. Datasource Extensions

Datasource Extension	Configuration Parameter	Description
org.jboss.jca.adapters.jdbc.spi.ExceptionSorter	<exception-sorter>	Checks whether an SQLException is fatal for the connection on which it was thrown

Datasource Extension	Configuration Parameter	Description
org.jboss.jca.adapters.jdbc.spi.StaleConnection	<stale-connection-checker>	Wraps stale SQLExceptions in a `org.jboss.jca.adapters.jdbc.StaleConnectionException`
org.jboss.jca.adapters.jdbc.spi.ValidConnection	<valid-connection-checker>	Checks whether a connection is valid for use by the application

JBoss EAP 6 also features implementations of these extensions for several supported databases.

Extension Implementations

Generic

- org.jboss.jca.adapters.jdbc.extensions.novendor.NullExceptionSorter
- org.jboss.jca.adapters.jdbc.extensions.novendor.NullStaleConnectionChecker
- org.jboss.jca.adapters.jdbc.extensions.novendor.NullValidConnectionChecker
- org.jboss.jca.adapters.jdbc.extensions.novendor.JDBC4ValidConnectionChecker

PostgreSQL

- org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter
- org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker

MySQL

- org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter
- org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLReplicationValidConnectionChecker
- org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionChecker

IBM DB2

- org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter
- org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionChecker
- org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionChecker

Sybase

- org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorter
- org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectionChecker

Microsoft SQLServer

- org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionChecker

Oracle

- org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorter
- org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnectionChecker
- org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectionChecker

Report a bug

14.4.4. View Datasource Statistics

You can view statistics from defined datasources for both the `jdbc` and `pool` using appropriately modified versions of the commands below:

Procedure 14.3.

-
  ```
  /subsystem=datasources/data-source=ExampleDS/statistics=jdbc:read-resource(include-runtime=true)
  ```

  ```
  /subsystem=datasources/data-source=ExampleDS/statistics=pool:read-resource(include-runtime=true)
  ```

NOTE

Ensure you specify the ***`include-runtime=true`*** argument, as all statistics are runtime only information and the default is `false`.

Report a bug

14.4.5. Datasource Statistics

Core Statistics

The following table contains a list of the supported datasource core statistics:

Table 14.14. Core Statistics

Name	Description
`ActiveCount`	The number of active connections. Each of the connections is either in use by an application or available in the pool
`AvailableCount`	The number of available connections in the pool.
`AverageBlockingTime`	The average time spent blocking on obtaining an exclusive lock on the pool. The value is in milliseconds.

Name	Description
`AverageCreationTime`	The average time spent creating a connection. The value is in milliseconds.
`CreatedCount`	The number of connections created.
`DestroyedCount`	The number of connections destroyed.
`InUseCount`	The number of connections currently in use.
`MaxCreationTime`	The maximum time it took to create a connection. The value is in milliseconds.
`MaxUsedCount`	The maximum number of connections used.
`MaxWaitCount`	The maximum number of requests waiting for a connection at the same time.
`MaxWaitTime`	The maximum time spent waiting for an exclusive lock on the pool.
`TimedOut`	The number of timed out connections.
`TotalBlockingTime`	The total time spent waiting for an exclusive lock on the pool. The value is in milliseconds.
`TotalCreationTime`	The total time spent creating connections. The value is in milliseconds.
`WaitCount`	The number of requests that had to wait for a connection.

JDBC Statistics

The following table contains a list of the supported datasource JDBC statistics:

Table 14.15. JDBC Statistics

Name	Description
`PreparedStatementCacheAccessCount`	The number of times that the statement cache was accessed.
`PreparedStatementCacheAddCount`	The number of statements added to the statement cache.
`PreparedStatementCacheCurrentSize`	The number of prepared and callable statements currently cached in the statement cache.

Name	Description
PreparedStatementCacheDeleteCount	The number of statements discarded from the cache.
PreparedStatementCacheHitCount	The number of times that statements from the cache were used.
PreparedStatementCacheMissCount	The number of times that a statement request could not be satisfied with a statement from the cache.

Report a bug

14.5. EXAMPLE DATASOURCES

14.5.1. Example PostgreSQL Datasource

Example 14.5.

The example below is a PostgreSQL datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <datasource jndi-name="java:jboss/PostgresDS" pool-name="PostgresDS">
    <connection-
url>jdbc:postgresql://localhost:5432/postgresdb</connection-url>
    <driver>postgresql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidCon
nectionChecker"></valid-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptio
nSorter"></exception-sorter>
    </validation>
  </datasource>
  <drivers>
    <driver name="postgresql" module="org.postgresql">
      <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>
```

The example below is a `module.xml` file for the PostgreSQL datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="org.postgresql">
  <resources>
    <resource-root path="postgresql-9.1-902.jdbc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.2. Example PostgreSQL XA Datasource

Example 14.6.

The example below is a PostgreSQL XA datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <xa-datasource jndi-name="java:jboss/PostgresXADS" pool-
name="PostgresXADS">
    <driver>postgresql</driver>
    <xa-datasource-property name="ServerName">localhost</xa-datasource-
property>
    <xa-datasource-property name="PortNumber">5432</xa-datasource-
property>
    <xa-datasource-property name="DatabaseName">postgresdb</xa-
datasource-property>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidCon
nectionChecker">
      </valid-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptio
nSorter">
      </exception-sorter>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="postgresql" module="org.postgresql">
      <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>
```

The example below is a `module.xml` file for the PostgreSQL XA datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="org.postgresql">
  <resources>
    <resource-root path="postgresql-9.1-902.jdbc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.3. Example MySQL Datasource

Example 14.7.

The example below is a MySQL datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <datasource jndi-name="java:jboss/MySqlDS" pool-name="MySqlDS">
    <connection-url>jdbc:mysql://mysql-
localhost:3306/jbossdb</connection-url>
    <driver>mysql</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionC
hecker"></valid-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"
></exception-sorter>
    </validation>
  </datasource>
  <drivers>
    <driver name="mysql" module="com.mysql">
      <xa-datasource-
class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-
class>
    </driver>
  </drivers>
</datasources>
```

The example below is a `module.xml` file for the MySQL datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.0.8-bin.jar"/>
  </resources>
```

```
    <dependencies>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
  </module>
```

Report a bug

14.5.4. Example MySQL XA Datasource

Example 14.8.

The example below is a MySQL XA datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <xa-datasource jndi-name="java:jboss/MysqlXADS" pool-
name="MysqlXADS">
  <driver>mysql</driver>
    <xa-datasource-property name="ServerName">localhost</xa-datasource-
property>
    <xa-datasource-property name="DatabaseName">mysqldb</xa-datasource-
property>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLValidConnectionC
hecker"></valid-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter"
></exception-sorter>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="mysql" module="com.mysql">
      <xa-datasource-
class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-
class>
    </driver>
  </drivers>
</datasources>
```

The example below is a **`module.xml`** file for the MySQL XA datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.0.8-bin.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.5. Example Oracle Datasource

NOTE

Prior to version 10.2 of the Oracle datasource, the <no-tx-separate-pools/> parameter was required, as mixing non-transactional and transactional connections would result in an error. This parameter may no longer be required for certain applications.

Example 14.9.

The example below is an Oracle datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <datasource jndi-name="java:/OracleDS" pool-name="OracleDS">
    <connection-url>jdbc:oracle:thin:@localhost:1521:XE</connection-
url>
    <driver>oracle</driver>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectio
nChecker"></valid-connection-checker>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnectio
nChecker"></stale-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorte
r"></exception-sorter>
    </validation>
  </datasource>
  <drivers>
    <driver name="oracle" module="com.oracle">
      <xa-datasource-
class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

The example below is a `module.xml` file for the Oracle datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
```

```
    <resources>
      <resource-root path="ojdbc6.jar"/>
    </resources>
    <dependencies>
      <module name="javax.api"/>
      <module name="javax.transaction.api"/>
    </dependencies>
  </module>
```

Report a bug

14.5.6. Example Oracle XA Datasource

NOTE

Prior to version 10.2 of the Oracle datasource, the <no-tx-separate-pools/> parameter was required, as mixing non-transactional and transactional connections would result in an error. This parameter may no longer be required for certain applications.

IMPORTANT

The following settings must be applied for the user accessing an Oracle XA datasource in order for XA recovery to operate correctly. The value **user** is the user defined to connect from JBoss to Oracle:

- GRANT SELECT ON sys.dba_pending_transactions TO user;
- GRANT SELECT ON sys.pending_trans$ TO user;
- GRANT SELECT ON sys.dba_2pc_pending TO user;
- GRANT EXECUTE ON sys.dbms_xa TO user; (If using Oracle 10g R2 (patched) or Oracle 11g)

  OR

  GRANT EXECUTE ON sys.dbms_system TO user; (If using an unpatched Oracle version prior to 11g)

Example 14.10.

The example below is an Oracle XA datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <xa-datasource jndi-name="java:/XAOracleDS" pool-name="XAOracleDS">
    <driver>oracle</driver>
    <xa-datasource-property name="URL">jdbc:oracle:oci8:@tc</xa-
datasource-property>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
      <no-tx-separate-pools />
    </xa-pool>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleValidConnectio
nChecker"></valid-connection-checker>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleStaleConnectio
nChecker"></stale-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.oracle.OracleExceptionSorte
r"></exception-sorter>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="oracle" module="com.oracle">
      <xa-datasource-
class>oracle.jdbc.xa.client.OracleXADataSource</xa-datasource-class>
    </driver>
  </drivers>
</datasources>
```

The example below is a **module.xml** file for the Oracle XA datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.oracle">
  <resources>
    <resource-root path="ojdbc6.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.7. Example Microsoft SQLServer Datasource

Example 14.11.

The example below is a Microsoft SQLServer datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <datasource jndi-name="java:/MSSQLDS" pool-name="MSSQLDS">
    <connection-
url>jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=MyDatabase</c
onnection-url>
    <driver>sqlserver</driver>
    <security>
```

```
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionC
hecker"></valid-connection-checker>
    </validation>
  </datasource>
  <drivers>
    <driver name="sqlserver" module="com.microsoft">
      <xa-datasource-
class>com.microsoft.sqlserver.jdbc.SQLServerXADataSource</xa-datasource-
class>
    </driver>
</datasources>
```

The example below is a **`module.xml`** file for the Microsoft SQLServer datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
  <resources>
    <resource-root path="sqljdbc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.8. Example Microsoft SQLServer XA Datasource

Example 14.12.

The example below is a Microsoft SQLServer XA datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <xa-datasource jndi-name="java:/MSSQLXADS" pool-name="MSSQLXADS">
    <driver>sqlserver</driver>
    <xa-datasource-property name="ServerName">localhost</xa-datasource-
property>
    <xa-datasource-property name="DatabaseName">mssqldb</xa-datasource-
property>
    <xa-datasource-property name="SelectMethod">cursor</xa-datasource-
property>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.mssql.MSSQLValidConnectionC
hecker"></valid-connection-checker>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="sqlserver" module="com.microsoft">
      <xa-datasource-
class>com.microsoft.sqlserver.jdbc.SQLServerXADataSource</xa-datasource-
class>
    </driver>
  </drivers>
</datasources>
```

The example below is a **`module.xml`** file for the Microsoft SQLServer XA datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.microsoft">
  <resources>
    <resource-root path="sqljdbc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.9. Example IBM DB2 Datasource

Example 14.13.

The example below is an IBM DB2 datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <datasource jndi-name="java:/DB2DS" pool-name="DB2DS">
    <connection-url>jdbc:db2:ibmdb2db</connection-url>
    <driver>ibmdb2</driver>
    <pool>
      <min-pool-size>0</min-pool-size>
      <max-pool-size>50</max-pool-size>
    </pool>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
```

```
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionCheck
er"></valid-connection-checker>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionCheck
er"></stale-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter"></e
xception-sorter>
    </validation>
  </datasource>
  <drivers>
    <driver name="ibmdb2" module="com.ibm">
      <xa-datasource-class>com.ibm.db2.jdbc.DB2XADataSource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>
```

The example below is a **`module.xml`** file for the IBM DB2 datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.10. Example IBM DB2 XA Datasource

Example 14.14.

The example below is an IBM DB2 XA datasource configuration. The datasource has been enabled, a user has been added and validation options have been set.

```
<datasources>
  <xa-datasource jndi-name="java:/DB2XADS" pool-name="DB2XADS">
    <driver>ibmdb2</driver>
    <xa-datasource-property name="DatabaseName">ibmdb2db</xa-
datasource-property>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <xa-pool>
      <is-same-rm-override>false</is-same-rm-override>
    </xa-pool>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ValidConnectionCheck
er"></valid-connection-checker>
      <stale-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2StaleConnectionCheck
er"></stale-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.db2.DB2ExceptionSorter"></e
xception-sorter>
    </validation>
    <recovery>
      <recover-plugin class-
name="org.jboss.jca.core.recovery.ConfigurableRecoveryPlugin">
        <config-property name="EnableIsValid">false</config-property>
        <config-property name="IsValidOverride">false</config-property>
        <config-property name="EnableClose">false</config-property>
      </recover-plugin>
    </recovery>
  </xa-datasource>
  <drivers>
    <driver name="ibmdb2" module="com.ibm">
      <xa-datasource-class>com.ibm.db2.jcc.DB2XADataSource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>
```

The example below is a **module.xml** file for the IBM DB2 XA datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.ibm">
  <resources>
    <resource-root path="db2jcc4.jar"/>
    <resource-root path="db2jcc_license_cisuz.jar"/>
    <resource-root path="db2jcc_license_cu.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.11. Example Sybase Datasource

Example 14.15.

The example below is a Sybase datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <datasource jndi-name="java:jboss/SybaseDB" pool-name="SybaseDB"
enabled="true">
```

```
    <connection-url>jdbc:sybase:Tds:localhost:5000/DATABASE?
JCONNECT_VERSION=6</connection-url>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectio
nChecker"></valid-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorte
r"></exception-sorter>
    </validation>
  </datasource>
  <drivers>
    <driver name="sybase" module="com.sybase">
      <datasource-
class>com.sybase.jdbc2.jdbc.SybDataSource</datasource-class>
      <xa-datasource-class>com.sybase.jdbc3.jdbc.SybXADataSource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>
```

The example below is a **`module.xml`** file for the Sybase datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.sybase">
  <resources>
    <resource-root path="jconn2.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

14.5.12. Example Sybase XA Datasource

Example 14.16.

The example below is a Sybase XA datasource configuration. The datasource has been enabled, a user has been added, and validation options have been set.

```
<datasources>
  <xa-datasource jndi-name="java:jboss/SybaseXADS" pool-
name="SybaseXADS" enabled="true">
    <xa-datasource-property name="NetworkProtocol">Tds</xa-datasource-
property>
    <xa-datasource-property name="ServerName">myserver</xa-datasource-
property>
```

```
    <xa-datasource-property name="PortNumber">4100</xa-datasource-
property>
    <xa-datasource-property name="DatabaseName">mydatabase</xa-
datasource-property>
    <security>
      <user-name>admin</user-name>
      <password>admin</password>
    </security>
    <validation>
      <background-validation>true</background-validation>
      <valid-connection-checker class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseValidConnectio
nChecker"></valid-connection-checker>
      <exception-sorter class-
name="org.jboss.jca.adapters.jdbc.extensions.sybase.SybaseExceptionSorte
r"></exception-sorter>
    </validation>
  </xa-datasource>
  <drivers>
    <driver name="sybase" module="com.sybase">
      <datasource-
class>com.sybase.jdbc2.jdbc.SybDataSource</datasource-class>
      <xa-datasource-class>com.sybase.jdbc3.jdbc.SybXADataSource</xa-
datasource-class>
    </driver>
  </drivers>
</datasources>
```

The example below is a **`module.xml`** file for the Sybase XA datasource above.

```
<module xmlns="urn:jboss:module:1.1" name="com.sybase">
  <resources>
    <resource-root path="jconn2.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
    <module name="javax.transaction.api"/>
  </dependencies>
</module>
```

Report a bug

CHAPTER 15. JDBC BASED CACHE STORES

Red Hat JBoss Data Grid offers several cache stores for use with common data storage formats. JDBC based cache stores are used with any cache store that exposes a JDBC driver. JBoss Data Grid offers the following JDBC based cache stores depending on the key to be persisted:

- **`JdbcBinaryStore`**.
- **`JdbcStringBasedStore`**.
- **`JdbcMixedStore`**.

Report a bug

15.1. JDBCBINARYSTORES

The **`JdbcBinaryStore`** supports all key types. It stores all keys with the same hash value (**`hashCode`** method on the key) in the same table row/blob. The hash value common to the included keys is set as the primary key for the table row/blob. As a result of this hash value, **`JdbcBinaryStore`** offers excellent flexibility but at the cost of concurrency and throughput.

As an example, if three keys (**`k1`**, **`k2`** and **`k3`**) have the same hash code, they are stored in the same table row. If three different threads attempt to concurrently update **`k1`**, **`k2`** and **`k3`**, they must do it sequentially because all three keys share the same row and therefore cannot be simultaneously updated.

Report a bug

15.1.1. JdbcBinaryStore Configuration (Remote Client-Server Mode)

The following is a configuration for **`JdbcBinaryStore`** using Red Hat JBoss Data Grid's Remote Client-Server mode with Passivation enabled.

Procedure 15.1. Configure the JdbcBinaryStore for Remote Client-Server Mode

1. **The binary-keyed-jdbc-store Element**
   The **`binary-keyed-jdbc-store`** element specifies the configuration for a binary keyed cache JDBC store.

   a. The ***`datasource`*** parameter defines the name of a JNDI for the datasource.

   b. The ***`passivation`*** parameter determines whether entries in the cache are passivated (**`true`**) or if the cache store retains a copy of the contents in memory (**`false`**).

   c. The ***`preload`*** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are **`true`** and **`false`**.

   d. The ***`purge`*** parameter specifies whether or not the cache store is purged when it is started. Valid values for this parameter are **`true`** and **`false`**.

```
<local-cache>
 ...
 <binary-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
```

```
    passivation="${true/false}"
    preload="${true/false]"
    purge="${true/false}">
```

2. **The binary-keyed-table Element**
   The **`binary-keyed-table`** element specifies information about the database table used to store binary cache entries.

   a. The ***`prefix`*** parameter specifies a prefix string for the database table name.

```
<local-cache>
 ...
 <binary-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
    passivation="${true/false}"
    preload="${true/false]"
    purge="${true/false}">
               <binary-keyed-table prefix="JDG">
```

3. **The id-column Element**
   The **`id-column`** element specifies information about a database column that holds cache entry IDs.

   a. The ***name*** parameter specifies the name of the database column.

   b. The ***type*** parameter specifies the type of the database column.

```
<local-cache>
 ...
 <binary-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
    passivation="${true/false}"
    preload="${true/false]"
    purge="${true/false}">
               <binary-keyed-table prefix="JDG">
                <id-column name="id"
       type="${id.column.type}"/>
```

4. **The data-column Element**
   The **`data-column`** element contains information about a database column that holds cache entry data.

   a. The ***name*** parameter specifies the name of the database column.

   b. The ***type*** parameter specifies the type of the database column.

```
<local-cache>
 ...
 <binary-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
    passivation="${true/false}"
    preload="${true/false]"
    purge="${true/false}">
               <binary-keyed-table prefix="JDG">
                <id-column name="id"
```

```
        type="${id.column.type}"/>
                  <data-column name="datum"
          type="${data.column.type}"/>
```

5. **The timestamp-column Element**
   The **`timestamp-column`** element specifies information about the database column that holds cache entry timestamps.

   a. The ***name*** parameter specifies the name of the database column.

   b. The ***type*** parameter specifies the type of the database column.

```
<local-cache>
 ...
 <binary-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
     passivation="${true/false}"
     preload="${true/false]"
     purge="${true/false}">
                 <binary-keyed-table prefix="JDG">
                  <id-column name="id"
        type="${id.column.type}"/>
                  <data-column name="datum"
          type="${data.column.type}"/>
                 <timestamp-column name="version"
        type="${timestamp.column.type}"/>
                </binary-keyed-table>
        </binary-keyed-jdbc-store>
</local-cache>
```

Report a bug

15.1.2. JdbcBinaryStore Configuration (Library Mode)

The following is a sample configuration for the **`JdbcBinaryStore`**:

Procedure 15.2. Configure the JdbcBinaryStore for Library Mode

1. **The binaryKeyedJdbcStore Element**
   The **`binaryKeyedJdbcStore`** element uses the following parameters to configure the cache store:

   a. The ***`fetchPersistentState`*** parameter determines whether the persistent state is fetched when joining a cluster. Set this to **`true`** if using a replication and invalidation in a clustered environment. Additionally, if multiple cache stores are chained, only one cache store can have this property enabled. If a shared cache store is used, the cache does not allow a persistent state transfer despite this property being set to **`true`**. The ***`fetchPersistentState`*** parameter is **`false`** by default.

   b. The **`ignoreModifications`** parameter determines whether operations that modify the cache (e.g. put, remove, clear, store, etc.) do not affect the cache store. As a result, the cache store can become out of sync with the cache.

c. The **purgeOnStartup** parameter specifies whether the cache store is purged when initially started.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <binaryKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false">
```

2. **The connectionPool Element**
The **connectionPool** element specifies a connection pool for the JDBC driver using the following parameters:

   a. The ***connectionUrl*** parameter specifies the JDBC driver-specific connection URL.

   b. The ***username*** parameter contains the username used to connect via the ***connectionUrl***.

   c. The ***driverClass*** parameter specifies the class name of the driver used to connect to the database.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <binaryKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
```

3. **The binaryKeyedTable Element**
The **binaryKeyedTable** element defines the table that stores cache entries. It uses the following parameters to configure the cache store:

a. The ***dropOnExit*** parameter specifies whether the database tables are dropped upon shutdown.

b. The ***createOnStart*** parameter specifies whether the database tables are created by the store on startup.

c. The ***prefix*** parameter defines the string prepended to name of the target cache when composing the name of the cache bucket table.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <binaryKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE">
```

4. **The idColumn Element**
   The **idColumn** element defines the column where the cache key or bucket ID is stored. It uses the following parameters:

a. Use the ***name*** parameter to specify the name of the column used.

b. Use the ***type*** parameter to specify the type of the column used.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <binaryKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false">
  <connectionPool
```

```
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
```

5. **The dataColumn Element**
   The **dataColumn** element specifies the column where the cache entry or bucket is stored.

   a. Use the ***name*** parameter to specify the name of the column used.

   b. Use the ***type*** parameter to specify the type of the column used.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <binaryKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
   <dataColumn name="DATA_COLUMN"
        type="BINARY" />
```

6. **The timestampColumn Element**
   The **timestampColumn** element specifies the column where the time stamp of the cache entry or bucket is stored.

   a. Use the ***name*** parameter to specify the name of the column used.

   b. Use the ***type*** parameter to specify the type of the column used.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
```

```
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <binaryKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                       fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
   <dataColumn name="DATA_COLUMN"
        type="BINARY" />
   <timestampColumn name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </binaryKeyedTable>
 </binaryKeyedJdbcStore>
</persistence>
```

Report a bug

15.1.3. JdbcBinaryStore Programmatic Configuration

The following is a sample configuration for the **`JdbcBinaryStore`**:

Procedure 15.3. JdbcBinaryStore Programmatic Configuration (Library Mode)

1. **Create a New Configuration Builder**
   Use the ***`ConfigurationBuilder`*** to create a new configuration object.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();
     builder.persistence()
   ```

2. **Add the `JdbcBinaryStoreConfigurationBuilder`**
   Add the **`JdbcBinaryStore`** configuration builder to build a specific configuration related to this store.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();
     builder.persistence()
        .addStore(JdbcBinaryStoreConfigurationBuilder.class)
   ```

3. **Set Up Persistence**
   `fetchPersistentState` determines whether or not to fetch the persistent state of a cache and apply it to the local cache store when joining the cluster. If the cache store is shared the

fetch persistent state is ignored, as caches access the same cache store. A configuration exception will be thrown when starting the cache service if more than one cache loader has this property set to **true**. The ***fetchPersistentState*** property is **false** by default.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
  builder.persistence()
     .addStore(JdbcBinaryStoreConfigurationBuilder.class)
     .fetchPersistentState(false)
```

4. **Set Modifications**
   ignoreModifications determines whether write methods are pushed to the specific cache loader by allowing write operations to the local file cache loader, but not the shared cache loader. In some cases, transient application data should only reside in a file-based cache loader on the same server as the in-memory cache. For example, this would apply with a further JDBC based cache loader used by all servers in the network. ***ignoreModifications*** is **false** by default.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
  builder.persistence()
     .addStore(JdbcBinaryStoreConfigurationBuilder.class)
     .fetchPersistentState(false)
     .ignoreModifications(false)
```

5. **Configure Purging**
   purgeOnStartup specifies whether the cache is purged when initially started.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
  builder.persistence()
     .addStore(JdbcBinaryStoreConfigurationBuilder.class)
     .fetchPersistentState(false)
     .ignoreModifications(false)
     .purgeOnStartup(false)
```

6. **Configure the Table**

   a. **Set Drop Table On Exit Method**
      dropOnExit determines if the table will be created when the cache store is stopped. This is set to **false** by default.

   b. **Set Create Table On Start Method**
      createOnStart creates the table when starting the cache store if no table currently exists. This method is **true** by default.

   c. **Set the Table Name Prefix**
      tableNamePrefix sets the prefix for the name of the table in which the data will be stored.

   d. ***idColumnName***
      The ***idColumnName*** property defines the column where the cache key or bucket ID is stored.

   e. ***dataColumnName***
      The ***dataColumnName*** property specifies the column where the cache entry or bucket is stored.

f. ***timestampColumnName***
The ***timestampColumnName*** element specifies the column where the time stamp of the cache entry or bucket is stored.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
  builder.persistence()
     .addStore(JdbcBinaryStoreConfigurationBuilder.class)
     .fetchPersistentState(false)
     .ignoreModifications(false)
     .purgeOnStartup(false)
     .table()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_BUCKET_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

.timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT
")
```

7. **The connectionPool Element**
The **connectionPool** element specifies a connection pool for the JDBC driver using the following parameters:

a. The ***connectionUrl*** parameter specifies the JDBC driver-specific connection URL.

b. The ***username*** parameter contains the user name used to connect via the ***connectionUrl***.

c. The ***driverClass*** parameter specifies the class name of the driver used to connect to the database.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
  builder.persistence()
     .addStore(JdbcBinaryStoreConfigurationBuilder.class)
     .fetchPersistentState(false)
     .ignoreModifications(false)
     .purgeOnStartup(false)
     .table()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_BUCKET_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

.timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT
")
     .connectionPool()

.connectionUrl("jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1")
        .username("sa")
        .driverClass("org.h2.Driver");
```

NOTE

Programmatic configurations can only be used with Red Hat JBoss Data Grid Library mode.

Report a bug

15.2. JDBCSTRINGBASEDSTORES

The `JdbcStringBasedStore` stores each entry in its own row in the table, instead of grouping multiple entries into each row, resulting in increased throughput under a concurrent load. It also uses a (pluggable) bijection that maps each key to a `String` object. The `Key2StringMapper` interface defines the bijection.

Red Hat JBoss Data Grid includes a default implementation called `DefaultTwoWayKey2StringMapper` that handles primitive types.

Report a bug

15.2.1. JdbcStringBasedStore Configuration (Remote Client-Server Mode)

The following is a sample `JdbcStringBasedStore` for Red Hat JBoss Data Grid's Remote Client-Server mode with Passivation enabled.

Procedure 15.4. Configure JdbcStringBasedStore in Remote Client-Server Mode

1. **The string-keyed-jdbc-store Element**
   The `string-keyed-jdbc-store` element specifies the configuration for a string based keyed cache JDBC store.

   a. The ***`datasource`*** parameter defines the name of a JNDI for the datasource.

   b. The ***`passivation`*** parameter determines whether entries in the cache are passivated (`true`) or if the cache store retains a copy of the contents in memory (`false`).

   c. The ***`preload`*** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are `true` and `false`.

   d. The ***`purge`*** parameter specifies whether or not the cache store is purged when it is started. Valid values for this parameter are `true` and `false`.

   e. The ***`shared`*** parameter is used when multiple cache instances share a cache store. This parameter can be set to prevent multiple cache instances writing the same modification multiple times. Valid values for this parameter are `true` and `false`.

   f. The ***`singleton`*** parameter enables a singleton store cache store. SingletonStore is a delegating cache store used when only one instance in a cluster can interact with the underlying store.

```
<local-cache>
 ...
 <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
     passivation="true"
     preload="false"
```

```
      purge="false"
      shared="false"
      singleton="true">
```

2. **The string-keyed-table Element**
   The **`string-keyed-table`** element specifies information about the database table used to store string based cache entries.

   a. The ***`prefix`*** parameter specifies a prefix string for the database table name.

```
<local-cache>
 ...
 <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
     passivation="true"
     preload="false"
     purge="false"
     shared="false"
     singleton="true">
                <string-keyed-table prefix="JDG">
```

3. **The id-column Element**
   The **`id-column`** element specifies information about a database column that holds cache entry IDs.

   a. The ***name*** parameter specifies the name of the ID column.

   b. The ***type*** parameter specifies the type of the ID column.

```
<local-cache>
 ...
 <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
     passivation="true"
     preload="false"
     purge="false"
     shared="false"
     singleton="true">
                <string-keyed-table prefix="JDG">
                 <id-column name="id"
       type="${id.column.type}"/>
```

4. **The data-column Element**
   The **`data-column`** element contains information about a database column that holds cache entry data.

   a. The ***name*** parameter specifies the name of the database column.

   b. The ***type*** parameter specifies the type of the database column.

```
<local-cache>
 ...
 <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
     passivation="true"
     preload="false"
     purge="false"
```

```
        shared="false"
        singleton="true">
                    <string-keyed-table prefix="JDG">
                     <id-column name="id"
          type="${id.column.type}"/>
      <data-column name="datum"
            type="${data.column.type}"/>
```

5. **The timestamp-column Element**
   The **`timestamp-column`** element specifies information about the database column that holds cache entry timestamps.

   a. The ***name*** parameter specifies the name of the database column.

   b. The ***type*** parameter specifies the type of the database column.

```
<local-cache>
 ...
 <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
     passivation="true"
     preload="false"
     purge="false"
     shared="false"
     singleton="true">
                <string-keyed-table prefix="JDG">
                 <id-column name="id"
       type="${id.column.type}"/>
   <data-column name="datum"
         type="${data.column.type}"/>
   <timestamp-column name="version"
       type="${timestamp.column.type}"/>
         </string-keyed-table>
 </string-keyed-jdbc-store>
</local-cache>
```

Report a bug

15.2.2. JdbcStringBasedStore Configuration (Library Mode)

The following is a sample configuration for the **`JdbcStringBasedStore`**:

Procedure 15.5. Configure JdbcStringBasedStore in Library Mode

1. **The stringKeyedJdbcStore Element**
   The **`stringKeyedJdbcStore`** element uses the following parameters to configure the cache store:

   a. The ***`fetchPersistentState`*** parameter determines whether the persistent state is fetched when joining a cluster. Set this to **`true`** if using a replication and invalidation in a clustered environment. Additionally, if multiple cache stores are chained, only one cache store can have this property enabled. If a shared cache store is used, the cache does not allow a persistent state transfer despite this property being set to **`true`**. The ***`fetchPersistentState`*** parameter is **`false`** by default.

b. The ***ignoreModifications*** parameter determines whether operations that modify the cache (e.g. put, remove, clear, store, etc.) do not affect the cache store. As a result, the cache store can become out of sync with the cache.

c. The ***purgeOnStartup*** parameter specifies whether the cache is purged when initially started.

d. The ***key2StringMapper*** parameter specifies the class name of the Key2StringMapper used to map keys to strings for the database tables.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <stringKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                       fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey
2StringMapper">
```

2. **The connectionPool Element**
The **connectionPool** element specifies a connection pool for the JDBC driver using the following parameters:

a. The ***connectionUrl*** parameter specifies the JDBC driver-specific connection URL.

b. The ***username*** parameter contains the user name used to connect via the ***connectionUrl***.

c. The ***driverClass*** parameter specifies the class name of the driver used to connect to the database.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <stringKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                        fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"
```

```
key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey
2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
```

3. **The stringKeyedTable Element**
   Add the **stringKeyedTable** element defines the table that stores cache entries. It uses the following parameters to configure the cache store:

   a. The ***dropOnExit*** parameter specifies whether the database tables are dropped upon shutdown.

   b. The ***createOnStart*** parameter specifies whether the database tables are created by the store on startup.

   c. The ***prefix*** parameter specifies a prefix string for the database table name.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <stringKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey
2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <stringKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_STRING_TABLE">
```

4. **The idColumn Element**
   The **idColumn** element defines the column where the cache key or bucket ID is stored. It uses the following parameters:

   a. Use the ***name*** parameter to specify the name of the ID column.

   b. Use the ***type*** parameter to specify the type of the ID column.

```
<infinispan
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <stringKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey
2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <stringKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_STRING_TABLE">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
```

5. **The dataColumn Element**
   The **`dataColumn`** element specifies the column where the cache entry or bucket is stored.

   a. Use the ***name*** parameter to specify the name of the database column.

   b. Use the ***type*** parameter to specify the type of the database column.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <stringKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey
2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <stringKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_STRING_TABLE">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
   <dataColumn name="DATA_COLUMN"
        type="BINARY" />
```

6. **The timestampColumn Element**
   The **`timestampColumn`** element specifies the column where the time stamp of the cache entry or bucket is stored.

   a. Use the ***name*** parameter to specify the name of the column used.

   b. Use the ***type*** parameter to specify the type of the column used.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <stringKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey
2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <stringKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_STRING_TABLE">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
   <dataColumn name="DATA_COLUMN"
        type="BINARY" />
   <timestampColumn name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </stringKeyedTable>
 </stringKeyedJdbcStore>
</persistence>
```

Report a bug

15.2.3. JdbcStringBasedStore Multiple Node Configuration (Remote Client-Server Mode)

The following is a configuration for the `JdbcStringBasedStore` in Red Hat JBoss Data Grid's Remote Client-Server mode. This configuration is used when multiple nodes must be used.

Procedure 15.6. JdbcStringBasedStore Multiple Node Configuration for Remote Client-Server Mode

1. **The string-keyed-jdbc-store Element**
   The `string-keyed-jdbc-store` element specifies the configuration for a string based keyed cache JDBC store.

   a. The ***`fetch-state`*** parameter determines whether or not to fetch the persistent state of a cache when joining a cluster. If multiple cache stores are chained, only one of them can have this property enabled.

   b. The ***`datasource`*** parameter defines the name of a JNDI for the datasource.

   c. The ***`passivation`*** parameter determines whether entries in the cache are passivated (`true`) or if the cache store retains a copy of the contents in memory (`false`).

   d. The ***`preload`*** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are `true` and `false`.

   e. The ***`purge`*** parameter specifies whether or not the cache store is purged when it is started. Valid values for this parameter are `true` and `false`.

   f. The ***`shared`*** parameter is used when multiple cache instances share a cache store. This parameter can be set to prevent multiple cache instances writing the same modification multiple times. Valid values for this parameter are `true` and `false`.

   g. The ***`singleton`*** parameter enables a singleton store cache store. SingletonStore is a delegating cache store used when only one instance in a cluster should interact with the underlying store.

```
<subsystem xmlns="urn:infinispan:server:core:5.2" default-cache-
container="default">
 <cache-container ... >
  ...
      <replicated-cache>
   ...
       <string-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
                     fetch-state="true"
                     passivation="false"
                     preload="false"
                     purge="false"
                     shared="false"
                     singleton="true">
```

2. **The string-keyed-table Element**
   The `string-keyed-table` element specifies information about the database table used to store string based cache entries.

   a. The ***`prefix`*** parameter specifies a prefix string for the database table name.

```
<subsystem xmlns="urn:infinispan:server:core:5.2" default-cache-
container="default">
```

```
 <cache-container ... >
  ...
      <replicated-cache>
   ...
       <string-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
                      fetch-state="true"
                      passivation="false"
                      preload="false"
                      purge="false"
                      shared="false"
                      singleton="true">
          <string-keyed-table prefix="JDG">
```

3. **The id-column Element**
   The **`id-column`** element specifies information about a database column that holds cache entry IDs.

   a. The ***name*** parameter specifies the name of the ID column.

   b. The ***type*** parameter specifies the type of the ID column.

```
<subsystem xmlns="urn:infinispan:server:core:5.2" default-cache-
container="default">
 <cache-container ... >
  ...
      <replicated-cache>
   ...
       <string-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
                      fetch-state="true"
                      passivation="false"
                      preload="false"
                      purge="false"
                      shared="false"
                      singleton="true">
          <string-keyed-table prefix="JDG">
              <id-column name="id"
                         type="${id.column.type}"/>>
```

4. **The data-column Element**
   The **`data-column`** element contains information about a database column that holds cache entry data.

   a. The ***name*** parameter specifies the name of the database column.

   b. The ***type*** parameter specifies the type of the database column.

```
<subsystem xmlns="urn:infinispan:server:core:5.2" default-cache-
container="default">
 <cache-container ... >
  ...
      <replicated-cache>
   ...
       <string-keyed-jdbc-store
```

```
datasource="java:jboss/datasources/JdbcDS"
                     fetch-state="true"
                     passivation="false"
                     preload="false"
                     purge="false"
                     shared="false"
                     singleton="true">
         <string-keyed-table prefix="JDG">
             <id-column name="id"
                        type="${id.column.type}"/>
             <data-column name="datum"
                          type="${data.column.type}"/>
```

5. **The timestamp-column Element**
   The **`timestamp-column`** element specifies information about the database column that holds cache entry timestamps.

   a. The ***name*** parameter specifies the name of the timestamp column.

   b. The ***type*** parameter specifies the type of the timestamp column.

```
<subsystem xmlns="urn:infinispan:server:core:5.2" default-cache-
container="default">
 <cache-container ... >
  ...
      <replicated-cache>
   ...
       <string-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
                     fetch-state="true"
                     passivation="false"
                     preload="false"
                     purge="false"
                     shared="false"
                     singleton="true">
         <string-keyed-table prefix="JDG">
             <id-column name="id"
                        type="${id.column.type}"/>
             <data-column name="datum"
                          type="${data.column.type}"/>
             <timestamp-column name="version"
                               type="${timestamp.column.type}"/>
    </string-keyed-table>
   </string-keyed-jdbc-store>
  </replicated-cache>
 </cache-container>
</subsystem>
```

Report a bug

15.2.4. JdbcStringBasedStore Programmatic Configuration

The following is a sample configuration for the **`JdbcStringBasedStore`**:

Procedure 15.7. Configure the JdbcStringBasedStore Programmatically

1. **Create a New Configuration Builder**
   Use the ***`ConfigurationBuilder`*** to create a new configuration object.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();
   ```

2. **Add `JdbcStringBasedStoreConfigurationBuilder`**
   Add the **`JdbcStringBasedStore`** configuration builder to build a specific configuration related to this store.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();

   builder.persistence().addStore(JdbcStringBasedStoreConfigurationBuil
   der.class)
   ```

3. **Set Up Persistence**
   `fetchPersistentState` determines whether or not to fetch the persistent state of a cache and apply it to the local cache store when joining the cluster. If the cache store is shared the fetch persistent state is ignored, as caches access the same cache store. A configuration exception will be thrown when starting the cache service if more than one cache loader has this property set to **`true`**. The ***`fetchPersistentState`*** property is **`false`** by default.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();

   builder.persistence().addStore(JdbcStringBasedStoreConfigurationBuil
   der.class)
         .fetchPersistentState(false)
   ```

4. **Set Modifications**
   `ignoreModifications` determines whether write methods are pushed to the specific cache loader by allowing write operations to the local file cache loader, but not the shared cache loader. In some cases, transient application data should only reside in a file-based cache loader on the same server as the in-memory cache. For example, this would apply with a further JDBC based cache loader used by all servers in the network. ***`ignoreModifications`*** is **`false`** by default.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();

   builder.persistence().addStore(JdbcStringBasedStoreConfigurationBuil
   der.class)
         .fetchPersistentState(false)
         .ignoreModifications(false)
   ```

5. **Configure Purging**
   `purgeOnStartup` specifies whether the cache is purged when initially started.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();

   builder.persistence().addStore(JdbcStringBasedStoreConfigurationBuil
   der.class)
   ```

```
      .fetchPersistentState(false)
      .ignoreModifications(false)
      .purgeOnStartup(false)
```

6. **Configure the Table**

   a. **Set Drop Table On Exit Method**
      dropOnExit determines if the table will be created when the cache store is stopped. This is set to `false` by default.

   b. **Set Create Table On Start Method**
      createOnStart creates the table when starting the cache store if no table currently exists. This method is `true` by default.

   c. **Set the Table Name Prefix**
      tableNamePrefix sets the prefix for the name of the table in which the data will be stored.

   d. ***idColumnName***
      The ***idColumnName*** property defines the column where the cache key or bucket ID is stored.

   e. ***dataColumnName***
      The ***dataColumnName*** property specifies the column where the cache entry or bucket is stored.

   f. ***timestampColumnName***
      The ***timestampColumnName*** element specifies the column where the time stamp of the cache entry or bucket is stored.

```
ConfigurationBuilder builder = new ConfigurationBuilder();

builder.persistence().addStore(JdbcStringBasedStoreConfigurationBuil
der.class)
      .fetchPersistentState(false)
      .ignoreModifications(false)
      .purgeOnStartup(false)
      .table()
         .dropOnExit(true)
         .createOnStart(true)
         .tableNamePrefix("ISPN_STRING_TABLE")
         .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
         .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

.timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT
")
```

7. **The connectionPool Element**
   The **connectionPool** element specifies a connection pool for the JDBC driver using the following parameters:

   a. The ***connectionUrl*** parameter specifies the JDBC driver-specific connection URL.

   b. The ***username*** parameter contains the username used to connect via the ***connectionUrl***.

c. The ***driverClass*** parameter specifies the class name of the driver used to connect to the database.

```
ConfigurationBuilder builder = new ConfigurationBuilder();

builder.persistence().addStore(JdbcStringBasedStoreConfigurationBuil
der.class)
     .fetchPersistentState(false)
     .ignoreModifications(false)
     .purgeOnStartup(false)
     .table()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_STRING_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

.timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT
")
     .connectionPool()

.connectionUrl("jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1")
        .username("sa")
        .driverClass("org.h2.Driver");
```

NOTE

Programmatic configurations can only be used with Red Hat JBoss Data Grid Library mode.

Report a bug

15.3. JDBCMIXEDSTORES

The **JdbcMixedStore** is a hybrid implementation that delegates keys based on their type to either the **JdbcBinaryStore** or **JdbcStringBasedStore**.

Report a bug

15.3.1. JdbcMixedStore Configuration (Remote Client-Server Mode)

The following is a configuration for a **JdbcMixedStore** for Red Hat JBoss Data Grid's Remote Client-Server mode with Passivation enabled.

Procedure 15.8. Configure JdbcMixedStore in Remote Client-Server Mode

1. **The mixed-keyed-jdbc-store Element**
   The **mixed-keyed-jdbc-store** element specifies the configuration for a mixed keyed cache JDBC store.

   a. The ***datasource*** parameter defines the name of a JNDI for the datasource.

b. The ***passivation*** parameter determines whether entries in the cache are passivated (`true`) or if the cache store retains a copy of the contents in memory (`false`).

c. The ***preload*** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are `true` and `false`.

d. The ***purge*** parameter specifies whether or not the cache store is purged when it is started. Valid values for this parameter are `true` and `false`.

```
<local-cache>
              <mixed-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
         passivation="true"
         preload="false"
         purge="false">
```

2. **The binary-keyed-table Element**
The **`binary-keyed-table`** element specifies information about the database table used to store mixed cache entries.

   a. The ***prefix*** parameter specifies a prefix string for the database table name.

```
<local-cache>
      <mixed-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
         passivation="true"
         preload="false"
         purge="false">
          <binary-keyed-table prefix="MIX_BKT2">
                     <id-column name="id"
                              type="${id.column.type}"/>
                   <data-column name="datum"
                               type="${data.column.type}"/>
                   <timestamp-column name="version"

type="${timestamp.column.type}"/>
            </binary-keyed-table>
```

3. **The string-keyed-table Element**
The **`string-keyed-table`** element specifies information about the database table used to store string based cache entries.

   a. The ***prefix*** parameter specifies a prefix string for the database table name.

```
<local-cache>
      <mixed-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
         passivation="true"
         preload="false"
         purge="false">
          <binary-keyed-table prefix="MIX_BKT2">
                     <id-column name="id"
                              type="${id.column.type}"/>
                   <data-column name="datum"
```

```
                            type="${data.column.type}"/>
                        <timestamp-column name="version"

type="${timestamp.column.type}"/>
                </binary-keyed-table>
                <string-keyed-table prefix="MIX_STR2">
```

4. **The id-column Element**
   The **`id-column`** element specifies information about a database column that holds cache entry IDs.

   a. The ***name*** parameter specifies the name of the ID column.

   b. The ***type*** parameter specifies the type of the ID column.

```
<local-cache>
       <mixed-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
         passivation="true"
         preload="false"
         purge="false">
          <binary-keyed-table prefix="MIX_BKT2">
                     <id-column name="id"
                              type="${id.column.type}"/>
                   <data-column name="datum"
                               type="${data.column.type}"/>
                   <timestamp-column name="version"

type="${timestamp.column.type}"/>
            </binary-keyed-table>
            <string-keyed-table prefix="MIX_STR2">
                     <id-column name="id"
                             type="${id.column.type}"/>
```

5. **The data-column Element**
   The **`data-column`** element contains information about a database column that holds cache entry data.

   a. The ***name*** parameter specifies the name of the database column.

   b. The ***type*** parameter specifies the type of the database column.

```
<local-cache>
        <mixed-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
         passivation="true"
         preload="false"
         purge="false">
          <binary-keyed-table prefix="MIX_BKT2">
                     <id-column name="id"
                              type="${id.column.type}"/>
                   <data-column name="datum"
                               type="${data.column.type}"/>
                   <timestamp-column name="version"
```

```
type="${timestamp.column.type}"/>
            </binary-keyed-table>
                 <string-keyed-table prefix="MIX_STR2">
                     <id-column name="id"
                             type="${id.column.type}"/>
                   <data-column name="datum"
                               type="${data.column.type}"/>
```

6. **The timestamp-column Element**
   The **`timestamp-column`** element specifies information about the database column that holds cache entry timestamps.

   a. The ***name*** parameter specifies the name of the timestamp column.

   b. The ***type*** parameter specifies the type of the timestamp column.

```
  <local-cache>
      <mixed-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
         passivation="true"
         preload="false"
         purge="false">
          <binary-keyed-table prefix="MIX_BKT2">
                     <id-column name="id"
                              type="${id.column.type}"/>
                   <data-column name="datum"
                               type="${data.column.type}"/>
                   <timestamp-column name="version"

type="${timestamp.column.type}"/>
            </binary-keyed-table>
                 <string-keyed-table prefix="MIX_STR2">
                     <id-column name="id"
                             type="${id.column.type}"/>
                   <data-column name="datum"
                               type="${data.column.type}"/>
               <timestamp-column name="version"
                               type="${timestamp.column.type}"/>
              </string-keyed-table>
        </mixed-keyed-jdbc-store>
   </local-cache>
    </cache-container>
</subsystem>
```

Report a bug

15.3.2. JdbcMixedStore Configuration (Library Mode)

The following is a sample configuration for the **`mixedKeyedJdbcStore`**:

Procedure 15.9. Configure JdbcMixedStore in Library Mode

1. **The mixedKeyedJdbcStore Element**
   The **`mixedKeyedJdbcStore`** element uses the following parameters to configure the cache

store:

a. The ***fetchPersistentState*** parameter determines whether the persistent state is fetched when joining a cluster. Set this to **true** if using a replication and invalidation in a clustered environment. Additionally, if multiple cache stores are chained, only one cache store can have this property enabled. If a shared cache store is used, the cache does not allow a persistent state transfer despite this property being set to **true**. The ***fetchPersistentState*** parameter is **false** by default.

b. The **ignoreModifications** parameter determines whether operations that modify the cache (for example put, remove, clear, store, etc.) do not affect the cache store. As a result, the cache store can become out of sync with the cache.

c. The **purgeOnStartup** parameter specifies whether the cache is purged when initially started.

d. The **key2StringMapper** parameter specifies the class name of the Key2StringMapper used to map keys to strings for the database tables.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <mixedKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                       fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.persistence.keymappers.DefaultTwoWa
yKey2StringMapper">
```

2. **The binaryKeyedTable and stringKeyedTable Elements**
The **binaryKeyedTable** and the **stringKeyedTable** element defines the table that stores cache entries. Each uses the following parameters to configure the cache store:

a. The ***dropOnExit*** parameter specifies whether the database tables are dropped upon shutdown.

b. The ***createOnStart*** parameter specifies whether the database tables are created by the store on startup.

c. The ***prefix*** parameter defines the string prepended to name of the target cache when composing the name of the cache bucket table.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
```

```
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <mixedKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.persistence.keymappers.DefaultTwoWa
yKey2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE_BINARY">
```

3. **The idColumn Element**
   The **`idColumn`** element defines the column where the cache key or bucket ID is stored. It uses the following parameters:
   a. Use the ***name*** parameter to specify the name of the column used.
   b. Use the ***type*** parameter to specify the type of the column used.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
   <persistence>
 <mixedKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.persistence.keymappers.DefaultTwoWa
yKey2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE_BINARY">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
```

▪

4. **The dataColumn Element**
   The **`dataColumn`** element specifies the column where the cache entry or bucket is stored.

   a. Use the ***name*** parameter to specify the name of the column used.

   b. Use the ***type*** parameter to specify the type of the column used.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <mixedKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.persistence.keymappers.DefaultTwoWa
yKey2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE_BINARY">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
   <dataColumn name="DATA_COLUMN"
        type="BINARY" />
```

5. **The timestampColumn Element**
   The **`timestampColumn`** element specifies the column where the time stamp of the cache entry or bucket is stored.

   a. Use the ***name*** parameter to specify the name of the column used.

   b. Use the ***type*** parameter to specify the type of the column used.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
```

```
<persistence>
 <mixedKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                      fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.persistence.keymappers.DefaultTwoWa
yKey2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE_BINARY">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
   <dataColumn name="DATA_COLUMN"
        type="BINARY" />
   <timestampColumn name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </binaryKeyedTable>
```

6. **The string-keyed-table Element**
   The **`string-keyed-table`** element specifies information about the database table used to store string based cache entries.

   a. The ***`prefix`*** parameter specifies a prefix string for the database table name.

```
<infinispan
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:config:6.0
http://www.infinispan.org/schemas/infinispan-config-6.0.xsd
            urn:infinispan:config:jdbc:6.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
6.0.xsd"
        xmlns="urn:infinispan:config:6.0">
        ...
<persistence>
 <mixedKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:6.0"
                         fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.persistence.keymappers.DefaultTwoWa
yKey2StringMapper">
  <connectionPool
connectionUrl="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1"
    username="sa"
    driverClass="org.h2.Driver"/>
  <binaryKeyedTable dropOnExit="true"
      createOnStart="true"
      prefix="ISPN_BUCKET_TABLE_BINARY">
   <idColumn name="ID_COLUMN"
```

```
        type="VARCHAR(255)" />
     <dataColumn name="DATA_COLUMN"
          type="BINARY" />
     <timestampColumn name="TIMESTAMP_COLUMN"
        type="BIGINT" />
   </binaryKeyedTable>
   <stringKeyedTable dropOnExit="true"
        createOnStart="true"
        prefix="ISPN_BUCKET_TABLE_STRING">
     <idColumn name="ID_COLUMN"
        type="VARCHAR(255)" />
     <dataColumn name="DATA_COLUMN"
          type="BINARY" />
     <timestampColumn name="TIMESTAMP_COLUMN"
        type="BIGINT" />
   </stringKeyedTable>
  </mixedKeyedJdbcStore>
</persistence>
```

Report a bug

15.3.3. JdbcMixedStore Programmatic Configuration

The following is a sample configuration for the **`JdbcMixedStore`**:

Procedure 15.10. Configure JdbcMixedStore Programmatically

1. **Create a New Configuration Builder**
   Use the ***`ConfigurationBuilder`*** to create a new configuration object.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();
   ```

2. **Add `JdbcMixedStoreConfigurationBuilder`**
   Add the **`JdbcMixedStore`** configuration builder to build a specific configuration related to this store.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();

   builder.persistence().addStore(JdbcMixedStoreConfigurationBuilder.cl
   ass)
   ```

3. **Set Up Persistence**
   `fetchPersistentState` determines whether or not to fetch the persistent state of a cache and apply it to the local cache store when joining the cluster. If the cache store is shared the state is ignored, as caches access the same cache store. A configuration exception will be thrown when starting the cache service if more than one cache loader has this property set to **`true`**. The ***`fetchPersistentState`*** property is **`false`** by default.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();

   builder.persistence().addStore(JdbcMixedStoreConfigurationBuilder.cl
   ass)
        .fetchPersistentState(false)
   ```

4. **Set Modifications**
   ignoreModifications determines whether write methods are pushed to the specific cache loader by allowing write operations to the local file cache loader, but not the shared cache loader. In some cases, transient application data should only reside in a file-based cache loader on the same server as the in-memory cache. For example, this would apply with a further JDBC based cache loader used by all servers in the network. ***ignoreModifications*** is **false** by default.

```
ConfigurationBuilder builder = new ConfigurationBuilder();

builder.persistence().addStore(JdbcMixedStoreConfigurationBuilder.cl
ass)
      .fetchPersistentState(false)
      .ignoreModifications(false)
```

5. **Configure Purging**
   purgeOnStartup specifies whether the cache is purged when initially started.

```
ConfigurationBuilder builder = new ConfigurationBuilder();

builder.persistence().addStore(JdbcMixedStoreConfigurationBuilder.cl
ass)
      .fetchPersistentState(false)
      .ignoreModifications(false)
      .purgeOnStartup(false)
```

6. **Configure the Table**

   a. **Set Drop Table On Exit Method**
      dropOnExit determines if the table will be created when the cache store is stopped. This is set to **false** by default.

   b. **Set Create Table On Start Method**
      createOnStart creates the table when starting the cache store if no table currently exists. This method is **true** by default.

   c. **Set the Table Name Prefix**
      tableNamePrefix sets the prefix for the name of the table in which the data will be stored.

   d. ***idColumnName***
      The ***idColumnName*** property defines the column where the cache key or bucket ID is stored.

   e. ***dataColumnName***
      The ***dataColumnName*** property specifies the column where the cache entry or bucket is stored.

   f. ***timestampColumnName***
      The ***timestampColumnName*** element specifies the column where the time stamp of the cache entry or bucket is stored.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
```

```
builder.persistence().addStore(JdbcMixedStoreConfigurationBuilder.cl
ass)
     .fetchPersistentState(false)
     .ignoreModifications(false)
     .purgeOnStartup(false)
     .stringTable()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_MIXED_STR_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

.timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT
")
```

7. **The connectionPool Element**
The **connectionPool** element specifies a connection pool for the JDBC driver using the following parameters:

   a. The ***connectionUrl*** parameter specifies the JDBC driver-specific connection URL.

   b. The ***username*** parameter contains the username used to connect via the ***connectionUrl***.

   c. The ***driverClass*** parameter specifies the class name of the driver used to connect to the database.

```
ConfigurationBuilder builder = new ConfigurationBuilder();

builder.persistence().addStore(JdbcMixedStoreConfigurationBuilder.cl
ass)
     .fetchPersistentState(false)
     .ignoreModifications(false)
     .purgeOnStartup(false)
     .stringTable()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_MIXED_STR_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

.timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT
")
     .binaryTable()
        .dropOnExit(true)
        .createOnStart(true)
        .tableNamePrefix("ISPN_MIXED_BINARY_TABLE")
        .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
        .dataColumnName("DATA_COLUMN").dataColumnType("BINARY")

.timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT
")
     .connectionPool()

.connectionUrl("jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-
1")
```

```
            .username("sa")
            .driverClass("org.h2.Driver");
```

NOTE

Programmatic configurations can only be used with Red Hat JBoss Data Grid Library mode.

Report a bug

15.4. CACHE STORE TROUBLESHOOTING

15.4.1. IOExceptions with JdbcStringBasedStore

An IOException **`Unsupported protocol version 48`** error when using **`JdbcStringBasedStore`** indicates that your data column type is set to **`VARCHAR`**, **`CLOB`** or something similar instead of the correct type, **`BLOB`** or **`VARBINARY`**. Despite its name, **`JdbcStringBasedStore`** only requires that the keys are strings while the values can be any data type, so that they can be stored in a binary column.

Report a bug

CHAPTER 16. CACHE LOADERS

The cache loader provides Red Hat JBoss Data Grid's connection to a persistent data store. The cache loader retrieves data from a data store when the required data is not present in the cache. If a cache loader is extended, it can be used as a cache store and can copy the modified data to the data store.

Cache loaders are associated with individual caches. Different caches attached to the same cache manager can have different cache store configurations.

Report a bug

16.1. CACHE LOADERS AND CACHE STORES

JBoss Cache originally shipped with a **`CacheLoader`** interface and a number of implementations. Red Hat JBoss Data Grid has divided these into two distinct interfaces, a **`CacheLoader`** and a **`CacheStore`**. The **`CacheLoader`** loads a previously existing state from another location, while the **`CacheStore`** (which extends **`CacheLoader`**) exposes methods to store states as well as loading them. This division allows easier definition of read-only sources.

JBoss Data Grid ships with several high performance implementations of these interfaces.

Report a bug

16.2. CACHE LOADER CONFIGURATION

16.2.1. Configuring the Cache Loader

Cache loaders can be configured in a chain. Cache read operations will check each of the cache loaders in the order configured until a valid non-null element of data has been located. Write operations affect all cache loaders unless the ***`ignoreModifications`*** element has been set to `"true"` for a specific cache loader.

Report a bug

16.2.2. Configure the Cache Loader using XML

The following example demonstrates cache loader configuration using XML.

Procedure 16.1. Configure the Cache Loader Using XML

1. **Create a New Cache Loader**
   Create a new cache loader, specifying ***`passivation`***, ***`shared`***, and ***`preload`*** settings.

   a. ***`passivation`*** affects the way in which Red Hat JBoss Data Grid interacts with loaders. Passivation removes an object from in-memory cache and writes it to a secondary data store, such as a system or database. Passivation is `false` by default.

   b. ***`shared`*** indicates that the cache loader is shared by different cache instances. For example, where all instances in a cluster use the same JDBC settings to talk to the same remote, shared database. ***`shared`*** is `false` by default. When set to `true`, it prevents duplicate data being written to the cache loader by different cache instances.

   c. ***`preload`*** is set to `false` by default. When set to `true` the data stored in the cache loader is preloaded into the memory when the cache starts. This allows data in the cache loader to be available immediately after startup and avoids cache operations delays as a result of loading data lazily. Preloaded data is only stored locally on the node, and there is no replication or distribution of the preloaded data. Red Hat JBoss Data Grid will only preload up to the maximum configured number of entries in eviction.

```
<persistence passivation="false" shared="false" preload="true">
```

2. **Set Up Persistence and Purging**

    a. ***`fetchPersistentState`*** determines whether or not to fetch the persistent state of a cache and apply it to the local cache store when joining the cluster. If the cache store is shared the fetch persistent state is ignored, as caches access the same cache store. A configuration exception will be thrown when starting the cache service if more than one cache loader has this property set to **`true`**. The ***`fetchPersistentState`*** property is **`false`** by default.

    b. ***`purgeSynchronously`*** controls whether expiration occurs in the eviction thread. When set to **`true`**, the eviction thread will block until the purge is finished, rather than bring returned immediately. The ***`purgeSychronously`*** property is set to **`false`** by default. If the cache loader supports multi-thread purge, ***`purgeThreads`*** are used to purge expired entries. ***`purgeThreads`*** is set to **`1`** by default. Check cache loader configuration to determine if multi-thread purge is supported.

    c. ***`ignoreModifications`*** determines whether write methods are pushed to the specific cache loader by allowing write operations to the local file cache loader, but not the shared cache loader. In some cases, transient application data should only reside in a file-based cache loader on the same server as the in-memory cache. For example, this would apply with a further JDBC based cache loader used by all servers in the network. ***`ignoreModifications`*** is **`false`** by default.

```
<persistence passivation="false" shared="false" preload="true">
   <fileStore
           fetchPersistentState="true"
           purgerThreads="3"
           purgeSynchronously="true"
           ignoreModifications="false"
           purgeOnStartup="false"
           location="${java.io.tmpdir}" />
```

3. **Asynchronous Settings**
These attributes configure aspects specific to each cache loader. For example, the ***`location`*** attribute points to where the SingleFileStore keeps files containing data. Other loaders may require more complex configuration.

```
<persistence passivation="false" shared="false" preload="true">
   <fileStore
           fetchPersistentState="true"
           purgerThreads="3"
           purgeSynchronously="true"
           ignoreModifications="false"
           purgeOnStartup="false"
           location="${java.io.tmpdir}" >
      <async
```

```
                enabled="true"
                flushLockTimeout="15000"
                threadPoolSize="5" />
    </fileStore>
```

4. **Configure Singletons and Push States**

   a. ***singletonStore*** enables modifications to be stored by only one node in the cluster. This node is called the coordinator. The coordinator pushes the caches in-memory states to disk. This function is activated by setting the ***enabled*** attribute to **`true`** in all nodes. The ***shared*** parameter cannot be defined with ***singletonStore*** enabled at the same time. The ***enabled*** attribute is **`false`** by default.

   b. ***pushStateWhenCoordinator*** is set to **`true`** by default. If **`true`**, this property will cause a node that has become the coordinator to transfer in-memory state to the underlying cache loader. This parameter is useful where the coordinator has crashed and a new coordinator is elected.

```
<persistence passivation="false" shared="false" preload="true">
   <fileStore
                fetchPersistentState="true"
                purgerThreads="3"
                purgeSynchronously="true"
                ignoreModifications="false"
                purgeOnStartup="false"
                location="${java.io.tmpdir}" >
      <async
                enabled="true"
                flushLockTimeout="15000"
                threadPoolSize="5" />
      <singletonStore
                enabled="true"
                pushStateWhenCoordinator="true"
                pushStateTimeout="20000" />
   </fileStore>
</persistence>
```

Report a bug

16.2.3. Configure the Cache Loader Programmatically

The following example demonstrates how to configure the cache loader programmatically.

Procedure 16.2. Configure the Cache Loader Programatically

1. **Create a New Configuration Builder**
   Use the ***ConfigurationBuilder*** to create a new configuration object.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
```

2. **Set Passivation**

passivation affects the way Red Hat JBoss Data Grid interacts with loaders. Passivation removes an object from in-memory cache and writes it to a secondary data loader, such as a system or database. Passivation is **false** by default.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
    .passivation(false)
```

3. **Set Up Sharing**
   shared indicates that the cache loader is shared by different cache instances. For example, where all instances in a cluster use the same JDBC settings to talk to the same remote, shared database. ***shared*** is **false** by default. When set to **true**, it prevents duplicate data being written to the cache loader by different cache instances.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();
   builder.persistence()
       .passivation(false)
       .shared(false)
   ```

4. **Set Up Preloading**
   preload is set to **false** by default. When set to **true** the data stored in the cache loader is preloaded into the memory when the cache starts. This allows data in the cache loader to be available immediately after startup and avoids cache operations delays as a result of loading data lazily. Preloaded data is only stored locally on the node, and there is no replication or distribution of the preloaded data. JBoss Data Grid will only preload up to the maximum configured number of entries in eviction.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();
   builder.persistence()
       .passivation(false)
       .shared(false)
       .preload(true)
   ```

5. **Configure the Cache Loader**
   addSingleFileStore() adds the SingleFileStore as the cache loader for this configuration. It is possible to create other stores, such as a JDBC Cache Store, which can be added using the ***addLoader*** method.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();
   builder.persistence()
       .passivation(false)
       .shared(false)
       .preload(true)
       .addSingleFileStore()
   ```

6. **Set Up Persistence**
   fetchPersistentState determines whether or not to fetch the persistent state of a cache and apply it to the local cache loader when joining the cluster. If the cache loader is shared the fetch persistent state is ignored, as caches access the same cache loader. A configuration exception will be thrown when starting the cache service if more than one cache loader has this property set to **true**. The ***fetchPersistentState*** property is **false** by default.

   ```
   ConfigurationBuilder builder = new ConfigurationBuilder();
   ```

```
builder.persistence()
    .passivation(false)
    .shared(false)
    .preload(true)
    .addSingleFileStore()
        .fetchPersistentState(true)
```

7. **Set Up Purging**
   purgeSynchronously controls whether expiration occurs in the eviction thread. When set to **`true`**, the eviction thread will block until the purge is finished, rather than bring returned immediately. The ***purgeSychronously*** property is set to **`false`** by default. If the cache loader supports multi-thread purge, ***purgeThreads*** are used to purge expired entries. ***purgeThreads*** is set to **`1`** by default. Check cache loader configuration to determine if multi-thread purge is supported.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
    .passivation(false)
    .shared(false)
    .preload(true)
    .addSingleFileStore()
        .fetchPersistentState(true)
        .purgerThreads(3)
        .purgeSynchronously(true)
```

8. **Set Modifications**
   ignoreModifications determines whether write methods are pushed to the specific cache loader by allowing write operations to the local file cache loader, but not the shared cache loader. In some cases, transient application data should only reside in a file-based cache loader on the same server as the in-memory cache. For example, this would apply with a further JDBC based cache loader used by all servers in the network. ***ignoreModifications*** is **`false`** by default.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
    .passivation(false)
    .shared(false)
    .preload(true)
    .addSingleFileStore()
        .fetchPersistentState(true)
        .purgerThreads(3)
        .purgeSynchronously(true)
        .ignoreModifications(false)
```

9. **Asynchronous Settings**
   These attributes configure aspects specific to each cache loader. For example, the ***location*** attribute points to where the SingleFileStore will keep files containing data. Other loaders may require more complex configuration.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
    .passivation(false)
    .shared(false)
    .preload(true)
```

```
        .addSingleFileStore()
            .fetchPersistentState(true)
            .purgerThreads(3)
            .purgeSynchronously(true)
            .ignoreModifications(false)
            .purgeOnStartup(false)
            .location(System.getProperty("java.io.tmpdir"))
            .async()
                .enabled(true)
                .flushLockTimeout(15000)
                .threadPoolSize(5)
```

10. **Configure Singletons**
    `singletonStore` enables modifications to be stored by only one node in the cluster. This node is called the coordinator. The coordinator pushes the caches in-memory states to disk. This function is activated by setting the ***`enabled`*** attribute to **`true`** in all nodes. The ***`shared`*** parameter cannot be defined with ***`singletonStore`*** enabled at the same time. The ***`enabled`*** attribute is **`false`** by default.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
    .passivation(false)
    .shared(false)
    .preload(true)
    .addSingleFileStore()
        .fetchPersistentState(true)
        .purgerThreads(3)
        .purgeSynchronously(true)
        .ignoreModifications(false)
        .purgeOnStartup(false)
        .location(System.getProperty("java.io.tmpdir"))
        .async()
            .enabled(true)
            .flushLockTimeout(15000)
            .threadPoolSize(5)
        .singletonStore()
           .enabled(true)
```

11. **Set Up Push States**
    `pushStateWhenCoordinator` is set to **`true`** by default. If **`true`**, this property will cause a node that has become the coordinator to transfer in-memory state to the underlying cache loader. This parameter is useful where the coordinator has crashed and a new coordinator is elected.

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.persistence()
    .passivation(false)
    .shared(false)
    .preload(true)
    .addSingleFileStore()
        .fetchPersistentState(true)
        .purgerThreads(3)
        .purgeSynchronously(true)
        .ignoreModifications(false)
        .purgeOnStartup(false)
```

```
          .location(System.getProperty("java.io.tmpdir"))
          .async()
              .enabled(true)
              .flushLockTimeout(15000)
              .threadPoolSize(5)
          .singletonStore()
             .enabled(true)
             .pushStateWhenCoordinator(true)
             .pushStateTimeout(20000);
```

Report a bug

16.3. SHARED CACHE LOADERS

A shared cache loader is a cache loader that is shared by multiple cache instances.

A cache loader is useful when all instances in a cluster communicate with the same remote, shared database using the same JDBC settings. In such an instance, configuring a shared cache loader prevents the unnecessary repeated write operations that occur when various cache instances attempt to write the same data to the cache loader.

Report a bug

16.3.1. Enable Shared Cache Loaders

Library Mode

In Red Hat JBoss Data Grid's Library mode, toggle cache loader sharing using the ***shared*** parameter within the `loader` element. This parameter is set to **FALSE** as a default. Enable cache loader sharing by setting the ***shared*** parameter to **TRUE**.

Remote Client-Server Mode

In JBoss Data Grid's Remote Client-Server mode, toggle cache loader sharing using the ***shared*** parameter within the `store` element. This parameter is set to **FALSE** as a default. Enable cache loader sharing by setting the ***shared*** parameter to **TRUE**. For example:

```
<jdbc-store shared="true">
...
</jdbc-store>
```

Report a bug

16.3.2. Invalidation Mode and Shared Cache Loaders

When used in conjunction with a shared cache loader, Red Hat JBoss Data Grid's invalidation mode causes remote caches to see the shared cache loader to retrieve modified data.

The benefits of using invalidation mode in conjunction with shared cache loaders include the following:

- Compared to replication messages, which contain the updated data, invalidation messages are much smaller and result in reduced network traffic.
- The remaining cluster caches look up modified data from the shared cache loader lazily and only when required to do so, resulting in further reduced network traffic.

Report a bug

16.3.3. The Cache Loader and Cache Passivation

In Red Hat JBoss Data Grid, a cache loader can be used to enforce the passivation of entries and to activate eviction in a cache. Whether passivation mode or activation mode are used, the configured cache loader both reads from and writes to the data store.

When passivation is disabled in JBoss Data Grid, after the modification, addition or removal of an element is carried out the cache loader steps in to persist the changes in the store.

Report a bug

16.3.4. Application Cacheloader Registration

It is not necessary to register an application cache loader for an isolated deployment. This is not a requirement in Red Hat JBoss Data Grid because lazy deserialization is used to work around this problem.

Report a bug

16.4. CONNECTION FACTORIES

In Red Hat JBoss Data Grid, all JDBC cache loaders rely on a `ConnectionFactory` implementation to obtain a database connection. This process is also known as connection management or pooling.

A connection factory can be specified using the ***`ConnectionFactoryClass`*** configuration attribute. JBoss Data Grid includes the following `ConnectionFactory` implementations:

- ManagedConnectionFactory
- SimpleConnectionFactory.

Report a bug

16.4.1. About ManagedConnectionFactory

`ManagedConnectionFactory` is a connection factory that is ideal for use within managed environments such as application servers. This connection factory can explore a configured location in the JNDI tree and delegate connection management to the `DataSource`. `ManagedConnectionFactory` is used within a managed environment that contains a `DataSource`. This `Datasource` is delegated the connection pooling.

Report a bug

16.4.2. About SimpleConnectionFactory

`SimpleConnectionFactory` is a connection factory that creates database connections on a per invocation basis. This connection factory is not designed for use in a production environment.

Report a bug

PART VII. SET UP PASSIVATION

CHAPTER 17. ACTIVATION AND PASSIVATION MODES

Activation is the process of loading an entry into memory and removing it from the cache store. Activation occurs when a thread attempts to access an entry that is in the store but not the memory (namely a passivated entry).

Passivation mode allows entries to be stored in the cache store after they are evicted from memory. Passivation prevents unnecessary and potentially expensive writes to the cache store. It is used for entries that are frequently used or referenced and therefore not evicted from memory.

While passivation is enabled, the cache store is used as an overflow tank, similar to virtual memory implementation in operating systems that swap memory pages to disk.

The passivation flag is used to toggle passivation mode, a mode that stores entries in the cache store only after they are evicted from memory.

Report a bug

17.1. PASSIVATION MODE BENEFITS

The primary benefit of passivation mode is that it prevents unnecessary and potentially expensive writes to the cache store. This is particularly useful if an entry is frequently used or referenced and therefore is not evicted from memory.

Report a bug

17.2. CONFIGURE PASSIVATION

In Red Hat JBoss Data Grid's Remote Client-Server mode, add the ***passivation*** parameter to the cache store element to toggle passivation for it:

```
<local-cache>
 ...
 <file-store passivation="true"
      ... />
 ...
</local-cache>
```

In Library mode, add the ***passivation*** parameter to the **loaders** element to toggle passivation:

```
<persistence passivation="true"
         ... />
   ...
</persistence>
```

Report a bug

17.3. EVICTION AND PASSIVATION

To ensure that a single copy of an entry remains, either in memory or in a cache store, use passivation in conjunction with eviction.

The primary reason to use passivation instead of a normal cache store is that updating entries require less resources when passivation is in use. This is because passivation does not require an update to the cache store.

Report a bug

17.3.1. Eviction and Passivation Usage

If the eviction policy caused the eviction of an entry from the cache while passivation is enabled, the following occur as a result:

- A notification regarding the passivated entry is emitted to the cache listeners.
- The evicted entry is stored.

When an attempt to retrieve an evicted entry is made, the entry is lazily loaded into memory from the cache loader. After the entry and its children are loaded, they are removed from the cache loader and a notification regarding the entry's activation is sent to the cache listeners.

Report a bug

17.3.2. Eviction Example when Passivation is Disabled

The following example indicates the state of the memory and the persistent store during eviction operations with passivation disabled.

Table 17.1. Eviction when Passivation is Disabled

Step	Key in Memory	Key on Disk
Insert **keyOne**	Memory: **keyOne**	Disk: **keyOne**
Insert **keyTwo**	Memory: **keyOne**, **keyTwo**	Disk: **keyOne**, **keyTwo**
Eviction thread runs, evicts **keyOne**	Memory: **keyTwo**	Disk: **keyOne**, **keyTwo**
Read **keyOne**	Memory: **keyOne**, **keyTwo**	Disk: **keyOne**, **keyTwo**
Eviction thread runs, evicts **keyTwo**	Memory: **keyOne**	Disk: **keyOne**, **keyTwo**
Remove **keyTwo**	Memory: **keyOne**	Disk: **keyOne**

Report a bug

17.3.3. Eviction Example when Passivation is Enabled

The following example indicates the state of the memory and the persistent store during eviction operations with passivation enabled.

Table 17.2. Eviction when Passivation is Enabled

Step	Key in Memory	Key on Disk
Insert **keyOne**	Memory: **keyOne**	Disk:
Insert **keyTwo**	Memory: **keyOne**, **keyTwo**	Disk:
Eviction thread runs, evicts **keyOne**	Memory: **keyTwo**	Disk: **keyOne**
Read **keyOne**	Memory: **keyOne**, **keyTwo**	Disk:
Eviction thread runs, evicts **keyTwo**	Memory: **keyOne**	Disk: **keyTwo**
Remove **keyTwo**	Memory: **keyOne**	Disk:

Report a bug

PART VIII. SET UP CACHE WRITING

CHAPTER 18. CACHE WRITING MODES

Red Hat JBoss Data Grid presents configuration options with a single or multiple cache stores. This allows it to store data in a persistent location, for example a shared JDBC database or a local file system. JBoss Data Grid supports two caching modes:

- Write-Through (Synchronous)
- Write-Behind (Asynchronous)

Report a bug

18.1. WRITE-THROUGH CACHING

The Write-Through (or Synchronous) mode in Red Hat JBoss Data Grid ensures that when clients update a cache entry (usually via a **`Cache.put()`** invocation), the call does not return until JBoss Data Grid has located and updated the underlying cache store. This feature allows updates to the cache store to be concluded within the client thread boundaries.

Report a bug

18.1.1. Write-Through Caching Benefits

The primary advantage of the Write-Through mode is that the cache and cache store are updated simultaneously, which ensures that the cache store remains consistent with the cache contents. This is at the cost of reduced performance for cache operations caused by the cache store accesses and updates during cache operations.

Report a bug

18.1.2. Write-Through Caching Configuration (Library Mode)

No specific configuration operations are required to configure a Write-Through or synchronous cache store. All cache stores are Write-Through or synchronous unless explicitly marked as Write-Behind or asynchronous. The following procedure demonstrates a sample configuration file of a Write-Through unshared local file cache store.

Procedure 18.1. Configure a Write-Through Local File Cache Store

1. **Identify the namedCache**
   The ***`name`*** parameter specifies the name of the **`namedCache`** to use.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns="urn:infinispan:config:5.0">
    <global />
    <default />
    <namedCache name="persistentCache">
   ```

2. **Configure the Cache Loader**
   The ***`shared`*** parameter is used when multiple cache instances share a cache store. This parameter can be set to prevent multiple cache instances writing the same modification multiple times. Valid values for this parameter are ***`true`*** and ***`false`***.

```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="urn:infinispan:config:5.0">
 <global />
 <default />
 <namedCache name="persistentCache">
  <persistence shared="false">
```

3. **Specify the Loader Class**
   The ***class*** attribute defines the class of the cache loader implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="urn:infinispan:config:5.0">
 <global />
 <default />
 <namedCache name="persistentCache">
  <persistence shared="false">
  <store class="org.infinispan.loaders.file.FileCacheStore"
```

4. **Configure the *fetchPersistentState* Parameter**
   The ***fetchPersistentState*** parameter determines whether the persistent state is fetched when joining a cluster. Set this to **true** if using a replication and invalidation in a clustered environment. Additionally, if multiple cache stores are chained, only one cache store can have this property enabled. If a shared cache store is used, the cache does not allow a persistent state transfer despite this property being set to **true**. The ***fetchPersistentState*** parameter is **false** by default.

```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="urn:infinispan:config:5.0">
 <global />
 <default />
 <namedCache name="persistentCache">
  <persistence shared="false">
  <store class="org.infinispan.loaders.file.FileCacheStore"
           fetchPersistentState="true"
```

5. **Set the *ignoreModifications* Parameter**
   The ***ignoreModifications*** parameter determines whether operations that modify the cache (e.g. put, remove, clear, store, etc.) do not affect the cache store. As a result, the cache store can become out of sync with the cache.

```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="urn:infinispan:config:5.0">
 <global />
 <default />
 <namedCache name="persistentCache">
  <persistence shared="false">
  <store class="org.infinispan.loaders.file.FileCacheStore"
           fetchPersistentState="true"
           ignoreModifications="false"
```

6. **Configure Purge On Startup**
   The ***purgeOnStartup*** parameter specifies whether the cache is purged when initially started.

```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="urn:infinispan:config:5.0">
 <global />
 <default />
 <namedCache name="persistentCache">
  <persistence shared="false">
  <store class="org.infinispan.loaders.file.FileCacheStore"
           fetchPersistentState="true"
           ignoreModifications="false"
           purgeOnStartup="false">
```

7. **The `property` Element**
   The **`property`** element contains information about properties related to the cache store.

   a. The ***name*** parameter specifies the name of the property.

```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns="urn:infinispan:config:5.0">
 <global />
 <default />
 <namedCache name="persistentCache">
  <persistence shared="false">
  <store class="org.infinispan.loaders.file.FileCacheStore"
           fetchPersistentState="true"
           ignoreModifications="false"
           purgeOnStartup="false">
    <properties>
     <property name="location"
        value="${java.io.tmpdir}" />
    </properties>
   </store>
  </persistence>
   </namedCache>
</infinispan>
```

Report a bug

18.2. WRITE-BEHIND CACHING

In Red Hat JBoss Data Grid's Write-Behind (Asynchronous) mode, cache updates are asynchronously written to the cache store. Asynchronous updates ensure that cache store updates are carried out by a thread different from the client thread interacting with the cache.

One of the foremost advantages of the Write-Behind mode is that the cache operation performance is not affected by the underlying store update. However, because of the asynchronous updates, for a brief period the cache store contains stale data compared to the cache.

Report a bug

18.2.1. About Unscheduled Write-Behind Strategy

In the Unscheduled Write-Behind Strategy mode, Red Hat JBoss Enterprise Data Grid attempts to store changes as quickly as possible by applying pending changes in parallel. This results in multiple threads waiting for modifications to conclude. Once these modifications are concluded, the threads become available and the modifications are applied to the underlying cache store.

This strategy is ideal for cache stores with low latency and low operational costs. An example of this is a local unshared file based cache store in which the cache store is local to the cache itself. Using this strategy the period of time where an inconsistency exists between the contents of the cache and the contents of the cache store is reduced to the shortest possible interval.

Report a bug

18.2.2. Unscheduled Write-Behind Strategy Configuration (Remote Client-Server Mode)

To set the write-behind strategy in Red Hat JBoss Data Grid's Remote Client-Server mode, add the **`write-behind`** element to the target cache store configuration as follows:

Procedure 18.2. The `write-behind` Element

The **`write-behind`** element uses the following configuration parameters:

1. **The *`modification-queue-size`* Parameter**
   The ***`modification-queue-size`*** parameter sets the modification queue size for the asynchronous store. If updates occur faster than the cache store can process the queue, the asynchronous store behaves like a synchronous store. The store behavior remains synchronous and blocks elements until the queue is able to accept them, after which the store behavior becomes asynchronous again.

   ```
   <file-store passivation="false"
               path="${PATH}"
               purge="true"
               shared="false">
       <write-behind modification-queue-size="1024" />
   ```

2. **The *`shutdown-timeout`* Parameter**
   The ***`shutdown-timeout`*** parameter specifies the time in milliseconds after which the cache store is shut down. When the store is stopped some modifications may still need to be applied. Setting a large timeout value will reduce the chance of data loss. The default value for this parameter is **`25000`**.

   ```
   <file-store passivation="false"
               path="${PATH}"
               purge="true"
               shared="false">
       <write-behind modification-queue-size="1024"
                     shutdown-timeout="25000" />
   ```

3. **The *`flush-lock-timeout`* Parameter**
   The ***`flush-lock-timeout`*** parameter specifies the time (in milliseconds) to acquire the lock that guards the state to be periodically flushed. The default value for this parameter is **`15000`**.

```
<file-store passivation="false"
            path="${PATH}"
            purge="true"
            shared="false">
    <write-behind modification-queue-size="1024"
                  shutdown-timeout="25000"
                  flush-lock-timeout="15000" />
```

4. **The *thread-pool-size* Parameter**
   The ***thread-pool-size*** parameter specifies the size of the thread pool. The threads in this thread pool apply modifications to the cache store. The default value for this parameter is **5**.

```
<file-store passivation="false"
            path="${PATH}"
            purge="true"
            shared="false">
    <write-behind modification-queue-size="1024"
                  shutdown-timeout="25000"
                  flush-lock-timeout="15000"
                  thread-pool-size="5" />
</file-store>
```

Report a bug

18.2.3. Unscheduled Write-Behind Strategy Configuration (Library Mode)

To enable the write-behind strategy of the cache entries to a store, add the **async** element to the store configuration as follows:

Procedure 18.3. The async Element

The **async** element uses the following configuration parameters:

1. The ***modificationQueueSize*** parameter sets the modification queue size for the asynchronous store. If updates occur faster than the cache store can process the queue, the asynchronous store behaves like a synchronous store. The store behavior remains synchronous and blocks elements until the queue is able to accept them, after which the store behavior becomes asynchronous again.

```
<persistence>
    <fileStore location="${LOCATION}">
        <async enabled="true"
                  modificationQueueSize="1024" />
```

2. The ***shutdownTimeout*** parameter specifies the time in milliseconds after which the cache store is shut down. This provides time for the asynchronous writer to flush data to the store when a cache is shut down. The default value for this parameter is **25000**.

```
<persistence>
    <fileStore location="${LOCATION}">
        <async enabled="true"
                  modificationQueueSize="1024"
                  shutdownTimeout="25000" />
```

3. The ***flushLockTimeout*** parameter specifies the time (in milliseconds) to acquire the lock that guards the state to be periodically flushed. The default value for this parameter is **15000**.

```
<persistence>
    <fileStore location="${LOCATION}">
        <async enabled="true"
                    modificationQueueSize="1024"
                    shutdownTimeout="25000"
                    flushLockTimeout="15000" />
```

4. The ***threadPoolSize*** parameter specifies the number of threads that concurrently apply modifications to the store. The default value for this parameter is **5**.

```
<persistence>
    <fileStore location="${LOCATION}">
        <async enabled="true"
                    modificationQueueSize="1024"
                    shutdownTimeout="25000"
                    flushLockTimeout="15000"
                    threadPoolSize="5"/>
    </fileStore>
</persistence>
```

Report a bug

PART IX. MONITOR CACHES AND CACHE MANAGERS

CHAPTER 19. SET UP JAVA MANAGEMENT EXTENSIONS (JMX)

19.1. ABOUT JAVA MANAGEMENT EXTENSIONS (JMX)

Java Management Extension (JMX) is a Java based technology that provides tools to manage and monitor applications, devices, system objects, and service oriented networks. Each of these objects is managed, and monitored by **`MBeans`**.

JMX is the de facto standard for middleware management and administration. As a result, JMX is used in Red Hat JBoss Data Grid to expose management and statistical information.

Report a bug

19.2. USING JMX WITH RED HAT JBOSS DATA GRID

Management in Red Hat JBoss Data Grid instances aims to expose as much relevant statistical information as possible. This information allows administrators to view the state of each instance. While a single installation can comprise of tens or hundreds of such instances, it is essential to expose and present the statistical information for each of them in a clear and concise manner.

In JBoss Data Grid, JMX is used in conjunction with JBoss Operations Network (JON) to expose this information and present it in an orderly and relevant manner to the administrator.

Report a bug

19.3. JMX STATISTIC LEVELS

JMX statistics can be enabled at two levels:

- At the cache level, where management information is generated by individual cache instances.
- At the **`CacheManager`** level, where the **`CacheManager`** is the entity that governs all cache instances created from it. As a result, the management information is generated for all these cache instances instead of individual caches.

IMPORTANT

In Red Hat JBoss Data Grid, statistics are enabled by default. While statistics are useful in assessing the status of JBoss Data Grid, they adversely affect performance and must be disabled if they are not required.

Report a bug

19.4. ENABLE JMX FOR CACHE INSTANCES

At the Cache level, JMX statistics can be enabled either declaratively or programmatically, as follows.

Enable JMX Declaratively at the Cache Level

Add the following snippet within either the <default> element for the default cache instance, or under the target <namedCache> element for a specific named cache:

```
<jmxStatistics enabled="true"/>
```

Enable JMX Programmatically at the Cache Level

Add the following code to programmatically enable JMX at the cache level:

```
Configuration configuration = ...
configuration.setExposeJmxStatistics(true);
```

Report a bug

19.5. ENABLE JMX FOR CACHEMANAGERS

At the **CacheManager** level, JMX statistics can be enabled either declaratively or programmatically, as follows.

Enable JMX Declaratively at the CacheManager Level

Add the following in the <global> element to enable JMX declaratively at the **CacheManager** level:

```
<globalJmxStatistics enabled="true"/>
```

Enable JMX Programmatically at the CacheManager Level

Add the following code to programmatically enable JMX at the **CacheManager** level:

```
GlobalConfiguration globalConfiguration = ...
globalConfiguration.setExposeGlobalJmxStatistics(true);
```

Report a bug

19.6. DISABLING THE CACHESTORE VIA JMX

Red Hat JBoss Data Grid allows the CacheStore to be disabled via JMX by invoking the ***disconnectSource*** operation on the **RollingUpgradeManager** MBean.

See Also:

- Section A.15, “RollingUpgradeManager”

Report a bug

19.7. MULTIPLE JMX DOMAINS

Multiple JMX domains are used when multiple **CacheManager** instances exist on a single virtual machine, or if the names of cache instances in different **CacheManagers** clash.

To resolve this issue, name each **CacheManager** in manner that allows it to be easily identified and used by monitoring tools such as JMX and JBoss Operations Network.

Set a CacheManager Name Declaratively

Add the following snippet to the relevant **CacheManager** configuration:

```
<globalJmxStatistics enabled="true" cacheManagerName="Hibernate2LC"/>
```

Set a CacheManager Name Programmatically

Add the following code to set the **CacheManager** name programmatically:

```
GlobalConfiguration globalConfiguration = ...
globalConfiguration.setExposeGlobalJmxStatistics(true);
globalConfiguration.setCacheManagerName("Hibernate2LC");
```

Report a bug

19.8. MBEANS

An **MBean** represents a manageable resource such as a service, component, device or an application.

Red Hat JBoss Data Grid provides **MBeans** that monitor and manage multiple aspects. For example, **MBeans** that provide statistics on the transport layer are provided. If a JBoss Data Grid server is configured with JMX statistics, an **MBean** that provides information such as the hostname, port, bytes read, bytes written and the number of worker threads exists at the following location:

```
jboss.infinispan:type=Server,name=<Memcached|Hotrod>,component=Transport
```

NOTE

A full list of available MBeans, their supported operations and attributes, is available in the Appendix

Report a bug

19.8.1. Understanding MBeans

When JMX reporting is enabled at either the Cache Manager or Cache level, use a standard JMX GUI such as JConsole or VisualVM to connect to a Java Virtual Machine running Red Hat JBoss Data Grid. When connected, the following **MBeans** are available:

- If Cache Manager-level JMX statistics are enabled, an **MBean** named ***jboss.infinispan:type=CacheManager,name="DefaultCacheManager"*** exists, with properties specified by the Cache Manager **MBean**.

- If the cache-level JMX statistics are enabled, multiple **MBeans** display depending on the configuration in use. For example, if a write behind cache store is configured, an **MBean** that exposes properties that belong to the cache store component is displayed. All cache-level **MBeans** use the same format:

  ```
  jboss.infinispan:type=Cache,name="<name-of-cache>(<cache-mode>)",manager="<name-of-cache-manager>",component=<component-name>
  ```

  In this format:

  - Specify the default name for the cache using the **cache-container** element's ***default-cache*** attribute.

- The ***cache-mode*** is replaced by the cache mode of the cache. The lower case version of the possible enumeration values represents the cache mode.
- The ***component-name*** is replaced by one of the JMX component names from the JMX reference documentation.

As an example, the cache store JMX component **MBean** for a default cache configured for synchronous distribution would be named as follows:

```
jboss.infinispan:type=Cache,name="default(dist_sync)",
manager="default",component=CacheStore
```

Each cache and cache manager name is within quotation marks to prevent the use of unsupported characters in these user-defined names.

Report a bug

19.8.2. Registering MBeans in Non-Default MBean Servers

The default location where all the MBeans used are registered is the standard JVM MBeanServer platform. Users can set up an alternative MBeanServer instance as well. Implement the MBeanServerLookup interface to ensure that the **getMBeanServer()** method returns the desired (non default) MBeanServer.

To set up a non default location to register your MBeans, create the implementation and then configure Red Hat JBoss Data Grid with the fully qualified name of the class. An example is as follows:

To Add the Fully Qualified Domain Name Declaratively

Add the following snippet:

```
<globalJmxStatistics enabled="true"
mBeanServerLookup="com.acme.MyMBeanServerLookup"/>
```

To Add the Fully Qualified Domain Name Programmatically

Add the following code:

```
GlobalConfiguration globalConfiguration = ...
globalConfiguration.setExposeGlobalJmxStatistics(true);
globalConfiguration.setMBeanServerLookup("com.acme.MyMBeanServerLookup")
```

Report a bug

CHAPTER 20. SET UP JBOSS OPERATIONS NETWORK (JON)

20.1. ABOUT JBOSS OPERATIONS NETWORK (JON)

The JBoss Operations Network (JON) is JBoss' administration and management platform used to develop, test, deploy and monitor the application life cycle. JBoss Operations Network is JBoss' enterprise management solution and is recommended for the management of multiple Red Hat JBoss Data Grid instances across servers. JBoss Operations Network's agent and auto discovery features facilitate monitoring the Cache Manager and Cache instances in JBoss Data Grid. JBoss Operations Network presents graphical views of key runtime parameters and statistics and allows administrators to set thresholds and be notified if usage exceeds or falls under the set thresholds.

IMPORTANT

In Red Hat JBoss Data Grid, statistics are enabled by default. While statistics are useful in assessing the status of JBoss Data Grid, they adversely affect performance and must be disabled if they are not required.

Report a bug

20.2. DOWNLOAD JBOSS OPERATIONS NETWORK (JON)

20.2.1. Prerequisites for Installing JBoss Operations Network (JON)

In order to install JBoss Operations Network in Red Hat JBoss Data Grid, the following is required:

- A Linux, Windows, or Mac OSX operating system, and an x86_64, i686, or ia64 processor.
- Java 6 or higher is required to run both the JBoss Operations Network Server and the JBoss Operations Network Agent.
- Synchronized clocks on JBoss Operations Network Servers and Agents.
- An external database must be installed.

Report a bug

20.2.2. Download JBoss Operations Network

Use the following procedure to download Red Hat JBoss Operations Network (JON) from the Customer Service Portal:

Procedure 20.1. Download JBoss Operations Network

1. **Access the Customer Service Portal**
   Log in to the Customer Service Portal at https://access.redhat.com
2. **Locate the Product**
   Mouse over **Downloads** and navigate to **JBoss Enterprise Middleware**.
3. **Select the Product**
   Select **JBoss Operations Network for JDG** from the menu.

4. **Download JBoss Operations Network**

   - Select the latest version of JBoss Operations Network Base Distribution and click the **`Download`** link.

   - Select the latest JBoss Data Grid Plugin Pack for JBoss Operations Network and click the **`Download`** link.

Report a bug

20.2.3. Remote JMX Port Values

A port value must be provided to allow Red Hat JBoss Data Grid instances to be located. The value itself can be any available port.

Provide unique (and available) remote JMX ports to run multiple JBoss Data Grid instances on a single machine. A locally running JBoss Operations Network agent can discover each instance using the remote port values.

Report a bug

20.2.4. Download JBoss Operations Network (JON) Plugin

Complete this task to download the JBoss Operations Network (JON) plugin for Red Hat JBoss Data Grid from the Red Hat Customer Portal.

Procedure 20.2. Download Installation Files

1. Open http://access.redhat.com in a web browser.

2. Click **`Downloads`** in the menu across the top of the page.

3. Click **`Downloads`** in the list under JBoss Enterprise Middleware.

4. Enter your login information.

   You are taken to the Software Downloads page.

5. **Download the JBoss Operations Network Plugin**
   If you intend to use the JBoss Operations Network plugin for JBoss Data Grid, select **`JBoss ON for JDG`** from either the Software Downloads drop-down box, or the menu on the left.

   a. Click the **`JBoss Operations Network`** ***`VERSION`*** **`Base Distribution`** download link.

   b. Click the **`Download`** link to start the Base Distribution download.

   c. Repeat the steps to download the **`JDG Plugin Pack for JBoss ON`** ***`VERSION`***

Report a bug

20.3. JBOSS OPERATIONS NETWORK SERVER INSTALLATION

The core of JBoss Operations Network is the server, which communicates with agents, maintains the inventory, manages resource settings, interacts with content providers, and provides a central management UI.

NOTE

For more detailed information about configuring JBoss Operations Network, see the JBoss Operations Network *Installation Guide*.

Report a bug

20.4. JBOSS OPERATIONS NETWORK AGENT

The JBoss Operations Network Agent is a standalone Java application. Only one agent is required per machine, regardless of how many resources you require the agent to manage.

The JBoss Operations Network Agent does not ship fully configured. Once the agent has been installed and configured it can be run as a Windows service from a console, or run as a daemon or `init.d` script in a UNIX environment.

A JBoss Operations Network Agent must be installed on each of the machines being monitored in order to collect data.

The JBoss Operations Agent is typically installed on the same machine on which Red Hat JBoss Data Grid is running, however where there are multiple machines an agent must be installed on each machine.

NOTE

For more detailed information about configuring JBoss Operations Network agents, see the JBoss Operations Network *Installation Guide*.

Report a bug

20.5. JBOSS OPERATIONS NETWORK FOR REMOTE CLIENT-SERVER MODE

In Red Hat JBoss Data Grid's Remote Client-Server mode, the JBoss Operations Network plug-in is used to

- initiate and perform installation and configuration operations.
- monitor resources and their metrics.

In Remote Client-Server mode, the JBoss Operations Network plug-in uses JBoss Enterprise Application Platform's management protocol to obtain metrics and perform operations on the JBoss Data Grid server.

Report a bug

20.5.1. Installing the JBoss Operations Network Plug-in (Remote Client-Server Mode)

The following procedure details how to install the JBoss Operations Network plug-ins for Red Hat JBoss Data Grid's Remote Client-Server mode.

1. **Install the plug-ins**
    - Copy the JBoss Data Grid server rhq plug-in to ***$JON_SERVER_HOME*/plugins**.
    - Copy the Wildfly 7 plug-in to ***$JON_SERVER_HOME*/plugins**.

    The server will automatically discover plug-ins here and deploy them. The plug-ins will be removed from the plug-ins directory after successful deployment.

2. **Obtain plug-ins**
    Obtain all available plug-ins from the JBoss Operations Network server. To do this, type the following into the agent's console:

    ```
    plugins update
    ```

3. **List installed plug-ins**
    Ensure the Wildfly 7 plug-in and the JBoss Data Grid server rhq plug-in are installed correctly using the following:

    ```
    plugins info
    ```

JBoss Operation Network can now discover running JBoss Data Grid servers.

Report a bug

20.6. JBOSS OPERATIONS NETWORK FOR LIBRARY MODE

In Red Hat JBoss Data Grid's Library mode, the JBoss Operations Network plug-in is used to

- initiate and perform installation and configuration operations.
- monitor resources and their metrics.

In Library mode, the JBoss Operations Network plug-in uses JMX to obtain metrics and perform operations on an application using the JBoss Data Grid library.

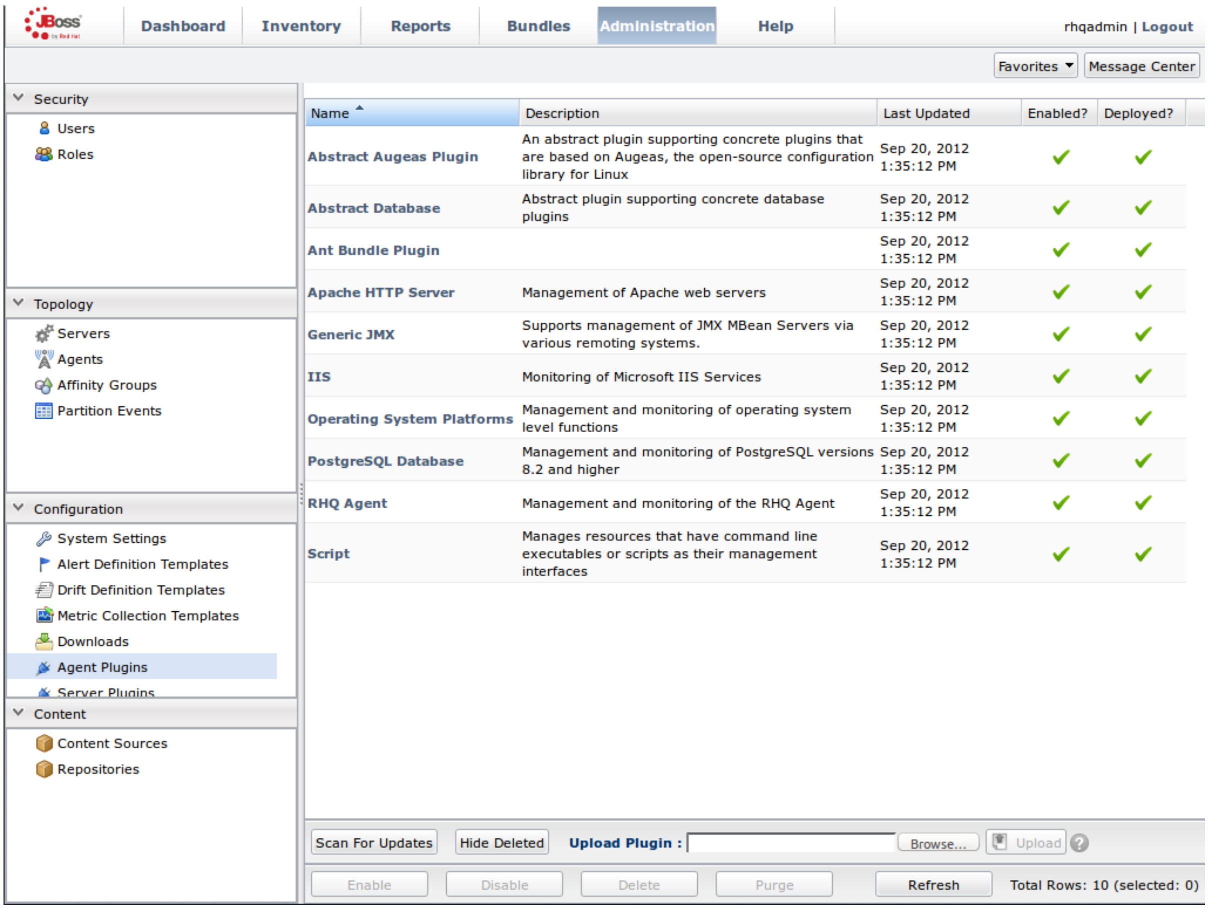
Report a bug

20.6.1. Installing the JBoss Operations Network Plug-in (Library Mode)

Use the following procedure to install the JBoss OperationsNetwork plug-in for Red Hat JBoss Data Grid's Library mode.

Procedure 20.3. Install JBoss Operations Network Library Mode Plug-in

1. **Open the JBoss Operations Network Console**
    a. From the JBoss Operations Network console, select **Administration**.
    b. Select **Agent Plugins** from the **Configuration** options on the left side of the console.



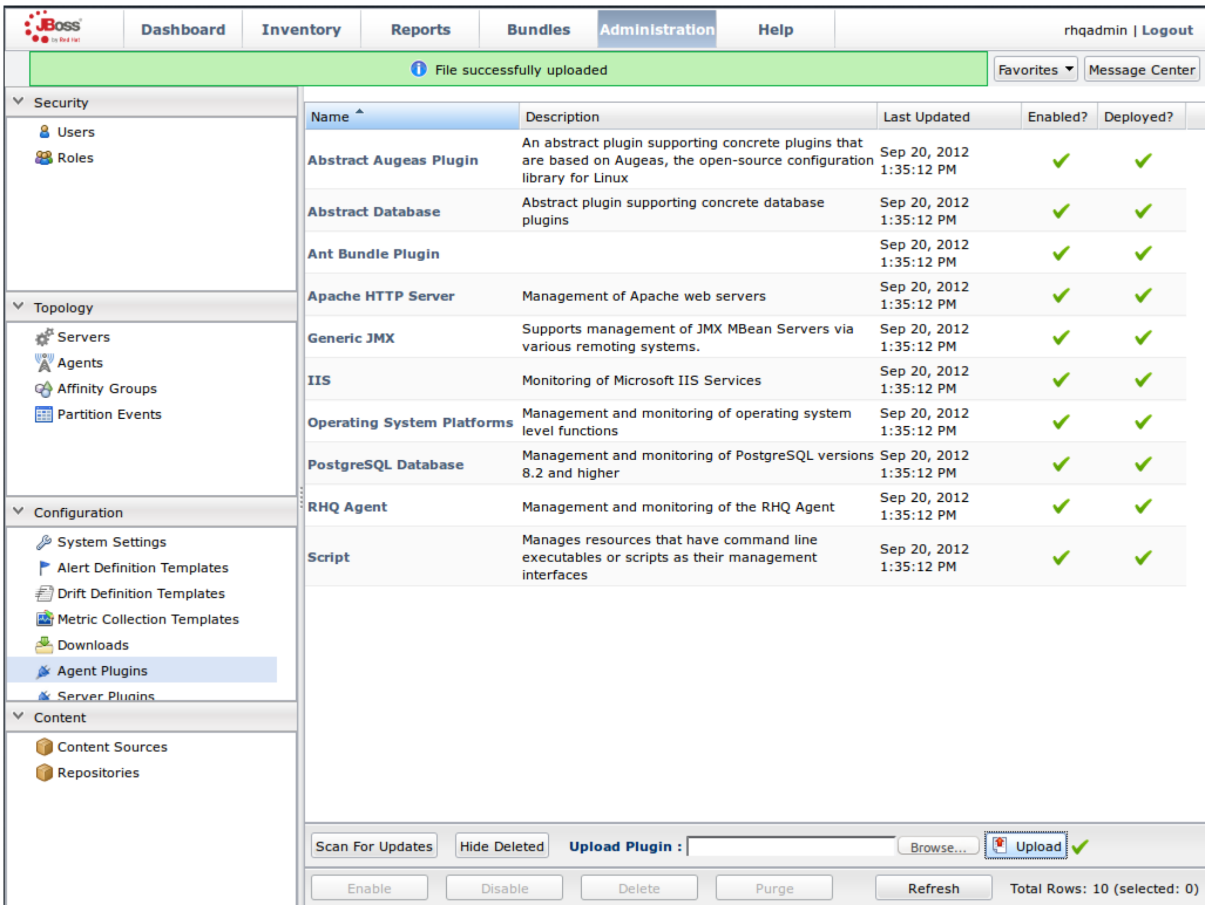
Name	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 1:35:12 PM	✓	✓
Ant Bundle Plugin		Sep 20, 2012 1:35:12 PM	✓	✓
Apache HTTP Server	Management of Apache web servers	Sep 20, 2012 1:35:12 PM	✓	✓
Generic JMX	Supports management of JMX MBean Servers via various remoting systems.	Sep 20, 2012 1:35:12 PM	✓	✓
IIS	Monitoring of Microsoft IIS Services	Sep 20, 2012 1:35:12 PM	✓	✓
Operating System Platforms	Management and monitoring of operating system level functions	Sep 20, 2012 1:35:12 PM	✓	✓
PostgreSQL Database	Management and monitoring of PostgreSQL versions 8.2 and higher	Sep 20, 2012 1:35:12 PM	✓	✓
RHQ Agent	Management and monitoring of the RHQ Agent	Sep 20, 2012 1:35:12 PM	✓	✓
Script	Manages resources that have command line executables or scripts as their management interfaces	Sep 20, 2012 1:35:12 PM	✓	✓

Figure 20.1. JBoss Operations Network Console for JBoss Data Grid

2. **Upload the Library Mode Plug-in**

   a. Click **`Browse`**, locate the **`InfinispanPlugin`** on your local file system.

   b. Click **`Upload`** to add the plug-in to the JBoss Operations Network Server.



File successfully uploaded

Name	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 1:35:12 PM	✓	✓
Ant Bundle Plugin		Sep 20, 2012 1:35:12 PM	✓	✓
Apache HTTP Server	Management of Apache web servers	Sep 20, 2012 1:35:12 PM	✓	✓
Generic JMX	Supports management of JMX MBean Servers via various remoting systems.	Sep 20, 2012 1:35:12 PM	✓	✓
IIS	Monitoring of Microsoft IIS Services	Sep 20, 2012 1:35:12 PM	✓	✓
Operating System Platforms	Management and monitoring of operating system level functions	Sep 20, 2012 1:35:12 PM	✓	✓
PostgreSQL Database	Management and monitoring of PostgreSQL versions 8.2 and higher	Sep 20, 2012 1:35:12 PM	✓	✓
RHQ Agent	Management and monitoring of the RHQ Agent	Sep 20, 2012 1:35:12 PM	✓	✓
Script	Manages resources that have command line executables or scripts as their management interfaces	Sep 20, 2012 1:35:12 PM	✓	✓

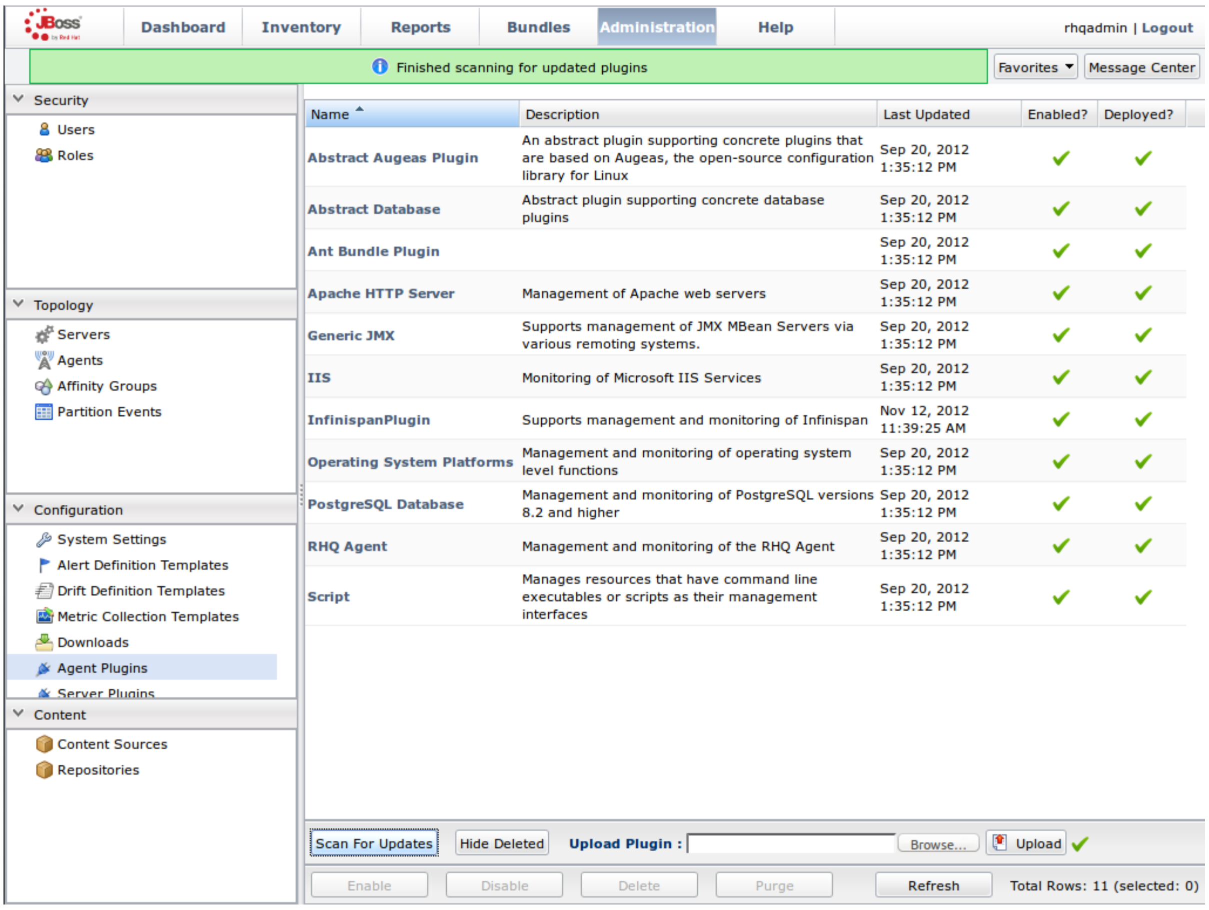
Total Rows: 10 (selected: 0)

Figure 20.2. Upload the `InfinispanPlugin`.

3. **Scan for Updates**

    a. Once the file has successfully uploaded, click **`Scan For Updates`** at the bottom of the screen.

    b. The **`InfinispanPlugin`** will now appear in the list of installed plug-ins.



Finished scanning for updated plugins

Name	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 1:35:12 PM	✓	✓
Ant Bundle Plugin		Sep 20, 2012 1:35:12 PM	✓	✓
Apache HTTP Server	Management of Apache web servers	Sep 20, 2012 1:35:12 PM	✓	✓
Generic JMX	Supports management of JMX MBean Servers via various remoting systems.	Sep 20, 2012 1:35:12 PM	✓	✓
IIS	Monitoring of Microsoft IIS Services	Sep 20, 2012 1:35:12 PM	✓	✓
InfinispanPlugin	Supports management and monitoring of Infinispan	Nov 12, 2012 11:39:25 AM	✓	✓
Operating System Platforms	Management and monitoring of operating system level functions	Sep 20, 2012 1:35:12 PM	✓	✓
PostgreSQL Database	Management and monitoring of PostgreSQL versions 8.2 and higher	Sep 20, 2012 1:35:12 PM	✓	✓
RHQ Agent	Management and monitoring of the RHQ Agent	Sep 20, 2012 1:35:12 PM	✓	✓
Script	Manages resources that have command line executables or scripts as their management interfaces	Sep 20, 2012 1:35:12 PM	✓	✓

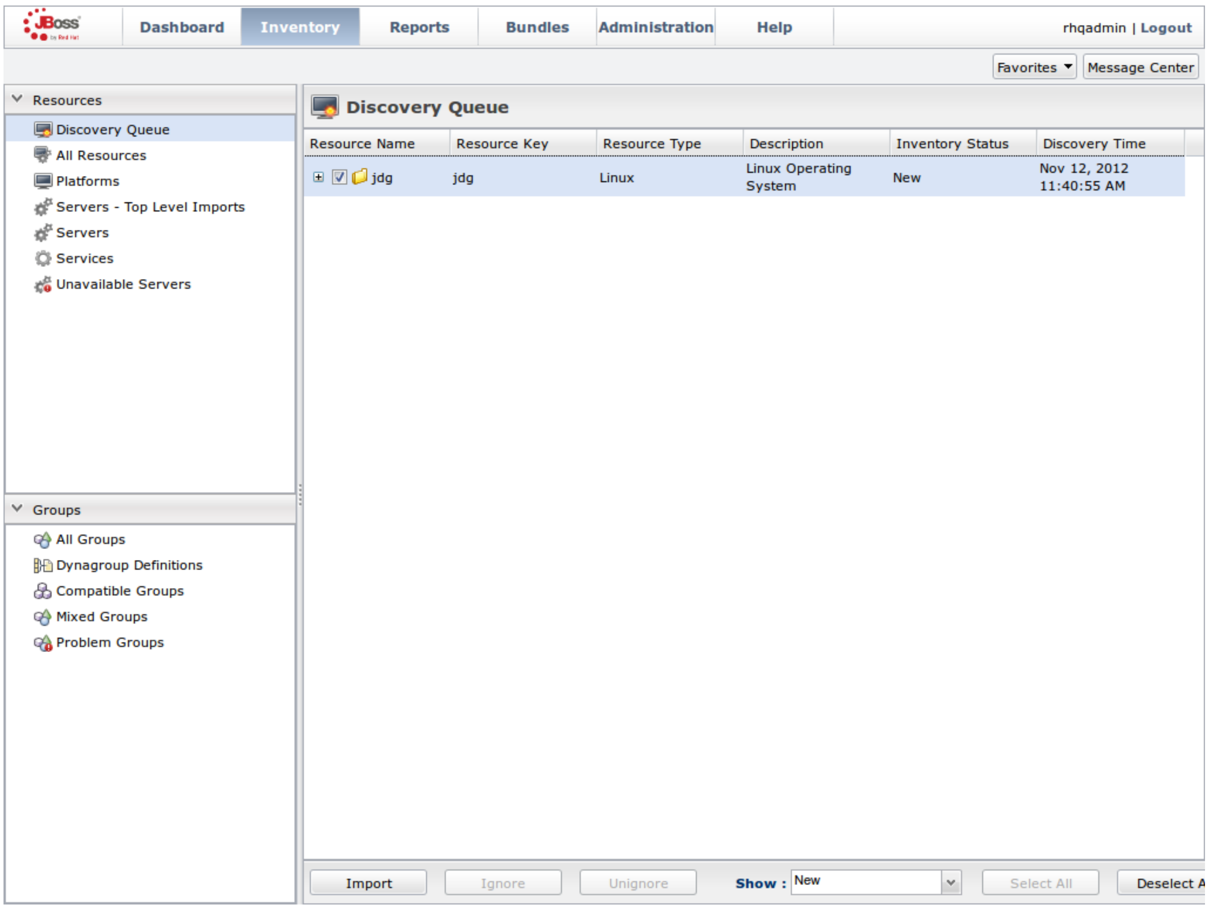
Total Rows: 11 (selected: 0)

Figure 20.3. Scan for Updated Plug-ins.

4. **Import the Platform**

   a. Navigate to the **Inventory** and select **Discovery Queue** from the **Resources** list on the left of the console.

   b. Select the platform on which the application is running and click `Import` at the bottom of the screen.



Discovery Queue

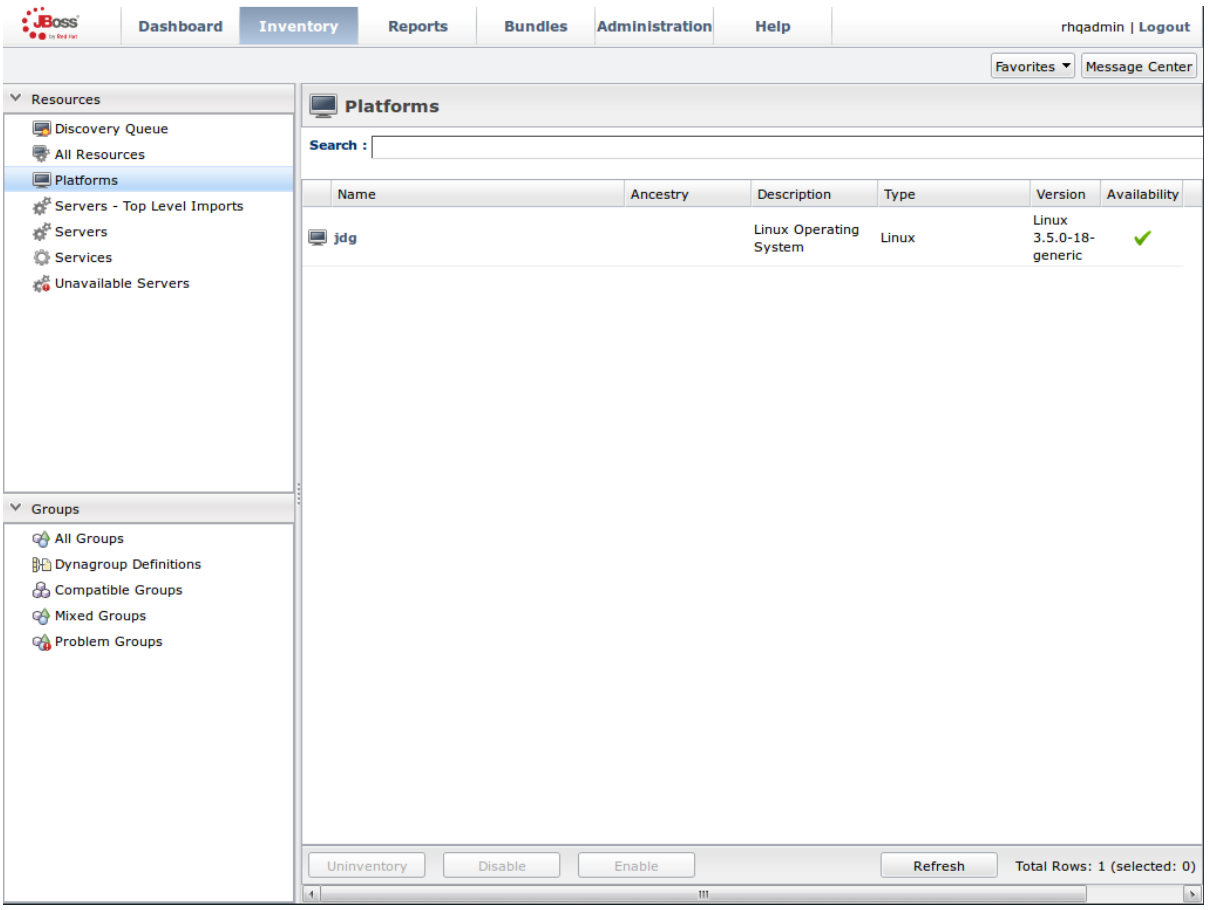
Resource Name	Resource Key	Resource Type	Description	Inventory Status	Discovery Time
jdg	jdg	Linux	Linux Operating System	New	Nov 12, 2012 11:40:55 AM

Figure 20.4. Import the Platform from the Discovery Queue.

5. **Access the Servers on the Platform**

    a. The `jdg` Platform now appears in the `Platforms` list.

    b. Click on the Platform to access the servers that are running on it.



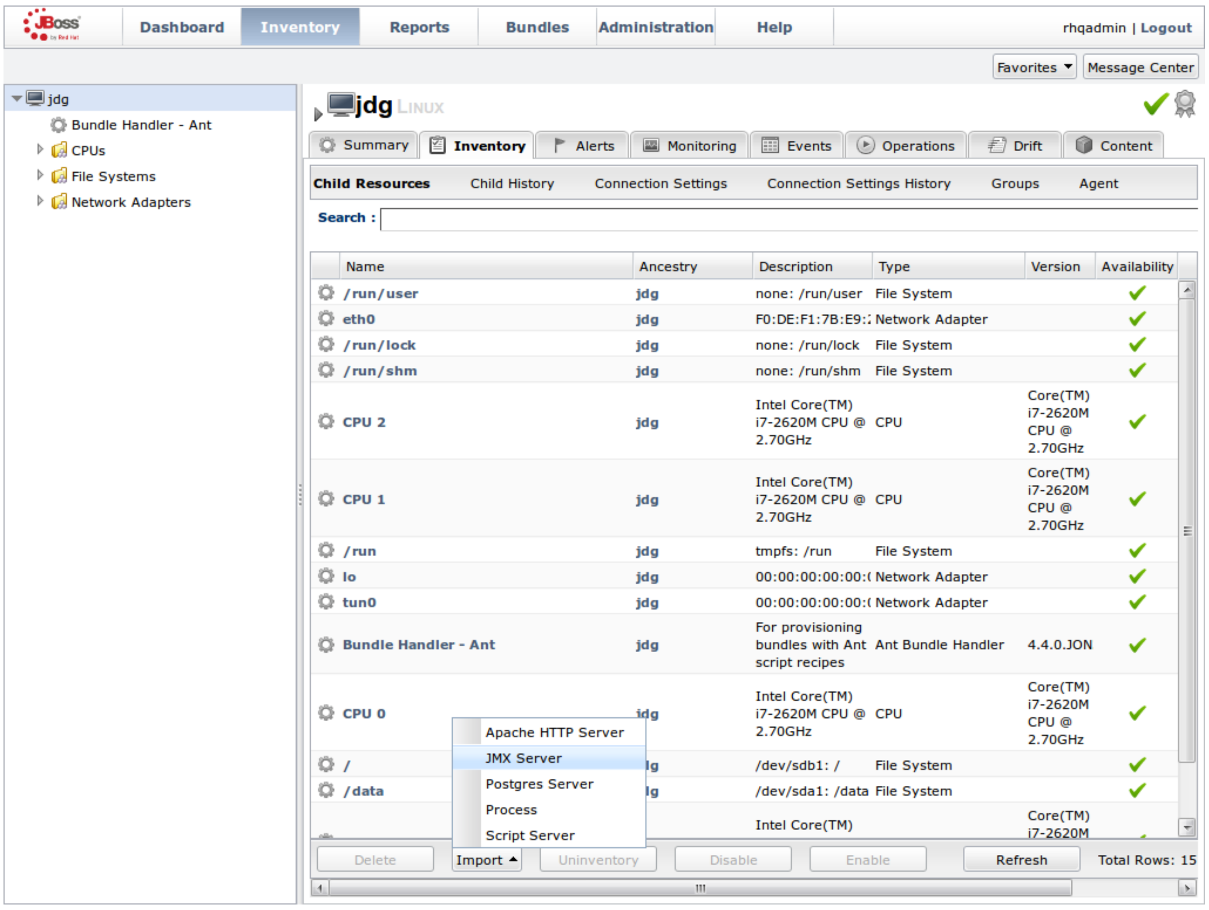
Name	Ancestry	Description	Type	Version	Availability
jdg		Linux Operating System	Linux	Linux 3.5.0-18-generic	✓

Figure 20.5. Open the `jdg` Platform to view the list of servers.

6. **Import the JMX Server**

   a. From the **Inventory** tab, select **Child Resources**.

   b. Click the `Import` button at the bottom of the screen and select the **JMX Server** option from the list.

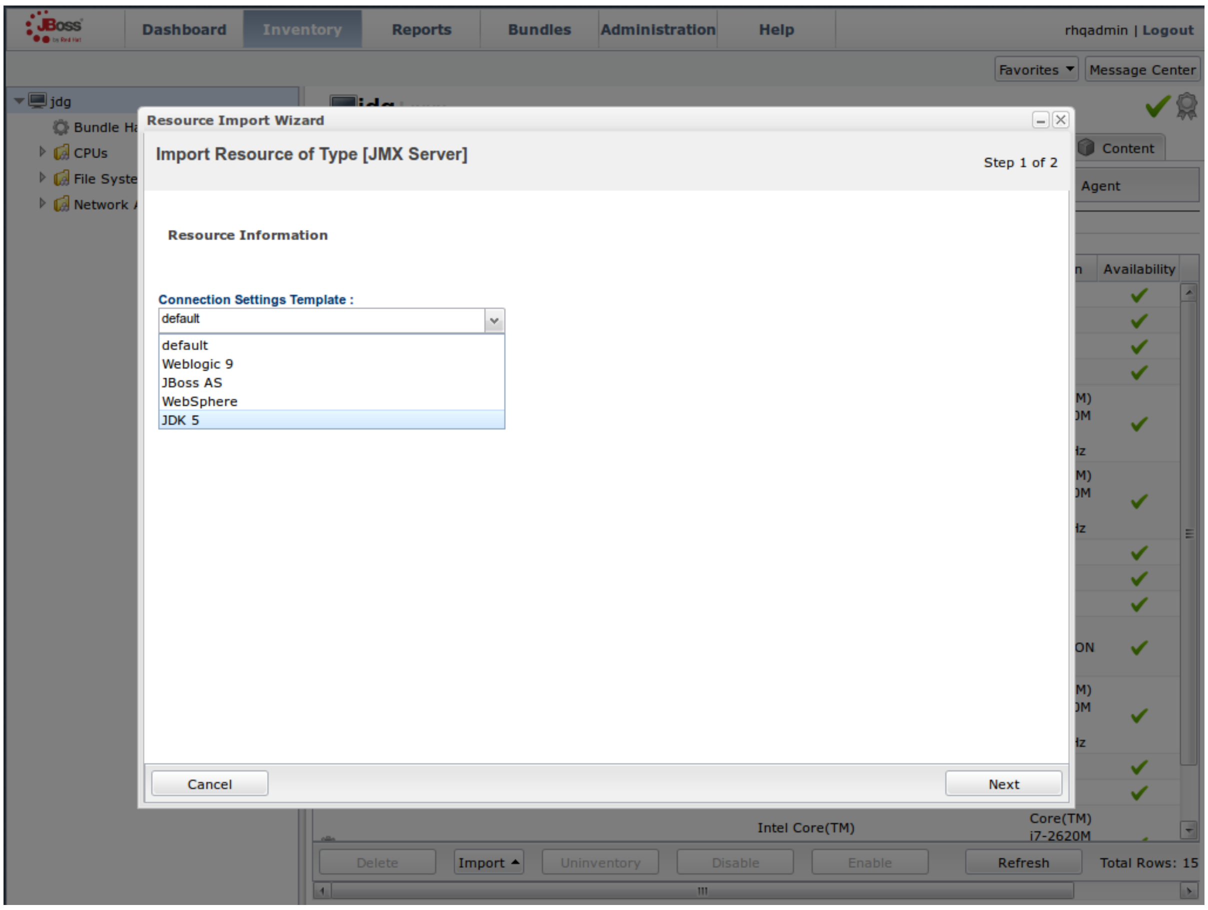


Name	Ancestry	Description	Type	Version	Availability
/run/user	jdg	none: /run/user	File System		✓
eth0	jdg	F0:DE:F1:7B:E9::	Network Adapter		✓
/run/lock	jdg	none: /run/lock	File System		✓
/run/shm	jdg	none: /run/shm	File System		✓
CPU 2	jdg	Intel Core(TM) i7-2620M CPU @ 2.70GHz	CPU	Core(TM) i7-2620M CPU @ 2.70GHz	✓
CPU 1	jdg	Intel Core(TM) i7-2620M CPU @ 2.70GHz	CPU	Core(TM) i7-2620M CPU @ 2.70GHz	✓
/run	jdg	tmpfs: /run	File System		✓
lo	jdg	00:00:00:00:00:(	Network Adapter		✓
tun0	jdg	00:00:00:00:00:(	Network Adapter		✓
Bundle Handler - Ant	jdg	For provisioning bundles with Ant script recipes	Ant Bundle Handler	4.4.0.JON	✓
CPU 0	jdg	Intel Core(TM) i7-2620M CPU @ 2.70GHz	CPU	Core(TM) i7-2620M CPU @ 2.70GHz	✓
/	lg	/dev/sdb1: /	File System		✓
/data	lg	/dev/sda1: /data	File System		✓
		Intel Core(TM)		Core(TM) i7-2620M	

Figure 20.6. Import the JMX Server

7. **Enable JDK Connection Settings**

   a. In the **Resource Import Wizard** window, specify **JDK 5** from the list of **Connection Settings Template** options.



Import Resource of Type [JMX Server]

Step 1 of 2

Resource Information

Connection Settings Template : default

default
Weblogic 9
JBoss AS
WebSphere
JDK 5

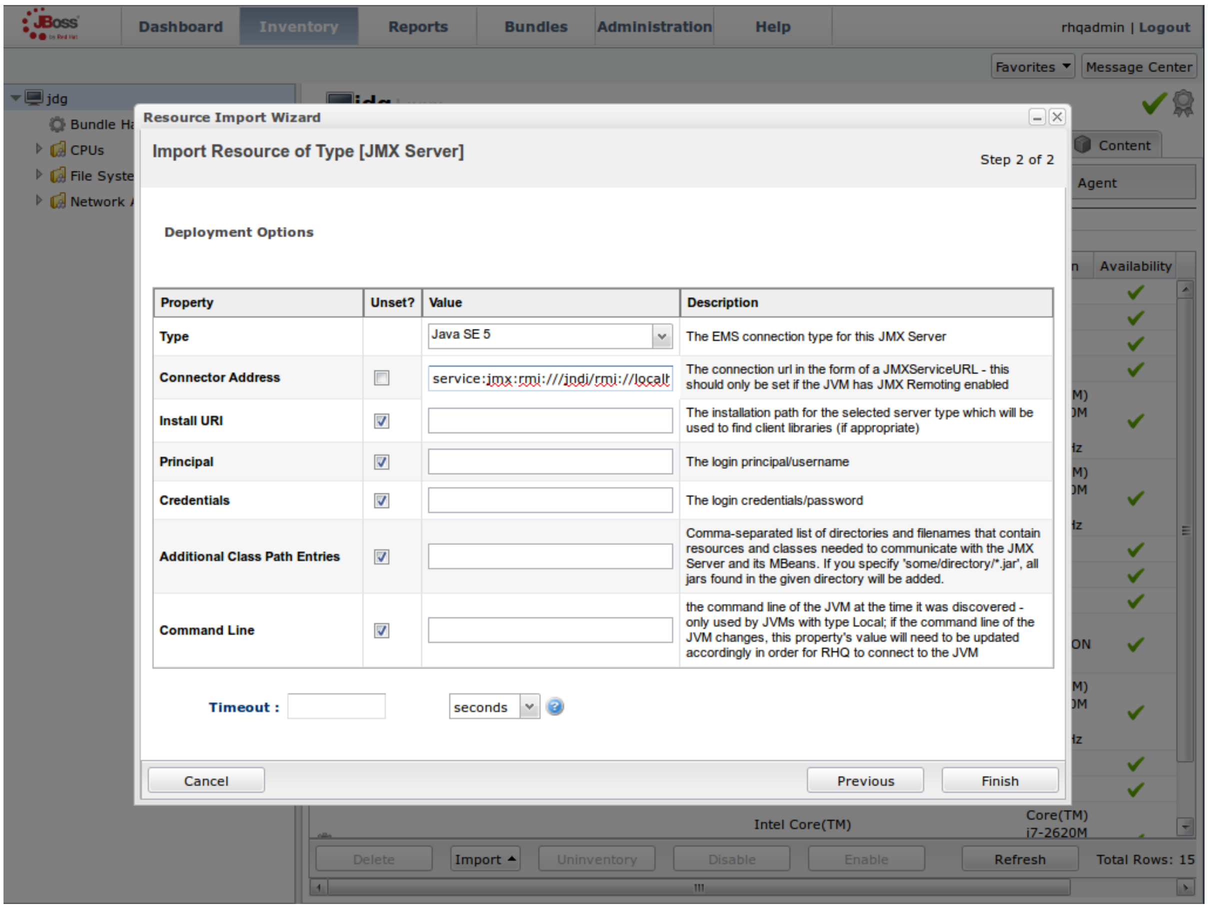
Figure 20.7. Select the JDK 5 Template.

8. **Modify the Connector Address**

   a. In the **Deployment Options** menu, modify the supplied **Connector Address** with the hostname and JMX port of the process containing the Infinispan Library.

   b. Specify the **Principal** and **Credentials** information if required.

   c. Click **`Finish`**.



Import Resource of Type [JMX Server]

Step 2 of 2

Deployment Options

Property	Unset?	Value	Description
Type		Java SE 5	The EMS connection type for this JMX Server
Connector Address	☐	service:jmx:rmi:///jndi/rmi://localt	The connection url in the form of a JMXServiceURL - this should only be set if the JVM has JMX Remoting enabled
Install URI	☑		The installation path for the selected server type which will be used to find client libraries (if appropriate)
Principal	☑		The login principal/username
Credentials	☑		The login credentials/password
Additional Class Path Entries	☑		Comma-separated list of directories and filenames that contain resources and classes needed to communicate with the JMX Server and its MBeans. If you specify 'some/directory/*.jar', all jars found in the given directory will be added.
Command Line	☑		the command line of the JVM at the time it was discovered - only used by JVMs with type Local; if the command line of the JVM changes, this property's value will need to be updated accordingly in order for RHQ to connect to the JVM

Timeout : seconds

Figure 20.8. Modify the values in the Deployment Options screen.

9. **View Cache Statistics and Operations**

   a. Click **`Refresh`** to refresh the list of servers.

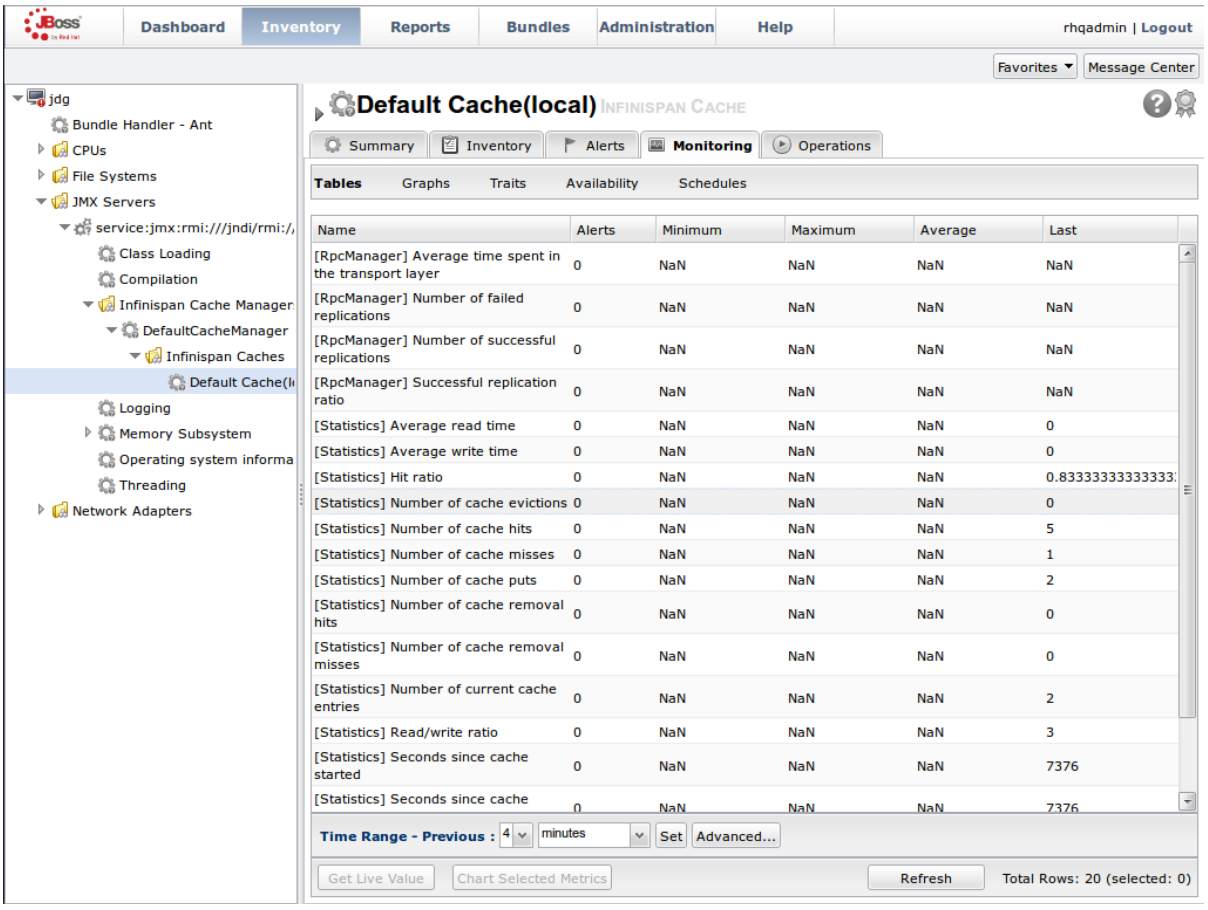
   b. The **JMX Servers** tree in the panel on the left side of the screen contains the **Infinispan Cache Managers** node, which contains the available cache managers. The available cache managers contain the available caches.

   c. Select a cache from the available caches to view metrics.

   d. Select the **Monitoring** tab.

   e. The **`Tables`** view shows statistics and metrics.

   f. The **Operations** tab provides access to the various operations that can be performed on the services.



Name	Alerts	Minimum	Maximum	Average	Last
[RpcManager] Average time spent in the transport layer	0	NaN	NaN	NaN	NaN
[RpcManager] Number of failed replications	0	NaN	NaN	NaN	NaN
[RpcManager] Number of successful replications	0	NaN	NaN	NaN	NaN
[RpcManager] Successful replication ratio	0	NaN	NaN	NaN	NaN
[Statistics] Average read time	0	NaN	NaN	NaN	0
[Statistics] Average write time	0	NaN	NaN	NaN	0
[Statistics] Hit ratio	0	NaN	NaN	NaN	0.83333333333333
[Statistics] Number of cache evictions	0	NaN	NaN	NaN	0
[Statistics] Number of cache hits	0	NaN	NaN	NaN	5
[Statistics] Number of cache misses	0	NaN	NaN	NaN	1
[Statistics] Number of cache puts	0	NaN	NaN	NaN	2
[Statistics] Number of cache removal hits	0	NaN	NaN	NaN	0
[Statistics] Number of cache removal misses	0	NaN	NaN	NaN	0
[Statistics] Number of current cache entries	0	NaN	NaN	NaN	2
[Statistics] Read/write ratio	0	NaN	NaN	NaN	3
[Statistics] Seconds since cache started	0	NaN	NaN	NaN	7376
[Statistics] Seconds since cache	0	NaN	NaN	NaN	7376

Figure 20.9. Metrics and operational data relayed through JMX is now available in the JBoss Operations Network console.

Report a bug

20.6.2. Manually Adding JBoss Data Grid Instances in Library Mode

To add Red Hat JBoss Data Grid instances to JBoss Operations Network manually, use the following procedure in the JBoss Operations Network interface.

1. Select **Resources** > **Platforms** > **localhost** > **Inventory**.
2. At the bottom of the page, open the drop-down menu next to the **Manually Add** section.
3. Select **Infinispan Cache Manager** and click **Ok**.
4. Select the **`default`** template on the next page.
5. **Manually add the JBoss Data Grid instance**
   a. Enter both the JMX connector address of the new JBoss Data Grid instance you want to monitor, and the Cache Manager Mbean object name. For example:

      Connector Address:

      ```
      service:jmx:rmi://127.0.0.1/jndi/rmi://127.0.0.1:7997/jmxrmi
      ```

   b. Object Name:

      ```
      org.infinispan:type=CacheManager,name="<name_of_cache_manager>
      ```

NOTE

The object name remains the same, however the connector address varies depending on the host and the JMX port assigned to the new instance. In this case, instances require the following system properties at start up:

```
-Dcom.sun.management.jmxremote.port=7997 -
Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
```

Report a bug

20.7. JBOSS OPERATIONS NETWORK REMOTE-CLIENT SERVER PLUGIN

20.7.1. JBoss Operations Network Plugin Metrics

Table 20.1. JBoss Operations Network Traits for the Cache Container (Cache Manager)

Metric Name	Display Name	Description
cache-manager-status	Cache Container Status	The current runtime status of a cache container.
cluster-name	Cluster Name	The name of the cluster.
coordinator-address	Coordinator Address	The coordinator node's address.
local-address	Local Address	The local node's address.

Table 20.2. JBoss Operations Network Metrics for the Cache

Metric Name	Display Name	Description
cache-status	Cache Status	The current runtime status of a cache.
number-of-locks-available	[LockManager] Number of locks available	The number of exclusive locks that are currently available.
concurrency-level	[LockManager] Concurrency level	The LockManager's configured concurrency level.
average-read-time	[Statistics] Average read time	Average number of milliseconds required for a read operation on the cache to complete.

Metric Name	Display Name	Description
hit-ratio	[Statistics] Hit ratio	The result (in percentage) when the number of hits (successful attempts) is divided by the total number of attempts.
elapsed-time	[Statistics] Seconds since cache started	The number of seconds since the cache started.
read-write-ratio	[Statistics] Read/write ratio	The read/write ratio (in percentage) for the cache.
average-write-time	[Statistics] Average write time	Average number of milliseconds a write operation on a cache requires to complete.
hits	[Statistics] Number of cache hits	Number of cache hits.
evictions	[Statistics] Number of cache evictions	Number of cache eviction operations.
remove-misses	[Statistics] Number of cache removal misses	Number of cache removals where the key was not found.
time-since-reset	[Statistics] Seconds since cache statistics were reset	Number of seconds since the last cache statistics reset.
number-of-entries	[Statistics] Number of current cache entries	Number of entries currently in the cache.
stores	[Statistics] Number of cache puts	Number of cache put operations
remove-hits	[Statistics] Number of cache removal hits	Number of cache removal operation hits.
misses	[Statistics] Number of cache misses	Number of cache misses.
success-ratio	[RpcManager] Successful replication ratio	Successful replications as a ratio of total replications in numeric double format.
replication-count	[RpcManager] Number of successful replications	Number of successful replications
replication-failures	[RpcManager] Number of failed replications	Number of failed replications

Metric Name	Display Name	Description
average-replication-time	[RpcManager] Average time spent in the transport layer	The average time (in milliseconds) spent in the transport layer.
commits	[Transactions] Commits	Number of transaction commits performed since the last reset.
prepares	[Transactions] Prepares	Number of transaction prepares performed since the last reset.
rollbacks	[Transactions] Rollbacks	Number of transaction rollbacks performed since the last reset.
invalidations	[Invalidation] Number of invalidations	Number of invalidations.
passivations	[Passivation] Number of cache passivations	Number of passivation events.
activations	[Activations] Number of cache entries activated	Number of activation events.
cache-loader-loads	[Activation] Number of cache store loads	Number of entries loaded from the cache store.
cache-loader-misses	[Activation] Number of cache store misses	Number of entries that did not exist in the cache store.
cache-loader-stores	[CacheStore] Number of cache store stores	Number of entries stored in the cache stores.

NOTE

Gathering of some of these statistics is disabled by default.

JBoss Operations Network Metrics for Connectors

The metrics provided by the JBoss Operations Network (JON) plugin for Red Hat JBoss Data Grid are for REST and Hot Rod endpoints only. For the REST protocol, the data must be taken from the Web subsystem metrics. For details about each of these endpoints, see the *Getting Started Guide*.

Table 20.3. JBoss Operations Network Metrics for the Connectors

Metric Name	Display Name	Description
bytesRead	Bytes Read	Number of bytes read.

Metric Name	Display Name	Description
bytesWritten	Bytes Written	Number of bytes written.

NOTE

Gathering of these statistics is disabled by default.

Report a bug

20.7.2. JBoss Operations Network Plugin Operations

Table 20.4. JBoss ON Plugin Operations for the Cache

Operation Name	Description
Clear Cache	Clears the cache contents.
Reset Statistics	Resets statistics gathered by the cache.
Reset Activation Statistics	Resets activation statistics gathered by the cache.
Reset Invalidation Statistics	Resets invalidations statistics gathered by the cache.
Reset Passivation Statistics	Resets passivation statistics gathered by the cache.
Reset Rpc Statistics	Resets replication statistics gathered by the cache.
Remove Cache	Removes the given cache from the cache-container.

JBoss Operations Network Plugin Operations for the Cache Backups

The cache backups used for these operations are configured using cross-datacenter replication. In the JBoss Operations Network (JON) User Interface, each cache backup is the child of a cache. For more information about cross-datacenter replication, see Section 27.1, “About Cross-Datacenter Replication”

Table 20.5. JBoss Operations Network Plugin Operations for the Cache Backups

Operation Name	Description
status	Display the site status.
bring-site-online	Brings the site online.
take-site-offline	Takes the site offline.

Cache (Transactions)

Red Hat JBoss Data Grid does not support using Transactions in Remote Client-Server mode. As a result, none of the endpoints can use transactions.

Report a bug

20.7.3. JBoss Operations Network Plugin Attributes

Table 20.6. JBoss ON Plugin Attributes for the Cache (Transport)

Attribute Name	Type	Description
cluster	string	The name of the group communication cluster.
executor	string	The executor used for the transport.
lock-timeout	long	The timeout period for locks on the transport. The default value is **`240000`**.
machine	string	A machine identifier for the transport.
rack	string	A rack identifier for the transport.
site	string	A site identifier for the transport.
stack	string	The JGroups stack used for the transport.

Report a bug

20.8. MONITOR JBOSS ENTERPRISE APPLICATION PLATFORM 6 APPLICATIONS USING LIBRARY MODE

20.8.1. Prerequisites

The following is a list of common prerequisites for both Section 20.8.2, “Monitor an Application Deployed in Standalone Mode” and Section 20.8.3, “Monitor an Application Deployed in Domain Mode”

- A correctly configured instance of JBoss Operations Network (JON) 3.1.x or better.
- A running instance of JBoss Operations Network (JON) Agent on the server where the application will run. For more information, see Section 20.4, “JBoss Operations Network Agent”
- An operational instance of the RHQ agent with a full JDK. Ensure that the agent has access to the **`tools.jar`** file from the JDK in particular. In the JBoss Operations Network (JON) agent's environment file (**`bin/rhq-env.sh`**), set the value of the **`RHQ_AGENT_JAVA_HOME`** property to a full JDK.

- The RHQ agent must have been initiated using the same user as the JBoss Enterprise Application Platform instance. As an example, running the JBoss Operations Network (JON) agent as a user with root privileges and the JBoss Enterprise Application Platform process under a different user does not work as expected and must be avoided.

- An installed JBoss Operations Network (JON) plugin for Library Mode. For more information, see Section 20.6.1, “Installing the JBoss Operations Network Plug-in (Library Mode)”

- A custom application using Red Hat JBoss Data Grid's Library mode. This application must have **`jmxStatistics`** enabled (either declaratively or programmatically). For more information, see Section 19.4, “Enable JMX for Cache Instances”

- The Java Virtual Machine (JVM) must be configured to expose the JMX MBean Server. For the Oracle/Sun JDK, see http://docs.oracle.com/javase/1.5.0/docs/guide/management/agent.html

- A correctly added and configured management user for JBoss Enterprise Application Platform.

Report a bug

20.8.2. Monitor an Application Deployed in Standalone Mode

Use the following instructions to monitor an application deployed in JBoss Enterprise Application Platform 6 using its standalone mode:

Procedure 20.4. Monitor an Application Deployed in Standalone Mode

1. **Start the JBoss Enterprise Application Platform Instance**
   Start the JBoss Enterprise Application Platform instance as follows:

   a. Enter the following command at the command line to add a new option to the standalone configuration file (**`/bin/standalone.conf`**):

   ```
   JAVA_OPTS="$JAVA_OPTS -Dorg.rhq.resourceKey=MyEAP"
   ```

   b. Start the JBoss Enterprise Application Platform instance in standalone mode as follows:

   ```
   $JBOSS_HOME/bin/standalone.sh
   ```

2. **Run JBoss Operations Network (JON) Discovery**
   Run the **`discovery --full`** command in the JBoss Operations Network (JON) agent.

3. **Locate Application Server Process**
   In the JBoss Operations Network (JON) web interface, the JBoss Enterprise Application Platform 6 process is listed as a JMX server.

4. **Import the Process Into Inventory**
   Import the process into the JBoss Operations Network (JON) inventory.

5. **Deploy the Red Hat JBoss Data Grid Application**
   Deploy the WAR file that contains the JBoss Data Grid Library mode application with **`globalJmxStatistics`** and **`jmxStatistics`** enabled.

6. **Optional: Run Discovery Again**
   If required, run the **discovery --full** command again to discover the new resources.

Result

The JBoss Data Grid Library mode application is now deployed in JBoss Enterprise Application Platform's standalone mode and can be monitored using the JBoss Operations Network (JON).

Report a bug

20.8.3. Monitor an Application Deployed in Domain Mode

Use the following instructions to monitor an application deployed in JBoss Enterprise Application Platform 6 using its domain mode:

Procedure 20.5. Monitor an Application Deployed in Domain Mode

1. **Edit the Host Configuration**
   Edit the **domain/configuration/host.xml** file to replace the **server** element with the following configuration:

```
<servers>
 <server name="server-one" group="main-server-group">
  <jvm name="default">
   <jvm-options>
    <option value="-Dorg.rhq.resourceKey=EAP1"/>
   </jvm-options>
  </jvm>
 </server>
 <server name="server-two" group="main-server-group" auto-
start="true">
  <socket-bindings port-offset="150"/>
  <jvm name="default">
   <jvm-options>
    <option value="-Dorg.rhq.resourceKey=EAP2"/>
   </jvm-options>
  </jvm>
 </server>
</servers>
```

2. **Start JBoss Enterprise Application Platform 6**
   Start JBoss Enterprise Application Platform 6 in domain mode:

```
$JBOSS_HOME/bin/domain.sh
```

3. **Deploy the Red Hat JBoss Data Grid Application**
   Deploy the WAR file that contains the JBoss Data Grid Library mode application with **globalJmxStatistics** and **jmxStatistics** enabled.

4. **Run Discovery in JBoss Operations Network (JON)**
   If required, run the **discovery --full** command for the JBoss Operations Network (JON) agent to discover the new resources.

Result

The JBoss Data Grid Library mode application is now deployed in JBoss Enterprise Application Platform's domain mode and can be monitored using the JBoss Operations Network (JON).

Report a bug

20.9. JBOSS OPERATIONS NETWORK PLUG-IN QUICKSTART

For testing or demonstrative purposes with a single JBoss Operations Network agent, upload the plug-in to the server then type "plugins update" at the agent command line to force a retrieval of the latest plugins from the server.

Report a bug

20.10. OTHER MANAGEMENT TOOLS AND OPERATIONS

Managing Red Hat JBoss Data Grid instances requires exposing significant amounts of relevant statistical information. This information allows administrators to get a clear view of each JBoss Data Grid node's state. A single installation can comprise of tens or hundreds of JBoss Data Grid nodes and it is important to provide this information in a clear and concise manner. JBoss Operations Network is one example of a tool that provides runtime visibility. Other tools, such as **JConsole** can be used where JMX is enabled.

Report a bug

20.10.1. Accessing Data via URLs

Caches that have been configured with a REST interface have access to Red Hat JBoss Data Grid using RESTful HTTP access.

The RESTful service only requires a HTTP client library, eliminating the need for tightly coupled client libraries and bindings.

HTTP **`put()`** and **`post()`** methods place data in the cache, and the URL used determines the cache name and key(s) used. The data is the value placed into the cache, and is placed in the body of the request.

A Content-Type header must be set for these methods. **`GET`** and **`HEAD`** methods are used for data retrieval while other headers control cache settings and behavior.

NOTE

It is not possible to have conflicting server modules interact with the data grid. Caches must be configured with a compatible interface in order to have access to JBoss Data Grid.

Report a bug

20.10.2. Limitations of Map Methods

Specific **`Map`** methods, such as **`size()`**, **`values()`**, **`keySet()`** and **`entrySet()`**, can be used with certain limitations with Red Hat JBoss Data Grid as they are unreliable. These methods do not acquire locks (global or local) and concurrent modification, additions and removals are excluded from consideration in these calls. Furthermore, the listed methods are only operational on the local data container and do not provide a global view of state.

If the listed methods acted globally, it would result in a significant impact on performance and would produce a scalability bottleneck. As a result, it is recommended that these methods are used for informational and debugging purposes only.

Report a bug

PART X. COMMAND LINE TOOLS

Red Hat JBoss Data Grid includes command line tools for interacting with the caches in the data grid. The first of these tools is the Infinispan Command Line tool.

Report a bug

CHAPTER 21. RED HAT JBOSS DATA GRID CLI

Red Hat JBoss Data Grid includes the Red Hat JBoss Data Grid Command Line Interface (CLI) that is used to inspect and modify data within caches and internal components (such as transactions, cross-datacenter replication sites, and rolling upgrades). The JBoss Data Grid CLI can also be used for more advanced operations such as transactions.

The CLI consists of a server-side module and a client command tool. The server-side module (`infinispan-cli-server-$VERSION.jar`) includes an interpreter for commands and must be included in the application.

Report a bug

21.1. START THE CLI (SERVER)

Start the Red Hat JBoss Data Grid CLI's server-side module with the `standalone` and `cluster` files. For Linux, use the `standlaone.sh` or `clustered.sh` script and for Windows, use the `standalone.bat` or `clustered.bat` file.

Report a bug

21.2. START THE CLI (CLIENT)

Start the Red Hat JBoss Data Grid CLI client using the `ispn-cli` file at `bin/`. For Linux, run `bin/ispn-cli.sh` and for Windows, run `bin/ispn-cli.bat`.

When starting up the CLI client, specific command line switches can be used to customize the start up.

Report a bug

21.3. CLI CLIENT SWITCHES FOR THE COMMAND LINE

The listed command line switches are appended to the command line when starting the Red Hat JBoss Data Grid CLI command:

Table 21.1. CLI Client Command Line Switches

Short Option	Long Option	Description
-c	--connect=${URL}	Connects to a running Red Hat JBoss Data Grid instance. For example, for JMX over RMI use `jmx://[username[:password]]@host:port[/container[/cache]]` and for JMX over JBoss Remoting use `remoting://[username[:password]]@host:port[/container[/cache]]`

Short Option	Long Option	Description
-f	--file=${FILE}	Read the input from the specified file rather than using interactive mode. If the value is set to `-` then the ***stdin*** is used as the input.
-h	--help	Displays the help information.
-v	--version	Displays the CLI version information.

Report a bug

21.4. CONNECT TO THE APPLICATION

Use the following command to connect to the application using the CLI:

```
[disconnected//]> connect jmx://localhost:12000
[jmx://localhost:12000/MyCacheManager/>
```

NOTE

The port value **12000** depends on the value the JVM is started with. For example, starting the JVM with the **-Dcom.sun.management.jmxremote.port=12000** command line parameter uses this port, but otherwise a random port is chosen. When the remoting protocol (**remoting://localhost:9999**) is used, the Red Hat JBoss Data Grid server administration port is used (the default is port **9999**).

The command line prompt displays the active connection information, including the currently selected **CacheManager**.

Use the **cache** command to select a cache before performing cache operations. The CLI supports tab completion, therefore using the **cache** and pressing the tab button displays a list of active caches:

```
[[jmx://localhost:12000/MyCacheManager/> cache
___defaultcache  namedCache
[jmx://localhost:12000/MyCacheManager/]> cache ___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]>
```

Additionally, pressing tab displays a list of valid commands for the CLI.

Report a bug

21.5. STOPPING A RED HAT JBOSS DATA GRID INSTANCE WITH THE CLI

Library Mode

When running a Red Hat JBoss Data Grid instance in a container as a library mode deployment, the life cycle of JBoss Data Grid is bound to the life cycle of the user deployment.

Stop or undeploy the user deployment using you container's management interfaces. For example, in Red Hat JBoss Enterprise Application Platform, use the Management CLI. See the *Management Interfaces* chapter in the JBoss Enterprise Application Platform *Administration and Configuration Guide* for more information.

Remote Client-Server Mode

A remote client-server JBoss Data Grid instance can be stopped using the following script:

```
jboss-datagrid-6.2.0-server/bin/init.d/jboss-datagrid.sh stop
```

Alternatively, you can use the **kill** commands directly:

```
kill -15 $pid # send the TERM signal
```

If after a period of time the **PID** is still there, use the following:

```
kill -9 $pid
```

Report a bug

21.6. CLI COMMANDS

21.6.1. The abort Command

The **abort** command aborts a running batch initiated using the **start** command. Batching must be enabled for the specified cache. The following is a usage example:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> abort
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
null
```

Report a bug

21.6.2. The begin Command

The **begin** command starts a transaction. This command requires transactions enabled for the cache it targets. An example of this command's usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> commit
```

Report a bug

21.6.3. The cache Command

The **cache** command specifies the default cache used for all subsequent operations. If invoked without any parameters, it shows the currently selected cache. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> cache ___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]> cache
___defaultcache
[jmx://localhost:12000/MyCacheManager/___defaultcache]>
```

Report a bug

21.6.4. The clear Command

The **clear** command clears all content from the cache. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> clear
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
null
```

Report a bug

21.6.5. The commit Command

The **commit** command commits changes to an ongoing transaction. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> commit
```

Report a bug

21.6.6. The container Command

The **container** command selects the default cache container (cache manager). When invoked without any parameters, it lists all available containers. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> container
MyCacheManager OtherCacheManager
[jmx://localhost:12000/MyCacheManager/namedCache]> container
OtherCacheManager
[jmx://localhost:12000/OtherCacheManager/]>
```

Report a bug

21.6.7. The create Command

The **create** command creates a new cache based on the configuration of an existing cache definition. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> create newCache like
namedCache
```

```
[jmx://localhost:12000/MyCacheManager/namedCache]> cache newCache
[jmx://localhost:12000/MyCacheManager/newCache]>
```

Report a bug

21.6.8. The disconnect Command

The **`disconnect`** command disconnects the currently active connection, which allows the CLI to connect to another instance. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> disconnect
[disconnected//]
```

Report a bug

21.6.9. The encoding Command

The **`encoding`** command sets a default codec to use when reading and writing entries to and from a cache. If invoked with no arguments, the currently selected codec is displayed. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding
none
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding --list
memcached
hotrod
none
rest
[jmx://localhost:12000/MyCacheManager/namedCache]> encoding hotrod
```

Report a bug

21.6.10. The end Command

The **end** command ends a running batch initiated using the **`start`** command. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> end
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
a
```

Report a bug

21.6.11. The evict Command

The **`evict`** command evicts an entry associated with a specific key from the cache. An example of it usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> evict a
```

Report a bug

21.6.12. The get Command

The **`get`** command shows the value associated with a specified key. For primitive types and Strings, the **`get`** command prints the default representation. For other objects, a JSON representation of the object is printed. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
a
```

Report a bug

21.6.13. The info Command

The **`info`** command displaysthe configuration of a selected cache or container. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> info
GlobalConfiguration{asyncListenerExecutor=ExecutorFactoryConfiguration{fac
tory=org.infinispan.executors.DefaultExecutorFactory@98add58},
asyncTransportExecutor=ExecutorFactoryConfiguration{factory=org.infinispan
.executors.DefaultExecutorFactory@7bc9c14c},
evictionScheduledExecutor=ScheduledExecutorFactoryConfiguration{factory=or
g.infinispan.executors.DefaultScheduledExecutorFactory@7ab1a411},
replicationQueueScheduledExecutor=ScheduledExecutorFactoryConfiguration{fa
ctory=org.infinispan.executors.DefaultScheduledExecutorFactory@248a9705},
globalJmxStatistics=GlobalJmxStatisticsConfiguration{allowDuplicateDomains
=true, enabled=true, jmxDomain='jboss.infinispan',
mBeanServerLookup=org.jboss.as.clustering.infinispan.MBeanServerProvider@6
c0dc01, cacheManagerName='local', properties={}},
transport=TransportConfiguration{clusterName='ISPN', machineId='null',
rackId='null', siteId='null', strictPeerToPeer=false,
distributedSyncTimeout=240000, transport=null, nodeName='null',
properties={}},
serialization=SerializationConfiguration{advancedExternalizers=
{1100=org.infinispan.server.core.CacheValue$Externalizer@5fabc91d,
1101=org.infinispan.server.memcached.MemcachedValue$Externalizer@720bffd,
1104=org.infinispan.server.hotrod.ServerAddress$Externalizer@771c7eb2},
marshaller=org.infinispan.marshall.VersionAwareMarshaller@6fc21535,
version=52,
classResolver=org.jboss.marshalling.ModularClassResolver@2efe83e5},
shutdown=ShutdownConfiguration{hookBehavior=DONT_REGISTER}, modules={},
site=SiteConfiguration{localSite='null'}}
```

Report a bug

21.6.14. The locate Command

The **`locate`** command displays the physical location of a specified entry in a distributed cluster. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> locate a
[host/node1,host/node2]
```

Report a bug

21.6.15. The put Command

The **put** command inserts an entry into the cache. If a mapping exists for a key, the **put** command overwrites the old value. The CLI allows control over the type of data used to store the key and value. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b 100
[jmx://localhost:12000/MyCacheManager/namedCache]> put c 4139l
[jmx://localhost:12000/MyCacheManager/namedCache]> put d true
[jmx://localhost:12000/MyCacheManager/namedCache]> put e {
"package.MyClass": {"i": 5, "x": null, "b": true } }
```

Optionally, the **put** can specify a life span and maximum idle time value as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a expires 10s
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a expires 10m
maxidle 1m
```

Report a bug

21.6.16. The replace Command

The **replace** command replaces an existing entry in the cache with a specified new value. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
b
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b c
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
c
[jmx://localhost:12000/MyCacheManager/namedCache]> replace a b d
[jmx://localhost:12000/MyCacheManager/namedCache]> get a
c
```

Report a bug

21.6.17. The rollback Command

The **rollback** command rolls back any changes made by an ongoing transaction. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> begin
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> rollback
```

–

Report a bug

21.6.18. The site Command

The **site** command performs administration tasks related to cross-datacenter replication. This command also retrieves information about the status of a site and toggles the status of a site. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --status NYC
online
[jmx://localhost:12000/MyCacheManager/namedCache]> site --offline NYC
ok
[jmx://localhost:12000/MyCacheManager/namedCache]> site --status NYC
offline
[jmx://localhost:12000/MyCacheManager/namedCache]> site --online NYC
```

Report a bug

21.6.19. The start Command

The **start** command initiates a batch of operations. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> start
[jmx://localhost:12000/MyCacheManager/namedCache]> put a a
[jmx://localhost:12000/MyCacheManager/namedCache]> put b b
[jmx://localhost:12000/MyCacheManager/namedCache]> end
```

Report a bug

21.6.20. The stats Command

The **stats** command displays statistics for the cache. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> stats
Statistics: {
  averageWriteTime: 143
  evictions: 10
  misses: 5
  hitRatio: 1.0
  readWriteRatio: 10.0
  removeMisses: 0
  timeSinceReset: 2123
  statisticsEnabled: true
  stores: 100
  elapsedTime: 93
  averageReadTime: 14
  removeHits: 0
  numberOfEntries: 100
  hits: 1000
}
LockManager: {
  concurrencyLevel: 1000
```

```
    numberOfLocksAvailable: 0
    numberOfLocksHeld: 0
  }
```

Report a bug

21.6.21. The upgrade Command

The **upgrade** command implements the rolling upgrade procedure. For details about rolling upgrades, see the *Rolling Upgrades* chapter in the *Red Hat JBoss Data Grid Developer Guide*

An example of the **upgrade** command's use is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> upgrade --
synchronize=hotrod --all
[jmx://localhost:12000/MyCacheManager/namedCache]> upgrade --
disconnectsource=hotrod --all
```

Report a bug

21.6.22. The version Command

The **version** command displays version information for the CLI client and server. An example of its usage is as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> version
Client Version 5.2.1.Final
Server Version 5.2.1.Final
```

Report a bug

PART XI. OTHER RED HAT JBOSS DATA GRID FUNCTIONS

CHAPTER 22. SET UP THE L1 CACHE

22.1. ABOUT THE L1 CACHE

The Level 1 (or L1) cache stores remote cache entries after they are initially accessed, preventing unnecessary remote fetch operations for each subsequent use of the same entries. The L1 cache is only available when Red Hat JBoss Data Grid's cache mode is set to distribution. In other cache modes any configuration related to the L1 cache is ignored.

When caches are configured with distributed mode, the entries are evenly distributed between all clustered caches. Each entry is copied to a desired number of owners, which can be less than the total number of caches. As a result, the system's scalability is improved but also means that some entries are not available on all nodes and must be fetched from their owner node. In this situation, configure the Cache component to use the L1 Cache to temporarily store entries that it does not own to prevent repeated fetching for subsequent uses.

Each time a key is updated an invalidation message is generated. This message is multicast to each node that contains data that corresponds to current L1 cache entries. The invalidation message ensures that each of these nodes marks the relevant entry as invalidated. Also, when the location of an entry changes in the cluster, the corresponding L1 cache entry is invalidated to prevent outdated cache entries.

Report a bug

22.2. L1 CACHE CONFIGURATION

22.2.1. L1 Cache Configuration (Library Mode)

The following sample configuration shows the L1 cache default values in Red Hat JBoss Data Grid's Library Mode.

```
<clustering mode="distribution">
 <sync/>
 <l1 enabled="true"
           lifespan="60000" />
</clustering>
```

The **l1** element configures the cache behavior in distributed cache instances. If used with non-distributed caches, this element is ignored.

- The ***enabled*** parameter enables the L1 cache.
- The ***lifespan*** parameter sets the maximum life span of an entry when it is placed in the L1 cache.

Report a bug

22.2.2. L1 Cache Configuration (Remote Client-Server Mode)

The following sample configuration shows the L1 cache default values in Red Hat JBoss Data Grid's Remote Client-Server mode.

```
<distributed-cache l1-lifespan="${VALUE}">
 ...
</distributed-cache>
```

The **`l1-lifespan`** element is added to a **`distributed-cache`** element to enable L1 caching and to set the life span of the L1 cache entries for the cache. This element is only valid for distributed caches.

If **`l1-lifespan`** is set to **`0`** or a negative number (**`-1`**), L1 caching is disabled. L1 caching is enabled when the **`l1-lifespan`** value is greater than **`0`**.

NOTE

When the cache is accessed remotely via the Hot Rod protocol, the client accesses the owner node directly. Therefore, using L1 Cache in this situation does not offer any performance improvement and is not recommended. Other remote clients (Memcached, REST) cannot target the owner, therefore, using L1 Cache may increase the performance (at the cost of higher memory consumption).

Report a bug

CHAPTER 23. SET UP TRANSACTIONS

23.1. ABOUT TRANSACTIONS

A transaction consists of a collection of interdependent or related operations or tasks. All operations within a single transaction must succeed for the overall success of the transaction. If any operations within a transaction fail, the transaction as a whole fails and rolls back any changes. Transactions are particularly useful when dealing with a series of changes as part of a larger operation.

In Red Hat JBoss Data Grid, transactions are only available in Library mode.

Report a bug

23.1.1. About the Transaction Manager

In Red Hat JBoss Data Grid, the Transaction Manager coordinates transactions across a single or multiple resources. The responsibilities of a Transaction Manager include:

- initiating and concluding transactions
- managing information about each transaction
- coordinating transactions as they operate over multiple resources
- recovering from a failed transaction by rolling back changes

Report a bug

23.1.2. XA Resources and Synchronizations

XA Resources are fully fledged transaction participants. In the prepare phase, the XA Resource returns a vote with either the value `OK` or `ABORT`. If the Transaction Manager receives `OK` votes from all XA Resources, the transaction is committed, otherwise it is rolled back.

Synchronizations are a type of listener that receive notifications about events leading to the transaction life cycle. Synchronizations receive an event before and after the operation completes.

Unless recovery is required, it is not necessary to register as a full XA resource. An advantage of synchronizations is that they allow the Transaction Manager to optimize 2PC (Two Phase Commit) with a 1PC (One Phase Commit) where only one other resource is enlisted with that transaction (last resource commit optimization). This makes registering a synchronization more efficient.

However, if the operation fails in the prepare phase within Red Hat JBoss Data Grid, the transaction is not rolled back and if there are more participants in the transaction, they can ignore this failure and commit. Additionally, errors encountered in the commit phase are not propagated to the application code that commits the transaction.

By default JBoss Data Grid registers to the transaction as a synchronization.

Report a bug

23.1.3. Optimistic and Pessimistic Transactions

Pessimistic transactions acquire the locks when the first write operation on the key executes. After the key is locked, no other transaction can modify the key until this transaction is committed or rolled back. It is up to the application code to acquire the locks in correct order to prevent deadlocks.

With optimistic transactions locks are acquired at transaction prepare time and are held until the transaction commits (or rolls back). Also, Red Hat JBoss Data Grid sorts keys for all entries modified within a transaction automatically, preventing any deadlocks occurring due to the incorrect order of keys being locked. This results in:

- less messages being sent during the transaction execution
- locks held for shorter periods
- improved throughput

NOTE

Read operations never acquire any locks. Acquiring the lock for a read operation on demand is possible only with pessimistic transactions, using the **`FORCE_WRITE_LOCK`** flag with the operation.

Report a bug

23.1.4. Write Skew Checks

A common use case for entries is that they are read and subsequently written in a transaction. However, a third transaction can modify the entry between these two operations. In order to detect such a situation and roll back a transaction Red Hat JBoss Data Grid offers entry versioning and write skew checks. If the modified version is not the same as when it was last read during the transaction, the write skew checks throws an exception and the transaction is rolled back.

Enabling write skew check requires the **REPEATABLE_READ** isolation level. Also, in clustered mode (distributed or replicated modes), set up entry versioning. For local mode, entry versioning is not required.

IMPORTANT

With optimistic transactions, write skew checks are required for (atomic) conditional operations.

Report a bug

23.1.5. Transactions Spanning Multiple Cache Instances

Each cache operates as a separate, standalone Java Transaction API (JTA) resource. However, components can be internally shared by Red Hat JBoss Data Grid for optimization, but this sharing does not affect how caches interact with a Java Transaction API (JTA) Manager.

Report a bug

23.2. CONFIGURE TRANSACTIONS

23.2.1. Configure Transactions (Library Mode)

In Red Hat JBoss Data Grid, transactions in Library mode can be configured with synchronization and transaction recovery. Transactions in their entirety (which includes synchronization and transaction recovery) are not available in Remote Client-Server mode.

In Library mode, transactions are configured as follows:

Procedure 23.1. Configure Transactions in Library Mode (XML Configuration)

1. **Set the Transaction Mode**
   See the table below this procedure for a list of available lookup classes.

```
<namedCache ...>
 <transaction transactionMode="{TRANSACTIONAL,NON_TRANSACTIONAL}">
        ...
</namedCache>
```

2. **Configure the Transaction Manager**
   The ***transactionMode*** element configures whether or not the cache is transactional.

```
<namedCache ...>
 <transaction transactionMode="TRANSACTIONAL"
       transactionManagerLookupClass="
{TransactionManagerLookupClass}">
</namedCache>
```

3. **Configure Locking Mode**
   The ***lockingMode*** parameter determines if the optimistic or pessimistic locking method is used. If the cache is non-transactional, the locking mode is ignored. The default value for this parameter is **`OPTIMISTIC`**.

```
<namedCache ...>
 <transaction transactionMode="TRANSACTIONAL"
       transactionManagerLookupClass="
{TransactionManagerLookupClass}"
       lockingMode="{OPTIMISTIC,PESSIMISTIC}">
</namedCache>
```

4. **Specify Synchronization**
   The **`useSynchronization`** element configures the cache to register a synchronization with the transaction manager, or register itself as an XA resource. The default value for this element is **`true`** (use synchronization).

```
<namedCache ...>
 <transaction transactionMode="TRANSACTIONAL"
       transactionManagerLookupClass="
{TransactionManagerLookupClass}"
       lockingMode="{OPTIMISTIC,PESSIMISTIC}"
       useSynchronization="{true,false}">
</namedCache>
```

5. **Configure Recovery**

The **recovery** element enables recovery for the cache when set to **true**.

The ***recoveryInfoCacheName*** parameter sets the name of the cache where recovery information is held. The default name of the cache is **__recoveryInfoCacheName__**.

```
<namedCache ...>
 <transaction transactionMode="TRANSACTIONAL"
       transactionManagerLookupClass="
{TransactionManagerLookupClass}"
       lockingMode="{OPTIMISTIC,PESSIMISTIC}"
       useSynchronization="{true,false}">
  <recovery enabled="true"
     recoveryInfoCacheName="{CacheName}" />
</namedCache>
```

6. **Configure the Write Skew Check**
   The **writeSkew** check determines if a modification to the entry from a different transaction should roll back the transaction. Write skew set to **true** requires ***isolation_level*** set to **REPEATABLE_READ**. The default value for ***writeSkew*** and **isolation_level** are **false** and **READ_COMMITTED** respectively.

```
<namedCache ...>
 <transaction ...>
 <locking isolation_level="{READ_COMMITTED,REPEATABLE_READ}"
    writeSkew="{true,false}" />
        ...
</namedCache>
```

7. **Configure Entry Versioning**
   For clustered caches, enable write skew check by enabling entry versioning and setting its value to **SIMPLE**.

```
<namedCache ...>
 <transaction ...>
 <locking ...>
 <versioning enabled="{true,false}"
      versioningScheme="{NONE|SIMPLE}"/>
        ...
</namedCache>
```

Procedure 23.2. Configure Transactions in Library Mode (Programmatic Configuration)

1. **Set the Transaction Mode**
   Set the transaction mode as follows:

```
Configuration config = new ConfigurationBuilder()/* ...
*/.transaction()
        .transactionMode(TransactionMode.TRANSACTIONAL);
```

2. **Configure the Transaction Manager**
   See the table below this procedure for a list of available lookup classes.

```
Configuration config = new ConfigurationBuilder()/* ...
```

```
*/.transaction()
        .transactionMode(TransactionMode.TRANSACTIONAL)
        .transactionManagerLookup(new
GenericTransactionManagerLookup());
```

3. **Configure Locking Mode**
   The **`lockingMode`** value determines whether optimistic or pessimistic locking is used. If the cache is non-transactional, the locking mode is ignored. The default value is **`OPTIMISTIC`**.

```
Configuration config = new ConfigurationBuilder()/* ...
*/.transaction()
        .transactionMode(TransactionMode.TRANSACTIONAL)
        .transactionManagerLookup(new
GenericTransactionManagerLookup());
        .lockingMode(LockingMode.OPTIMISTIC);
```

4. **Specify Synchronization**
   The **`useSynchronization`** value configures the cache to register a synchronization with the transaction manager, or register itself as an XA resource. The default value is **`true`** (use synchronization).

```
Configuration config = new ConfigurationBuilder()/* ...
*/.transaction()
        .transactionMode(TransactionMode.TRANSACTIONAL)
        .transactionManagerLookup(new
GenericTransactionManagerLookup());
        .lockingMode(LockingMode.OPTIMISTIC)
        .useSynchronization(true);
```

5. **Configure Recovery**
   The ***`recovery`*** parameter enables recovery for the cache when set to **`true`**.

   The **`recoveryInfoCacheName`** sets the name of the cache where recovery information is held. The default name of the cache is specified by ***`RecoveryConfiguration.DEFAULT_RECOVERY_INFO_CACHE`***.

```
Configuration config = new ConfigurationBuilder()/* ...
*/.transaction()
        .transactionMode(TransactionMode.TRANSACTIONAL)
        .transactionManagerLookup(new
GenericTransactionManagerLookup());
        .lockingMode(LockingMode.OPTIMISTIC)
        .useSynchronization(true)
        .recovery()
            .recoveryInfoCacheName("anotherRecoveryCacheName");
```

6. **Configure Write Skew Check**
   The ***`writeSkew`*** check determines if a modification to the entry from a different transaction should roll back the transaction. Write skew set to **`true`** requires ***`isolation_level`*** set to **REPEATABLE_READ**. The default value for ***`writeSkew`*** and ***`isolation_level`*** are **`false`** and **READ_COMMITTED** respectively.

```
Configuration config = new ConfigurationBuilder()/* ... */.locking()
```

```
.isolationLevel(IsolationLevel.REPEATABLE_READ).writeSkewCheck(true)
;
```

7. **Configure Entry Versioning**
   For clustered caches, enable write skew check by enabling entry versioning and setting its value to **SIMPLE**.

```
Configuration config = new ConfigurationBuilder()/* ...
*/.versioning()
         .enable()
         .scheme(VersioningScheme.SIMPLE);
```

Table 23.1. Transaction Manager Lookup Classes

Class Name	Details
org.infinispan.transaction.lookup.DummyTransactionManagerLookup	Used primarily for testing environments. This testing transaction manager is not for use in a production environment and is severely limited in terms of functionality, specifically for concurrent transactions and recovery.
org.infinispan.transaction.lookup.JBossStandaloneJTAManagerLookup	The default transaction manager when Red Hat JBoss Data Grid runs in a standalone environment. It is a fully functional JBoss Transactions based transaction manager that overcomes the functionality limits of the **DummyTransactionManager**.
org.infinispan.transaction.lookup.GenericTransactionManagerLookup	GenericTransactionManagerLookup is used by default when no transaction lookup class is specified. This lookup class is recommended when using JBoss Data Grid with Java EE-compatible environment that provides a TransactionManager interface, and is capable of locating the Transaction Manager in most Java EE application servers. If no transaction manager is located, it defaults to **DummyTransactionManager**.
org.infinispan.transaction.lookup.JBossTransactionManagerLookup	The **JbossTransactionManagerLookup** finds the standard transaction manager running in the application server. This lookup class uses JNDI to look up the TransactionManager instance, and is recommended when custom caches are being used in JTA transactions.

Report a bug

23.2.2. Configure Transactions (Remote Client-Server Mode)

Red Hat JBoss Data Grid does not offer transactions in Remote Client-Server mode. The default and only supported configuration is non-transactional, which is set as follows:

```
<cache>
 ...
  <transaction mode="NONE" />
 ...
</cache>
```

Report a bug

23.3. TRANSACTION RECOVERY

The Transaction Manager coordinates the recovery process and works with Red Hat JBoss Data Grid to determine which transactions require manual intervention to complete operations. This process is known as transaction recovery.

JBoss Data Grid uses JMX for operations that explicitly force transactions to commit or roll back. These methods receive byte arrays that describe the XID instead of the number associated with the relevant transactions.

The System Administrator can use such JMX operations to facilitate automatic job completion for transactions that require manual intervention. This process uses the Transaction Manager's transaction recovery process and has access to the Transaction Manager's XID objects.

Report a bug

23.3.1. Transaction Recovery Process

The following process outlines the transaction recovery process in Red Hat JBoss Data Grid.

Procedure 23.3. The Transaction Recovery Process

1. The Transaction Manager creates a list of transactions that require intervention.

2. The system administrator, connected to JBoss Data Grid using JMX, is presented with the list of transactions (including transaction IDs) using email or logs. The status of each transaction is either **`COMMITTED`** or **`PREPARED`**. If some transactions are in both **`COMMITTED`** and **`PREPARED`** states, it indicates that the transaction was committed on some nodes while in the preparation state on others.

3. The System Administrator visually maps the XID received from the Transaction Manager to a JBoss Data Grid internal ID. This step is necessary because the XID (a byte array) cannot be conveniently passed to the JMX tool and then reassembled by JBoss Data Grid without this mapping.

4. The system administrator forces the commit or rollback process for a transaction based on the mapped internal ID.

Report a bug

23.3.2. Transaction Recovery Example

The following example describes how transactions are used in a situation where money is transferred from an account stored in a database to an account stored in Red Hat JBoss Data Grid.

Example 23.1. Money Transfer from an Account Stored in a Database to an Account in JBoss Data Grid

1. The `TransactionManager.commit()` method is invoked to run the two phase commit protocol between the source (the database) and the destination (JBoss Data Grid) resources.

2. The `TransactionManager` tells the database and JBoss Data Grid to initiate the prepare phase (the first phase of a Two Phase Commit).

During the commit phase, the database applies the changes but JBoss Data Grid fails before receiving the Transaction Manager's commit request. As a result, the system is in an inconsistent state due to an incomplete transaction. Specifically, the amount to be transferred has been subtracted from the database but is not yet visible in JBoss Data Grid because the prepared changes could not be applied.

Transaction recovery is used here to reconcile the inconsistency between the database and JBoss Data Grid entries.

NOTE

To use JMX to manage transaction recoveries, JMX support must be explicitly enabled.

Report a bug

23.4. DEADLOCK DETECTION

A deadlock occurs when multiple processes or tasks wait for the other to release a mutually required resource. Deadlocks can significantly reduce the throughput of a system, particularly when multiple transactions operate against one key set.

Red Hat JBoss Data Grid provides deadlock detection to identify such deadlocks. Deadlock detection is set to **`disabled`** by default.

Report a bug

23.4.1. Enable Deadlock Detection

Deadlock detection in Red Hat JBoss Data Grid is set to **`disabled`** by default but can be enabled and configured for each cache using the ***`namedCache`*** configuration element by adding the following:

```
<deadlockDetection enabled="true" spinDuration="100"/>
```

The ***`spinDuration`*** attribute defines how often lock acquisition is attempted within the maximum time allowed to acquire a particular lock (in milliseconds).

Deadlock detection can only be applied to individual caches. Deadlocks that are applied on more than one cache cannot be detected by JBoss Data Grid.

Report a bug

CHAPTER 24. CONFIGURE JGROUPS

24.1. ABOUT JGROUPS

JGroups is the underlying group communication library used to connect Red Hat JBoss Data Grid instances.

Report a bug

24.2. CONFIGURE RED HAT JBOSS DATA GRID INTERFACE BINDING (REMOTE CLIENT-SERVER MODE)

24.2.1. Interfaces

Red Hat JBoss Data Grid allows users to specify an interface type rather than a specific (unknown) IP address.

- ***`link-local`***: Uses a **`169.`*`x`*`.`*`x`*`.`*`x`*** or **`254.`*`x`*`.`*`x`*`.`*`x`*** address. This suits the traffic within one box.

```
<interfaces>
    <interface name="link-local">
        <link-local-address/>
    </interface>
    ...
</interfaces>
```

- ***`site-local`***: Uses a private IP address, for example **`192.168.`*`x`*`.`*`x`***. This prevents extra bandwidth charged from GoGrid, and similar providers.

```
<interfaces>
    <interface name="site-local">
        <site-local-address/>
    </interface>
    ...
</interfaces>
```

- ***`global`***: Picks a public IP address. This should be avoided for replication traffic.

```
<interfaces>
    <interface name="global">
        <any-address/>
    </interface>
    ...
</interfaces>
```

- ***`non-loopback`***: Uses the first address found on an active interface that is not a **`127.`*`x`*`.`*`x`*`.`*`x`*** address.

```
<interfaces>
    <interface name="non-loopback">
        <not>
     <loopback />
```

```
  </not>
     </interface>
</interfaces>
```

Report a bug

24.2.2. Binding Sockets

Socket bindings provide a named the combination of interface and port. Sockets can be bound to the interface either individually or using a socket binding group.

Report a bug

24.2.2.1. Binding a Single Socket Example

The following is an example depicting the use of JGroups interface socket binding to bind an individual socket using the ***socket-binding*** element.

```
<socket-binding name="jgroups-udp" ... interface="site-local"/>
```

Report a bug

24.2.2.2. Binding a Group of Sockets Example

The following is an example depicting the use of Groups interface socket bindings to bind a group, using the ***socket-binding-group*** element:

```
<socket-binding-group name="ha-sockets" default-interface="global">
 ...
 <socket-binding name="jgroups-tcp" port="7600"/>
 <socket-binding name="jgroups-tcp-fd" port="57600"/>
 ...
</socket-binding-group>
```

The two sample socket bindings in the example are bound to the same ***default-interface*** (**global**), therefore the interface attribute does not need to be specified.

Report a bug

24.2.3. Configure JGroups Socket Binding

Each JGroups stack, configured in the JGroups subsystem, uses a specific socket binding. Set up the socket binding as follows:

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.2" default-stack="udp">
    <stack name="udp">
        <transport type="UDP" socket-binding="jgroups-udp">
            ...
        </transport>
        <!-- rest of protocols -->
    </stack>
</subsystem>
```

IMPORTANT

When using UDP as the JGroups transport, the socket binding has to specify the regular (unicast) port, multicast address, and multicast port.

Report a bug

24.3. CONFIGURE JGROUPS (LIBRARY MODE)

Red Hat JBoss Data Grid must have an appropriate JGroups configuration in order to operate in clustered mode.

To configure JGroups programmatically use the following:

```
GlobalConfiguration gc = new GlobalConfigurationBuilder()
  .transport()
  .defaultTransport()
  .addProperty("configurationFile","jgroups.xml")
  .build();
```

To configure JGroups using XML use the following:

```
<infinispan>
  <global>
    <transport>
      <properties>
        <property name="configurationFile" value="jgroups.xml" />
      </properties>
    </transport>
  </global>

  ...

</infinispan>
```

In either programmatic or XML configuration methods, JBoss Data Grid searches for `jgroups.xml` in the classpath before searching for an absolute path name if it is not found in the classpath.

Report a bug

24.3.1. JGroups Transport Protocols

A transport protocol is the protocol at the bottom of a protocol stack. Transport Protocols are responsible for sending and receiving messages from the network.

Red Hat JBoss Data Grid ships with both UDP and TCP transport protocols.

Report a bug

24.3.1.1. The UDP Transport Protocol

UDP is a transport protocol that uses:

- IP multicasting to send messages to all members of a cluster.

- UDP datagrams for unicast messages, which are sent to a single member.

When the UDP transport is started, it opens a unicast socket and a multicast socket. The unicast socket is used to send and receive unicast messages, the multicast socket sends and receives multicast sockets. The physical address of the channel with be the same as the address and port number of the unicast socket.

Report a bug

24.3.1.2. The TCP Transport Protocol

TCP/IP is a replacement transport for UDP in situations where IP multicast cannot be used, such as operations over a WAN where routers may discard IP multicast packets.

TCP is a transport protocol used to send unicast and multicast messages.

- When sending multicast messages, TCP sends multiple unicast messages.
- When using TCP, each message to all cluster members is sent as multiple unicast messages, or one to each member.

As IP multicasting cannot be used to discover initial members, another mechanism must be used to find initial membership.

Red Hat JBoss Data Grid's Hot Rod is a custom TCP client/server protocol.

Report a bug

24.3.1.3. Using the TCPPing Protocol

Some networks only allow TCP to be used. The pre-configured **jgroups-tcp.xml** includes the **MPING** protocol, which uses **UDP** multicast for discovery. When **UDP** multicast is not available, the **MPING** protocol, has to be replaced by a different mechanism. The recommended alternative is the **TCPPING** protocol. The **TCPPING** configuration contains a static list of IP addresses which are contacted for node discovery.

The following is an example of how to configure the JGroups subsystem to use **TCPPING**.

```
<TCP bind_port="7800" />
<TCPPING timeout="3000"

initial_hosts="${jgroups.tcpping.initial_hosts:HostA[7800],HostB[7801]}"
        port_range="1"
        num_initial_members="3"/>
```

Report a bug

24.3.2. Pre-Configured JGroups Files

Red Hat JBoss Data Grid ships with a number of pre-configured JGroups files packaged in **infinispan-core.jar**, and are available on the classpath by default. In order to use one of these files, specify one of these file names instead of using **jgroups.xml**.

The JGroups configuration files shipped with JBoss Data Grid are intended to be used as a starting point for a working project. JGroups will usually require fine-tuning for optimal network performance.

Available configurations are:

- `jgroups-udp.xml`
- `jgroups-tcp.xml`
- `jgroups-ec2.xml`

Report a bug

24.3.2.1. jgroups-udp.xml

`jgroups-udp.xml` is a pre-configured JGroups file in Red Hat JBoss Data Grid. The `jgroups-udp.xml` configuration

- uses UDP as a transport and UDP multicast for discovery.
- is suitable for large clusters (over 8 nodes).
- is suitable if using Invalidation or Replication modes.

The behavior of some of these settings can be altered by adding certain system properties to the JVM at startup. The settings that can be configured are shown in the following table.

Table 24.1. jgroups-udp.xml System Properties

System Property	Description	Default	Required?
jgroups.udp.mcast_addr	IP address to use for multicast (both for communications and discovery). Must be a valid Class D IP address, suitable for IP multicast.	228.6.7.8	No
jgroups.udp.mcast_port	Port to use for multicast socket	46655	No
j jgroups.udp.ip_ttl	Specifies the time-to-live (TTL) for IP multicast packets. The value here refers to the number of network hops a packet is allowed to make before it is dropped	2	No

Report a bug

24.3.2.2. jgroups-tcp.xml

`jgroups-tcp.xml` is a pre-configured JGroups file in Red Hat JBoss Data Grid. The `jgroups-tcp.xml` configuration

- uses TCP as a transport and UDP multicast for discovery.
- is generally only used where multicast UDP is not an option.

- TCP does not perform as well as UDP for clusters of eight or more nodes. Clusters of four nodes or fewer result in roughly the same level of performance for both UDP and TCP.

As with other pre-configured JGroups files, the behavior of some of these settings can be altered by adding certain system properties to the JVM at startup. The settings that can be configured are shown in the following table.

Table 24.2. jgroups-udp.xml System Properties

System Property	Description	Default	Required?
jgroups.tcp.address	IP address to use for the TCP transport.	127.0.0.1	No
jgroups.tcp.port	Port to use for TCP socket	7800	No
jgroups.udp.mcast_addr	IP address to use for multicast (for discovery). Must be a valid Class D IP address, suitable for IP multicast.	228.6.7.8	No
jgroups.udp.mcast_port	Port to use for multicast socket	46655	No
jgroups.udp.ip_ttl	Specifies the time-to-live (TTL) for IP multicast packets. The value here refers to the number of network hops a packet is allowed to make before it is dropped	2	No

Report a bug

24.3.2.3. jgroups-ec2.xml

`jgroups-ec2.xml` is a pre-configured JGroups file in Red Hat JBoss Data Grid. The `jgroups-ec2.xml` configuration

- uses TCP as a transport and S3_PING for discovery.
- is suitable on Amazon EC2 nodes where UDP multicast isn't available.

As with other pre-configured JGroups files, the behavior of some of these settings can be altered by adding certain system properties to the JVM at startup. The settings that can be configured are shown in the following table.

Table 24.3. jgroups-ec2.xml System Properties

System Property	Description	Default	Required?
jgroups.tcp.address	IP address to use for the TCP transport.	127.0.0.1	No

System Property	Description	Default	Required?
jgroups.tcp.port	Port to use for TCP socket	7800	No
jgroups.s3.access_key	The Amazon S3 access key used to access an S3 bucket		Yes
jgroups.s3.secret_access_key	The Amazon S3 secret key used to access an S3 bucket		Yes
jgroups.s3.bucket	Name of the Amazon S3 bucket to use. Must be unique and must already exist		Yes
jgroups.s3.pre_signed_delete_url	The pre-signed URL to be used for the DELETE operation.		Yes
jgroups.s3.pre_signed_put_url	The pre-signed URL to be used for the PUT operation.		Yes
jgroups.s3.prefix	If set, S3_PING searches for a bucket with a name that starts with the prefix value.		No

Report a bug

24.4. TEST MULTICAST USING JGROUPS

Learn how to ensure that the system has correctly configured multicasting within the cluster.

Report a bug

24.4.1. Testing With Different Red Hat JBoss Data Grid Versions

The following table details which Red Hat JBoss Data Grid versions are compatible with this multicast test:

Table 24.4. Testing with Different JBoss Data Grid Versions

Version	Test Case	Details
JBoss Data Grid 6.0	Not Available	This version of JBoss Data Grid is based on JBoss Enterprise Application Server 6.0, which does not include the test classes used for this test.

Version	Test Case	Details
JBoss Data Grid 6.0.1	Not Available	This version of JBoss Data Grid is based on JBoss Enterprise Application Platform 6.0, which does not include the test classes used for this test.
JBoss Data Grid 6.1	Available	This version of JBoss Data Grid is based on JBoss Enterprise Application Platform 6.0.1, but contains a newer version of the JGroups JAR file than the JAR file included in JBoss Enterprise Application Platform. As a result, the test class required for this test is available in the **`$JBOSS_HOME/modules/org/jgroups/main`** directory for JBoss Data Grid 6.1.
JBoss Data Grid 6.2	Available	This version of JBoss Data Grid is based on JBoss Enterprise Application Platform 6.1. The JAR file is named according to the version, for example **`jgroups-3.2.7.Final-redhat-1.jar`** and is available in the **`$JBOSS_HOME/modules/system/layers/base/org/jgroups/main`** directory.

Report a bug

24.4.2. Testing Multicast Using JGroups

The following procedure details the steps to test multicast using JGroups if you are using Red Hat JBoss Data Grid 6.2:

Prerequisites

Ensure that the following prerequisites are met before starting the testing procedure.

1. Set the ***`bind_addr`*** value to the appropriate IP address for the instance.

2. For added accuracy, set ***`mcast_addr`*** and ***`port`*** values that are the same as the cluster communication values.

3. Start two command line terminal windows. Navigate to the location of the JGroups JAR file for one of the two nodes in the first terminal and the same location for the second node in the second terminal.

Procedure 24.1. Test Multicast Using JGroups

1. **Run the Multicast Server on Node One**
   Run the following command on the command line terminal for the first node:

   ```
   java -cp jgroups.jar org.jgroups.tests.McastReceiverTest -mcast_addr
   230.1.2.3 -port 5555 -bind_addr $YOUR_BIND_ADDRESS
   ```

2. **Run the Multicast Server on Node Two**
   Run the following command on the command line terminal for the second node:

   ```
   java -cp jgroups.jar org.jgroups.tests.McastSenderTest -mcast_addr
   230.1.2.3 -port 5555 -bind_addr $YOUR_BIND_ADDRESS
   ```

3. **Transmit Information Packets**
   Enter information on instance for node two (the node sending packets) and press enter to send the information.

4. **View Receives Information Packets**
   View the information received on the node one instance. The information entered in the previous step should appear here.

5. **Confirm Information Transfer**
   Repeat steps 3 and 4 to confirm all transmitted information is received without dropped packets.

6. **Repeat Test for Other Instances**
   Repeat steps 1 to 4 for each combination of sender and receiver. Repeating the test identifies other instances that are incorrectly configured.

Result

All information packets transmitted from the sender node must appear on the receiver node. If the sent information does not appear as expected, multicast is incorrectly configured in the operating system or the network.

Report a bug

CHAPTER 25. USE RED HAT DATA GRID WITH AMAZON WEB SERVICES

25.1. THE S3_PING JGROUPS DISCOVERY PROTOCOL

`S3_PING` is a discovery protocol that is ideal for use with Amazon's Elastic Compute Cloud (EC2) because EC2 does not allow multicast and therefore **`MPING`** is not allowed.

Each EC2 instance adds a small file to an S3 data container, known as a bucket. Each instance then reads the files in the bucket to discover the other members of the cluster.

Report a bug

25.2. S3_PING CONFIGURATION OPTIONS

Red Hat JBoss Data Grid works with Amazon Web Services in two ways:

- In Library mode, use JGroups' **`jgroups-ec2.xml`** file (see Section 24.3.2.3, "jgroups-ec2.xml" for details) or use the **`S3_PING`** protocol.
- In Remote Client-Server mode, use JGroups' **`S3_PING`** protocol.

In Library and Remote Client-Server mode, there are three ways to configure the **`S3_PING`** protocol for clustering to work in Amazon AWS:

- Use Private S3 Buckets. These buckets use Amazon AWS credentials.
- Use Pre-Signed URLs. These pre-assigned URLs are assigned to buckets with private write and public read rights.
- Use Public S3 Buckets. These buckets do not have any credentials.

Report a bug

25.2.1. Using Private S3 Buckets

This configuration requires access to a private bucket that can only be accessed with the appropriate AWS credentials. To confirm that the appropriate permissions are available, confirm that the user has the following permissions for the bucket:

- List
- Upload/Delete
- View Permissions
- Edit Permissions

Ensure that the **`S3_PING`** configuration includes the following properties:

- either the ***`location`*** or the ***`prefix`*** property to specify the bucket, but not both. If the ***`prefix`*** property is set, **`S3_PING`** searches for a bucket with a name that starts with the prefix value. If a bucket with the prefix at the beginning of the name is found, **`S3_PING`** uses that bucket. If a bucket with the prefix is not found, **S3_PING** creates a bucket using the AWS credentials and names it based on the prefix and a UUID (the naming format is *{prefix value}-{UUID}*).

- the ***access_key*** and ***secret_access_key*** properties for the AWS user.

NOTE

If a **403** error displays when using this configuration, verify that the properties have the correct values. If the problem persists, confirm that the system time in the EC2 node is correct. Amazon S3 rejects requests with a time stamp that is more than **15** minutes old compared to their server's times for security purposes.

Example 25.1. Start the Red Hat JBoss Data Grid Server with a Private Bucket

Run the following command from the top level of the server directory to start the Red Hat JBoss Data Grid server using a private S3 bucket:

```
bin/clustered.sh -Djboss.bind.address={server_ip_address} -
Djboss.bind.address.management={server_ip_address} -
Djboss.default.jgroups.stack=s3 -Djgroups.s3.bucket={s3_bucket_name} -
Djgroups.s3.access_key={access_key} -
Djgroups.s3.secret_access_key={secret_access_key}
```

1. Replace *{server_ip_address}* with the server's IP address.
2. Replace *{s3_bucket_name}* with the appropriate bucket name.
3. Replace *{access_key}* with the user's access key.
4. Replace *{secret_access_key}* with the user's secret access key.

Report a bug

25.2.2. Using Pre-Signed URLs

For this configuration, create a publically readable bucket in S3 by setting the **List** permissions to **Everyone** to allow public read access. Each node in the cluster generates a pre-signed URL for put and delete operations, as required by the **S3_PING** protocol. This URL points to a unique file and can include a folder path within the bucket.

NOTE

Longer paths will cause errors in **S3_PING**. For example, a path such as **my_bucket/DemoCluster/node1** works while a longer path such as **my_bucket/Demo/Cluster/node1** will not.

Report a bug

25.2.2.1. Generating Pre-Signed URLs

JGroup's **S3_PING** class includes a utility method to generate pre-signed URLs. The last argument for this method is the time when the URL expires expressed in the number of seconds since the Unix epoch (January 1, 1970).

The syntax to generate a pre-signed URL is as follows:

```
String Url = S3_PING.generatePreSignedUrl("{access_key}",
"{secret_access_key}", "{operation}", "{bucket_name}", "{path}",
{seconds});
```

1. Replace *{operation}* with either **PUT** or **DELETE**.
2. Replace *{access_key}* with the user's access key.
3. Replace *{secret_access_key}* with the user's secret access key.
4. Replace *{bucket_name}* with the name of the bucket.
5. Replace *{path}* with the desired path to the file within the bucket.
6. Replace *{seconds}* with the number of seconds since the Unix epoch (January 1, 1970) that the path remains valid.

Example 25.2. Generate a Pre-Signed URL

```
String putUrl = S3_PING.generatePreSignedUrl("access_key",
"secret_access_key", "put", "my_bucket", "DemoCluster/node1",
1234567890);
```

Ensure that the **S3_PING** configuration includes the ***pre_signed_put_url*** and ***pre_signed_delete_url*** properties generated by the call to **S3_PING.generatePreSignedUrl()**. This configuration is more secure than one using private S3 buckets, because the AWS credentials are not stored on each node in the cluster

NOTE

If a pre-signed URL is entered into an XML file, then the & characters in the URL must be replaced with its XML entity (**&amp;**).

Report a bug

25.2.2.2. Set Pre-Signed URLs Using the Command Line

To set the pre-signed URLs using the command line, use the following guidelines:

- Enclose the URL in double quotation marks (**""**).
- In the URL, each occurrence of the ampersand (&) character must be escaped with a backslash (\)

Example 25.3. Start a JBoss Data Grid Server with a Pre-Signed URL

```
bin/clustered.sh -Djboss.bind.address={server_ip_address} -
Djboss.bind.address.management={server_ip_address} -
Djboss.default.jgroups.stack=s3 -
Djgroups.s3.pre_signed_put_url="http://{s3_bucket_name}.s3.amazonaws.com
/ node1?
AWSAccessKeyId={access_key}\&Expires={expiration_time}\&Signature={signa
ture}"-
Djgroups.s3.pre_signed_delete_url="http://{s3_bucket_name}.s3.amazonaws.
com/ node1?
AWSAccessKeyId={access_key}\&Expires={expiration_time}\&Signature={signa
ture}"
```

In the provided example, the ***{signatures}*** values are generated by the **`S3_PING.generatePreSignedUrl()`** method. Additionally, the *{expiration_time}* values are the expiration time for the URL that are passed into the **`S3_PING.generatePreSignedUrl()`** method.

Report a bug

25.2.3. Using Public S3 Buckets

This configuration involves an S3 bucket that has public read and write permissions, which means that **`Everyone`** has permissions to **`List`**, **`Upload/Delete`**, **`View Permissions`**, and **`Edit Permissions`** for the bucket.

The ***location*** property must be specified with the bucket name for this configuration. This configuration method is the least secure because any user who knows the name of the bucket can upload and store data in the bucket and the bucket creator's account is charged for this data.

To start the Red Hat JBoss Data Grid server, use the following command:

```
bin/clustered.sh -Djboss.bind.address={server_ip_address} -
Djboss.bind.address.management={server_ip_address} -
Djboss.default.jgroups.stack=s3 -Djgroups.s3.bucket={s3_bucket_name}
```

Report a bug

25.2.4. Troubleshooting S3_PING Warnings

Depending on the **`S3_PING`** configuration type used, the following warnings may appear when starting the JBoss Data Grid Server:

```
15:46:03,468 WARN  [org.jgroups.conf.ProtocolConfiguration] (MSC service
thread 1-7) variable "${jgroups.s3.pre_signed_put_url}" in S3_PING could
not be substituted; pre_signed_put_url is removed from properties
```

```
15:46:03,469 WARN  [org.jgroups.conf.ProtocolConfiguration] (MSC service
thread 1-7) variable "${jgroups.s3.prefix}" in S3_PING could not be
substituted; prefix is removed from properties
```

```
15:46:03,469 WARN  [org.jgroups.conf.ProtocolConfiguration] (MSC service
thread 1-7) variable "${jgroups.s3.pre_signed_delete_url}" in S3_PING
could not be substituted; pre_signed_delete_url is removed from properties
```

In each case, ensure that the property listed as missing in the warning is not needed by the **`S3_PING`** configuration.

Report a bug

CHAPTER 26. HIGH AVAILABILITY USING SERVER HINTING

In Red Hat JBoss Data jGrid, Server Hinting ensures that backed up copies of data are not stored on the same physical server, rack, or data center as the original. Server Hinting does not apply to total replication because total replication mandates complete replicas on every server, rack, and data center.

Data distribution across nodes is controlled by the Consistent Hashing mechanism. JBoss Data Grid offers a pluggable policy to specify the consistent hashing algorithm. For details see Section 26.4, “ConsistentHashFactories”

Setting a ***`machineId`***, ***`rackId`***, or ***`siteId`*** in the transport configuration will trigger the use of **`TopologyAwareConsistentHashFactory`**, which is the equivalent of the **`DefaultConsistentHashFactory`** with Server Hinting enabled.

Server Hinting is particularly important when ensuring the high availability of your JBoss Data Grid implementation.

Report a bug

26.1. ESTABLISHING SERVER HINTING WITH JGROUPS

When setting up a clustered environment in Red Hat JBoss Data Grid, Server Hinting is configured when establishing JGroups configuration.

JBoss Data Grid ships with several JGroups files pre-configured for clustered mode, and using Server Hinting. These files can be used as a starting point when configuring clustered modes in JBoss Data Grid.

See Also:

- Section 24.3.2, “Pre-Configured JGroups Files”

Report a bug

26.2. CONFIGURE SERVER HINTING (REMOTE CLIENT-SERVER MODE)

In Red Hat JBoss Data Grid's Remote Client-Server mode, Server Hinting is configured in the JGroups subsystem on the `transport` element for the default stack, as follows:

Procedure 26.1. Configure Server Hinting in Remote Client-Server Mode

1. **Find the JGroups Subsystem Configuration**

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.1"
    default-stack="${jboss.default.jgroups.stack:udp}">
 <stack name="udp">
```

2. **Enable Server Hinting via the `transport` Element**

    a. **Set the Site ID**

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.1"
    default-stack="${jboss.default.jgroups.stack:udp}">
```

```
<stack name="udp">
<transport type="UDP"
    socket-binding="jgroups-udp"
    site="${jboss.jgroups.transport.site:s1}">
```

b. **Set the Rack ID**

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.1"
    default-stack="${jboss.default.jgroups.stack:udp}">
 <stack name="udp">
  <transport type="UDP"
      socket-binding="jgroups-udp"
      site="${jboss.jgroups.transport.site:s1}"
      rack="${jboss.jgroups.transport.rack:r1}">
```

c. **Set the Machine ID**

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.1"
    default-stack="${jboss.default.jgroups.stack:udp}">
 <stack name="udp">
  <transport type="UDP"
      socket-binding="jgroups-udp"
      site="${jboss.jgroups.transport.site:s1}"
      rack="${jboss.jgroups.transport.rack:r1}"
      machine="${jboss.jgroups.transport.machine:m1}">
      ...
  </transport>
 </stack>
</subsystem>
```

Report a bug

26.3. CONFIGURE SERVER HINTING (LIBRARY MODE)

In Red Hat JBoss Data Grid's Library mode, Server Hinting is configured at the transport level. The following is a Server Hinting sample configuration:

Procedure 26.2. Configure Server Hinting for Library Mode

The following configuration attributes are used to configure Server Hinting in JBoss Data Grid.

1. **Set the *clusterName* Attribute**
   The ***clusterName*** attribute specifies the name assigned to the cluster.

   ```
   <transport clusterName = "MyCluster" />
   ```

2. **Add the *machineId***
   The ***machineId*** attribute specifies the JVM instance that contains the original data. This is particularly useful for nodes with multiple JVMs and physical hosts with multiple virtual hosts.

   ```
   <transport clusterName = "MyCluster"
              machineId = "LinuxServer01" />
   ```

3. **Add the *rackId***

   The ***rackId*** parameter specifies the rack that contains the original data, so that other racks are used for backups.

   ```
   <transport clusterName = "MyCluster"
              machineId = "LinuxServer01"
              rackId = "Rack01" />
   ```

4. **Add the *siteId***

   The ***siteId*** parameter differentiates between nodes in different data centers replicating to each other.

   ```
   <transport clusterName = "MyCluster"
              machineId = "LinuxServer01"
              rackId = "Rack01"
              siteId = "US-WestCoast" />
   ```

The listed parameters are optional in a JBoss Data Grid configuration.

If ***machineId***, ***rackId***, or ***siteId*** are included in the configuration, **`TopologyAwareConsistentHashFactory`** is selected automatically, enabling Server Hinting. However, if Server Hinting is not configured, JBoss Data Grid's distribution algorithms are allowed to store replications in the same physical machine/rack/data center as the original data.

Report a bug

26.4. CONSISTENTHASHFACTORIES

Red Hat JBoss Data Grid offers a pluggable mechanism for selecting the consistent hashing algorithm. It is shipped with four implementations but a custom implementation can also be used.

JBoss Data Grid ships with four ConsistentHashFactory implementations:

- **`DefaultConsistentHashFactory`** - keeps segments balanced evenly across all the nodes, however the key mapping is not guaranteed to be same across caches,as this depends on the history of each cache.

- **`SyncConsistentHashFactory`** - guarantees that the key mapping is the same for each cache, provided the current membership is the same. This has a drawback in that a node joining the cache can cause the existing nodes to also exchange segments, resulting in either additional state transfer traffic, the distribution of the data becoming less even, or both.

- **`TopologyAwareConsistentHashFactory`** - equivalent of **`DefaultConsistentHashFactory`**, but with server hinting enabled.

- **`TopologyAwareSyncConsistentHashFactory`** - equivalent of **`SyncConsistentHashFactory`**, but with server hinting enabled.

The consistent hash implementation can be selected via the hash configuration:

```
<hash
consistentHashFactory="org.infinispan.distribution.ch.TopologyAwareSyncCon
sistentHashFactory"/>
```

This configuration guarantees caches with the same members have the same consistent hash, and if the ***machineId***, ***rackId***, or ***siteId*** attributes are specified in the transport configuration it also spreads backup copies across physical machines/racks/data centers.

It has a potential drawback in that it can move a greater number of segments than necessary during re-balancing. This can be mitigated by using a larger number of segments.

Another potential drawback is that the segments are not distributed as evenly as possible, and actually using a very large number of segments can make the distribution of segments worse.

Report a bug

26.4.1. Implementing a ConsistentHashFactory

A custom **ConsistentHashFactory** must implement the **org.infinispan.distribution.ch.ConsistenHashFactory** interface with the following methods (all of which return an implementation of **org.infinispan.distribution.ch.ConsistentHash**):

```
create(Hash hashFunction, int numOwners, int numSegments, List<Address>
members,Map<Address, Float> capacityFactors)
updateMembers(ConsistentHash baseCH, List<Address> newMembers,
Map<Address,
Float> capacityFactors)
rebalance(ConsistentHash baseCH)
union(ConsistentHash ch1, ConsistentHash ch2)
```

Currently it is not possible to pass custom parameters to **ConsistentHashFactory** implementations.

Report a bug

CHAPTER 27. SET UP CROSS-DATACENTER REPLICATION

27.1. ABOUT CROSS-DATACENTER REPLICATION

In Red Hat JBoss Data Grid, Cross-Datacenter Replication allows the administrator to create data backups in multiple clusters. These clusters can be at the same physical location or different ones. JBoss Data Grid's Cross-Site Replication implementation is based on JGroups' **RELAY2** protocol.

Cross-Datacenter Replication ensures data redundancy across clusters. Ideally, each of these clusters must be in a different physical location than the others.

Report a bug

27.2. CROSS-DATACENTER REPLICATION OPERATIONS

Red Hat JBoss Data Grid's Cross-Datacenter Replication operation is explained through the use of an example, as follows:

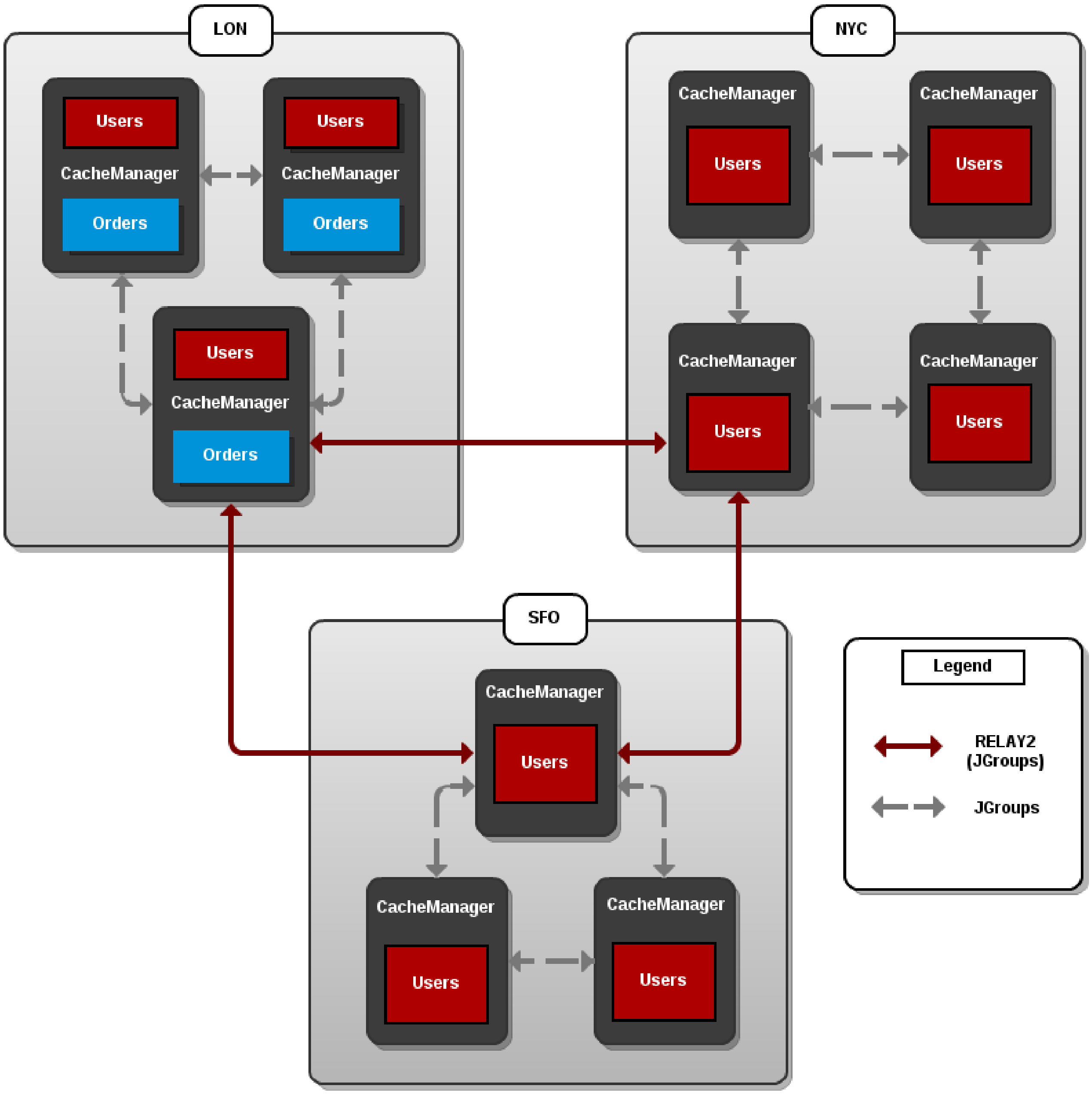


Figure 27.1. Cross-Datacenter Replication Example

Three sites are configured in this example: **LON**, **NYC** and **SFO**. Each site hosts a running JBoss Data Grid cluster made up of three to four physical nodes.

The **`Users`** cache is active in all three sites. Changes to the **`Users`** cache at the **LON** site is replicated at the other two sites. The **`Orders`** cache, however, is only available locally at the **LON** site because it is not replicated to the other sites.

The **`Users`** cache can use different replication mechanisms each site. For example, it can back up data synchronously to **SFO** and asynchronously to **NYC** and **LON**.

The **`Users`** cache can also have a different configuration from one site to another. For example, it can be configured as a distributed cache with ***`numOwners`*** set to **`2`** in the **LON** site, as a replicated cache in the **NYC** site and as a distributed cache with ***`numOwners`*** set to **`1`** in the **SFO** site.

JGroups is used for communication within each site as well as inter-site communication. Specifically, a JGroups protocol called **RELAY2** facilitates communication between sites. For more information, see Section B.4, “About RELAY2”

Report a bug

27.3. CONFIGURE CROSS-DATACENTER REPLICATION

27.3.1. Configure Cross-Datacenter Replication (Remote Client-Server Mode)

In Red Hat JBoss Data Grid's Remote Client-Server mode, cross-datacenter replication is set up as follows:

Procedure 27.1. Set Up Cross-Datacenter Replication

1. **Set Up RELAY**
   Add the following configuration to the `standalone.xml` file to set up **RELAY**:

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.2"
    default-stack="udp">
 <stack name="udp">
  <transport type="UDP"
      socket-binding="jgroups-udp"/>
  ...

  <relay site="LON">
     <remote-site name="NYC" stack="tcp" cluster="global"/>
     <remote-site name="SFO" stack="tcp" cluster="global"/>
     <property name="relay_multicasts">false</property>
  </relay>
 </stack>
</subsystem>
```

   The **RELAY** protocol creates an additional stack (running parallel to the existing **TCP** stack) to communicate with the remote site. If a **TCP** based stack is used for the local cluster, two **TCP** based stack configurations are required: one for local communication and one to connect to the remote site. For an illustration, see Section 27.2, “Cross-Datacenter Replication Operations”

2. **Set Up Sites**
   Use the following configuration in the `standalone.xml` file to set up sites for each distributed cache in the cluster:

```
<distributed-cache>
     ...
     <backups>
        <backup site="{FIRSTSITENAME}" strategy="{SYNC/ASYNC}" />
        <backup site="{SECONDSITENAME}" strategy="{SYNC/ASYNC}" />
     </backups>
</distributed-cache>
```

3. **Configure Local Site Transport**
   Add the name of the local site in the `transport` element to configure transport:

```
<transport executor="infinispan-transport"
           lock-timeout="60000"
           cluster="LON"
           stack="udp"/>
```

Report a bug

27.3.2. Configure Cross-Data Replication (Library Mode)

27.3.2.1. Configure Cross-Datacenter Replication Declaratively

When configuring Cross-Datacenter Replication, the `relay.RELAY2` protocol creates an additional stack (running parallel to the existing **TCP** stack) to communicate with the remote site. If a **TCP**-based stack is used for the local cluster, two **TCP** based stack configurations are required: one for local communication and one to connect to the remote site.

In JBoss Data Grid's Library mode, cross-datacenter replication is set up as follows:

Procedure 27.2. Setting Up Cross-Datacenter Replication

1. **Configure the Local Site**
   Add the `site` element to the `global` element to add the local site (in this example, the local site is named `LON`).

```
<infinispan>
   <global>
      ...
      <site local="LON" />
      ...
   </global>
</infinispan>
```

2. **Configure JGroups for the Local Site**
   Cross-site replication requires a non-default JGroups configuration. Add the `transport` element and set up the path to the configuration file as the ***`configurationFile`*** property. In this example, the JGroups configuration file is named `jgroups-with-relay.xml`.

```
<infinispan>
   <global>
      ...
      <site local="LON" />
      <transport clusterName="default">
         <properties>
             <property name="configurationFile" value="jgroups-
with-relay.xml" />
          </properties>
      </transport>
      ...
   </global>
</infinispan>
```

3. **Configure the LON Cache**
   Configure the cache in site **LON** to back up to the sites **NYC** and **SFO**:

```
<infinispan>
   <global>
      <site local="LON" />
      ...
   </global>
   ...
   <namedCache name="lon">
      <sites>
         <backups>
            <backup site="NYC"
      strategy="SYNC"
      backupFailurePolicy="WARN" />
            <backup site="SFO"
      strategy="ASYNC"
      backupFailurePolicy="IGNORE"/>
         </backups>
      </sites>
   </namedCache>
</infinispan>
```

4. **Configure the Back Up Caches**

   a. Configure the cache in site **NYC** to receive back up data from **LON**:

```
<infinispan>
   <global>
      <site local="NYC" />
      ...
   </global>
   ...
   <namedCache name="lonBackup">
      <sites>
         <backupFor remoteSite="LON"
      remoteCache="lon" />
      </sites>
   </namedCache>
</infinispan>
```

   b. Configure the cache in site **SFO** to receive back up data from **LON**:

```
<infinispan>
   <global>
      <site local="SFO" />
      ...
   </global>
   ...
   <namedCache name="lonBackup">
      <sites>
         <backupFor remoteSite="LON"
      remoteCache="lon" />
      </sites>
   </namedCache>
</infinispan>
```

5. **Add the Contents of the Configuration File**
   As a default, Red Hat JBoss Data Grid includes JGroups configuration files such as `jgroups-tcp.xml` and `jgroups-udp.xml` in the `infinispan-core-{VERSION}.jar` package.

   Copy the JGroups configuration to a new file (in this example, it is named `jgroups-with-relay.xml`) and add the provided configuration information to this file. Note that the `relay.RELAY2` protocol configuration must be the last protocol in the configuration stack.

```
<config>
    ...
    <relay.RELAY2 site="LON"
              config="relay.xml"
              relay_multicasts="false" />
</config>
```

6. **Configure the relay.xml File**
   Set up the `relay.RELAY2` configuration in the `relay.xml` file. This file describes the global cluster configuration.

```
<RelayConfiguration>
    <sites>
        <site name="LON"
              id="0">
            <bridges>
                <bridge config="jgroups-global.xml"
                        name="global"/>
                </bridges>
        </site>
        <site name="NYC"
              id="1">
            <bridges>
                <bridge config="jgroups-global.xml"
                        name="global"/>
                </bridges>
        </site>
        <site name="SFO"
              id="2">
            <bridges>
                <bridge config="jgroups-global.xml"
                        name="global"/>
            </bridges>
        </site>
    </sites>
</RelayConfiguration>
```

7. **Configure the Global Cluster**
   The file `jgroups-global.xml` referenced in `relay.xml` contains another JGroups configuration which is used for the global cluster: communication between sites.

   The global cluster configuration is usually `TCP`-based and uses the `TCPPING` protocol (instead of `PING` or `MPING`) to discover members. Copy the contents of `jgroups-tcp.xml` into `jgroups-global.xml` and add the following configuration in order to configure `TCPPING`:

```
<config>
```

```
    <TCP bind_port="7800" ... />
    <TCPPING
initial_hosts="lon.hostname[7800],nyc.hostname[7800],sfo.hostname[78
00]"
             num_initial_members="3"
             ergonomics="false" />
          <!-- Rest of the protocols -->
</config>
```

Replace the hostnames (or IP addresses) in ***TCPPING.initial_hosts*** with those used for your site masters. The ports (**7800** in this example) must match the ***TCP.bind_port***.

For more information about the **TCPPING** protocol, see Section 24.3.1.3, "Using the TCPPing Protocol"

Report a bug

27.3.2.2. Configure Cross-Datacenter Replication Programmatically

The programmatic method to configure cross-datacenter replication in Red Hat JBoss Data Grid is as follows:

Procedure 27.3. Configure Cross-Datacenter Replication Programmatically

1. **Identify the Node Location**
   Declare the site the node resides in:

   ```
   globalConfiguration.site().localSite("LON");
   ```

2. **Configure JGroups**
   Configure JGroups to use the **RELAY** protocol:

   ```
   globalConfiguration.transport().addProperty("configurationFile",
   jgroups-with-relay.xml);
   ```

3. **Set Up the Remote Site**
   Set up JBoss Data Grid caches to replicate to the remote site:

   ```
   ConfigurationBuilder lon = new ConfigurationBuilder();
   lon.sites().addBackup()
         .site("NYC")
         .backupFailurePolicy(BackupFailurePolicy.WARN)
         .strategy(BackupConfiguration.BackupStrategy.SYNC)
         .replicationTimeout(12000)
         .sites().addInUseBackupSite("NYC")
       .sites().addBackup()
         .site("SFO")
         .backupFailurePolicy(BackupFailurePolicy.IGNORE)
         .strategy(BackupConfiguration.BackupStrategy.ASYNC)
         .sites().addInUseBackupSite("SFO")
   ```

4. **Optional: Configure the Backup Caches**
   JBoss Data Grid implicitly replicates data to a cache with same name as the remote site. If a backup cache on the remote site has a different name, users must specify a ***backupFor*** cache

to ensure data is replicated to the correct cache.

NOTE

This step is optional and only required if the remote site's caches are named differently from the original caches.

a. Configure the cache in site **NYC** to receive backup data from **LON**:

```
ConfigurationBuilder NYCbackupOfLon = new ConfigurationBuilder();
lonBackup.sites().backupFor().remoteCache("lon").remoteSite("LON"
);
```

b. Configure the cache in site **SFO** to receive backup data from **LON**:

```
ConfigurationBuilder SFObackupOfLon = new ConfigurationBuilder();
lonBackup.sites().backupFor().remoteCache("lon").remoteSite("LON"
);
```

5. **Add the Contents of the Configuration File**
As a default, Red Hat JBoss Data Grid includes JGroups configuration files such as **`jgroups-tcp.xml`** and **`jgroups-udp.xml`** in the **`infinispan-core-{VERSION}.jar`** package.

Copy the JGroups configuration to a new file (in this example, it is named **`jgroups-with-relay.xml`**) and add the provided configuration information to this file. Note that the **`relay.RELAY2`** protocol configuration must be the last protocol in the configuration stack.

```
<config>
    ...
    <relay.RELAY2 site="LON"
              config="relay.xml"
              relay_multicasts="false" />
</config>
```

6. **Configure the relay.xml File**
Set up the **`relay.RELAY2`** configuration in the **`relay.xml`** file. This file describes the global cluster configuration.

```
<RelayConfiguration>
    <sites>
        <site name="LON"
              id="0">
            <bridges>
                <bridge config="jgroups-global.xml"
                        name="global"/>
                </bridges>
        </site>
        <site name="NYC"
              id="1">
            <bridges>
                <bridge config="jgroups-global.xml"
                        name="global"/>
                </bridges>
        </site>
        <site name="SFO"
              id="2">
            <bridges>
                <bridge config="jgroups-global.xml"
                        name="global"/>
            </bridges>
        </site>
    </sites>
</RelayConfiguration>
```

7. **Configure the Global Cluster**
   The file **jgroups-global.xml** referenced in **relay.xml** contains another JGroups configuration which is used for the global cluster: communication between sites.

   The global cluster configuration is usually **TCP**-based and uses the **TCPPING** protocol (instead of **PING** or **MPING**) to discover members. Copy the contents of **jgroups-tcp.xml** into **jgroups-global.xml** and add the following configuration in order to configure **TCPPING**:

```
<config>
    <TCP bind_port="7800" ... />
    <TCPPING
initial_hosts="lon.hostname[7800],nyc.hostname[7800],sfo.hostname[78
00]"
             num_initial_members="3"
             ergonomics="false" />
          <!-- Rest of the protocols -->
</config>
```

   Replace the hostnames (or IP addresses) in ***TCPPING.initial_hosts*** with those used for your site masters. The ports (**7800** in this example) must match the ***TCP.bind_port***.

   For more information about the **TCPPING** protocol, see Section 24.3.1.3, "Using the TCPPing Protocol"

Report a bug

27.4. TAKING A SITE OFFLINE

In Red Hat JBoss Data Grid's Cross-datacenter replication configuration, if backing up to one site fails a certain number of times during a time interval, that site can be marked as offline automatically. This feature removes the need for manual intervention by an administrator to mark the site as offline.

It is possible to configure JBoss Data Grid to take down a site automatically when specified confiscations are met, or for an administrator to manually take down a site:

- Configure automatically taking a site offline:
  - Declaratively in Remote Client-Server mode.
  - Declaratively in Library mode.
  - Using the programmatic method.
- Manually taking a site offline:

- Using JBoss Operations Network (JON).
- Using the JBoss Data Grid Command Line Interface (CLI).

Report a bug

27.4.1. Taking a Site Offline (Remote Client-Server Mode)

In Red Hat JBoss Data Grid's Remote Client-Server mode, the **take-offline** element is added to the **backup** element to configure when a site is automatically taken offline. An example of this configuration is as follows:

```
<backup>
 <take-offline after-failures="${NUMBER}"
        min-wait="${PERIOD}" />
</backup>
```

The **take-offline** element use the following parameters to configure when to take a site offline:

- The ***after-failures*** parameter specifies the number of times attempts to contact a site can fail before the site is taken offline.
- The ***min-wait*** parameter specifies the number (in milliseconds) to wait to mark an unresponsive site as offline. The site is offline when the ***min-wait*** period elapses after the first attempt, and the number of failed attempts specified in the ***after-failures*** parameter occur.

Report a bug

27.4.2. Taking a Site Offline (Library Mode)

In Red Hat JBoss Data Grid's Library mode, use the **backupFor** element after defining all back up sites within the **backups** element:

```
<backup>
        <takeOffline afterFailures="${NUM}"
                     minTimeToWait="${PERIOD}"/>
</backup>
```

Add the **takeOffline** element to the **backup** element to configure automatically taking a site offline.

- The ***afterFailures*** parameter specifies the number of times attempts to contact a site can fail before the site is taken offline. The default value (**0**) allows an infinite number of failures if ***minTimeToWait*** is less than **0**. If the ***minTimeToWait*** is not less than **0**, ***afterFailures*** behaves as if the value is negative. A negative value for this parameter indicates that the site is taken offline after the time specified by ***minTimeToWait*** elapses.
- The ***minTimeToWait*** parameter specifies the number (in milliseconds) to wait to mark an unresponsive site as offline. The site is taken offline after the number attempts specified in the ***afterFailures*** parameter conclude and the time specified by ***minTimeToWait*** after the first failure has elapsed. If this parameter is set to a value smaller than or equal to **0**, this parameter is disregarded and the site is taken offline based solely on the ***afterFailures*** parameter.

Report a bug

27.4.3. Taking a Site Offline (Programmatically)

To configure taking a Cross-datacenter replication site offline automatically in Red Hat JBoss Data Grid programmatically:

```
lon.sites().addBackup()
      .site("NYC")
      .backupFailurePolicy(BackupFailurePolicy.FAIL)
      .strategy(BackupConfiguration.BackupStrategy.SYNC)
      .takeOffline()
         .afterFailures(500)
         .minTimeToWait(10000);
```

Report a bug

27.4.4. Taking a Site Offline via JBoss Operations Network (JON)

A site can be taken offline in Red Hat JBoss Data Grid using the JBoss Operations Network operations. For a list of the metrics, see Section 20.7.2, “JBoss Operations Network Plugin Operations”

Report a bug

27.4.5. Taking a Site Offline via the CLI

Use Red Hat JBoss Data Grid's Command Line Interface (CLI) to manually take a site from a cross-datacenter replication configuration down if it is unresponsive using the ***`site`*** command.

The **`site`** command can be used to check the status of a site as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --status
${SITENAME}
```

The result of this command would either be **`online`** or **`offline`** according to the current status of the named site.

The command can be used to bring a site online or offline by name as follows:

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --offline
${SITENAME}
```

```
[jmx://localhost:12000/MyCacheManager/namedCache]> site --online
${SITENAME}
```

If the command is successful, the output **ok** displays after the command.

For more information about the JBoss Data Grid CLI and its commands, see the *Developer Guide*'s chapter on the JBoss Data Grid Command Line Interface (CLI)

Report a bug

27.4.6. Bring a Site Back Online

After a site is taken offline, currently the only way to bring the site back online is using the JMX console to invoke the **`bringSiteOnline(`*`siteName`*`)`** operation on the **`XSiteAdmin`** MBean. For details about this MBean, see Section A.21, “XSiteAdmin”

Report a bug

27.5. CONFIGURE MULTIPLE SITE MASTERS

A standard Red Hat JBoss Data Grid cross-datacenter replication configuration includes one master node for each site. The master node is a gateway for other nodes to communicate with the master nodes at other sites.

This standard configuration works for a simple cross-datacenter replication configuration. However, with a larger volume of traffic between the sites, passing traffic through a single master node can create a bottleneck, which slows communication across nodes.

In JBoss Data Grid, configure multiple master nodes for each site to optimize traffic across multiple sites.

Report a bug

27.5.1. Multiple Site Master Operations

When multiple site masters are enabled and configured, the master nodes in each site joins the local cluster (i.e. the local site) as well as the global cluster (which includes nodes that are members of multiple sites).

Each node that acts as a site master and maintains a routing table that consists of a list of target sites and site masters. When a message arrives, a random master node for the destination site is selected. The message is then forwarded to the random master node, where it is sent to the destination node (unless the randomly selected node was the destination).

Report a bug

27.5.2. Configure Multiple Site Masters (Remote Client-Server Mode)

Prerequisites

Configure Cross-Datacenter Replication for Red Hat JBoss Data Grid's Remote Client-Server Mode.

Procedure 27.4. Set Multiple Site Masters in Remote Client-Server Mode

1. **Locate the Target Configuration**
   Locate the target site's configuration in the **`clustered-xsite.xml`** example configuration file. The sample configuration looks like the following example:

   ```
   <relay site="LON">
    <remote-site name="NYC" stack="tcp" cluster="global"/>
    <remote-site name="SFO" stack="tcp" cluster="global"/>
    <property name="relay_multicasts">false</property>
   </relay>
   ```

2. **Configure Maximum Sites**
   Use the ***`max_site_masters`*** property to determine the maximum number of master nodes within the site. Set this value to the number of nodes in the site to make every node a master.

```
<relay site="LON">
 <remote-site name="NYC" stack="tcp" cluster="global"/>
 <remote-site name="SFO" stack="tcp" cluster="global"/>
 <property name="relay_multicasts">false</property>
 <property name="max_site_masters">16</property>
</relay>
```

3. **Configure Site Master**
   Use the ***`can_become_site_master`*** property to allow the node to become the site master. This flag is set to **`true`** as a default. Setting this flag to **`false`** prevents the node from becoming a site master. This is required in situations where the node does not have a network interface connected to the external network.

```
<relay site="LON">
 <remote-site name="NYC" stack="tcp" cluster="global"/>
 <remote-site name="SFO" stack="tcp" cluster="global"/>
 <property name="relay_multicasts">false</property>
 <property name="max_site_masters">16</property>
 <property name="can_become_site_master">true</property>
</relay>
```

Report a bug

27.5.3. Configure Multiple Site Masters (Library Mode)

To configure multiple site masters in Red Hat JBoss Data Grid's Library Mode:

Procedure 27.5. Configure Multiple Site Masters (Library Mode)

1. **Configure Cross-Datacenter Replication**
   Configure Cross-Datacenter Replication in JBoss Data Grid. Use the instructions in Section 27.3.2.1, "Configure Cross-Datacenter Replication Declaratively" for an XML configuration or the instructions in Section 27.3.2.2, "Configure Cross-Datacenter Replication Programmatically" for a programmatic configuration.

2. **Add the Contents of the Configuration File**
   Add the ***`can_become_site_master`*** and ***`max_site_masters`*** parameters to the configuration as follows:

```
<config>
    ...
    <relay.RELAY2 site="LON"
              config="relay.xml"
              relay_multicasts="false"
              can_become_site_master="true"
              max_site_masters="16"/>
</config>
```

   Set the ***`max_site_masters`*** value to the number of nodes in the cluster to make all nodes masters.

Report a bug

APPENDIX A. JMX MBEANS IN REDHAT JBOSS DATA GRID

A.1. ACTIVATION

`org.infinispan.eviction.ActivationManagerImpl`

Activates entries that have been passivated to the CacheStore by loading the entries into memory.

Table A.1. Attributes

Name	Description	Type	Writable
activations	Number of activation events.	String	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table A.2. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.2. CACHE

`org.infinispan.CacheImpl`

The Cache component represents an individual cache instance.

Table A.3. Attributes

Name	Description	Type	Writable
cacheName	Returns the cache name.	String	No
cacheStatus	Returns the cache status.	String	No
configurationAsProperties	Returns the cache configuration in form of properties.	java.util.Properties	No

Name	Description	Type	Writable
version	Returns the version of Infinispan	java.lang.String	No

Table A.4. Operations

Name	Description	Signature
start	Starts the cache.	void start()
stop	Stops the cache.	void stop()
clear	Clears the cache.	void clear()

Report a bug

A.3. CACHELOADER

`org.infinispan.interceptors.CacheLoaderInterceptor`

This component loads entries from a CacheStore into memory.

Table A.5. Attributes

Name	Description	Type	Writable
cacheLoaderLoads	Number of entries loaded from the cache store.	long	No
cacheLoaderMisses	Number of entries that did not exist in cache store.	long	No
stores	Returns a collection of cache loader types which are configured and enabled.	Collection	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table A.6. Operations

Name	Description	Signature
disableStore	Disable all cache loaders of a given type, where type is a fully qualified class name of the cache loader to disable.	void disableStore(String storeType)
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.4. CACHEMANAGER

`org.infinispan.manager.DefaultCacheManager`

The CacheManager component acts as a manager, factory, and container for caches in the system.

Table A.7. Attributes

Name	Description	Type	Writable
cacheManagerStatus	The status of the cache manager instance.	String	No
clusterMembers	Lists members in the cluster.	String	No
clusterName	Cluster name.	String	No
clusterSize	Size of the cluster in the number of nodes.	int	No
createdCacheCount	The total number of created caches, including the default cache.	String	No
definedCacheCount	The total number of defined caches, excluding the default cache.	String	No
definedCacheNames	The defined cache names and their statuses. The default cache is not included in this representation.	String	No

Name	Description	Type	Writable
name	The name of this cache manager.	String	No
nodeAddress	The network address associated with this instance.	String	No
physicalAddresses	The physical network addresses associated with this instance.	String	No
runningCacheCount	The total number of running caches, including the default cache.	String	No
version	Infinispan version.	String	No.

Table A.8. Operations

Name	Description	Signature
startCache	Starts the default cache associated with this cache manager.	void startCache()
startCache	Starts a named cache from this cache manager.	void startCache (String p0)

Report a bug

A.5. CACHESTORE

`org.infinispan.interceptors.CacheWriterInterceptor`

The CacheStore component stores entries to a CacheStore from memory.

Table A.9. Attributes

Name	Description	Type	Writable
writesToTheStores	Number of writes to the store.	long	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table A.10. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.6. DEADLOCKDETECTINGLOCKMANAGER

`org.infinispan.util.concurrent.locks.DeadlockDetectingLockManager`

This component provides information about the number of deadlocks that were detected.

Table A.11. Attributes

Name	Description	Type	Writable
detectedLocalDeadlocks	Number of local transactions that were rolled back due to deadlocks.	long	No
detectedRemoteDeadlocks	Number of remote transactions that were rolled back due to deadlocks.	long	No
overlapWithNotDeadlockAwareLockOwners	Number of situations when we try to determine a deadlock and the other lock owner is NOT a transaction. In this scenario we cannot run the deadlock detection mechanism.	long	No
totalNumberOfDetectedDeadlocks	Total number of local detected deadlocks.	long	No
concurrencyLevel	The concurrency level that the MVCC Lock Manager has been configured with.	int	No
numberOfLocksAvailable	The number of exclusive locks that are available.	int	No

Name	Description	Type	Writable
numberOfLocksHeld	The number of exclusive locks that are held.	int	No

Table A.12. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.7. DISTRIBUTIONMANAGER

`org.infinispan.distribution.DistributionManagerImpl`

The DistributionManager component handles the distribution of content across a cluster.

NOTE

The DistrubutionManager component is only available in clustered mode.

Table A.13. Operations

Name	Description	Signature
isAffectedByRehash	Determines whether a given key is affected by an ongoing rehash.	boolean isAffectedByRehash(Object p0)
isLocatedLocally	Indicates whether a given key is local to this instance of the cache. Only works with String keys.	boolean isLocatedLocally(String p0)
locateKey	Locates an object in a cluster. Only works with String keys.	List locateKey(String p0)

Report a bug

A.8. INTERPRETER

`org.infinispan.cli.interpreter.Interpreter`

The Interpreter component executes command line interface (CLI operations).

Table A.14. Attributes

Name	Description	Type	Writable
cacheNames	Retrieves a list of caches for the cache manager.	String[]	No

Table A.15. Operations

Name	Description	Signature
createSessionId	Creates a new interpreter session.	String createSessionId(String cacheName)
execute	Parses and executes IspnCliQL statements.	String execute(String p0, String p1)

Report a bug

A.9. INVALIDATION

`org.infinispan.interceptors.InvalidationInterceptor`

The Invalidation component invalidates entries on remote caches when entries are written locally.

Table A.16. Attributes

Name	Description	Type	Writable
invalidations	Number of invalidations.	long	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table A.17. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.10. LOCKMANAGER

`org.infinispan.util.concurrent.locks.LockManagerImpl`

The LockManager component handles MVCC locks for entries.

Table A.18. Attributes

Name	Description	Type	Writable
concurrencyLevel	The concurrency level that the MVCC Lock Manager has been configured with.	int	No
numberOfLocksAvailabl e	The number of exclusive locks that are available.	int	No
numberOfLocksHeld	The number of exclusive locks that are held.	int	No

Report a bug

A.11. LOCALTOPOLOGYMANAGER

`org.infinispan.topology.LocalTopologyManagerImpl`

The LocalTopologyManager component controls the cache membership and state transfer in Red Hat JBoss Data Grid.

NOTE

The LocalTopologyManager component is only available in clustered mode.

Table A.19. Attributes

Name	Description	Type	Writable
rebalancingEnabled	If false, newly started nodes will not join the existing cluster nor will the state be transferred to them. If any of the current cluster members are stopped when rebalancing is disabled, the nodes will leave the cluster but the state will not be rebalanced among the remaining nodes. This will result in fewer copies than specified by the **`numOwners`** attribute until rebalancing is enabled again.	boolean	Yes

Report a bug

A.12. MASSINDEXER

`org.infinispan.query.MassIndexer`

The MassIndexer component rebuilds the index using cached data.

Table A.20. Operations

Name	Description	Signature
start	Starts rebuilding the index.	void start()

NOTE

This operation is available only for caches with indexing enabled. For more information, see the Red Hat JBoss Data Grid *Infinispan Query Guide*

Report a bug

A.13. PASSIVATION

`org.infinispan.interceptors.PassivationInterceptor`

The Passivation component handles the passivation of entries to a CacheStore on eviction.

Table A.21. Attributes

Name	Description	Type	Writable
passivations	Number of passivation events.	String	No
statisticsEnabled	Enables or disables the gathering of statistics by this component	boolean	Yes

Table A.22. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.14. RECOVERYADMIN

`org.infinispan.transaction.xa.recovery.RecoveryAdminOperations`

The RecoveryAdmin component exposes tooling for handling transaction recovery.

Table A.23. Operations

Name	Description	Signature
forceCommit	Forces the commit of an in-doubt transaction.	String forceCommit(long p0)
forceCommit	Forces the commit of an in-doubt transaction	String forceCommit(int p0, byte[] p1, byte[] p2)
forceRollback	Forces the rollback of an in-doubt transaction.	String forceRollback(long p0)
forceRollback	Forces the rollback of an in-doubt transaction	String forceRollback(int p0, byte[] p1, byte[] p2)
forget	Removes recovery info for the given transaction.	String forget(long p0)
forget	Removes recovery info for the given transaction.	String forget(int p0, byte[] p1, byte[] p2)
showInDoubtTransactions	Shows all the prepared transactions for which the originating node crashed.	String showInDoubtTransactions()

Report a bug

A.15. ROLLINGUPGRADEMANAGER

`org.infinispan.upgrade.RollingUpgradeManager`

The RollingUpgradeManager component handles the control hooks in order to migrate data from one version of Red Hat JBoss Data Grid to another.

Table A.24. Operations

Name	Description	Signature
disconnectSource	Disconnects the target cluster from the source cluster according to the specified migrator.	void disconnectSource(String p0)
recordKnownGlobalKeyset	Dumps the global known keyset to a well-known key for retrieval by the upgrade process.	void recordKnownGlobalKeyset()

Name	Description	Signature
synchronizeData	Synchronizes data from the old cluster to this using the specified migrator.	long synchronizeData(String p0)

Report a bug

A.16. RPCMANAGER

`org.infinispan.remoting.rpc.RpcManagerImpl`

The RpcManager component manages all remote calls to remote cache instances in the cluster.

NOTE

The RpcManager component is only available in clustered mode.

Table A.25. Attributes

Name	Description	Type	Writable
averageReplicationTime	The average time spent in the transport layer, in milliseconds.	long	No
committedViewAsString	Retrieves the committed view.	String	No
pendingViewAsString	Retrieves the pending view.	String	No
replicationCount	Number of successful replications.	long	No
replicationFailures	Number of failed replications.	long	No
successRatio	Successful replications as a ratio of total replications.	String	No
successRatioFloatingPoint	Successful replications as a ratio of total replications in numeric double format.	double	No

Name	Description	Type	Writable
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table A.26. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()
setStatisticsEnabled	Whether statistics should be enabled or disabled (true/false)	void setStatisticsEnabled(boolean enabled)

Report a bug

A.17. STATETRANSFERMANAGER

`org.infinispan.statetransfer.StateTransferManager`

The StateTransferManager component handles state transfer in Red Hat JBoss Data Grid.

NOTE

The StateTransferManager component is only available in clustered mode.

Table A.27. Attributes

Name	Description	Type	Writable
joinComplete	If true, the node has successfully joined the grid and is considered to hold state. If false, the join process is still in progress..	boolean	No
stateTransferInProgres s	Checks whether there is a pending inbound state transfer on this cluster member.	boolean	No

Report a bug

A.18. STATISTICS

`org.infinispan.interceptors.CacheMgmtInterceptor`

This component handles general statistics such as timings, hit/miss ratio, etc.

Table A.28. Attributes

Name	Description	Type	Writable
averageReadTime	Average number of milliseconds for a read operation on the cache.	long	No
averageWriteTime	Average number of milliseconds for a write operation in the cache.	long	No
elapsedTime	Number of seconds since cache started.	long	No
evictions	Number of cache eviction operations.	long	No
hitRatio	Percentage hit/(hit+miss) ratio for the cache.	double	No
hits	Number of cache attribute hits.	long	No
misses	Number of cache attribute misses.	long	No
numberOfEntries	Number of entries currently in the cache.	int	No
readWriteRatio	Read/writes ratio for the cache.	double	No
removeHits	Number of cache removal hits.	long	No
removeMisses	Number of cache removals where keys were not found.	long	No
stores	Number of cache attribute PUT operations.	long	No

Name	Description	Type	Writable
timeSinceReset	Number of seconds since the cache statistics were last reset.	long	No

Table A.29. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.19. TRANSACTIONS

`org.infinispan.interceptors.TxInterceptor`

The Transactions component manages the cache's participation in JTA transactions.

Table A.30. Attributes

Name	Description	Type	Writable
commits	Number of transaction commits performed since last reset.	long	No
prepares	Number of transaction prepares performed since last reset.	long	No
rollbacks	Number of transaction rollbacks performed since last reset.	long	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table A.31. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.20. TRANSPORT

org.infinispan.server.core.transport.NettyTransport

The Transport component manages read and write operations to and from the server.

Table A.32. Attributes

Name	Description	Type	Writable
hostName	Returns the host to which the transport binds.	String	No
idleTimeout	Returns the idle timeout.	String	No
numberOfGlobalConnections	Returns a count of active connections in the cluster. This operation will make remote calls to aggregate results, so latency may have an impact on the speed of calculation for this attribute.	Integer	false
numberOfLocalConnections	Returns a count of active connections this server.	Integer	No
numberWorkerThreads	Returns the number of worker threads.	String	No
port	Returns the port to which the transport binds.	String	
receiveBufferSize	Returns the receive buffer size.	String	No
sendBufferSize	Returns the send buffer size.	String	No
totalBytesRead	Returns the total number of bytes read by the server from clients, including both protocol and user information.	String	No

Name	Description	Type	Writable
totalBytesWritten	Returns the total number of bytes written by the server back to clients, including both protocol and user information.	String	No
tcpNoDelay	Returns whether TCP no delay was configured or not.	String	No

Report a bug

A.21. XSITEADMIN

`org.infinispan.xsite.XSiteAdminOperations`

The XSiteAdmin component exposes tooling for backing up data to remote sites.

Table A.33. Operations

Name	Description	Signature
bringSiteOnline	Brings the given site back online on all the cluster.	String bringSiteOnline(String p0)
amendTakeOffline	Amends the values for 'TakeOffline' functionality on all the nodes in the cluster.	String amendTakeOffline(String p0, int p1, long p2)
getTakeOfflineAfterFailures	Returns the value of the 'afterFailures' for the 'TakeOffline' functionality.	String getTakeOfflineAfterFailures(Strin g p0)
getTakeOfflineMinTimeToWait	Returns the value of the 'minTimeToWait' for the 'TakeOffline' functionality.	String getTakeOfflineMinTimeToWait(St ring p0)
setTakeOfflineAfterFailures	Amends the values for 'afterFailures' for the 'TakeOffline' functionality on all the nodes in the cluster.	String setTakeOfflineAfterFailures(Strin g p0, int p1)
setTakeOfflineMinTimeToWait	Amends the values for 'minTimeToWait' for the 'TakeOffline' functionality on all the nodes in the cluster.	String setTakeOfflineMinTimeToWait(St ring p0, long p1)

Name	Description	Signature
siteStatus	Check whether the given backup site is offline or not.	String siteStatus(String p0)
status	Returns the the status(offline/online) of all the configured backup sites.	String status()
takeSiteOffline	Takes this site offline in all nodes in the cluster.	String takeSiteOffline(String p0)

Report a bug

APPENDIX B. REFERENCES

B.1. ABOUT CONSISTENCY

Consistency is the policy that states whether it is possible for a data record on one node to vary from the same data record on another node.

For example, due to network speeds, it is possible that a write operation performed on the master node has not yet been performed on another node in the store, a strong consistency guarantee will ensure that data which is not yet fully replicated is not returned to the application.

Report a bug

B.2. ABOUT CONSISTENCY GUARANTEE

Despite the locking of a single owner instead of all owners, Red Hat JBoss Data Grid's consistency guarantee remains intact. Consider the following situation:

1. If Key **K** is hashed to nodes `{A,B}` and transaction **TX1** acquires a lock for **K** on, for example, node **A** and

2. If another cache access occurs on node **B**, or any other node, and **TX2** attempts to lock **K**, this access attempt fails with a timeout because the transaction **TX1** already holds a lock on `K`.

This lock acquisition attempt always fails because the lock for key **K** is always deterministically acquired on the same node of the cluster, irrespective of the transaction's origin.

Report a bug

B.3. ABOUT JBOSS CACHE

Red Hat JBoss Cache is a tree-structured, clustered, transactional cache that can also be used in a standalone, non-clustered environment. It caches frequently accessed data in-memory to prevent data retrieval or calculation bottlenecks that occur while enterprise features such as Java Transactional API (JTA) compatibility, eviction and persistence are provided.

JBoss Cache is the predecessor to Infinispan and Red Hat JBoss Data Grid.

Report a bug

B.4. ABOUT RELAY2

The **RELAY** protocol bridges two remote clusters by creating a connection between one node in each site. This allows multicast messages sent out in one site to be relayed to the other and vice versa.

JGroups includes the **RELAY2** protocol, which is used for communication between sites in Red Hat JBoss Data Grid's Cross-Site Replication.

The **RELAY2** protocol works similarly to **RELAY** but with slight differences. Unlike **RELAY**, the **RELAY2** protocol:

- connects more than two sites.

- connects sites that operate autonomously and are unaware of each other.

- offers both unicasts and multicast routing between sites.

Report a bug

B.5. ABOUT RETURN VALUES

Values returned by cache operations are referred to as return values. In Red Hat JBoss Data Grid, these return values remain reliable irrespective of which cache mode is employed and whether synchronous or asynchronous communication is used.

Report a bug

B.6. ABOUT RUNNABLE INTERFACES

A Runnable Interface (also known as a Runnable) declares a single `run()` method, which executes the active part of the class' code. The Runnable object can be executed in its own thread after it is passed to a thread constructor.

Report a bug

B.7. ABOUT TWO PHASE COMMIT (2PC)

A Two Phase Commit protocol (2PC) is a consensus protocol used to atomically commit or roll back distributed transactions. It is successful when faced with cases of temporary system failures, including network node and communication failures, and is therefore widely utilized.

Report a bug

B.8. ABOUT KEY-VALUE PAIRS

A key-value pair (KVP) is a set of data consisting of a key and a value.

- A key is unique to a particular data entry and is composed from data attributes of the particular entry it relates to.

- A value is the data assigned to and identified by the key.

Report a bug

B.9. THE EXTERNALIZER

B.9.1. About Externalizer

An `Externalizer` is a class that can:

- Marshall a given object type to a byte array.

- Unmarshall the contents of a byte array into an instance of the object type.

Externalizers are used by Red Hat JBoss Data Grid and allow users to specify how their object types are serialized. The marshalling infrastructure used in JBoss Data Grid builds upon JBoss Marshalling and provides efficient payload delivery and allows the stream to be cached. The stream caching allows data to be accessed multiple times, whereas normally a stream can only be read once.

Report a bug

B.9.2. Internal Externalizer Implementation Access

Externalizable objects should not access Red Hat JBoss Data Grids Externalizer implementations. The following is an example of incorrect usage:

```
public static class ABCMarshallingExternalizer implements
AdvancedExternalizer<ABCMarshalling> {
   @Override
   public void writeObject(ObjectOutput output, ABCMarshalling object)
throws IOException {
      MapExternalizer ma = new MapExternalizer();
      ma.writeObject(output, object.getMap());
   }

   @Override
   public ABCMarshalling readObject(ObjectInput input) throws IOException,
ClassNotFoundException {
      ABCMarshalling hi = new ABCMarshalling();
      MapExternalizer ma = new MapExternalizer();
      hi.setMap((ConcurrentHashMap<Long, Long>) ma.readObject(input));
      return hi;
   }

   ...
```

End user externalizers do not need to interact with internal externalizer classes. The following is an example of correct usage:

```
public static class ABCMarshallingExternalizer implements
AdvancedExternalizer<ABCMarshalling> {
   @Override
   public void writeObject(ObjectOutput output, ABCMarshalling object)
throws IOException {
      output.writeObject(object.getMap());
   }

   @Override
   public ABCMarshalling readObject(ObjectInput input) throws IOException,
ClassNotFoundException {
      ABCMarshalling hi = new ABCMarshalling();
      hi.setMap((ConcurrentHashMap<Long, Long>) input.readObject());
      return hi;
   }

   ...
}
```

Report a bug

B.10. HASH SPACE ALLOCATION

B.10.1. About Hash Space Allocation

Red Hat JBoss Data Grid is responsible for allocating a portion of the total available hash space to each node. During subsequent operations that must store an entry, JBoss Data Grid creates a hash of the relevant key and stores the entry on the node that owns that portion of hash space.

Report a bug

B.10.2. Locating a Key in the Hash Space

Red Hat JBoss Data Grid always uses an algorithm to locate a key in the hash space. As a result, the node that stores the key is never manually specified. This scheme allows any node to know which node owns a particular key without such ownership information being distributed. This scheme reduces the amount of overhead and, more importantly, improves redundancy because the ownership information does not need to be replicated in case of node failure.

Report a bug

B.10.3. Requesting a Full Byte Array

How can I request the Red Hat JBoss Data Grid return a full byte array instead of partial byte array contents?

As a default, JBoss Data Grid only partially prints byte arrays to logs to avoid unnecessarily printing large byte arrays. This occurs when either:

- JBoss Data Grid caches are configured for lazy deserialization. Lazy deserialization is not available in JBoss Data Grid's Remote Client-Server mode.
- A **`Memcached`** or **`Hot Rod`** server is run.

In such cases, only the first ten positions of the byte array display in the logs. To display the complete contents of the byte array in the logs, pass the ***`-Dinfinispan.arrays.debug=true`*** system property at start up.

Example B.1. Partial Byte Array Log

```
2010-04-14 15:46:09,342 TRACE [ReadCommittedEntry] (HotRodWorker-1-1)
Updating entry
(key=CacheKey{data=ByteArray{size=19, hashCode=1b3278a,
array=[107, 45, 116, 101, 115, 116, 82, 101, 112, 108, ..]}}
removed=false valid=true changed=true created=true
value=CacheValue{data=ByteArray{size=19,
array=[118, 45, 116, 101, 115, 116, 82, 101, 112, 108, ..]},
version=281483566645249}]
And here's a log message where the full byte array is shown:
2010-04-14 15:45:00,723 TRACE [ReadCommittedEntry] (Incoming-
2,Infinispan-Cluster,eq-6834) Updating entry
(key=CacheKey{data=ByteArray{size=19, hashCode=6cc2a4,
array=[107, 45, 116, 101, 115, 116, 82, 101, 112, 108, 105, 99, 97, 116,
101, 100, 80, 117, 116]}}
removed=false valid=true changed=true created=true
value=CacheValue{data=ByteArray{size=19,
array=[118, 45, 116, 101, 115, 116, 82, 101, 112, 108, 105, 99, 97, 116,
101, 100, 80, 117, 116]},
version=281483566645249}]
```

Report a bug

APPENDIX C. REVISION HISTORY

Revision 6.2.1-7 **Thu Sep 04 2014** **Rakesh Ghatvisave**

BZ-1122298: Updated revision number to reflect the change.

Revision 6.2.1-6 **Wed Sep 03 2014** **Rakesh Ghatvisave**

BZ-1122298: Added multiple XML schemas in JDBC cache store configurations.

Revision 6.2.1-5 **Wed Aug 06 2014** **Rakesh Ghatvisave**

BZ-1122292: Removed Loader XML elements.

Revision 6.2.1-4 **Wed Apr 02 2014** **Misha Husnain Ali**

BZ-1069361: Added information about additional properties for S3_PING.

Revision 6.2.1-3 **Tue Mar 11 2014** **Mandar Joshi**

BZ-1072723: Corrected a typo in the topic "Eviction Strategies".

BZ-1072749: Rewrote the topic "LRU Eviction Algorithm Limitations" for more clarity.

BZ-1073911: Updated the code snippet in the topic "Configure Local Mode (Remote Client-Server Mode)".

BZ-1075278: Added missing word in sentence.

BZ-1075559: Changed instances of "Centre" to "Center."

Revision 6.2.1-2 **Thu Mar 06 2014** **Rakesh Ghatvisave**

BZ-997230: Added note for clustered mode only JMX attributes. Added LocalTopologyManager MBean to Appendix.

BZ-1071900: Added relay_multicasts=false attribute for Cross-Datacentre replication.

BZ-1063532: Added topic about stopping JBoss Data Grid instance via CLI.

BZ-1069361: Added a chapter about using S3_PING with JBoss Data Grid.

BZ-1069143: Modified JON topics and rearranged sections.

BZ-1061671: Corrected JON usage.

Revision 6.2.1-1 **Wed Feb 26 2014** **Misha Husnain Ali**

BZ-1069920: Removed an inconsistent line about the new FileCacheStore.

BZ-1068769: Added CLI tools part. This part has been moved here from the Developer Guide.

BZ-1069143: Removed JON content.

BZ-1061684: Word improvement.