Red Hat JBoss Data Grid 6.1

Administration and Configuration Guide

For use with Red Hat JBoss Data Grid 6.1

Edition 2

Last Updated: 2017-11-21

Red Hat JBoss Data Grid 6.1 Administration and Configuration Guide

For use with Red Hat JBoss Data Grid 6.1
Edition 2

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

Darrin Mison

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

Legal Notice

Abstract

This guide presents information about the administration and configuration of Red Hat JBoss Data Grid 6.1

Table of Contents

PREFACE

CHAPTER 1. JBOSS DATA GRID

1.1. ABOUT JBOSS DATA GRID

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

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

JBoss Data Grid 6.1 and JBoss Data Grid 6.1 Beta is based on Infinispan version 5.2.

Report a bug

1.2. JBOSS DATA GRID SUPPORTED CONFIGURATIONS

The set of supported features, configurations, and integrations for JBoss Data Grid (6.0 and 6.1) are available at the Supported Configurations page at https://access.redhat.com/knowledge/articles/115883.

Report a bug

1.3. JBOSS DATA GRID USAGE MODES

1.3.1. JBoss Data Grid Usage Modes

JBoss Data Grid offers two usage modes:

- Remote Client-Server mode
- Library mode

Report a bug

1.3.2. Remote Client-Server Mode

Remote Client-Server mode provides a managed, distributed and clusterable data grid server. Applications can remotely access the data grid server using **Hot Rod**, **Memcached** or **REST** client APIs.

All JBoss Data Grid operations in Remote Client-Server mode are non-transactional. As a result, a number of features cannot be performed when running JBoss Data Grid in Remote Client-Server mode.

However, there are a number of benefits to running JBoss Data Grid in Remote Client-Server mode if you do not require any features that require Library mode. Remote Client-Server mode is client language agnostic, provided there is a client library for your chosen protocol. As a result, Remote Client-Server mode provides:

- easier scaling of the data grid.

- easier upgrades of the data grid without impact on client applications.

Report a bug

1.3.3. Library Mode

Library mode allows the user to build and deploy a custom runtime environment. The Library usage mode hosts a single data grid node in the applications process, with remote access to nodes hosted in other JVMs. Tested containers for JBoss Data Grid 6 Library mode includes Tomcat 7 and JBoss Enterprise Application Platform 6. Library mode is supported in JBoss Data Grid 6.0.1.

A number of features in JBoss Data Grid can be used in Library mode, but not Remote Client-Server mode.

Use Library mode if you require:

- transactions.
- listeners and notifications.

Report a bug

1.4. JBOSS DATA GRID BENEFITS

JBoss Data Grid provides the following benefits:

Benefits of JBoss Data Grid

Performance

Accessing objects from local memory is faster than accessing objects from remote data stores (such as a database). JBoss Data Grid provides an efficient way to store in-memory objects coming from a slower data source, resulting in faster performance than a remote data store. JBoss Data Grid also offers optimization for both clustered and non clustered caches to further improve performance.

Consistency

Storing data in a cache carried the inherent risk: at the time it is accessed, the data may be outdated (stale). To address this risk, JBoss Data Grid uses mechanisms such as cache invalidation and expiration to remove stale data entries from the cache. Additionally, JBoss Data Grid supports JTA, distributed (XA) and two-phase commit transactions along with transaction recovery and a version API to remove or replace data according to saved versions.

Massive Heap and High Availability

In JBoss Data Grid, applications no longer need to delegate the majority of their data lookup processes to a large single server database for performance benefits. JBoss Data Grid employs techniques such as replication and distribution to completely remove the bottleneck that exists in the majority of current enterprise applications.

> **Example 1.1. Massive Heap and High Availability Example**
>
> In a sample grid with 16 blade servers, each node has 2 GB storage space dedicated for a replicated cache. In this case, all the data in the grid is copies of the 2 GB data. In contrast, using a distributed grid (assuming the requirement of one copy per data item, resulting in the capacity of the overall heap being divided by two) the resulting memory backed virtual heap contains 16 GB

> data. This data can now be effectively accessed from anywhere in the grid. In case of a server failure, the grid promptly creates new copies of the lost data and places them on operational servers in the grid.

Scalability

A significant benefit of a distributed data grid over a replicated clustered cache is that a data grid is scalable in terms of both capacity and performance. Add a node to JBoss Data Grid to increase throughput and capacity for the entire grid. JBoss Data Grid uses a consistent hashing algorithm that limits the impact of adding or removing a node to a subset of the nodes instead of every node in the grid.

Due to the even distribution of data in JBoss Data Grid, the only upper limit for the size of the grid is the group communication on the network. The network's group communication is minimal and restricted only to the discovery of new nodes. Nodes are permitted by all data access patterns to communicate directly via peer-to-peer connections, facilitating further improved scalability. JBoss Data Grid clusters can be scaled up or down in real time without requiring an infrastructure restart. The result of the real time application of changes in scaling policies results in an exceptionally flexible environment.

Data Distribution

JBoss Data Grid uses consistent hash algorithms to determine the locations for keys in clusters. Benefits associated with consistent hashing include:

- cost effectiveness.
- speed.
- deterministic location of keys with no requirements for further metadata or network traffic.

Data distribution ensures that sufficient copies exist within the cluster to provide durability and fault tolerance, while not an abundance of copies, which would reduce the environment's scalability.

Persistence

JBoss Data Grid exposes a `CacheStore` interface and several high-performance implementations, including the JDBC Cache stores and file system based cache stores. Cache stores can be used to populate the cache when it starts and to ensure that the relevant data remains safe from corruption. The cache store also overflows data to the disk when required if a process runs out of memory.

Language bindings

JBoss Data Grid supports both the popular Memcached protocol, with existing clients for a large number of popular programming languages, as well as an optimized JBoss Data Grid specific protocol called Hot Rod. As a result, instead of being restricted to Java, JBoss Data Grid can be used for any major website or application. Additionally, remote caches can be accessed using the HTTP protocol via a RESTful API.

Management

In a grid environment of several hundred or more servers, management is an important feature. JBoss Operations Network, the enterprise network management software, is the best tool to manage multiple JBoss Data Grid instances. JBoss Operations Network's features allow easy and effective monitoring of the Cache Manager and cache instances.

Remote Data Grids

Rather than scale up the entire application server architecture to scale up your data grid, JBoss Data Grid provides a Remote Client-Server mode which allows the data grid infrastructure to be upgraded independently from the application server architecture. Additionally, the data grid server can be assigned different resources than the application server and also allow independent data grid upgrades and application redeployment within the data grid.

Report a bug

1.5. JBOSS DATA GRID PREREQUISITES

The only prerequisites to set up JBoss Data Grid is a Java Virtual Machine (compatible with Java 6.0 or better) and that the most recent supported version of the product is installed on your system.

Report a bug

1.6. JBOSS DATA GRID VERSION INFORMATION

JBoss Data Grid is based on Infinispan, the open source community version of the data grid software. Infinispan uses code, designs and ideas from JBoss Cache, which has been tried, tested and proved in high stress environments. As a result, JBoss Data Grid's first release is version 6.0 as a result of its deployment history.

Report a bug

1.7. JBOSS DATA GRID CACHE ARCHITECTURE

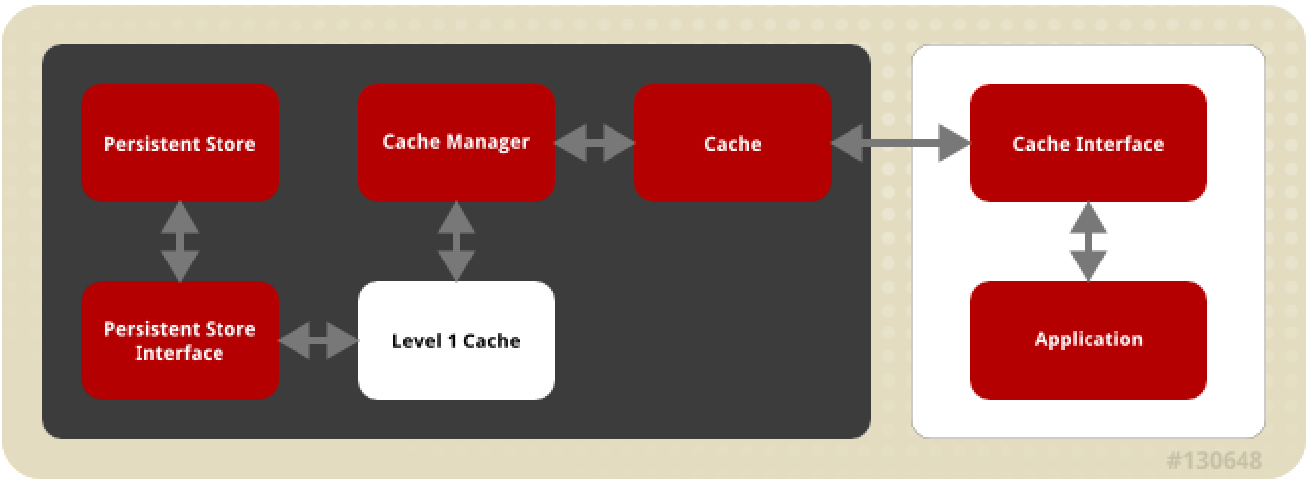


Figure 1.1. JBoss Data Grid Cache Architecture

JBoss Data Grid's cache infrastructure depicts the individual elements and their interaction with each other. For clarity, the cache architecture diagram is separated into two parts:

- Elements that a user cannot directly interact with (depicted within a dark box), which includes the Cache, Cache Manager, Level 1 Cache, Persistent Store Interfaces and the Persistent Store.
- Elements that a user can interact directly with (depicted within a white box), which includes Cache Interfaces and the Application.

Cache Architecture Elements

JBoss Data Grid's cache architecture includes the following elements:

1. The Persistent Store permanently stores cache instances and entries.
2. JBoss Data Grid offers two Persistent Store Interfaces to access the persistent store. Persistent store interfaces can be either:
   - A cache loader is a read only interface that provides a connection to a persistent data store. A cache loader can locate and retrieve data from cache instances and from the persistent store. For details, see Chapter 13, *Cache Loaders*.
   - A cache store extends the cache loader functionality to include write capabilities by exposing methods that allow the cache loader to load and store states. For details, see Chapter 12, *Cache Stores*.
3. The Level 1 Cache (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. For details, see Chapter 17, *The L1 Cache*.
4. The Cache Manager is the primary mechanism used to retrieve a Cache instance in JBoss Data Grid, and can be used as a starting point for using the Cache. For details, see Chapter 14, *Cache Managers*.
5. The Cache contains cache instances retrieved by a Cache Manager.
6. Cache Interfaces use protocols such as Memcached and Hot Rod, or REST to interface with the cache. For details about the remote interfaces, refer to the *Developer Guide*.
   - Memcached is an in-memory caching system used to improve response and operation times for database-driver websites. The Memcached caching system defines a text based, client-server caching protocol called the Memcached protocol.
   - Hot Rod is a binary TCP client-server protocol used in JBoss Data Grid. It was created to overcome deficiencies in other client/server protocols, such as Memcached. Hot Rod enables clients to do smart routing of requests in partitioned or distributed JBoss Data Grid server clusters.
   - The REST protocol eliminates the need for tightly coupled client libraries and bindings. The REST API introduces an overhead, and requires a REST client or custom code to understand and create REST calls.
7. An application allows the user to interact with the cache via a cache interface. Browsers are a common example of such end-user applications.

Report a bug

1.8. JBOSS DATA GRID APIS

1.8.1. JBoss Data Grid APIs

JBoss Data Grid provides the following programmable APIs:

- Cache
- Batching
- Grouping

- CacheStore and ConfigurationBuilder
- Externalizable
- Notification (also known as the Listener API because it deals with Notifications and Listeners)

JBoss Data Grid offers the following APIs to interact with the data grid in Remote-Client Server mode:

- The Asynchronous API (can only be used in conjunction with the Hot Rod Client in Remote Client-Server Mode)
- The REST Interface
- The Memcached Interface
- The Hot Rod Interface
  - The RemoteCache API

Report a bug

1.8.2. About the Asynchronous API

In addition to synchronous API methods, JBoss Data Grid also offers an asynchronous API that provides the same functionality in a non-blocking fashion.

The asynchronous method naming convention is similar to their synchronous counterparts, with **`Async`** appended to each method name. Asynchronous methods return a Future that contains the result of the operation.

For example, in a cache parameterized as ***`Cache(String, String)`***, ***`Cache.put(String key, String value)`*** returns a String, while ***`Cache.putAsync(String key, String value)`*** returns a **`Future(String)`**.

Report a bug

1.8.3. About the Batching API

The Batching API is used when the JBoss Data Grid cluster is the sole participant in a transaction. However, Java Transaction API (JTA) transactions (which use the Transaction Manager) should be used when multiple systems are participants in the transaction.

NOTE

The Batching API cannot be used in JBoss Data Grid's Remote Client-Server mode.

Report a bug

1.8.4. About the Cache API

The Cache interface provides simple methods for the addition, retrieval and removal of entries, which includes atomic mechanisms exposed by the JDK's **`ConcurrentMap`** interface. How entries are stored depends on the cache mode in use. For example, an entry may be replicated to a remote node or an entry may be looked up in a cache store.

The Cache API is used in the same manner as the JDK Map API for basic tasks. This simplifies the process of migrating from Map-based, simple in-memory caches to JBoss Data Grid's cache.

NOTE

This API is not available in JBoss Data Grid's Remote Client-Server Mode

Report a bug

1.8.5. About the RemoteCache Interface

The RemoteCache Interface allows clients outside JBoss Data Grid to access the Hot Rod server module within JBoss Data Grid. The RemoteCache Interface offers optional features such as distribution and eviction.

Report a bug

1.9. TOOLS AND OPERATIONS

1.9.1. About Management Tools

Managing 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.

See Also:

- Chapter 21, *Management Tools in JBoss Data Grid*

Report a bug

1.9.2. Accessing Data via URLs

Caches that have been configured with a REST interface have access to 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

1.9.3. Limitations of Map Methods

Specific **`Map`** methods, such as **`size()`**, **`values()`**, **`keySet()`** and **`entrySet()`**, can be used with certain limitations with 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

CHAPTER 2. LOGGING IN JBOSS DATA GRID

2.1. ABOUT LOGGING

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

2.2. SUPPORTED APPLICATION LOGGING FRAMEWORKS

2.2.1. Supported Application Logging Frameworks

JBoss LogManager supports the following logging frameworks:

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

Report a bug

2.2.2. 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, JBoss Data Grid 6 also uses JBoss Logging.

JBoss Logging provides an easy way to add logging to an application. You add code to your 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

2.2.3. 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

2.3. CONFIGURE LOGGING

2.3.1. About Boot Logging

The boot log is the record of events that occur while the server is starting up (or "booting"). JBoss Data Grid 6 also includes a server log, which includes log entries generated after the server concludes the boot process.

Report a bug

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

Report a bug

2.3.3. Default Log File Locations

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

Table 2.1. 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

2.4. LOGGING ATTRIBUTES

2.4.1. 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.

JBoss Data Grid 6 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):

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 level 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

2.4.2. Supported Log Levels

The following table lists log levels that are supported in 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 2.2. 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 6 to shut down.

Report a bug

2.4.3. 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 lower are captured by the log category and sent to the specified log handlers. As an example, the **DEBUG** log level results in log values of **300**, **400** and **500** are captured.

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

Report a bug

2.4.4. 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

2.4.5. About Log Handlers

Log handlers define how captured log messages are recorded by 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

2.4.6. Log Handler Types

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

Table 2.3. 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 6 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

2.4.7. Selecting Log Handlers

The following are the most common uses for each of the log handler types available for 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

2.4.8. 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.

Refer to: http://docs.oracle.com/javase/6/docs/api/java/util/Formatter.html

Report a bug

2.5. LOGGING CONFIGURATION PROPERTIES

2.5.1. Root Logger Properties

Table 2.4. Root Logger Properties

Property	Datatype	Description
level	string	The maximum level of log message that the root logger records.
handlers	list of strings	A list of log handlers that are used by the root logger.

Report a bug

2.5.2. Log Category Properties

Table 2.5. Log Category Properties

Property	Datatype	Description
level	string	The maximum level of log message that the log category records.
handlers	list of strings	A list of log handlers that are used by the root logger.
use-parent-handlers	boolean	If set to true, this category will use the log handlers of the root logger in addition to any other assigned handlers.
category	string	The log category from which log messages will be captured.

Report a bug

2.5.3. Console Log Handler Properties

Table 2.6. Console Log Handler Properties

Property	Datatype	Description
level	string	The maximum level of log message the log handler records.
encoding	string	The character encoding scheme to be used for the output.
formatter	string	The log formatter used by this log handler.
target	string	The system output stream where the output of the log handler goes. This can be System.err or System.out for the system error stream or standard out stream respectively.
autoflush	boolean	If set to true the log messages will be sent to the handlers target immediately upon receipt.

Property	Datatype	Description
name	string	The unique identifier for this log handler.

Report a bug

2.5.4. File Log Handler Properties

Table 2.7. File Log Handler Properties

Property	Datatype	Description
level	string	The maximum level of log message the log handler records.
encoding	string	The character encoding scheme to be used for the output.
formatter	string	The log formatter used by this log handler.
append	boolean	If set to true then all messages written by this handler will be appended to the file if it already exists. 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.
autoflush	boolean	If set to true the log messages will be sent to the handlers assigned file immediately upon receipt. Changes to **`autoflush`** require a server reboot to take effect.
name	string	The unique identifier for this log handler.
file	object	The object that represents the file where the output of this log handler is written to. It has two configuration properties, **`relative-to`** and **`path`**.
relative-to	string	This is a property of the file object and 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.
path	string	This is a property of the file object and 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.

Report a bug

2.5.5. Periodic Log Handler Properties

Table 2.8. Periodic Log Handler Properties

Property	Datatype	Description
append	boolean	If set to true then all messages written by this handler will be appended to the file if it already exists. 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.
autoflush	boolean	If set to true the log messages will be sent to the handlers assigned file immediately upon receipt. Changes to autoflush require a server reboot to take effect.
encoding	string	The character encoding scheme to be used for the output.
formatter	string	The log formatter used by this log handler.
level	string	The maximum level of log message the log handler records.
name	string	The unique identifier for this log handler.
file	object	Object that represents the file where the output of this log handler is written to. It has two configuration properties, ***`relative-to`*** and ***`path`***.
relative-to	string	This is a property of the file object and is the directory where the log file is written to. File path variables can be specified here. The ***`jboss.server.log.dir`*** variable points to the **`log/`** directory of the server.
path	string	This is a property of the file object and 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.
suffix	string	This is a string with is both appended to 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 the suffix **`.yyyy-MM-dd`** will result in daily log rotation. Refer to http://docs.oracle.com/javase/6/docs/api/index.html?java/text/SimpleDateFormat.html

Report a bug

2.5.6. Size Log Handler Properties

Table 2.9. Size Log Handler Properties

Property	Datatype	Description
append	boolean	If set to true then all messages written by this handler will be appended to the file if it already exists. 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.

Property	Datatype	Description
autoflush	boolean	If set to true the log messages will be sent to the handlers assigned file immediately upon receipt. Changes to autoflush require a server reboot to take effect.
encoding	string	The character encoding scheme to be used for the output.
formatter	string	The log formatter used by this log handler.
level	string	The maximum level of log message the log handler records.
name	string	The unique identifier for this log handler.
file	object	Object that represents the file where the output of this log handler is written to. It has two configuration properties, ***`relative-to`*** and ***`path`***.
relative-to	string	This is a property of the file object and is the directory where the log file is written to. File path variables can be specified here. The **`jboss.server.log.dir`** variable points to the **`log/`** directory of the server.
path	string	This is a property of the file object and 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.
rotate-size	integer	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. Eg. **`50m`** for 50 megabytes.
max-backup-index	integer	The maximum number of rotated logs that are kept. When this number is reached, the oldest log is reused.

Report a bug

2.5.7. Async Log Handler Properties

Table 2.10. Async Log Handler Properties

Property	Datatype	Description
level	string	The maximum level of log message the log handler records.
name	string	The unique identifier for this log handler.
Queue-length	integer	Maximum number of log messages that will be held by this handler while waiting for sub-handlers to respond.

Property	Datat ype	Description
overflow-action	string	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.
subhandlers	list of strings	This is the list of log handlers to which this async handler passes its log messages.

Report a bug

2.6. LOGGING SAMPLE CONFIGURATIONS

2.6.1. Sample XML Configuration for the Root Logger

```
<subsystem xmlns="urn:jboss:domain:logging:1.1">

   <root-logger>
      <level name="INFO"/>
      <handlers>
         <handler name="CONSOLE"/>
         <handler name="FILE"/>
      </handlers>
   </root-logger>

</subsystem>
```

Report a bug

2.6.2. Sample XML Configuration for a Log Category

```
<subsystem xmlns="urn:jboss:domain:logging:1.1">

   <logger category="com.company.accounts.rec">
      <handlers>
         <handler name="accounts-rec"/>
      </handlers>
   </logger>

</subsystem>
```

Report a bug

2.6.3. Sample XML Configuration for a Console Log Handler

```
<subsystem xmlns="urn:jboss:domain:logging:1.1">

   <console-handler name="CONSOLE">
```

```
      <level name="INFO"/>
      <formatter>
         <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
      </formatter>
   </console-handler>

</subsystem>
```

Report a bug

2.6.4. Sample XML Configuration for a File Log Handler

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

Report a bug

2.6.5. Sample XML Configuration for a Periodic Log Handler

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

2.6.6. Sample XML Configuration for a Size Log Handler

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

Report a bug

2.6.7. Sample XML Configuration for a Async Log Handler

```
<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

CHAPTER 3. STATE TRANSFER AND HIGH AVAILABILITY USING SERVER HINTING

3.1. STATE TRANSFER

3.1.1. About State Transfer

State transfer occurs automatically in 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

3.1.2. Non-Blocking State Transfer

Non-Blocking State Transfer in 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

3.2. HIGH AVAILABILITY USING SERVER HINTING

3.2.1. About Server Hinting

In JBoss Data Grid, Server Hinting is a topology aware consistent hashing algorithm that ensures backup 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.

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

3.2.2. Establishing Server Hinting with JGroups

When setting up a clustered environment in 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 20.4.2, “Pre-Configured JGroups Files”

Report a bug

3.2.3. Configure Server Hinting (Remote Client-Server Mode)

In JBoss Data Grid's Remote Client-Server mode, Server Hinting is configured using the subsystem tags, as follows:

```
<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

3.2.4. Configure Server Hinting (Library Mode)

In JBoss Data Grid's Library mode, Server Hinting is configured at the transport level. The following is a Server Hinting sample configuration:

```
<transport clusterName = "MyCluster"
           machineId = "LinuxServer01"
```

```
            rackId = "Rack01"
            siteId = "US-WestCoast" />
```

As illustrated in the sample configuration, the following configuration attributes are used to configure Server Hinting in JBoss Data Grid:

- The ***clusterName*** attribute specifies the name assigned to the cluster.
- 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.
- The ***rackId*** parameter specifies the rack that contains the original data, so that other racks are used for backups.
- The ***siteId*** parameter differentiates between nodes in different data centers replicating to each other.

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 centre as the original data.

Report a bug

3.2.5. ConsistentHashFactory

3.2.5.1. TopologyAwareConsistentHashFactory and TopologyAwareSyncConsistentHashFactory

The **TopologyAwareConsistentHashFactory** implementation enables Server Hinting. This attempts to distribute segments based on the topology information in the Transport configuration.

The **TopologyAwareConsistentHashFactory** implementation can be selected by setting one or more of the parameters in the transport configuration.

The **TopologyAwareSyncConsistentHashFactory** implementation can also 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

3.2.5.2. Implementing a ConsistentHashFactory

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.

In addition to these, a **`ConsistentHashFactory`** can be implemented manually. The custom **`ConsistentHashFactory`** must implement the following methods:

```
create(Hash hashFunction,
          int numOwners,
          int numSegments,
          List<Address> members)
// create a new consistent hash instance

updateMembers(ConsistentHash baseCH,
          List<Address> newMembers)
// update the list of members of an existing consistent hash instance; the
implementation should not assign new segments to a node, unless that
segments doesn't have any owners

rebalance(ConsistentHash baseCH)
// rebalance the segments between existing members (so a node joining
requires both updateMembers and rebalance)

union(ConsistentHash ch1, ConsistentHash ch2)
// create a consistent hash instance which, for each segment, has the same
owners as both consistent hash parameters. The result consistent hash is
used for write operations during state transfer.
```

Consistent hash instances can be created with the following parameters:

- The ***`hashFunction`*** parameter is used on top of the keys' own hashCode() implementation.
- The ***`numOwners`*** parameter is the ideal number of owners for each key. The created consistent hash can have a greater or fewer number of owners, however each key will have at least one owner.

- The ***`numSegments`*** defines the number of hash-space arguments. The implementation may either round up the number of segments for performance, or it may ignore the parameter altogether.

- The ***`members`*** parameter provides a list of addresses representing the new cache members.

Currently it is not possible to pass custom parameters to **`ConsistentHashFactory`** implementations.

Report a bug

CHAPTER 4. CACHE MODES

4.1. ABOUT CACHE MODES

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 comparision 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, refer to Section 20.4, “JGroups for Clustered Modes”

Report a bug

4.2. ABOUT CACHE CONTAINERS

Cache containers are used in 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 is a sample cache container configuration:

```
<subsystem xmlns="urn:infinispan:server:core:5.2"
    default-cache-container="default">
 <cache-container name="default"
    default-cache="default"
    listener-executor="infinispan-listener"
    start="EAGER">
  <local-cache ... >
   ...
  </local-cache>
 </cache-container>
</subsystem>
```

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

- The ***`name`*** parameter defines the name of the cache container.
- The ***`default-cache`*** parameter defines the name of the default cache used with the cache container.
- The ***`listener-executor`*** defines the executor used for asynchronous cache listener notifications.
- 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`**.

Report a bug

4.3. ABOUT LOCAL MODE

4.3.1. Local Mode Operations

Using 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-biased 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

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

A local cache can be added to any cache container. For example:

```
<cache-container name="local"
   default-cache="default" >
 <local-cache name="default"
       start="EAGER">
  <locking isolation="NONE"
    acquire-timeout="30000"
    concurrency-level="1000"
    striping="false" />
  <transaction mode="NONE" />
 </local-cache>
</cache-container>
```

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.

The local-cache Element

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

- The ***name*** parameter specifies the name of the local cache to use.
- The ***start*** parameter indicates where the cache starts, i.e. whether it will start lazily when requested or when the server starts up. Valid values for this parameter are **EAGER** and **LAZY**.
- The ***batching*** parameter specifies whether batching is enabled for the local cache.
- The ***indexing*** parameter specifies the type of indexing used for the local cache. Valid values for this parameter are **NONE**, **LOCAL** and **ALL**.

Report a bug

4.3.3. Configure Local Mode (Library Mode)

In 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

4.4. ABOUT CLUSTERED MODES

4.4.1. Clustered Mode Operations

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 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

4.4.2. 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

4.4.3. Cache Mode Troubleshooting

4.4.3.1. 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

4.4.3.2. About Asynchronous Communications

In 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" >
                ...
</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

4.4.3.3. 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

CHAPTER 5. DISTRIBUTION MODE

5.1. ABOUT DISTRIBUTION MODE

When enabled, 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

5.2. 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

5.3. LOCATING ENTRIES IN DISTRIBUTION MODE

The consistent hash algorithm used in 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

5.4. RETURN VALUES IN DISTRIBUTION MODE

In 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

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

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

```
<cache-container name="local"
   default-cache="default" >
 <distributed-cache name="default"
      mode="{SYNC/ASYNC}"
      segments="${NUMBER}"
      start="EAGER">
  <locking isolation="NONE"
    acquire-timeout="30000"
    concurrency-level="1000"
    striping="false" />
  <transaction mode="NONE" />
 </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, refer to the appropriate chapter.

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

- The **`name`** parameter provides a unique identifier for the cache.
- The **`mode`** parameter sets the clustered cache mode. Valid values are **`SYNC`** (synchronous) and **`ASYNC`** (asynchronous).
- 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`**.
- The **`start`** parameter specifies whether the cache starts when the server starts up or when it is requested or deployed.

See Also:

- Section 20.4, “JGroups for Clustered Modes”

Report a bug

5.6. CONFIGURE DISTRIBUTION MODE (LIBRARY MODE)

In JBoss Data Grid's Library mode, a distributed cache configuration is as follows:

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

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

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

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

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

- 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.

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

- 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.

- 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.

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

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

- 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.

- The ***`strictPeerToPeer`*** parameter, when set to **`true`** ensures that replication operations fail if the named cache does not exist. If set to **`false`**, the operation completes and logs messages about specific named caches not found and that replication was not performed on the nodes as a result.

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

Report a bug

5.7. SYNCHRONOUS AND ASYNCHRONOUS DISTRIBUTION

5.7.1. About 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 5.1. 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

5.8. GET AND PUT USAGE IN DISTRIBUTION MODE

5.8.1. About GET and PUT Operations 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 JBoss Data Grid must wait for the return value.

Report a bug

5.8.2. 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.

Disable 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 6. REPLICATION MODE

6.1. ABOUT REPLICATION MODE

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.

Report a bug

6.2. 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.

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

Report a bug

6.3. RETURN VALUES IN REPLICATION MODE

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

Report a bug

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

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

```
<cache-container name="local"
   default-cache="default" >
 <replicated-cache name="default"
    mode="{SYNC/ASYNC}"
     start="EAGER">
  <locking isolation="NONE"
    acquire-timeout="30000"
    concurrency-level="1000"
    striping="false" />
  <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, refer to the appropriate chapter.

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

- The **`name`** parameter provides a unique identifier for the cache.
- The **`mode`** parameter sets the clustered cache mode. Valid values are **`SYNC`** (synchronous) and **`ASYNC`** (asynchronous).
- The **`start`** parameter specifies whether the cache starts when the server starts up or when it is requested or deployed.

See Also:

- Section 20.4, “JGroups for Clustered Modes”

Report a bug

6.5. CONFIGURE REPLICATION MODE (LIBRARY MODE)

In JBoss Data Grid's Library mode, a replicated cache configuration is as follows:

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

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

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

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

- 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.

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

- 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.

- 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.

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

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

- 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.

- The ***`strictPeerToPeer`*** parameter, when set to **`true`** ensures that replication operations fail if the named cache does not exist. If set to **`false`**, the operation completes and logs messages about specific named caches not found and that replication was not performed on the nodes as a result.

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

Report a bug

6.6. SYNCHRONOUS AND ASYNCHRONOUS REPLICATION

6.6.1. 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

6.6.2. 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 you enable a replication queue (which is recommended for asynchronous mode).

Report a bug

6.7. THE REPLICATION QUEUE

6.7.1. Replication Queue

In replication mode, 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 replication queue is used in conjunction with asynchronous mode.

Report a bug

6.7.2. Replication Queue Operations

When using the replication queue, the primary disadvantage 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. With the replication queue disabled, however, the data is directly transmitted and therefore arrives at the cluster nodes faster.

Report a bug

6.7.3. 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"
                  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

6.8. FREQUENTLY ASKED QUESTIONS

6.8.1. About Replication Guarantees

In a clustered cache, the user can receive synchronous replication guarantees as well as the parallelism associated with asynchronous replication. 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

6.8.2. 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 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 7. INVALIDATION MODE

7.1. 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.

Report a bug

7.2. USING INVALIDATION MODE

Invalidation mode requires a second permanent store for data, such as a database. JBoss Data Grid, in such a situation, is used as an optimization for a read-heavy system 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

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

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

```
<cache-container name="local"
   default-cache="default" >
 <invalidated-cache name="default"
      mode="{SYNC/ASYNC}"
      start="EAGER">
  <locking isolation="NONE"
    acquire-timeout="30000"
    concurrency-level="1000"
    striping="false" />
  <transaction mode="NONE" />
 </invalidated-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, refer to the appropriate chapter.

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

- The **name** parameter provides a unique identifier for the cache.

- The **mode** parameter sets the clustered cache mode. Valid values are **SYNC** (synchronous) and **ASYNC** (asynchronous).
- The **start** parameter specifies whether the cache starts when the server starts up or when it is requested or deployed.

See Also:

- Section 20.4, “JGroups for Clustered Modes”

Report a bug

7.4. CONFIGURE INVALIDATION MODE (LIBRARY MODE)

In JBoss Data Grid's Library mode, an invalidated cache configuration is as follows:

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

The **clustering** element's ***mode*** parameter's value determines the clustering mode selected for the cache.

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

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

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

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

- The ***`clusterName`*** parameter specifies the name of the cluster. Nodes can only connect to clusters that share the same name.
- 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.
- The ***`strictPeerToPeer`*** parameter, when set to **`true`** ensures that replication operations fail if the named cache does not exist. If set to **`false`**, the operation completes and logs messages about specific named caches not found and that replication was not performed on the nodes as a result.
- The ***`transportClass`*** parameter specifies a class that represents a network transport for the cache.

Report a bug

7.5. SYNCHRONOUS/ASYNCHRONOUS INVALIDATION

In 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

7.6. THE L1 CACHE INVALIDATION

7.6.1. 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

CHAPTER 8. CACHE WRITING MODES

8.1. WRITE-THROUGH AND WRITE-BEHIND CACHING

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

8.2. WRITE-THROUGH CACHING

8.2.1. About Write-Through Caching

The Write-Through (or Synchronous) mode in 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

8.2.2. 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

8.2.3. 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 is a sample configuration file of a Write-Through unshared local file cache store:

```
<?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">
  <loaders shared="false">
   <loader class="org.infinispan.loaders.file.FileCacheStore"
          fetchPersistentState="true" ignoreModifications="false"
          purgeOnStartup="false">
    <properties>
     <property name="location"
```

```
            value="${java.io.tmpdir}" />
        </properties>
      </loader>
    </loaders>
      </namedCache>
</infinispan>
```

Report a bug

8.3. WRITE-BEHIND CACHING

8.3.1. About Write-Behind Caching

In 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

8.3.2. About Unscheduled Write-Behind Strategy

In the Unscheduled Write-Behind Strategy mode, 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 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

8.3.3. Unscheduled Write-Behind Strategy Configuration (Remote Client-Server Mode)

To set the write-behind stragy in JBoss Data Grid's Remote Client-Server mode, add the **`write-behind`** element to the target cache store configuration as follows:

```
<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>
```

The **`write-behind`** element uses the following configuration parameters:

- The ***`modification-queue-size`*** parameter specifies the maximum number of entries in the asynchronous queue. If the queue is full, the cache uses the write-through strategy until it is able to accept new entries again. The default value for this parameter is **`1024`** bytes.
- The ***`shutdown-timeout`*** parameter specifies the time in milliseconds after which the cache store is shut down. The default value for this parameter is **`25000`**.
- The ***`flush-lock-timeout`*** 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.
- The ***`thread-pool-size`*** parameter specified 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`**.

Report a bug

8.3.4. 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:

```
<loaders>
    <fileStore location="${LOCATION}">
        <async enabled="true"
                    modificationQueueSize="1024"
                    shutdownTimeout="25000"
                    flushLockTimeout="15000"
                    threadPoolSize="5"/>
    </fileStore>
</loaders>
```

The **`async`** element uses the following configuration parameters:

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

Report a bug

CHAPTER 9. LOCKING

9.1. ABOUT LOCKING

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

9.2. CONFIGURE LOCKING (REMOTE CLIENT-SERVER MODE)

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

The following is a sample of a basic locking configuration for a default cache in JBoss Data Grid's Library mode:

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

- The ***`isolation`*** parameter defines the isolation level used for the local cache. Valid values for this parameter are **`REPEATABLE_READ`** and **`READ_COMMITTED`**.
- The ***`acquire-timeout`*** parameter specifies the number of milliseconds after which lock acquisition will time out.
- The ***`concurrency-level`*** parameter defines the number of lock stripes used by the LockManager.
- The ***`striping`*** parameter specifies whether lock striping will be used for the local cache.

Report a bug

9.3. 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:

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

```
      writeSkewCheck="${TRUE/FALSE}"
      supportsConcurrentUpdates="${TRUE/FALSE}" />
    ...
   </configuration>
  </default>
</infinispan>
```

- 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.

- The ***isolationLevel*** parameter specifies the cache's isolation level. Valid isolation levels are **READ_COMMITTED** and **REPEATABLE_READ**. For details about isolation levels, refer to Chapter 11, *Isolation Levels*

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

- 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, refer to Chapter 10, *Lock Striping*

- 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.

- The ***supportsConcurrentUpdates*** parameter is only valid for non-transactional caches. If this parameter is set to **TRUE** (default value), the cache ensures that the data remains consistent despite concurrent updates. If the application is not expected to perform concurrent writes, it is recommended that this parameter is set to **FALSE** to improve performance.

Report a bug

9.4. LOCKING TYPES

9.4.1. 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

9.4.2. 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

9.4.3. Pessimistic Locking Types

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

9.4.4. 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:

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

1. When the line **`cache.lock(K)`** executes, a cluster-wide lock is acquired on **`K`**.
2. When the line **`cache.put(K,V5)`** executes, it guarantees success.
3. When the line **`tx.commit()`** executes, the locks held for this process are released.

Report a bug

9.4.5. Implicit Pessimistic Locking Example

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

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

–

1. When the line **`cache.put(K,V)`** executes, a cluster-wide lock is acquired on **`K`**.
2. When the line **`cache.put(K2,V2)`** executes, a cluster-wide lock is acquired on **`K2`**.
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.
4. When the line **`tx.commit()`** executes, all locks held for this transaction are released.

Report a bug

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

To configure a locking mode in 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

9.4.7. Configure Locking Mode (Library Mode)

In 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">
  <recovery enabled="true"
     recoveryInfoCacheName="{CacheName}" />
</transaction>
```

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

Report a bug

9.5. LOCKING OPERATIONS

9.5.1. 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 10, *Lock Striping*

Report a bug

9.5.2. About Lock Acquisition

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

9.5.3. About Concurrency Levels

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 10. LOCK STRIPING

10.1. 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 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

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

Lock striping in 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

10.3. CONFIGURE LOCK STRIPING (LIBRARY MODE)

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

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

Report a bug

CHAPTER 11. ISOLATION LEVELS

11.1. 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 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, refer to the lock striping configuration samples:

- Refer to Section 10.2, “Configure Lock Striping (Remote Client-Server Mode)” for a Remote Client-Server mode configuration sample.
- Refer to Section 10.3, “Configure Lock Striping (Library Mode)” for a Library mode configuration sample.

Report a bug

11.2. ABOUT READ_COMMITTED

`READ_COMMITTED` is one of two isolation modes available in 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

11.3. ABOUT REPEATABLE_READ

`REPEATABLE_READ` is one of two isolation modes available in 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

CHAPTER 12. CACHE STORES

12.1. ABOUT CACHE STORES

The cache store connects 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

12.2. FILE CACHE STORES

12.2.1. About File System Based Cache Stores

JBoss Data Grid includes one file system based cache store: the **`FileCacheStore`**.

The **`FileCacheStore`** is a simple, file system based implementation.

Due to its limitations, **`FileCacheStore`** can be used in a limited capacity in production environments. It should 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.

The **`FileCacheStore`** is ideal for testing usage and is not suited to use in highly concurrent, transactional or stress-based environments.

Report a bug

12.2.2. File Cache Store Configuration (Remote Client-Server Mode)

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

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

- The ***`name`*** parameter of the **`local-cache`** attribute is used to specify a name for the cache.
- The **`file-store`** element specifies configuration information for the file cache store. Attributes for this element include the ***`relative-to`*** parameter used to define a named path, and the ***`path`*** parameter used to specify a directory within ***`relative-to`***.

Report a bug

12.2.3. File Cache Store Configuration (Library Mode)

In JBoss Data Grid's Library mode, configure a File Cache Store within the loaders element as follows:

```
<loaders>
    ...
 <fileStore location="${java.io.tmpdir}"
                  streamBufferSize="${SIZE}"
                  fsyncMode="${DEFAULT/PER_WRITE/PERIODIC}"
                  fsyncInterval="${TIME}" />
    ...
</loaders>
```

The **`fileStore`** element is used to configure the File Cache Store with the following parameters:

- The ***`location`*** parameter provides a location on the disk where the file store can write internal files. The default value for this parameter is the **`Infinispan-Filestore`** directory in the working directory.
- The ***`streamBufferSize`*** parameter sets the size of the buffered stream used to write the state to the disk. Larger buffer sizes result in faster performance but occupy more temporary memory. The default value for this parameter is **`8192`** bytes.
- The ***`fsyncMode`*** parameter sets how the file changes synchronize with the underlying file system. Valid values for this parameter are **`DEFAULT`** (synchronize when the operating system buffer is full or when the bucket is read), **`PER_WRITE`** (synchronize after each write request) and **`PERIODIC`** (synchronizes after a defined interval or just before a bucket is read).
- The ***`fsyncInterval`*** parameter specifies the time period after which the file changes in the cache are flushed. This parameter is used only when the periodic fsync mode is in use. The default value for this parameter is **`1`** second.

Report a bug

12.3. REMOTE CACHE STORES

12.3.1. About Remote Cache Stores

The **`RemoteCacheStore`** is an implementation of the cache loader that stores data in a remote 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

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

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

```
<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>
```

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

- The ***cache*** parameter defines the name for the remote cache. If left undefined, the default cache is used instead.
- 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.
- The ***tcp-no-delay*** sets whether **TCP_NODELAY** applies on socket connections to remote Hot Rod servers.
- The ***hotrod-wrapping*** sets whether a wrapper is required for Hot Rod on the remote store.

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

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

Report a bug

12.3.3. Remote Cache Store Configuration (Library Mode)

Prerequisites:

Create and include a file named **hotrod.properties** in the relevant classpath.

Remote Cache Store Configuration

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

```
<loaders shared="true"
  preload="true">
 <loader class="org.infinispan.loaders.remote.RemoteCacheStore">
  <properties>
   <property name="remoteCacheName"
      value="default"/>
   <property name="hotRodClientPropertiesFile"
      value="hotrod.properties" />
  </properties>
 </loader>
</loaders>
```

IMPORTANT

Define the outbound socket for the remote cache store when using this configuration. The remote cache store property named **hotRodClientPropertiesFile** refers to the **hotrod.properties** file. This file must be defined for the Remote Cache Store to operate correctly.

Report a bug

12.3.4. The hotrod.properties File

To use a Remote Cache Store configuration, the hotrod.properties file must be created and included in the relevant classpath for a Remote Cache Store configuration.

The hotrod.properties file contains one or more properties. The most simple version of a working hotrod.properties file can contain the following:

```
infinispan.client.hotrod.server_list=remote-server:11222
```

Properties that can be included in hotrod.properties are:

`infinispan.client.hotrod.request_balancing_strategy`

For replicated (vs distributed) Hot Rod server clusters, the client balances requests to the servers according to this strategy.

The default value for this property is **`org.infinispan.client.hotrod.impl.transport.tcp.RoundRobinBalancingStrategy`**.

`infinispan.client.hotrod.server_list`

This is the initial list of Hot Rod servers to connect to, specified in the following format: host1:port1;host2:port2... At least one host:port must be specified.

The default value for this property is **`127.0.0.1:11222`**.

`infinispan.client.hotrod.force_return_values`

Whether or not to enable Flag.FORCE_RETURN_VALUE for all calls.

The default value for this property is **`false`**.

`infinispan.client.hotrod.tcp_no_delay`

Affects TCP NODELAY on the TCP stack.

The default value for this property is **`true`**.

`infinispan.client.hotrod.ping_on_startup`

If true, a ping request is sent to a back end server in order to fetch cluster's topology.

The default value for this property is **`true`**.

`infinispan.client.hotrod.transport_factory`

Controls which transport will be used. Currently only the TcpTransport is supported.

The default value for this property is **`org.infinispan.client.hotrod.impl.transport.tcp.TcpTransportFactory`**.

`infinispan.client.hotrod.marshaller`

Allows you to specify a custom Marshaller implementation to serialize and deserialize user objects.

The default value for this property is **`org.infinispan.marshall.jboss.GenericJBossMarshaller`**.

`infinispan.client.hotrod.async_executor_factory`

Allows you to specify a custom asynchronous executor for async calls.

The default value for this property is **`org.infinispan.client.hotrod.impl.async.DefaultAsyncExecutorFactory`**.

`infinispan.client.hotrod.default_executor_factory.pool_size`

If the default executor is used, this configures the number of threads to initialize the executor with.

The default value for this property is **`10`**.

`infinispan.client.hotrod.default_executor_factory.queue_size`

If the default executor is used, this configures the queue size to initialize the executor with.

The default value for this property is **`100000`**.

`infinispan.client.hotrod.hash_function_impl.1`

This specifies the version of the hash function and consistent hash algorithm in use, and is closely tied with the Hot Rod server version used.

The default value for this property is the **`Hash function specified by the server in the responses as indicated in ConsistentHashFactory`**.

`infinispan.client.hotrod.key_size_estimate`

This hint allows sizing of byte buffers when serializing and deserializing keys, to minimize array resizing.

The default value for this property is **`64`**.

`infinispan.client.hotrod.value_size_estimate`

This hint allows sizing of byte buffers when serializing and deserializing values, to minimize array resizing.

The default value for this property is **`512`**.

`infinispan.client.hotrod.socket_timeout`

This property defines the maximum socket read timeout before giving up waiting for bytes from the server.

The default value for this property is **`60000 (equals 60 seconds)`**.

`infinispan.client.hotrod.protocol_version`

This property defines the protocol version that this client should use. Other valid values include 1.0.

The default value for this property is **`1.1`**.

`infinispan.client.hotrod.connect_timeout`

This property defines the maximum socket connect timeout before giving up connecting to the server.

The default value for this property is **`60000 (equals 60 seconds)`**.

Report a bug

12.3.5. 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

12.4. JDBC BASED CACHE STORES

12.4.1. About JDBC Based Cache Stores

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:

- **`JdbcBinaryCacheStore`**.
- **`JdbcStringBasedCacheStore`**.
- **`JdbcMixedCacheStore`**.

For information about supported SQL databases, refer to Section 1.2, “JBoss Data Grid Supported Configurations”

Report a bug

12.4.2. JdbcBinaryCacheStores

12.4.2.1. About JdbcBinaryCacheStore

The **`JdbcBinaryCacheStore`** 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, **JdbcBinaryCacheStore** 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

12.4.2.2. JdbcBinaryCacheStore Configuration (Remote Client-Server Mode)

The following is a configuration for **JdbcBinaryCacheStore** using JBoss Data Grid's Remote Client-Server mode with Passivation enabled.

```
<local-cache>
 ...
 <binary-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
     passivation="${true/false}"
     preload="${true/false]"
     purge="${true/false}">
                <property name="databaseType">${database.type}</property>
                <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>
```

The binary-keyed-jdbc-store Element

The **binary-keyed-jdbc-store** element specifies the configuration for a binary keyed cache JDBC store.

- The ***datasource*** parameter defines the name of a JNDI for the datasource.
- 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**).
- The ***preload*** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are **true** and **false**.
- 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**.

The property Element

The **property** element contains information about properties related to the cache store.

- The ***name*** parameter specifies the name of the cache store.

- The value *${database.type}* must be replaced by a valid database type value, such as **DB2_390**, **SQL_SERVER**, **MYSQL**, **ORACLE**, **POSTGRES** or **SYBASE**.

The binary-keyed-table Element

The **binary-keyed-table** element specifies information about the database table used to store binary cache entries.

- The ***prefix*** parameter specifies a prefix string for the database table name.

The id-column Element

The **id-column** element specifies information about a database column that holds cache entry IDs.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

The data-column Element

The **data-column** element contains information about a database column that holds cache entry data.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

The timestamp-column Element

The **timestamp-column** element specifies information about the database column that holds cache entry timestamps.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

Report a bug

12.4.2.3. JdbcBinaryCacheStore Configuration (Library Mode)

The following is a sample configuration for the **JdbcBinaryCacheStore**:

```
<loaders>
 <binaryKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:5.2"
        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>
</loaders>
```

The binaryKeyedJdbcStore Element

The **`binaryKeyedJdbcStore`** element uses the following parameters to configure the cache store:

- 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 propety 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 **`ignoreModifications`** parameter determines whether operations that modify the cache (e.g. put, remove, clear, store, etc.) do not affect the cache. As a result, the cache store can become out of sync with the cache.
- The **`purgeOnStartup`** parameter specifies whether the cache is purged when initally started.

The connectionPool Element

The **`connectionPool`** element specifies a connection pool for the JDBC drive using the following parameters:

- The ***`connectionUrl`*** parameter specifies the JDBC driver-specfic connection URL.
- The ***`username`*** parameter contains the username used to connect via the ***`connectionUrl`***.
- The ***`driverClass`*** parameter specifies the class name of the driver used to connect to the database.

The binaryKeyedTable Element

Add the **`binaryKeyedTable`** element defines the table that stores cache entries. It uses the following parameters to configure the cache store:

- The ***`dropOnExit`*** parameter specifies whether the database tables are dropped upon shutdown.
- The ***`createOnStart`*** parameter specifies whether the database tables are created by the store on startup.
- The ***`prefix`*** parameter defines the string prepended to name of the target cache when composing the name of the cache bucket table.

The idColumn Element

The **`idColumn`** element defines the column where the cache key or bucket ID is stored. It used the following parameters:

- Use the ***`name`*** parameter to specify the name of the column used.
- Use the ***`type`*** parameter to specify the type of the column used.

The dataColumn Element

The **`dataColumn`** element specifies the column where the cache entry or bucket is stored.

- Use the ***name*** parameter to specify the name of the column used.
- Use the ***type*** parameter to specify the type of the column used.

The timestampColumn Element

The **`timestampColumn`** element specifies the column where the time stamp of the cache entry or bucket is stored.

- Use the ***name*** parameter to specify the name of the column used.
- Use the ***type*** parameter to specify the type of the column used.

Report a bug

12.4.3. JdbcStringBasedCacheStores

12.4.3.1. About JdbcStringBasedCacheStore

The **`JdbcStringBasedCacheStore`** stores each entry 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.

JBoss Data Grid includes a default implementation called **`DefaultTwoWayKey2StringMapper`** that handles primitive types.

Report a bug

12.4.3.2. JdbcStringBasedCacheStore Configuration (Remote Client-Server Mode)

The following is a sample **`JdbcStringBasedCacheStore`** for JBoss Data Grid's Remote Client-Server mode with Passivation enabled.

```
 <local-cache>
...
<string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
    passivation="true"
    preload="false"
    purge="false">
        <property name="databaseType">${database.type}</property>
               <string-keyed-table prefix="JDG">
                <id-column name="id"
      type="${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>
```

The string-keyed-jdbc-store Element

The **string-keyed-jdbc-store** element specifies the configuration for a string based keyed cache JDBC store.

- The ***datasource*** parameter defines the name of a JNDI for the datasource.
- 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**).
- The ***preload*** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are **true** and **false**.
- 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**.
- 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 **ENABLED** and **DISABLED**.
- The ***singleton*** parameter enables a singleton store that is used if a cluster interacts with the underlying store.

The property Element

The **property** element contains information about properties related to the cache store.

- The ***name*** parameter specifies the name of the cache store.
- The value *${database.type}* must be replaced by a valid database type value, such as **DB2_390**, **SQL_SERVER**, **MYSQL**, **ORACLE**, **POSTGRES** or **SYBASE**.

The string-keyed-table Element

The **string-keyed-table** element specifies information about the database table used to store string based cache entries.

- The ***prefix*** parameter specifies a prefix string for the database table name.

The id-column Element

The **id-column** element specifies information about a database column that holds cache entry IDs.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

The data-column Element

The **data-column** element contains information about a database column that holds cache entry data.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

The timestamp-column Element

The **timestamp-column** element specifies information about the database column that holds cache entry timestamps.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

Report a bug

12.4.3.3. JdbcStringBasedCacheStore Configuration (Library Mode)

The following is a sample configuration for the **`JdbcStringBasedCacheStore`**:

```
<loaders>
 <stringKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:5.2"
         fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey2Strin
gMapper">
  <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_BUCKET_TABLE">
   <idColumn name="ID_COLUMN"
      type="VARCHAR(255)" />
   <dataColumn name="DATA_COLUMN"
       type="BINARY" />
   <timestampColumn name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </stringKeyedTable>
 </stringKeyedJdbcStore>
</loaders>
```

The stringKeyedJdbcStore Element

The **`stringKeyedJdbcStore`** element uses the following parameters to configure the cache store:

- 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 propety 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 **`ignoreModifications`** parameter determines whether operations that modify the cache (e.g. put, remove, clear, store, etc.) do not affect the cache. As a result, the cache store can become out of sync with the cache.
- The **`purgeOnStartup`** parameter specifies whether the cache is purged when initally started.
- The **`key2StringMapper`** parameter specifies the class name of the Key2StringMapper used to map keys to strings for the database tables.

The stringKeyedTable Element

Add the **`stringKeyedTable`** element defines the table that stores cache entries. It uses the following parameters to configure the cache store:

- The ***`dropOnExit`*** parameter specifies whether the database tables are dropped upon shutdown.
- The ***`createOnStart`*** parameter specifies whether the database tables are created by the store on startup.
- The ***`prefix`*** parameter defines the string prepended to name of the target cache when composing the name of the cache bucket table.

The idColumn Element

The **`idColumn`** element defines the column where the cache key or bucket ID is stored. It used the following parameters:

- Use the ***`name`*** parameter to specify the name of the column used.
- Use the ***`type`*** parameter to specify the type of the column used.

The dataColumn Element

The **`dataColumn`** element specifies the column where the cache entry or bucket is stored.

- Use the ***`name`*** parameter to specify the name of the column used.
- Use the ***`type`*** parameter to specify the type of the column used.

The timestampColumn Element

The **`timestampColumn`** element specifies the column where the time stamp of the cache entry or bucket is stored.

- Use the ***`name`*** parameter to specify the name of the column used.
- Use the ***`type`*** parameter to specify the type of the column used.

Report a bug

12.4.3.4. JdbcStringBasedCacheStore Multiple Node Configuration (Remote Client-Server Mode)

The following is a configuration for the **`JdbcStringBasedCacheStore`** in JBoss Data Grid's Remote Client-Server mode. This configuration is used when multiple nodes must be used.

```
<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">
     <property name="databaseType">${database.type}</property>
                  <string-keyed-table prefix="JDG">
                              <id-column name="id"
              type="${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>
```

The string-keyed-jdbc-store Element

The **`string-keyed-jdbc-store`** element specifies the configuration for a string based keyed cache JDBC store.

- The ***`datasource`*** parameter defines the name of a JNDI for the datasource.
- 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`**).
- The ***`preload`*** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are **`true`** and **`false`**.
- 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`**.
- 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`**.
- The ***`singleton`*** parameter enables a singleton store that is used if a cluster interacts with the underlying store.

The property Element

The **`property`** element contains information about properties related to the cache store.

- The ***`name`*** parameter specifies the name of the cache store.
- The value *${database.type}* must be replaced by a valid database type value, such as **`DB2_390`**, **`SQL_SERVER`**, **`MYSQL`**, **`ORACLE`**, **`POSTGRES`** or **`SYBASE`**.

The string-keyed-table Element

The **`string-keyed-table`** element specifies information about the database table used to store string based cache entries.

- The ***`prefix`*** parameter specifies a prefix string for the database table name.

The id-column Element

The **`id-column`** element specifies information about a database column that holds cache entry IDs.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

The data-column Element

The **data-column** element contains information about a database column that holds cache entry data.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

The timestamp-column Element

The **timestamp-column** element specifies information about the database column that holds cache entry timestamps.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

Report a bug

12.4.4. JdbcMixedCacheStore

12.4.4.1. About JdbcMixedCacheStore

The **JdbcMixedCacheStore** is a hybrid implementation that delegates keys based on their type to either the **JdbcBinaryCacheStore** or **JdbcStringBasedCacheStore**.

Report a bug

12.4.4.2. JdbcMixedCacheStore Configuration (Remote Client-Server Mode)

The following is a configuration for a **JdbcMixedCacheStore** for JBoss Data Grid's Remote Client-Server mode with Passivation enabled.

```
  <subsystem xmlns="urn:infinispan:server:core:5.2" default-cache-
container="default">
    <cache-container ... >
 <local-cache ... >
            ...
            <mixed-keyed-jdbc-store
datasource="java:jboss/datasources/JdbcDS"
         passivation="true"
         preload="false"
         purge="false">
   <property name="databaseType">${database.type}</property>
          <binary-keyed-table prefix="MIX_BKT2">
                     <id-column name="id"
                  type="${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>
```

The mixed-keyed-jdbc-store Element

The **mixed-keyed-jdbc-store** element specifies the configuration for a mixed keyed cache JDBC store.

- The ***datasource*** parameter defines the name of a JNDI for the datasource.
- 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**).
- The ***preload*** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are **true** and **false**.
- 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**.

The property Element

The **property** element contains information about properties related to the cache store.

- The ***name*** parameter specifies the name of the cache store.
- The value *${database.type}* must be replaced by a valid database type value, such as **DB2_390**, **SQL_SERVER**, **MYSQL**, **ORACLE**, **POSTGRES** or **SYBASE**.

The mixed-keyed-table Element

The **mixed-keyed-table** element specifies information about the database table used to store mixed cache entries.

- The ***prefix*** parameter specifies a prefix string for the database table name.

The string-keyed-table Element

The **string-keyed-table** element specifies information about the database table used to store string based cache entries.

- The ***prefix*** parameter specifies a prefix string for the database table name.

The id-column Element

The **id-column** element specifies information about a database column that holds cache entry IDs.

- The ***name*** parameter specifies the name of the database column.

- The ***type*** parameter specifies the type of the database column.

The data-column Element

The **`data-column`** element contains information about a database column that holds cache entry data.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

The timestamp-column Element

The **`timestamp-column`** element specifies information about the database column that holds cache entry timestamps.

- The ***name*** parameter specifies the name of the database column.
- The ***type*** parameter specifies the type of the database column.

Report a bug

12.4.4.3. JdbcMixedCacheStore Configuration (Library Mode)

The following is a sample configuration for the **`mixedKeyedJdbcStore`**:

```
<loaders>
 <mixedKeyedJdbcStore xmlns="urn:infinispan:config:jdbc:5.2"
         fetchPersistentState="false"
         ignoreModifications="false"
         purgeOnStartup="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey2Strin
gMapper">
  <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>
</loaders>
```

The mixedKeyedJdbcStore Element

The **`mixedKeyedJdbcStore`** element uses the following parameters to configure the cache store:

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

- The **`purgeOnStartup`** parameter specifies whether the cache is purged when initally started.

- The **`key2StringMapper`** parameter specifies the class name of the Key2StringMapper used to map keys to strings for the database tables.

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:

- The ***`dropOnExit`*** parameter specifies whether the database tables are dropped upon shutdown.

- The ***`createOnStart`*** parameter specifies whether the database tables are created by the store on startup.

- The ***`prefix`*** parameter defines the string prepended to name of the target cache when composing the name of the cache bucket table.

The idColumn Element

The **`idColumn`** element defines the column where the cache key or bucket ID is stored. It used the following parameters:

- Use the ***`name`*** parameter to specify the name of the column used.

- Use the ***`type`*** parameter to specify the type of the column used.

The dataColumn Element

The **`dataColumn`** element specifies the column where the cache entry or bucket is stored.

- Use the ***`name`*** parameter to specify the name of the column used.

- Use the ***`type`*** parameter to specify the type of the column used.

The timestampColumn Element

The **`timestampColumn`** element specifies the column where the time stamp of the cache entry or bucket is stored.

- Use the ***name*** parameter to specify the name of the column used.
- Use the ***type*** parameter to specify the type of the column used.

Report a bug

12.4.5. Custom Cache Stores

12.4.5.1. About Custom Cache Stores

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

Report a bug

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

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

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

Report a bug

12.4.5.3. Custom Cache Store Configuration (Library Mode)

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

```
<loaders shared="true"
  preload="true">
 <loader class="org.infinispan.custom.CustomCacheStore">
  <properties>
   <property name="CustomCacheName"
      value="default"/>
  </properties>
 </loader>
</loaders>
```

Report a bug

12.5. FREQUENTLY ASKED QUESTIONS

12.5.1. About Asynchronous Cache Store Modifications

In 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

12.6. CACHE STORE TROUBLESHOOTING

12.6.1. IOExceptions with JdbcStringBasedCacheStore

An IOException **`Unsupported protocol version 48`** error when using **`JdbcStringBasedCacheStore`** 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, **`JdbcStringBasedCacheStore`** 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 13. CACHE LOADERS

13.1. ABOUT CACHE LOADERS

The cache loader provides 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

13.2. CACHE LOADERS AND CACHE STORES

JBoss Cache originally shipped with a **`CacheLoader`** interface and a number of implementations. 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

13.3. SHARED CACHE LOADERS

13.3.1. About 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

13.3.2. Enable Shared Cache Loaders

Library Mode

In 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

13.3.3. Invalidation Mode and Shared Cache Loaders

When used in conjunction with a shared cache loader, JBoss Data Grid's invalidation mode causes remote caches to refer to 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

13.3.4. The Cache Loader and Cache Passivation

In 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

13.3.5. Application Cacheloader Registration

It is not necessary to register an application cache loader for an isolated deployment. This is not a requirement in JBoss Data Grid because lazy deserialization is used to work around this problem.

Report a bug

13.4. CONNECTION FACTORIES

13.4.1. About Connection Factories

In 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

13.4.2. 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

13.4.3. 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

CHAPTER 14. CACHE MANAGERS

14.1. ABOUT CACHE MANAGERS

A Cache Manager is the primary mechanism used to retrieve a cache instance in JBoss Data Grid, and can be used as a starting point for using the cache.

Cache Managers are resource intensive and therefore a single cache manager for each Java Virtual Machine (JVM) is recommended unless a specific configuration requires multiple cache managers.

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

- it can create 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

14.2. MULTIPLE CACHE MANAGERS

14.2.1. Create Multiple Caches with a Single Cache Manager

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

14.2.2. Using Multiple Cache Managers

JBoss Data Grid allows multiple cache managers to be used. In most cases, such as with RPC 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

CHAPTER 15. EVICTION

15.1. 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.

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.

Report a bug

15.2. EVICTION OPERATIONS

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.

Report a bug

15.3. EVICTION USAGE

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.

See Also:

- Section 16.3, “Eviction and Expiration Comparison”

Report a bug

15.4. EVICTION STRATEGIES

15.4.1. About Eviction Strategies

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

Table 15.1. Eviction Strategies

Strategy Name	Operations	Use Cases
`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.	LRU is JBoss Data Grid's default eviction algorithm because it suits a large variety of production use cases.
`EvictionStrategy.UNORDERED`	Unordered eviction strategy. This strategy evicts entries without any ordered algorithm and may therefore evict entries that are required later. However, this entry 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.	-

Report a bug

15.4.2. LRU Eviction Algorithm Limitations

Despite being simple to understand, the Least Recently Used (LRU) eviction algorithm does not perform optimally in specific weak access locality use cases. 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

15.5. USING EVICTION

15.5.1. 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`***, JBoss Data Grid runs out of memory.

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

Procedure 15.1. 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

15.5.2. Default Eviction Configuration

In 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

15.5.3. Eviction Configuration Examples

Configure eviction in 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

Note that 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

15.5.4. Eviction Configuration Troubleshooting

In 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

15.6. EVICTION AND PASSIVATION

15.6.1. About 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.

See Also:

- Section 18.4, “Eviction and Passivation”

Report a bug

CHAPTER 16. EXPIRATION

16.1. ABOUT EXPIRATION

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

16.2. EXPIRATION OPERATIONS

Expiration in 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

16.3. EVICTION AND EXPIRATION COMPARISON

Expiration is a top-level construct in 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

16.4. CACHE ENTRY EXPIRATION NOTIFICATIONS

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

16.5. CONFIGURE EXPIRATION

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

Procedure 16.1. 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

16.6. MORTAL AND IMMORTAL DATA

16.6.1. About Data Mortality

In addition to storing entities, 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.

Report a bug

16.6.2. Default Data Mortality

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.

Report a bug

16.6.3. Configure Data Mortality

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

16.7. TROUBLESHOOTING

16.7.1. Expiration Troubleshooting

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 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

CHAPTER 17. THE L1 CACHE

17.1. 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 created when JBoss Data Grid's cache mode is set to distribution. The L1 cache is not available with other cache modes therefore any configuration related to L1 cache is ignored.

The temporary location used by the L1 cache can be changed from the default location.

Report a bug

17.2. L1 CACHE ENTRIES

17.2.1. L1 Cache Entries

The L1 cache contains entries retrieved from a remote cache. When the location of an entry changes in the cluster, the corresponding L1 cache entry is invalidated to prevent outdated cache entries.

When L1 is enabled, the cache consults the L1 cache before fetching an entry from a remote location to prevent an unnecessary fetch operation.

Report a bug

17.2.2. L1 Cache Configuration (Library Mode)

The following sample configuration shows the L1 cache default values in 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

17.2.3. L1 Cache Configuration (Remote Client-Server Mode)

The following sample configuration shows the L1 cache default values in 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**.

Report a bug

17.3. L1 CACHE OPERATIONS

17.3.1. 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

17.3.2. Using the L1 Cache with GET Operations

Multiple **GET** operations performed on the same key generate repeated remote calls. To reduce the number of unnecessary **GET** operations on the same key, enable L1 caching.

The L1 cache provides a local (and temporary) cache that stores values retrieved from remote caches. The location used for the L1 cache can be configured.

Report a bug

CHAPTER 18. ACTIVATION AND PASSIVATION MODES

18.1. ABOUT ACTIVATION AND PASSIVATION

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

18.2. 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

18.3. CONFIGURE PASSIVATION

In 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:

```
<loaders passivation="true"
        ... />
   ...
</loaders>
```

Report a bug

18.4. EVICTION AND PASSIVATION

18.4.1. About 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

18.4.2. 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

18.4.3. 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 18.1. 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

18.4.4. 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 18.2. 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

CHAPTER 19. TRANSACTIONS

19.1. 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.

IMPORTANT

With JBoss Data Grid 6.0.x, it is recommended to disable transactions in Remote Client-Server Mode. However, if an error displays warning of an **`ExceptionTimeout`** where JBoss Data Grid is **`Unable to acquire lock after {time} on key {key} for requester {thread}`**, enable transactions. This occurs because non-transactional caches acquire locks on each node they write on. Using transactions prevents deadlocks because caches acquire locks on a single node. This problem is resolved in JBoss Data Grid 6.1.

Report a bug

19.2. ABOUT TRANSACTION SYNCHRONIZATIONS

Synchronizations are a type of listener that receives notifications about events leading to the transaction life cycle. Synchronizations receive an event just before and just after completion of an operation.

Synchronizations are primarily useful when specific tasks must be triggered before or after an event. An common application for synchronizations are clearing out an application's caches after an operation (such as a transaction) concludes.

In JBoss Data Grid, synchronizations are only available in Library mode.

Report a bug

19.3. ABOUT THE TRANSACTION MANAGER

In 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

19.4. TRANSACTION RECOVERY

19.4.1. About Transaction Recovery

The Transaction Manager coordinates the recovery process and works with JBoss Data Grid to determine which transactions require manual intervention to complete operations. This process is known as transaction recovery.

Report a bug

19.4.2. Obtain the Transaction Manager From the Cache

Use the following configuration after initialization to obtain the TransactionManager from the cache:

Procedure 19.1. Obtain the Transaction Manager from the Cache

1. Define a **`transactionManagerLookupClass`** by adding the following property to your **`BasicCacheContainer`**'s configuration location properties:

   ```
   Configuration config = new ConfigurationBuilder()
   ...
     .transaction().transactionManagerLookup(new
   GenericTransactionManagerLookup())
   ```

2. Call **`TransactionManagerLookup.getTransactionManager`** as follows:

   ```
   TransactionManager tm =
   cache.getAdvancedCache().getTransactionManager();
   ```

Report a bug

19.4.3. Transaction Manager and XAResources

Despite being specific to the Transaction Manager, the transaction recovery process must provide a reference to a **`XAResource`** implementation to run **`XAResource.recover`** on it.

Report a bug

19.4.4. Obtain a XAResource Reference

To obtain a reference to a JBoss Data Grid **`XAResource`**, use the following API:

```
XAResource xar = cache.getAdvancedCache().getXAResource();
```

Report a bug

19.5. CONFIGURE TRANSACTIONS

19.5.1. Configure Transactions (Library Mode)

In 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:

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

Report a bug

19.5.2. Configure Transactions (Remote Client-Server Mode)

JBoss Data Grid transactions are configured at the cache level. The following is the syntax used to configure transactions in Remote Client-Server mode:

```
<cache>
 ...
  <transaction mode="{NONE,NON_XA,NON_DURABLE_XA,FULL_XA}" />
 ...
</cache>
```

The ***mode*** attribute sets the cache transaction mode. While valid values for this attribute are **`NONE`**, **`NON_XA`**, **`NON_DURABLE_XA`**, **`FULL_XA`**, JBoss Data Grid 6.1 supports only the **`NONE`** value because transactions are not available in Remote Client-Server mode.

Report a bug

19.6. TRANSACTION BEHAVIOR

19.6.1. Transaction Recovery Process

The following process outlines the transaction recovery process in JBoss Data Grid.

Procedure 19.2. 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

19.6.2. 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 JBoss Data Grid.

Example 19.1. Money Transfer from an Account Stored in a Database to an Account in JBoss Data Grid

1. The **`TransactionManager.commit()`** method is invoked to 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.

Report a bug

19.6.3. Default Distributed Transaction Behavior

JBoss Data Grid's default behavior is to register itself as a first class participant in distributed transactions through **`XAResource`**. In situations where JBoss Data Grid does not need to be a participant in a transaction, it can be notified about the lifecycle status (for example, prepare, complete, etc.) of the transaction via a synchronization.

Report a bug

19.6.4. Transaction/Batching and Invalidation Messages

When making modifications in invalidation mode without the use of batching or transactions, invalidation messages are dispatched after each modification occurs. If transactions or batching are in use, invalidation messages are sent following a successful commit.

Using invalidation with transactions or batching provides greater efficiency as transmitting messages in bulk results in less network traffic.

Report a bug

19.6.5. Transaction Memory and JMX Support

It is important to note that to use JMX to manage transaction recoveries, JMX support must be explicitly enabled.

Report a bug

19.6.6. Forced Commit and Rollback Operations

JBoss Data Grid uses JMX for operations that explicitly force transactions to commit or rollback. 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

19.6.7. Transactions and Exceptions

If a cache method returns a **`CacheException`** (or a subclass of the **`CacheException`**) within the scope of a JTA transaction, the transaction is automatically marked to be rolled back.

Report a bug

19.6.8. Transactions Spanning Multiple Cache Instances

Each cache operates as a separate, standalone Java Transaction API (JTA) resource. However, components can be internally shared by JBoss Data Grid for optimization, but this sharing does not affect how caches interact with a Java Transaction API (JTA) Manager.

Report a bug

19.7. TRANSACTION SYNCHRONIZATION

19.7.1. About Transaction Synchronizations

Synchronizations are a type of listener that receives notifications about events leading to the transaction life cycle. Synchronizations receive an event just before and just after completion of an operation.

Synchronizations are primarily useful when specific tasks must be triggered before or after an event. An common application for synchronizations are clearing out an application's caches after an operation (such as a transaction) concludes.

In JBoss Data Grid, synchronizations are only available in Library mode.

Report a bug

19.8. DEADLOCK DETECTION

19.8.1. About 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.

JBoss Data Grid provides deadlock detection to identify such deadlocks. Deadlock detection is set to **`disabled`** by default.

Report a bug

19.8.2. Enable Deadlock Detection

Deadlock detection in 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="1000"/>
```

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.

NOTE

JBoss Data Grid only allows deadlock detection to be configured in Library mode. This configuration is not available in Remote Client-Server mode.

Report a bug

CHAPTER 20. JGROUPS INTERFACES

20.1. ABOUT JGROUPS INTERFACES

JGroups is the underlying group communication library used to connect JBoss Data Grid instances. JGroups allows users to bind to an interface type rather than a specific (unknown) IP address.

Report a bug

20.2. CONFIGURE JGROUPS INTERFACES

Before JGroups Interface configuration, configure the JBoss Data Grid configuration (**`clustered.xml`** or **`standalone.xml`** depending on the type of deployment) file to a key word rather than a dotted decimal or symbolic IP address as follows:

```
<socket-binding name="jgroups-udp" ...  interface="site-local"/>
```

Then, configure the JGroups Interface as follows:

```
<interfaces>
 <interface name="link-local"><link-local-address/></interface>
 <interface name="site-local"><site-local-address/></interface>
 <interface name="global"><any-address/></interface>
 <interface name="non-loopback"><not><loopback/></not></interface>
</interfaces>
```

The configuration values used in the example are as follows:

- ***`link-local`***: Uses a **`169.x.x.x`** or **`254.x.x.x`** address. This suits the traffic within one box.
- ***`site-local`***: Uses a private IP address, for example **`192.168.x.x`**. This prevents extra bandwidth charged from GoGrid, and similar providers.
- ***`global`***: Picks a public IP address. This should be avoided for replication traffic.
- ***`non-loopback`***: Uses the first address found on an active interface that is not a**`127.x.x.x`** address.

Report a bug

20.3. ABOUT BINDING SOCKETS

20.3.1. About Group and Individual Socket Binding

Group interfaces can be used to bind all socket bindings within a given socket binding group or to bind individual sockets.

Report a bug

20.3.2. 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

20.3.3. 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

20.4. JGROUPS FOR CLUSTERED MODES

20.4.1. Configure JGroups for Clustered Modes

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

20.4.2. Pre-Configured JGroups Files

20.4.2.1. Using a Pre-Configured JGroups File

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`**

Report a bug

20.4.2.2. jgroups-udp.xml

`jgroups-udp.xml` is a pre-configured JGroups file in 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.
- minimizes inefficient use of sockets.

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 20.1. 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

System Property	Description	Default	Required?
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

20.4.2.3. jgroups-tcp.xml

`jgroups-tcp.xml` is a pre-configured JGroups file in JBoss Data Grid. The **`jgroups-tcp.xml`** configuration

- uses TCP as a transport and UDP multicast for discovery.
- is better suited to smaller clusters (less than 8 nodes) only when using distribution mode. This is because TCP is more efficient as a point-to-point protocol.

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 20.2. 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

CHAPTER 21. MANAGEMENT TOOLS IN JBOSS DATA GRID

21.1. JAVA MANAGEMENT EXTENSIONS (JMX)

21.1.1. 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 JBoss Data Grid to expose management and statistical information.

Report a bug

21.1.2. Using JMX with JBoss Data Grid

Management in 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

21.1.3. 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.

Report a bug

21.1.4. 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

21.1.5. 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

21.1.6. Disabling the CacheStore via JMX

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

21.1.7. 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

21.1.8. About MBeans

An **MBean** represents a manageable resource such as a service, component, device or an application.

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

See Also:

- Appendix A, *List of JMX MBeans in JBoss Data Grid*

Report a bug

21.1.9. 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 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

21.1.10. 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 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 22. JBOSS OPERATIONS NETWORK (JON)

22.1. 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 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.

Report a bug

22.2. DOWNLOAD JBOSS OPERATIONS NETWORK (JON)

22.2.1. Prerequisites for Installing JBoss Operations Network (JON)

In order to install JBoss Operations Network in 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

22.2.2. Download JBoss Operations Network

Use the following procedure to download JBoss Operations Network from the Customer Service Portal:

Procedure 22.1. 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 ON 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

22.2.3. Remote JMX Port Values

A port value must be provided to allow 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

22.2.4. Download JBoss Operations Network (JON) Plugin for JBoss Data Grid

Complete this task to download the JBoss Operations Network (JON) plugin for JBoss Data Grid from the Red Hat Customer Portal.

Procedure 22.2. 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

22.3. JBOSS OPERATIONS NETWORK SERVER INSTALLATION

22.3.1. Installing JBoss Operations Network Server Prerequisites

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.

Prerequisites

In order to install the JBoss Operations Network, you must have:

- Downloaded the JBoss Operations Network Base Distribution.

- Downloaded and installed Java 6 or Java 7 JDK.
- Properly installed PostgreSQL database for JBoss Operations Network.

Additionally, the following are prerequisites to monitor JBoss Data Grid instances:

- Downloaded both the JBoss Data Grid server RHQ plug-in and the JBoss Application Server 7 plug-in (for Remote Client-Server Mode).
- Downloaded the JBoss Data Grid Library RHQ plug-in (for Library Mode).

Report a bug

22.3.2. Installing the JBoss Operations Network Server on Linux

Use the following procedure to install the JBoss Operations Network Server on Linux.

Procedure 22.3. Installing the Server on Linux

1. Stop any currently running JBoss Operations Network instances.
2. Download the JBoss Operations Network binaries from the Customer Support Portal at https://access.redhat.com.
   - In the Customer Support Portal, click **`Software`**, and then select **`JBoss Operations Network`** in the product drop-down box.
   - Download the **`JBoss Operations Network 3.1.2 Base Distribution`** package by clicking the **`Download`** icon.
   - There are additional plug-in packs available for EAP, EDS, EWS, and SOA-P. If any of those plug-ins will be used with the JBoss Operations Network server, then download them as well.
3. Unzip the server distribution to the directory where will be executed from.

   ```
   cd /opt

   unzip jon-server-3.1.2.0.GA1.zip
   ```

   This creates a version-specific installation directory, **`/opt/jon-server-3.1.2.0.GA1`**. A directory with this name should not exist prior to the unzip operation.
4. Run the JBoss Operations Network server:

   ```
   serverRoot/jon-server-3.1.2.0.GA1/bin/rhq-server.sh start
   ```
5. Set up the JBoss Operations Network server using the web installer, available at **`http://localhost:7080/`**, or by editing the configuration file.

   For more detailed information about configuring JBoss Operations Network, refer to the JBoss Operations Network *Installation Guide*.

Report a bug

22.3.3. Installing the JBoss Operations Network Server on Windows

Use the following procedure to install the JBoss Operations Network Server on Windows.

Procedure 22.4. Installing the Server on Windows

1. Stop any currently running JBoss Operations Network instances.
2. Download the JBoss Operations Network binaries from the Customer Support Portal at https://access.redhat.com.
   - In the Customer Support Portal, click **`Software`**, and then select **`JBoss Operations Network`** in the product drop-down box.
   - Download the **`JBoss Operations Network 3.1.2 Base Distribution`** package by clicking the **`Download`** icon.
3. Create a directory for the server to be installed in.

   Use a relatively short name. Path names longer than 19 characters can cause problems running the server or executing some tasks.
4. Unzip the server distribution to the desired installation directory.

   ```
   C:> winzip32 -e jon-server-3.1.2.0.GA1.zip C:\jon\jon-server-
   3.1.2.0.GA1
   ```
5. Set the directory path to the JDK installation for a 32-bit JDK. For example:

   ```
   set RHQ_SERVER_JAVA_HOME=C:\Program Files\Java\jdk1.6.0_29
   ```

   The default Java service wrapper included with JBoss Operations network requires a 32-bit JVM, so the Java preference set for the server must be a 32-bit JDK.

   NOTE

   The JBoss Operations Network server must use a 32-bit JVM even on 64-bit systems.

   Running the server or agent with a 32-bit JVM does not in any way affect how JBoss Operations Network manages other resources which may run with a 64-bit JVM. JBoss Operations Network can still manage those resources and those resources can still use the 64-bit Java libraries for their own processes.
6. Install the JBoss Operations Network server as a Windows service. This action must be "Run as Administrator."

   ```
   C:\rhq\jon-server-3.1.2.0.GA1\bin\rhq-server.bat install
   ```
7. Start the JBoss ON server. This action must be "Run as Administrator."

   ```
   C:\rhq\jon-server-3.1.2.0.GA1\bin\rhq-server.bat start
   ```

8. Set up the JBoss Operations Network server using the web installer, available at **`http://localhost:7080/`**, or by editing the configuration file.

   For more detailed information about configuring JBoss Operations Network, refer to the JBoss Operations Network *Installation Guide*.

Report a bug

22.4. JBOSS OPERATIONS NETWORK AGENT

22.4.1. About the 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.

Report a bug

22.4.2. JBoss Operations Network Agent Installation Prerequisites

The following steps are prerequisites for installing the JBoss Operations Network Agent in JBoss Data Grid.

1. Install one or more JBoss Operations Network servers.

2. Upgrade any existing pre-installed JBoss Operations Network Agents.

3. Preconfigure multiple agents for easily automated installs of multiple JBoss Operations Network Agents.

Report a bug

22.4.3. Installing the JBoss Operations Network Agent

The JBoss Operations Network ships with a bundled JBoss Operations Network Agent.

Use the following procedure to install the JBoss Operations Network Agent using the **`agent update binary`**.

Procedure 22.5. Install the JBoss Operations Network Agent

1. **Download the agent .jar**
   Download the JBoss Operations Network agent **`.jar`** file from **`http://JONserverAddress:7080/agentupdate/download`**.

2. **Install the agent**
   Unpack and install the agent using the following command:

   ```
   java -jar downloaded_agent_jar_file.jar --install
   ```

3. **Set server discovery frequency**

Add the following to the **`agentRoot/conf/agent-configuration.xml`** file:

```
<!-- how often server discovery is run -->
<entry key="rhq.agent.plugins.server-discovery.period-secs"
value="20"/>
```

This step will ensure less time between automatic resource discovery attempts.

4. **Start the agent**
   Start the JBoss Operations Network agent by running:

```
agentRoot/rhq-agent/bin/rhq-agent.sh  (to start with clean config ->
--cleanconfig )
```

For more detailed information, refer to the JBoss Operations Network *Installation Guide*.

Report a bug

22.4.4. Configure the JBoss Operations Network Agent

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 JBoss Data Grid is running, however where there are multiple machines an agent must be installed on each machine.

Procedure 22.6. Basic JBoss Operations Network Agent Configuration

To start a basic JBoss Operations Network operation, activate the embedded agent that ships with the JBoss Operations Network distribution:

1. Locate the **`rhq-server-properties`** in the **`/bin`** folder of the JBoss Operations Network distribution.

2. Enable the agent by changing the following property in the configuration file:

```
#Embedded RHQ Agent
rhq.server.embedded-agent.enabled=true
```

Report a bug

22.4.5. Tools and Operations

22.4.5.1. About Management Tools

Managing 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.

See Also:

- Chapter 21, *Management Tools in JBoss Data Grid*

Report a bug

22.4.5.2. Accessing Data via URLs

Caches that have been configured with a REST interface have access to 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

22.4.5.3. Limitations of Map Methods

Specific **`Map`** methods, such as **`size()`**, **`values()`**, **`keySet()`** and **`entrySet()`**, can be used with certain limitations with 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

22.5. JBOSS OPERATIONS NETWORK FOR REMOTE CLIENT-SERVER MODE

22.5.1. JBoss Operations Network in Remote Client-Server Mode

In 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

22.5.2. 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 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 JBoss Application Server 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 JBoss Application Server 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

22.6. JBOSS OPERATIONS NETWORK FOR LIBRARY MODE

22.6.1. JBoss Operations Network in Library Mode

In 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

22.6.2. Installing the JBoss Operations Network Plug-in (Library Mode)

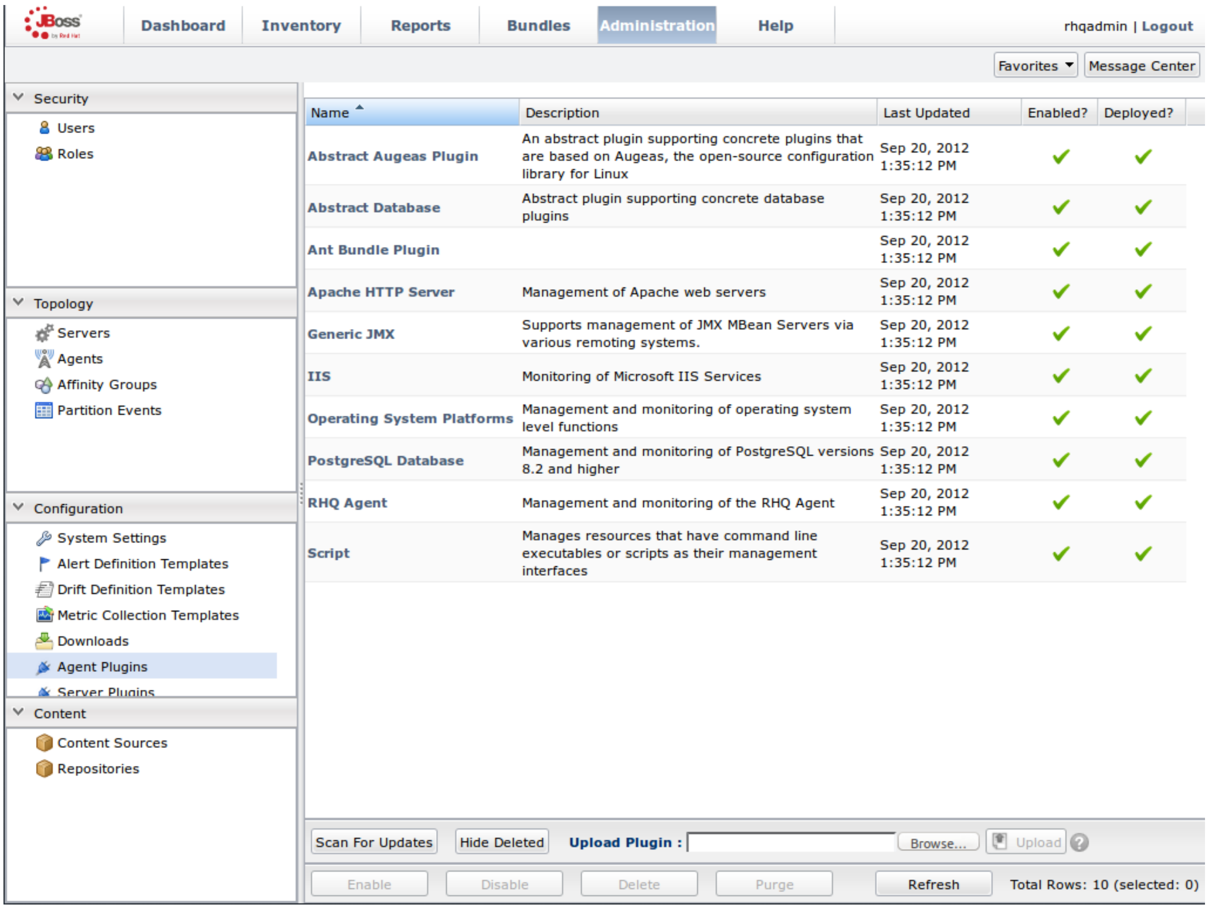
Use the following procedure to install the JBoss Operations Network plug-in for JBoss Data Grid's Library mode.

Procedure 22.7. 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.



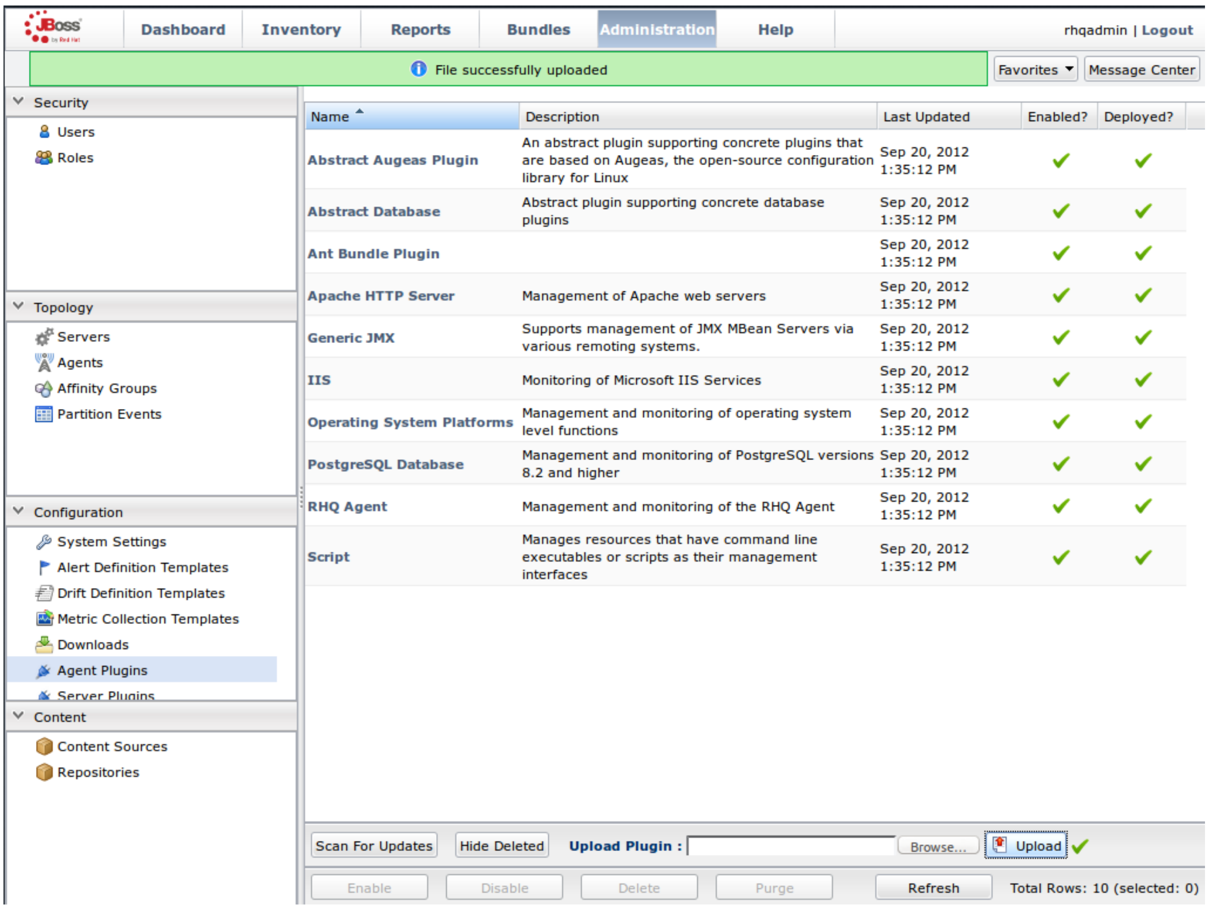
Name	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 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 22.1. 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	✓	✓

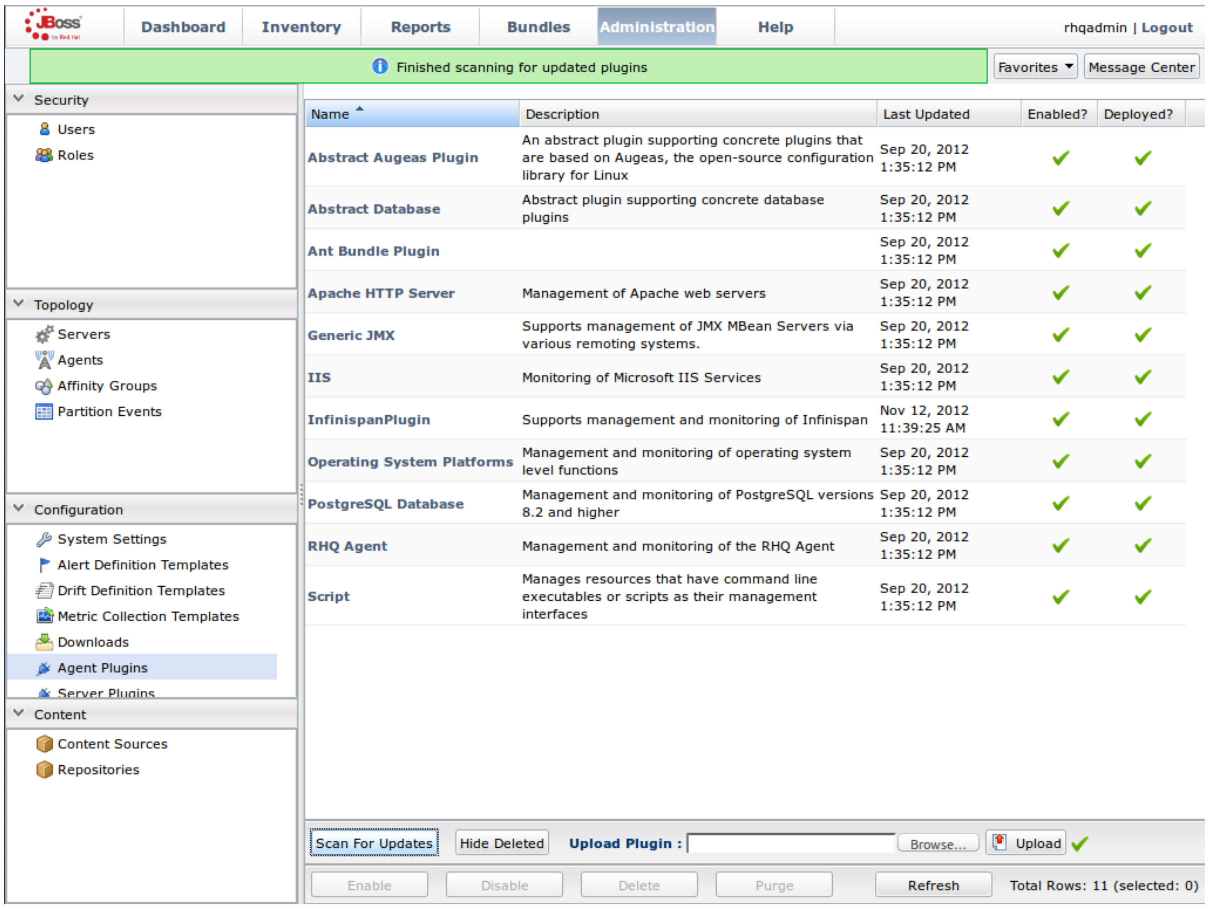
Total Rows: 10 (selected: 0)

Figure 22.2. 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	✓	✓

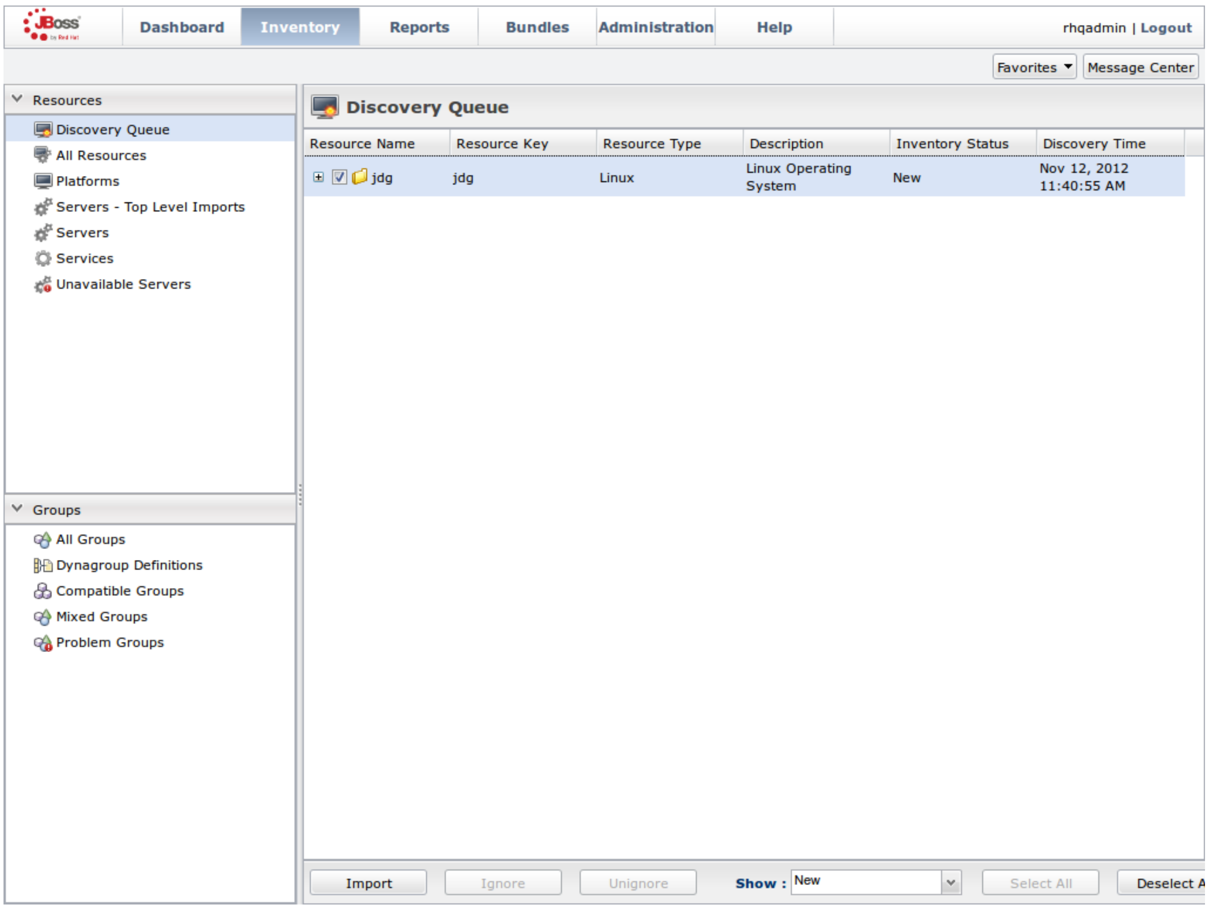
Total Rows: 11 (selected: 0)

Figure 22.3. 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

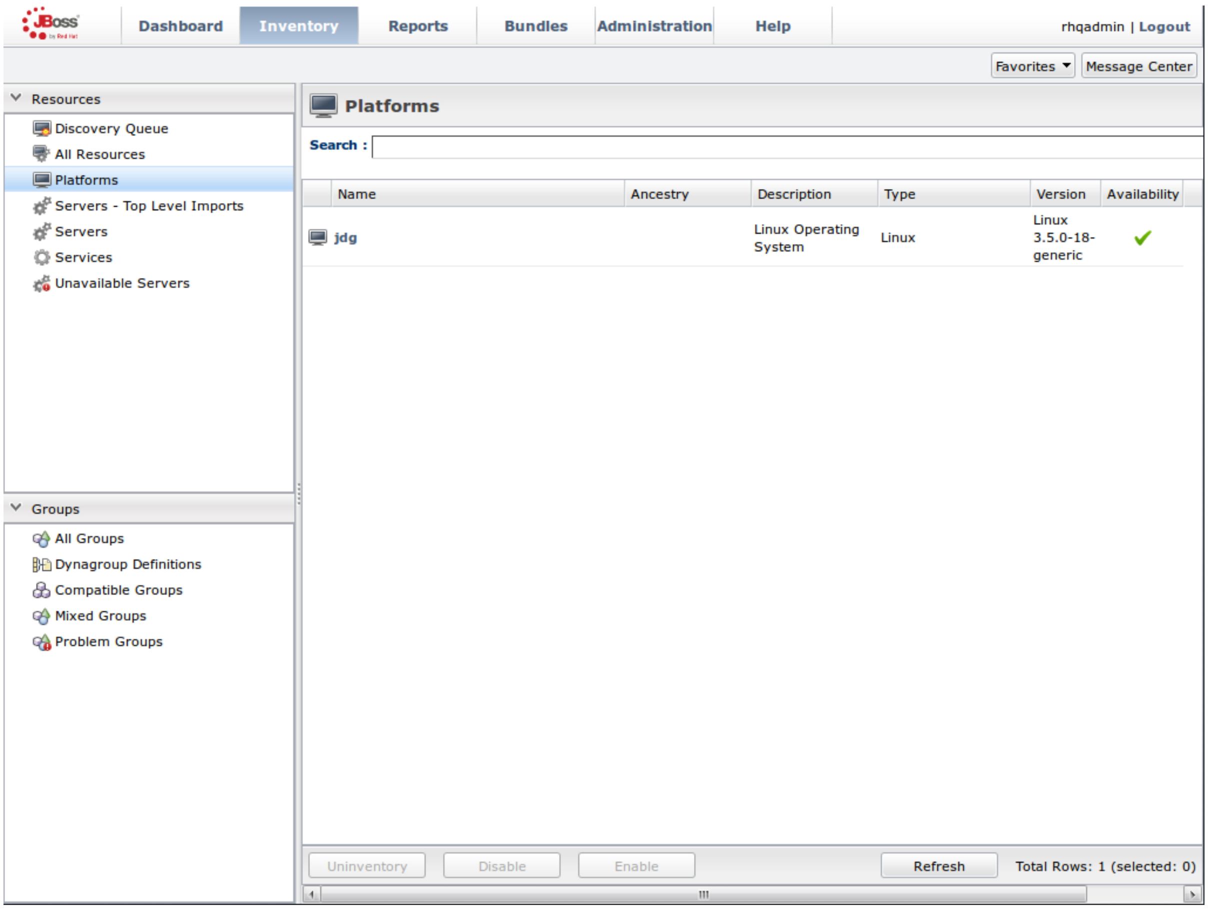
Resource Name	Resource Key	Resource Type	Description	Inventory Status	Discovery Time
jdg	jdg	Linux	Linux Operating System	New	Nov 12, 2012 11:40:55 AM

Figure 22.4. 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.



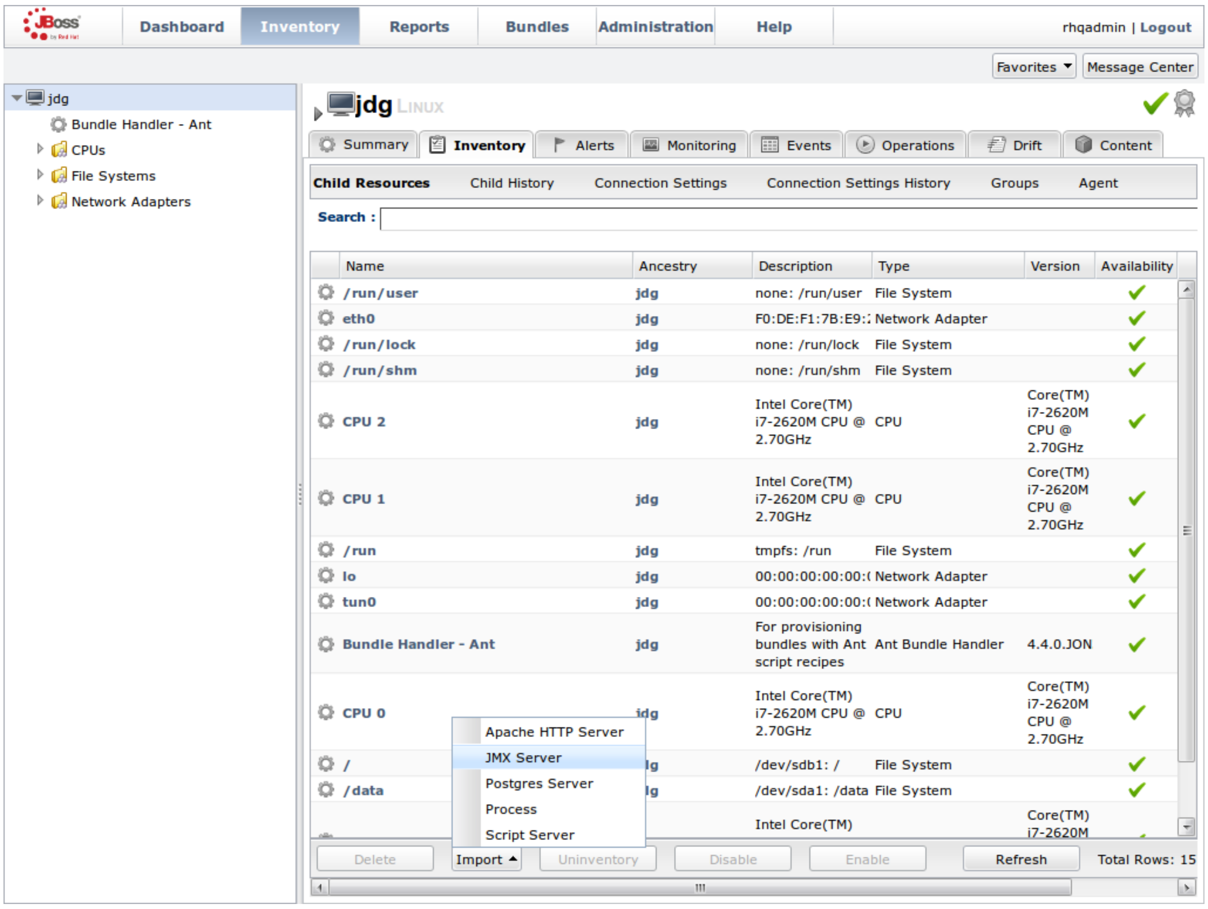
Name	Ancestry	Description	Type	Version	Availability
jdg		Linux Operating System	Linux	Linux 3.5.0-18-generic	✓

Figure 22.5. 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 22.6. 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.

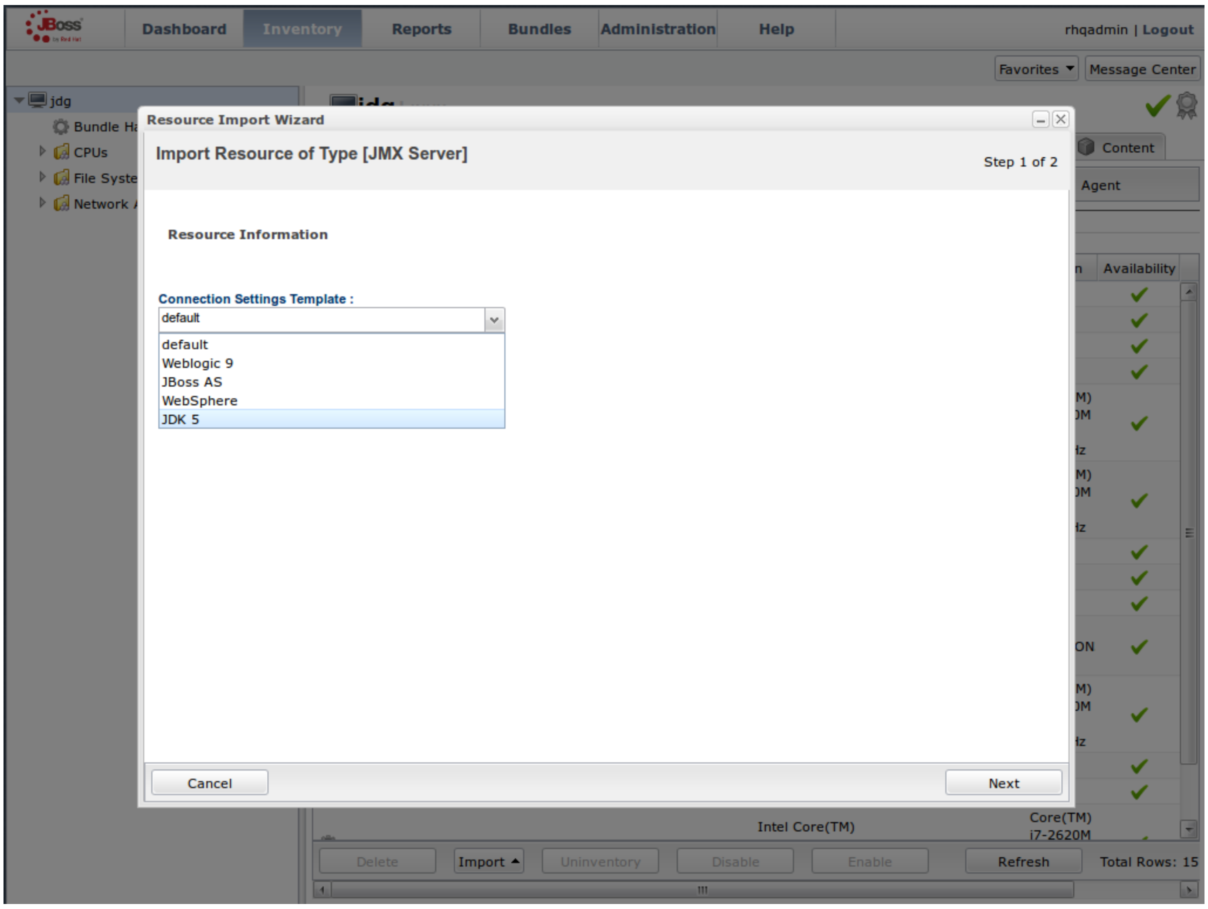


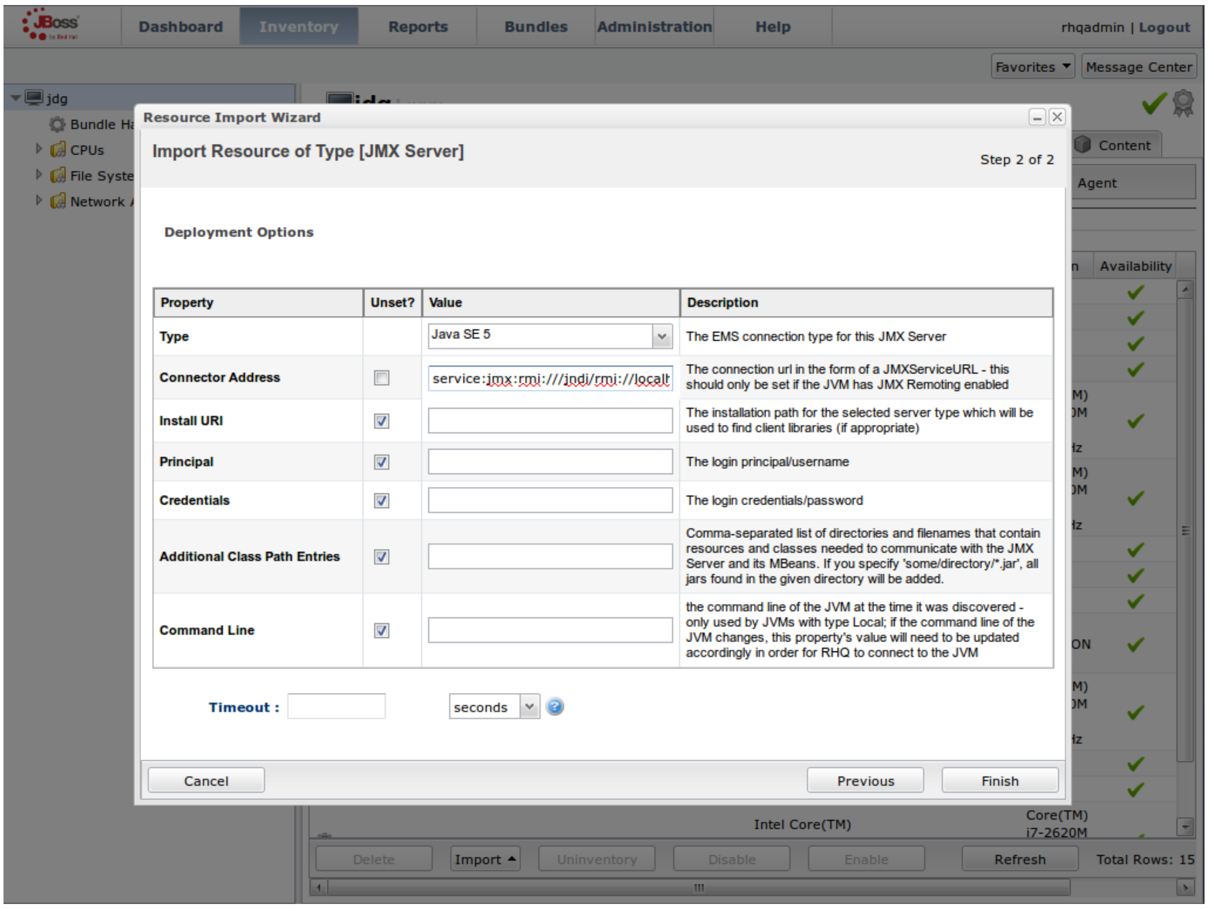
Figure 22.7. Select the JDK 5 Template.

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 22.8. Modify the values in the Deployment Options screen.

9. **View Cache Statistics and Operations**

   a. Click **`Refresh`** to refresh the list of servers.

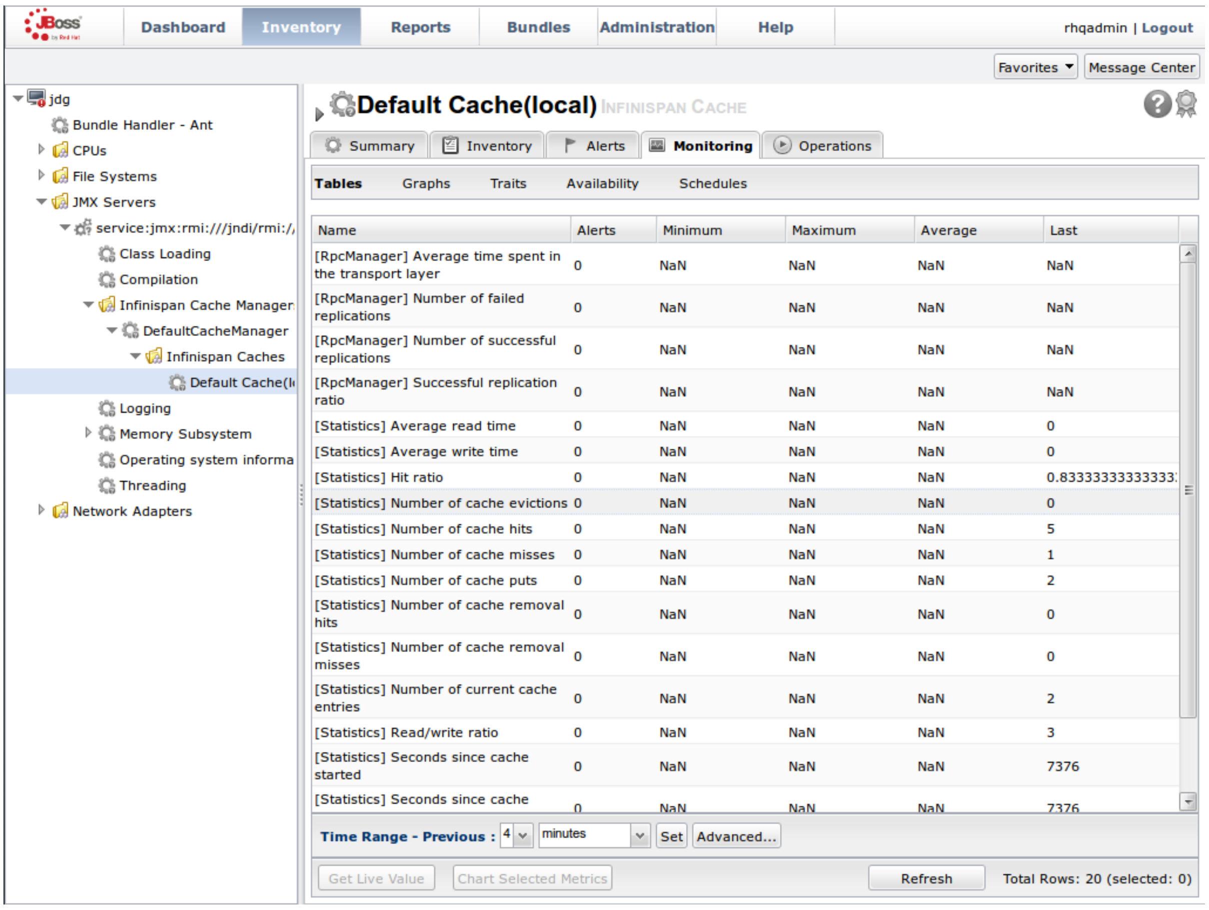
   b. The **JMX Servers** tree in the panel on the left side of the screen contains the **Infinispan Cache Managers** node, which contains the available cache managers. The available cache managers contain the available caches.

   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 22.9. Metrics and operational data relayed through JMX is now available in the JBoss Operations Network console.

Report a bug

22.6.3. Manually Adding JBoss Data Grid Instances in Library Mode

To add 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

22.7. JBOSS OPERATIONS NETWORK REMOTE-CLIENT SERVER PLUGIN

22.7.1. JBoss Operations Network Plugin Metrics

Table 22.1. JBoss ON Metrics 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 22.2. JBoss ON 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.

JBoss ON Metrics for Connectors

The metrics provided by the JBoss Operations Network (JON) plugin for 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, refer to the *Getting Started Guide*.

Table 22.3. JBoss ON Metrics for the Connectors

Metric Name	Display Name	Description
bytesRead	Bytes Read	Number of bytes read.
bytesWritten	Bytes Written	Number of bytes written.

Report a bug

22.7.2. JBoss Operations Network Plugin Operations

Table 22.4. 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.

JBoss ON Plugin Operations for the Cache Backups

The cache backups used for these operations are configured using cross-datacentre replication. In the JBoss Operations Network (JON) User Interface, each cache backup is the child of a cache. For more information about cross-datacentre replication, refer to Section 23.1, “About Cross-Datacenter Replication”

Table 22.5. JBoss ON 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)

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

22.7.3. JBoss Operations Network Plugin Attributes

Table 22.6. JBoss ON Plugin Attributes for the Cache (Transport)

Attribute Name	Type	Description
cluster	string	The name of the group communication cluster.

Attribute Name	Type	Description
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

22.8. MONITOR JBOSS ENTERPRISE APPLICATION PLATFORM 6 APPLICATIONS USING LIBRARY MODE

22.8.1. Prerequisites

The following is a list of common prerequisites for both Section 22.8.2, “Monitor an Application Deployed in Standalone Mode” and Section 22.8.3, “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, refer to Section 22.4.1, “About the 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 Server instance. As an example, running the JBoss Operations Network (JON) agent as a user with root priviledges and the JBoss Enterprise Application Platform process under a different user does not work as expected and should be avoided.
- An installed JBoss Operations Network (JON) plugin for Library Mode. For more information, refer to Section 22.6.2, “Installing the JBoss Operations Network Plug-in (Library Mode)”
- A custom application using JBoss Data Grid's Library mode. This application must have **`jmxStatistics`** enabled (either declaratively or programmatically). For more information, refer to Section 21.1.4, “Enable JMX for Cache Instances”

- The Java Virtual Machine (JVM) must be configured to expose the JMX MBean Server. For the Oracle/Sun JDK, refer to 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

22.8.2. 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 22.8. 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 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

22.8.3. 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 22.9. 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 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

22.9. 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

CHAPTER 23. CROSS-DATACENTER REPLICATION

23.1. ABOUT CROSS-DATACENTER REPLICATION

In 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 should be in a different physical location than the others.

Report a bug

23.2. CROSS-DATACENTER REPLICATION OPERATIONS

JBoss Data Grid's Cross-Datacenter Replication operation is explained through the use of an example, as follows:

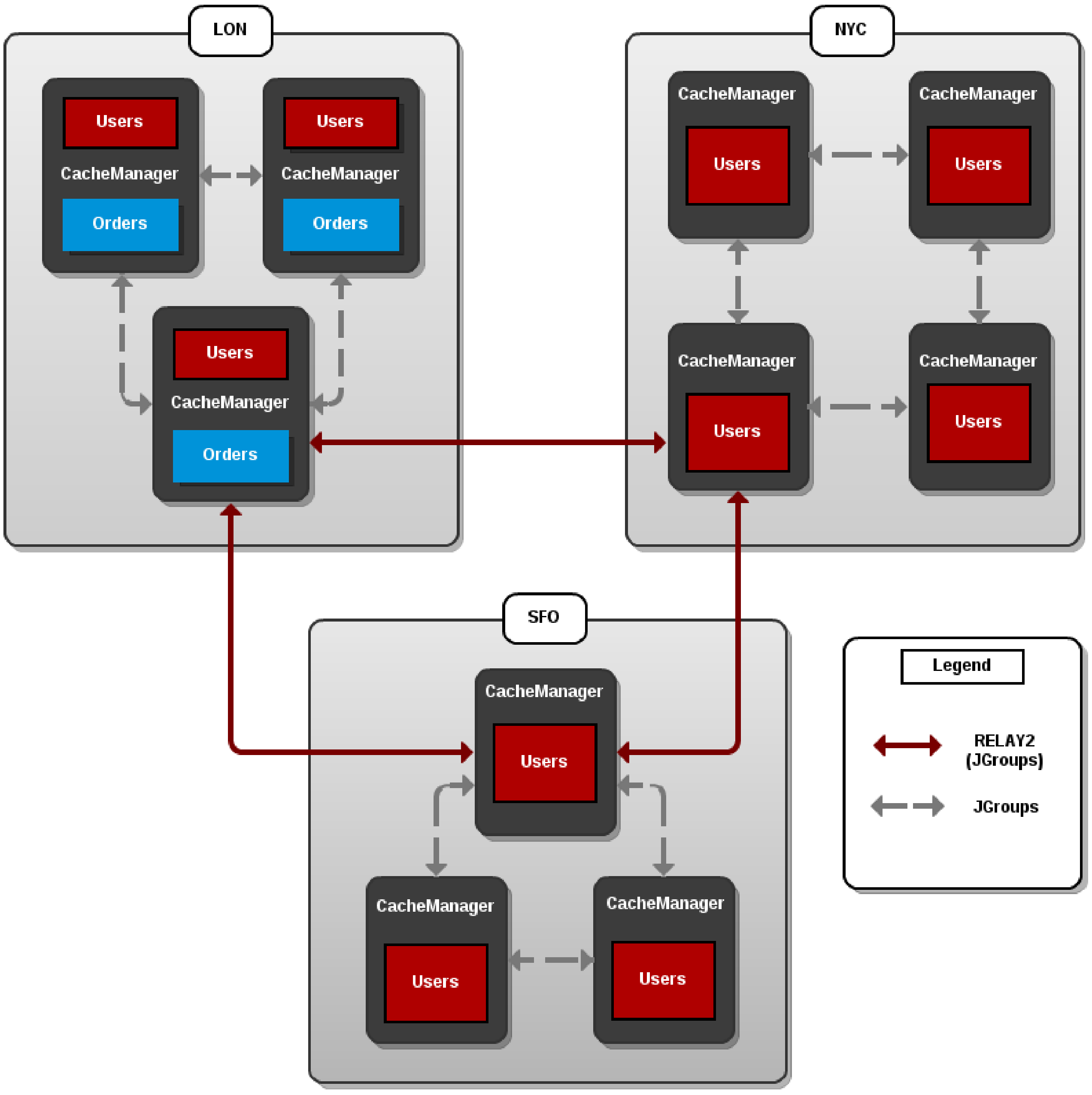


Figure 23.1. Cross-Datacenter Replication Example

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 about **RELAY2**, refer to Section B.7, “About RELAY2”

Report a bug

23.3. CONFIGURE CROSS DATACENTRE REPLICATION

23.3.1. Configure Cross-Datacentre Replication (Remote Client-Server Mode)

In Red Hat JBoss Data Grid's Remote Client-Server mode, cross-datacentre replication is set up as follows:

Procedure 23.1. Set Up Cross-Datacentre 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"/>
  </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 23.2, “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

23.3.2. Configure Cross-Datacentre Replication (Library Mode)

When configuring Cross-Datacentre 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 Red Hat JBoss Data Grid's Library mode, cross-datacentre replication is set up as follows:

Procedure 23.2. Configure Cross-Datacentre Replication (Library Mode)

1. **Configure the Local Site**

   a. 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>
```

   b. 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>
```

2. **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"
                  can_become_site_master="true"
                  max_site_masters="1"/>
</config>>
```

3. **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>
```

4. **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`***.

5. **File Locations**
   Ensure all the created files are on the classpath before using the new configurations.

Report a bug

23.3.3. Configure Cross-Datacentre Replication Programmatically

The programmatic method to configure cross-datacentre replication in JBoss Data Grid is as follows:

```
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")
```

The configuration uses the same three site example mentioned previously (where **`NYC`** and **`SFO`** are backup sites for local site **`LON`**).

Next, to programmatically configure which sites **`NYC`** and **`SFO`** are backups for, use the following:

```
ConfigurationBuilder cb = new ConfigurationBuilder();
cb.sites().backupFor().remoteCache("users").remoteSite("LON");
```

Report a bug

23.4. TAKING A SITE OFFLINE

23.4.1. About Taking Sites Offline

In JBoss Data Grid's Cross-datacentre 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

23.4.2. Taking a Site Offline (Remote Client-Server Mode)

In 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

23.4.3. Taking a Site Offline (Library Mode)

In 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

23.4.4. Taking a Site Offline (Programmatically)

To configure taking a Cross-datacentre replication site offline automatically in JBoss Data Grid programmatically:

```
lon.sites().addBackup()
      .site("NYC")
      .backupFailurePolicy(BackupFailurePolicy.FAIL)
      .strategy(BackupConfiguration.BackupStrategy.SYNC)
      .takeOffline()
         .afterFailures(500)
         .minTimeToWait(10000);
```

Report a bug

23.4.5. Taking a Site Offline via JBoss Operations Network (JON)

A site can be taken offline in JBoss Data Grid using the JBoss Operations Network Metrics. For a list of the metrics, refer to Section 22.7.1, “JBoss Operations Network Plugin Metrics”

Report a bug

23.4.6. Taking a Site Offline via the CLI

Use JBoss Data Grid's Command Line Interface (CLI) to manually take a site from a cross-datacentre 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, refer to the *Developer Guide*'s chapter on the JBoss Data Grid Command Line Interface (CLI)

Report a bug

23.4.7. 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, refer to Section A.21, “XSiteAdmin”

Report a bug

APPENDIX A. LIST OF JMX MBEANS IN 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
getActivations	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
ConfigurationAsXmlString	Returns the cache configuration as XML string.	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
CacheLoaders	Returns a collection of cache loader types which are configured and enabled.	Collection	No

Table A.6. Operations

Name	Description	Signature
disableCacheLoader	Disable all cache loaders of a given type, where type is a fully qualified class name of the cache loader to disable.	void disableCacheLoader(String p0)
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	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

Name	Description	Type	Writable
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.CacheStoreInterceptor`

The CacheStore component stores entries to a CacheStore from memory.

Table A.9. Attributes

Name	Description	Type	Writable
CacheLoaderStores	Number of cache loader stores.	long	No

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 transaction that were roll backed due to deadlocks.	long	No
DetectedRemoteDeadlocks	Number of remote transaction that were roll backed due to deadlocks.	long	No
LocallyInterruptedTransactions	Number of locally originated transactions that were interrupted as a deadlock situation was detected.	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

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.

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 IspnQL 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. JMXSTATSCOMMANDINTERCEPTOR

`org.infinispan.interceptors.base.JmxStatsCommandInterceptor`

Table A.18. Attributes

Name	Description	Type	Writable
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table A.19. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.11. LOCKMANAGER

`org.infinispan.util.concurrent.locks.LockManagerImpl`

The LockManager component handles MVCC locks for entries.

Table A.20. Attributes

Name	Description	Type	Writable
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
NumberOfLocksHeld	The number of exclusive locks that are held.	int	No

Report a bug

A.12. MASSINDEXER

`org.infinispan.query.MassIndexer`

The MassIndexer component rebuilds the index using cached data.

Table A.21. TOperations

Name	Description	Signature
start	Starts rebuilding the index.	void start()

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.22. Attributes

Name	Description	Type	Writable
Passivations	Number of passivation events.	String	No

Table A.23. 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.24. 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 JBoss Data Grid to another.

Table A.25. Operations

Name	Description	Signature
disconnectSource	Disconnects the target cluster from the source cluster according to the specified migrator.	void disconnectSource(String p0)

Name	Description	Signature
recordKnownGlobalKeyset	Dumps the global known keyset to a well-known key for retrieval by the upgrade process.	void recordKnownGlobalKeyset()
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.

Table A.26. 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
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table A.27. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.17. STATETRANSFERMANAGER

`org.infinispan.statetransfer.StateTransferManager`

The StateTransferManager component handles state transfer in JBoss Data Grid.

Table A.28. 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
StateTransferInProgress	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.29. 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

Name	Description	Type	Writable
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
TimeSinceReset	Number of seconds since the cache statistics were last reset.	long	No

Table A.30. 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.31. 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.32. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

Report a bug

A.20. TRANSPORT

`org.infinispan.server.core.transport.Transport`

The Transport component manages read and write operations to and from the server.

Table A.33. Attributes

Name	Description	Type	Writable
HostName	Returns the host to which the transport binds.	String	No
IdleTimeout	Returns the idle timeout.	String	No

Name	Description	Type	Writable
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
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.34. 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(String p0)
getTakeOfflineMinTimeToWait	Returns the value of the 'minTimeToWait' for the 'TakeOffline' functionality.	String getTakeOfflineMinTimeToWait(String p0)
setTakeOfflineAfterFailures	Amends the values for 'afterFailures' for the 'TakeOffline' functionality on all the nodes in the cluster.	String setTakeOfflineAfterFailures(String p0, int p1)
setTakeOfflineMinTimeToWait	Amends the values for 'minTimeToWait' for the 'TakeOffline' functionality on all the nodes in the cluster.	String setTakeOfflineMinTimeToWait(String p0, long p1)
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, JBoss Data Grid's consistency guarantee remains intact. The consistency guarantee is as follows:

1. If Key **`K`** is hashed to nodes **`{A,B}`** and transaction **`TX1`** acquires a lock for **`K`** on, for example, node **`A`**.

2. If another cache access occurs on node **`B`**, or any other node, and **`TX2`** attempts to lock **`K`**, it 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 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 JBoss Data Grid to expose management and statistical information.

Report a bug

B.4. ABOUT JBOSS CACHE

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 JBoss Data Grid.

Report a bug

B.5. ABOUT JSON

The JavaScript Object Notation (JSON) is a lightweight data exchange format. It is readable and writable by both humans and machines. Despite being derived from JavaScript, JSON is language-independent.

Report a bug

B.6. ABOUT LUCENE DIRECTORY

The **Lucene Directory** is the Input Output API for **Apache Lucene** to store the query indexes.

The most common **Lucene Directory** implementations used with JBoss Data Grid's Query Module are:

- Ram - stores the index in a local map to the node. This index cannot be shared.
- File system - stores the index in a locally mounted file system. This could be a network shared file system, however sharing in this manner is not recommended.
- JBoss Data Grid - stores the indexes in a different set of dedicated JBoss Data Grid caches. These caches can be configured as replicated or distributed in order to share the index between nodes.

The Query Module is not aware of where indexes are stored.

NOTE

The Lucene Directory provided by JBoss Data Grid is not limited to the Query Module. It can seamlessly replace any other requirement to store Lucene indexes where your application uses Lucene directly.

The JBoss Data Grid Query Module ships with several **Lucene Directory** implementations, and accepts third party implementations.

IMPORTANT

The Query Module is currently only available as a Technical Preview for JBoss Data Grid 6.1.

Report a bug

B.7. ABOUT RELAY2

JGroups includes the **RELAY2** protocol, which is used for communication between sites in JBoss Data Grid's Cross-Site Replication.

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.

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.8. ABOUT RETURN VALUES

Values returned by cache operations are referred to as return values. In 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.9. 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.10. 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.11. 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.12. THE EXTERNALIZER

B.12.1. 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 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.12.2. Internal Externalizer Implementation Access

Externalizable objects should not access JBoss Data Grids Externalizer implementations. The following is an example of the incorrect method to deal with this:

```
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 the correct method to deal with this situation:

```
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.13. HASH SPACE ALLOCATION

B.13.1. About Hash Space Allocation

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.13.2. Locating a Key in the Hash Space

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.13.3. Requesting a Full Byte Array

How can I request the 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 6.1.0-23	**Thu Jan 30 2014**	**Misha Husnain Ali**
Revision 6.1.0-22.400	**2013-10-31**	**Rüdiger Landmann**
Rebuild with publican 4.0.0		
Revision 6.1.0-22	**Tue Oct 22 2013**	**Misha Husnain Ali**
Updated Cross DataCentre Configuration for Library mode.		
Revision 6.1.0-21	**Tue Aug 06 2013**	**Misha Husnain Ali**
Updated product name.		
Revision 6.1.0-20	**Sun Apr 07 2013**	**Misha Husnain Ali**
Updated with patch for BZ-948988.		