



Red Hat JBoss Data Grid 7.0

Getting Started Guide

For use with Red Hat JBoss Data Grid 7.0

Red Hat JBoss Data Grid 7.0 Getting Started Guide

For use with Red Hat JBoss Data Grid 7.0

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

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

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

Christian Huffman
Red Hat Engineering Content Services
chuffman@redhat.com

Legal Notice

Copyright © 2016 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide outlines introductory concepts and operations within Red Hat JBoss Data Grid 7.0

Table of Contents

PART I. INTRODUCING RED HAT JBOSS DATA GRID	5
CHAPTER 1. RED HAT JBOSS DATA GRID	6
1.1. SUPPORTED CONFIGURATIONS	6
1.2. COMPONENTS AND VERSIONS	6
1.3. RED HAT JBOSS DATA GRID USAGE MODES	6
1.4. RED HAT JBOSS DATA GRID BENEFITS	8
1.5. RED HAT JBOSS DATA GRID VERSION INFORMATION	9
1.6. RED HAT JBOSS DATA GRID CACHE ARCHITECTURE	10
1.7. RED HAT JBOSS DATA GRID APIS	12
PART II. DOWNLOAD AND INSTALL RED HAT JBOSS DATA GRID	13
CHAPTER 2. DOWNLOAD RED HAT JBOSS DATA GRID	14
2.1. RED HAT JBOSS DATA GRID INSTALLATION PREREQUISITES	14
2.2. JAVA VIRTUAL MACHINE	14
2.3. INSTALL OPENJDK ON RED HAT ENTERPRISE LINUX	14
2.4. DOWNLOAD AND INSTALL JBOSS DATA GRID	15
CHAPTER 3. INSTALL AND USE THE MAVEN REPOSITORIES	18
3.1. ABOUT MAVEN	18
3.2. REQUIRED MAVEN REPOSITORIES	18
3.3. INSTALL THE MAVEN REPOSITORY	18
3.4. CONFIGURE THE MAVEN REPOSITORY	20
PART III. SUPPORTED CONTAINERS FOR JBOSS DATA GRID	23
CHAPTER 4. USING JBOSS DATA GRID WITH SUPPORTED CONTAINERS	24
4.1. DEPLOY JBOSS DATA GRID IN JBOSS EAP (LIBRARY MODE)	24
4.2. DEPLOY JBOSS DATA GRID IN JBOSS EAP (REMOTE CLIENT-SERVER MODE)	26
4.3. DEPLOY JBOSS DATA GRID IN JBOSS ENTERPRISE WEB SERVER	28
4.4. DEPLOY WEB APPLICATIONS ON WEBLOGIC SERVER (LIBRARY MODE)	28
4.5. DEPLOY WEB APPLICATIONS ON WEBLOGIC SERVER (REMOTE CLIENT-SERVER MODE)	29
4.6. RUNNING RED HAT JBOSS DATA GRID IN KARAF (OSGI)	30
CHAPTER 5. RUNNING RED HAT JBOSS DATA GRID WITH APACHE CAMEL	33
5.1. THE CAMEL-JBOSSDATAGRID COMPONENT	33
5.2. ROUTING WITH CAMEL IN JBOSS DATA GRID	43
5.3. REMOTE QUERY	44
5.4. CUSTOM LISTENERS FOR EMBEDDED CACHE	45
5.5. CUSTOM LISTENERS FOR REMOTE CACHE	46
5.6. RED HAT JBOSS DATA GRID AND RED HAT JBOSS FUSE	48
5.7. RED HAT JBOSS DATA GRID AND RED HAT JBOSS EAP	51
PART IV. RUNNING RED HAT JBOSS DATA GRID	57
CHAPTER 6. RUN RED HAT JBOSS DATA GRID JAR FILES WITH MAVEN	58
6.1. RUN JBOSS DATA GRID (REMOTE CLIENT-SERVER MODE)	58
6.2. RUN JBOSS DATA GRID (LIBRARY MODE)	59
CHAPTER 7. RUN RED HAT JBOSS DATA GRID IN REMOTE CLIENT-SERVER MODE	61
7.1. PREREQUISITES	61
7.2. RUN RED HAT JBOSS DATA GRID IN STANDALONE MODE	61
7.3. RUN RED HAT JBOSS DATA GRID IN CLUSTERED MODE	61

7.4. RUN RED HAT JBOSS DATA GRID IN A MANAGED DOMAIN	61
7.5. RUN RED HAT JBOSS DATA GRID WITH A CUSTOM CONFIGURATION	62
7.6. SET AN IP ADDRESS TO RUN RED HAT JBOSS DATA GRID	62
7.7. RUNNING RED HAT JBOSS DATA GRID	63
CHAPTER 8. RUN A RED HAT JBOSS DATA GRID AS A NODE WITHOUT ENDPOINTS	64
8.1. BENEFITS OF A NODE WITHOUT ENDPOINTS	64
8.2. SAMPLE CONFIGURATION FOR A NODE WITHOUT ENDPOINTS	64
8.3. CONFIGURE A NODE WITH NO ENDPOINTS	64
CHAPTER 9. RUN RED HAT JBOSS DATA GRID IN LIBRARY MODE	66
9.1. CREATE A NEW RED HAT JBOSS DATA GRID PROJECT	66
9.2. ADD DEPENDENCIES TO YOUR PROJECT	66
9.3. ADD A PROFILE TO YOUR PROJECT	66
CHAPTER 10. RUN RED HAT JBOSS DATA GRID IN LIBRARY MODE (SINGLE-NODE SETUP)	69
10.1. RUN THE QUICKSTART CLASS	69
10.2. USE THE DEFAULT CACHE	69
CHAPTER 11. RUN RED HAT JBOSS DATA GRID IN LIBRARY MODE (MULTI-NODE SETUP)	74
11.1. SHARING JGROUP CHANNELS	74
11.2. RUN RED HAT JBOSS DATA GRID IN A CLUSTER	74
CHAPTER 12. MONITOR RED HAT JBOSS DATA GRID APPLICATIONS IN RED HAT JBOSS EAP .	80
12.1. PREREQUISITES	80
12.2. MONITOR RED HAT JBOSS DATA GRID APPLICATIONS IN RED HAT JBOSS EAP	80
PART V. SET UP A CACHE MANAGER	82
CHAPTER 13. CACHE MANAGERS	83
13.1. TYPES OF CACHE MANAGERS	83
13.2. CREATING CACHEMANAGERS	83
13.3. MULTIPLE CACHE MANAGERS	84
PART VI. RED HAT JBOSS DATA GRID QUICKSTARTS	86
CHAPTER 14. THE HELLO WORLD QUICKSTART	88
14.1. QUICKSTART PREREQUISITES	88
14.2. START TWO APPLICATION SERVER INSTANCES	88
14.3. BUILD AND DEPLOY THE HELLO WORLD QUICKSTART	89
14.4. ACCESS THE RUNNING APPLICATION	90
14.5. TEST REPLICATION ON THE APPLICATION	90
14.6. REMOVE THE APPLICATION	91
CHAPTER 15. THE CARMART QUICKSTARTS	93
15.1. ABOUT THE CARMART TRANSACTIONAL QUICKSTART	93
15.2. DIFFERENCES BETWEEN THE CARMART AND TRANSACTIONAL QUICKSTARTS	94
15.3. THE (NON-TRANSACTIONAL) CARMART QUICKSTART USING JBOSS EAP	94
15.4. THE (NON-TRANSACTIONAL) CARMART QUICKSTART USING JBOSS ENTERPRISE WEB SERVER	96
15.5. THE (NON-TRANSACTIONAL) CARMART QUICKSTART IN REMOTE CLIENT-SERVER MODE (JBOSS EAP)	97
15.6. THE (NON-TRANSACTIONAL) CARMART QUICKSTART IN REMOTE CLIENT-SERVER MODE (JBOSS ENTERPRISE WEB SERVER)	100
15.7. THE (TRANSACTIONAL) CARMART QUICKSTART USING JBOSS EAP	102
15.8. THE (TRANSACTIONAL) CARMART QUICKSTART USING JBOSS ENTERPRISE WEB SERVER	104

CHAPTER 16. THE FOOTBALL QUICKSTART ENDPOINT EXAMPLES	108
16.1. QUICKSTART PREREQUISITES	108
16.2. BUILD THE FOOTBALL APPLICATION	108
CHAPTER 17. THE RAPID STOCK MARKET QUICKSTART	113
17.1. BUILD AND RUN THE RAPID STOCK MARKET QUICKSTART	113
CHAPTER 18. THE CLUSTER APP QUICKSTART	115
18.1. EAP CLUSTER APP PREREQUISITES	115
18.2. START THE APPLICATION SERVER INSTANCES	115
18.3. BUILD THE APPLICATION	118
18.4. RUN THE APPLICATION	119
18.5. DEBUG THE APPLICATION	120
CHAPTER 19. THE CAMEL-JBOSSDATAGRID-FUSE QUICKSTART	121
19.1. QUICKSTART PREREQUISITES	121
19.2. SETUP	121
19.3. TESTING THE CAMEL-JBOSSDATAGRID-FUSE QUICKSTART	122
PART VII. UNINSTALL RED HAT JBOSS DATA GRID	124
CHAPTER 20. REMOVE RED HAT JBOSS DATA GRID	125
20.1. REMOVE RED HAT JBOSS DATA GRID FROM YOUR LINUX SYSTEM	125
20.2. REMOVE RED HAT JBOSS DATA GRID FROM YOUR WINDOWS SYSTEM	125
APPENDIX A. REFERENCES	127
A.1. ABOUT KEY-VALUE PAIRS	127
APPENDIX B. MAVEN CONFIGURATION INFORMATION	128
B.1. INSTALL THE JBOSS ENTERPRISE APPLICATION PLATFORM REPOSITORY USING NEXUS	128
B.2. MAVEN REPOSITORY CONFIGURATION EXAMPLE	129
B.3. DETERMINING THE URL OF THE JBOSS DATA GRID REPOSITORY	130
APPENDIX C. REVISION HISTORY	131

PART I. INTRODUCING RED HAT JBOSS DATA GRID

CHAPTER 1. RED HAT JBOSS DATA GRID

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

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

[Report a bug](#)

1.1. SUPPORTED CONFIGURATIONS

The set of supported features, configurations, and integrations for Red Hat JBoss Data Grid (current and past versions) are available at the Supported Configurations page at <https://access.redhat.com/articles/115883>.

[Report a bug](#)

1.2. COMPONENTS AND VERSIONS

Red Hat JBoss Data Grid includes many components for Library and Remote Client-Server modes. A comprehensive (and up to date) list of components included in each of these usage modes and their versions is available in the *Red Hat JBoss Data Grid Component Details* page at <https://access.redhat.com/articles/488833>

[Report a bug](#)

1.3. RED HAT JBOSS DATA GRID USAGE MODES

Red Hat JBoss Data Grid offers two usage modes:

- Remote Client-Server mode
- Library mode

[Report a bug](#)

1.3.1. Remote Client-Server Mode

Remote Client-Server mode provides a managed, distributed, and clusterable data grid server. In Client-Server mode the server runs as a self-contained process, utilizing a container based on JBoss EAP, allowing client applications to remotely access the data grid server using **Hot Rod**, **Memcached** or **REST** client APIs.

All Red Hat 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.

There are a number of benefits to running JBoss Data Grid in Remote Client-Server mode if Library mode features are not required. 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.

Run the following commands to start a standalone JBoss Data Grid instance in Remote Client-Server mode.

For Linux:

```
$JBOSS_HOME/bin/standalone.sh
```

For Windows:

```
$JBOSS_HOME\bin\standalone.bat
```

Run the following commands to start a managed domain JBoss Data Grid instance in Remote Client-Server mode.

For Linux:

```
$JBOSS_HOME/bin/domain.sh
```

For Windows:

```
$JBOSS_HOME\bin\domain.bat
```

[Report a bug](#)

1.3.2. Library Mode

Library mode allows building and deploying a custom runtime environment. The Library mode hosts a single data grid node in the applications process, with remote access to nodes hosted in other JVMs. Tested containers for Red Hat JBoss Data Grid 6 Library mode includes Red Hat JBoss Enterprise Web Server 2.x and JBoss Enterprise Application Platform 6.x.

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

The following features require Library mode:

- transactions
- listeners and notifications

JBoss Data Grid can also be run as a standalone application in Java SE. Standalone mode is a supported alternative to running JBoss Data Grid in a container.

[Report a bug](#)

1.4. RED HAT JBOSS DATA GRID BENEFITS

Red Hat 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 carries 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. Adding a node to JBoss Data Grid increases 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 to prevent running 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. RED HAT JBOSS DATA GRID VERSION INFORMATION

Red Hat 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 have 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.

The following table lists the correlation between JBoss Data Grid and Infinispan versions.

Table 1.1. JBoss Data Grid and Infinispan Correlation

JBoss Data Grid Product	Infinispan Version
JBoss Data Grid 6.0.0	Infinispan 5.1.5
JBoss Data Grid 6.0.1	Infinispan 5.1.7
JBoss Data Grid 6.1.0	Infinispan 5.2.4
JBoss Data Grid 6.2.0	Infinispan 6.0.1
JBoss Data Grid 6.3.0	Infinispan 6.1.0
JBoss Data Grid 6.3.1	Infinispan 6.1.1
JBoss Data Grid 6.4.0	Infinispan 6.2.0
JBoss Data Grid 6.4.1	Infinispan 6.2.1
JBoss Data Grid 6.5.0	Infinispan 6.3.0
JBoss Data Grid 6.5.1	Infinispan 6.3.1
JBoss Data Grid 6.6.0	Infinispan 6.4.0
JBoss Data Grid 7.0.0	Infinispan 8.3.0



NOTE

From Red Hat JBoss Data Grid 6.2.0 onwards, there is no direct corresponding version of Infinispan for each JBoss Data Grid release. The Infinispan version provided for JBoss Data Grid 6.3.0 and later are based on an internal version number (possibly an unreleased version) of Infinispan.

[Report a bug](#)

1.6. RED HAT JBOSS DATA GRID CACHE ARCHITECTURE

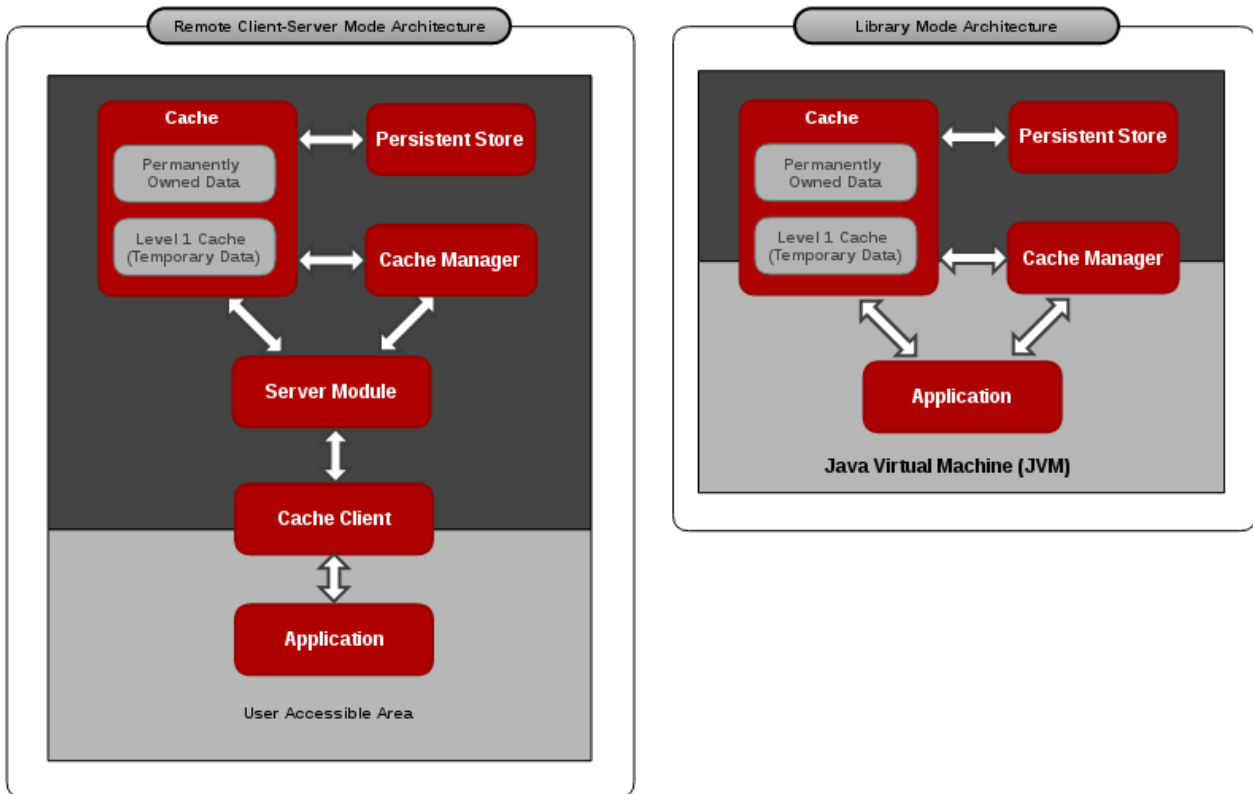


Figure 1.1. Red Hat JBoss Data Grid Cache Architecture

Red Hat JBoss Data Grid's cache infrastructure depicts the individual elements and their interaction with each other in each JBoss Data Grid Usage Mode (Library and Remote Client-Server). For clarity, each cache architecture diagram is separated into two parts:

- Elements that a user cannot directly interact with are depicted within a dark grey box in the diagram. In Remote Client-Server mode, this includes Persistent Store, Cache, Cache Manager, L1 Cache, and Server Module. In Library mode, user cannot directly interact with Persistent Store and L1 Cache.
- Elements that a user can interact directly with are depicted in a light grey box in the diagram. In Remote Client-Server mode, this includes the Application and the Cache Client. In Library mode, users are allowed to interact with the Cache and Cache Manager, as well as the Application.

Cache Architecture Elements

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

1. The Persistent Store is an optional component. It can permanently store the cached entries for restoration after a data grid shutdown.
2. 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.
3. The Cache Manager controls the life cycle of Cache instances and can store and retrieve them when required.
4. The Cache is the main component for storage and retrieval of the key-value entries.

Library and Remote Client-Server Mode Architecture

In Library mode, the Application (user code) can interact with the Cache and Cache Manager components directly. In this case the Application resides in the same Java Virtual Machine (JVM) and can call Cache and Cache Manager Java API methods directly.

In Remote Client-Server mode, the Application does not directly interact with the cache. Additionally, the Application usually resides in a different JVM, on different physical host, or does not need to be a Java Application. In this case, the Application uses a Cache Client that communicates with a remote JBoss Data Grid Server over the network using one of the supported protocols such as Memcached, Hot Rod, or REST. The appropriate server module handles the communication on the server side. When a request is sent to the server remotely, it translates the protocol back to the concrete operations performed on the cache component to store and retrieve data.

[Report a bug](#)

1.7. RED HAT JBOSS DATA GRID APIS

Red Hat JBoss Data Grid provides the following programmable APIs:

- Cache
- Batching
- Grouping
- Persistence (formerly CacheStore)
- 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](#)

PART II. DOWNLOAD AND INSTALL RED HAT JBOSS DATA GRID

CHAPTER 2. DOWNLOAD RED HAT JBOSS DATA GRID

2.1. RED HAT JBOSS DATA GRID INSTALLATION PREREQUISITES

The only prerequisite to set up Red Hat JBoss Data Grid is an installed Java Virtual Machine (compatible with Java 8.0 or later).

[Report a bug](#)

2.2. JAVA VIRTUAL MACHINE

A Java Virtual Machine (JVM) is a virtual environment that runs and executes Java programs on a host operating system. The JVM acts as an abstract computer and is a platform-independent execution environment that converts the Java programming code into machine language. A Java Virtual Machine (JVM) makes Java portable by providing an abstraction layer between the compiled Java program and the underlying hardware platform and operating system.

Red Hat recommends using OpenJDK Java platform as it is an open source, supported Java Virtual Machine that runs well on Red Hat Enterprise Linux systems. For Windows users, Oracle JDK 1.8 installation is recommended.

[Report a bug](#)

2.3. INSTALL OPENJDK ON RED HAT ENTERPRISE LINUX

Procedure 2.1. Install OpenJDK on Red Hat Enterprise Linux

1. **Subscribe to the Base Channel**

Obtain the OpenJDK from the RHN base channel. Your installation of Red Hat Enterprise Linux is subscribed to this channel by default.

2. **Install the Package**

Use the yum utility to install OpenJDK:

```
$ sudo yum install java-1.8.0-openjdk-devel
```

3. **Verify that OpenJDK is the System Default**

Ensure that the correct JDK is set as the system default as follows:

- a. Log in as a user with root privileges and run the alternatives command:

```
$ /usr/sbin/alternatives --config java
```

- b. Select `/usr/lib/jvm/java-1.8.0-openjdk-{java-version}.x86_64/bin/java`.

- c. Use the following command to set `javac`:

```
$ /usr/sbin/alternatives --config javac
```

- d. Select `/usr/lib/jvm/java-1.8.0-openjdk-{java-version}.x86_64/bin/javac`.

[Report a bug](#)

2.4. DOWNLOAD AND INSTALL JBOSS DATA GRID

Use the following steps to download and install Red Hat JBoss Data Grid:

1. Download JBoss Data Grid from the Red Hat Customer Portal.
2. Verify the downloaded files.
3. Install JBoss Data Grid.

[Report a bug](#)

2.4.1. Download Red Hat JBoss Data Grid

Follow the listed steps to download Red Hat JBoss Data Grid from the Customer Portal:

Procedure 2.2. Download JBoss Data Grid

1. Log into the Customer Portal at <https://access.redhat.com>.
2. Click the **Downloads** button near the top of the page.
3. In the **Product Downloads** page, click **Red Hat JBoss Data Grid**.
4. Select the appropriate JBoss Data Grid version from the **Version:** drop down menu.
5. Download the appropriate files from the list that is displayed.

[Report a bug](#)

2.4.2. About the Red Hat Customer Portal

The *Red Hat Customer Portal* is the centralized platform for Red Hat knowledge and subscription resources. Use the *Red Hat Customer Portal* to do the following:

- Manage and maintain Red Hat entitlements and support contracts.
- Download officially-supported software.
- Access product documentation and the Red Hat Knowledgebase.
- Contact Global Support Services.
- File bugs against Red Hat products.

The Customer Portal is available here: <https://access.redhat.com>.

[Report a bug](#)

2.4.3. Checksum Validation

Checksum validation is used to ensure a downloaded file has not been corrupted. Checksum validation employs algorithms that compute a fixed-size datum (or checksum) from an arbitrary block of digital data. If two parties compute a checksum of a particular file using the same algorithm, the results will be identical. Therefore, when computing the checksum of a downloaded file using the same algorithm as the supplier, if the checksums match, the integrity of the file is confirmed. If there is a discrepancy, the file has been corrupted in the download process.

[Report a bug](#)

2.4.4. Verify the Downloaded File

Procedure 2.3. Verify the Downloaded File

1. To verify that a file downloaded from the Red Hat Customer Portal is error-free, access the portal site and go to that package's **Software Details** page. The Software Details page displays the **MD5** and **SHA256** "checksum" values. Use the checksum values to check the integrity of the file.
2. Open a terminal window and run either the **md5sum** or **sha256sum** command, with the downloaded file as an argument. The program displays the checksum value for the file as the output for the command.
3. Compare the checksum value returned by the command to the corresponding value displayed on the **Software Details** page for the file.



NOTE

Microsoft Windows does not come equipped with a checksum tool. Windows operating system users have to download a third-party product instead.

Result

If the two checksum values are identical then the file has not been altered or corrupted and is, therefore, safe to use.

If the two checksum values are not identical, then download the file again. A difference between the checksum values means that the file has either been corrupted during download or has been modified since it was uploaded to the server. If, after several downloads, the checksum will still not successfully validate, contact Red Hat Support for assistance.

[Report a bug](#)

2.4.5. Install Red Hat JBoss Data Grid

Prerequisite

Locate the appropriate version, platform, and file type and download Red Hat JBoss Data Grid from the Customer Portal.

Procedure 2.4. Install JBoss Data Grid

1. Copy the downloaded JBoss Data Grid package to the preferred location on your machine.
2. Run the following command to extract the downloaded JBoss Data Grid package:

```
$ unzip JDG_PACKAGE
```

Replace *JDG_PACKAGE* with the name of the JBoss Data Grid usage mode package downloaded from the Red Hat Customer Portal.

3. The resulting unzipped directory will now be referred to as *\$JDG_HOME*.

[Report a bug](#)

2.4.6. Red Hat Documentation Site

Red Hat's official documentation site is available at <https://access.redhat.com/site/documentation/>. There you will find the latest version of every book, including this one.

[Report a bug](#)

CHAPTER 3. INSTALL AND USE THE MAVEN REPOSITORIES

3.1. ABOUT MAVEN

Apache Maven is a distributed build automation tool used in Java application development to create, manage, and build software projects. Maven uses standard configuration files called Project Object Model (POM) files to define projects and manage the build process. POM files describe the module and component dependencies, build order, and targets for the resulting project packaging and output using an XML file. This ensures that the project is built correctly and in a uniform manner.



IMPORTANT

Red Hat JBoss Data Grid requires Maven 3 (or better) for all quickstarts and general use.

Visit the Maven Download page (<http://maven.apache.org/download.html>) for instructions to download and install Maven.

[Report a bug](#)

3.2. REQUIRED MAVEN REPOSITORIES

Red Hat JBoss Data Grid Quickstarts require the following Maven repositories to be set up as a prerequisite:

- The JBoss Data Grid Maven Repository
- The **techpreview-all-repository** (<https://maven.repository.redhat.com/techpreview/all/>)

Both Maven repositories are installed in the same way. As a result, the subsequent instructions are for both repositories.

[Report a bug](#)

3.3. INSTALL THE MAVEN REPOSITORY

There are three ways to install the required repositories:

1. On your local file system ([Section 3.3.1, “Local File System Repository Installation”](#)).
2. On Apache Web Server ([Section 3.3.2, “Apache httpd Repository Installation”](#)).
3. With a Maven repository manager ([Section 3.3.3, “Maven Repository Manager Installation”](#)).

Use the option that best suits your environment.

[Report a bug](#)

3.3.1. Local File System Repository Installation

This option is best suited for initial testing in a small team. Follow the outlined procedure to extract the Red Hat JBoss Data Grid and JBoss Enterprise Application Platform Maven repositories to a directory in your local file system:

Procedure 3.1. Local File System Repository Installation (JBoss Data Grid)

1. Log Into the Customer Portal

In a browser window, navigate to the Customer Portal page (<https://access.redhat.com/home>) and log in.

2. Download the JBoss Data Grid Repository File

Download the `jboss-datagrid-{VERSION}-maven-repository.zip` file from the Red Hat Customer Portal.

3. Unzip the file to a directory on your local file system (for example `$JDG_HOME/projects/maven-repositories/`).

The above procedure will create a new `jboss-datagrid-{jdg-version}-maven-repository` directory, which contains the Maven repository in a subdirectory entitled `maven-repository/`.

[Report a bug](#)

3.3.2. Apache httpd Repository Installation

This example will cover the steps to download the JBoss Data Grid Maven Repository for use with Apache httpd. This option is good for multi-user and cross-team development environments because any developer that can access the web server can also access the Maven repository.



NOTE

Apache httpd must be configured for the following steps to work. For instructions on configuring Apache refer to the [Apache HTTP Server Project](#)

1. Open a web browser and access <https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?product=data.grid>
2. Find **Red Hat JBoss Data Grid 7.0.0 Maven Repository** in the list.
3. Click the **Download** button to download a .zip file containing the repository.
4. Unzip the files in a directory that is web accessible on the Apache server.
5. Configure Apache to allow read access and directory browsing in the created directory.

[Report a bug](#)

3.3.3. Maven Repository Manager Installation

This option is ideal if you are already using a repository manager.

The Red Hat JBoss Data Grid and JBoss Enterprise Application Server repositories are installed using a Maven repository manager using its documentation. Examples of such repository managers are:

- Apache Archiva: <http://archiva.apache.org/>
- JFrog Artifactory: <http://www.jfrog.com/products.php>
- Sonatype Nexus: <http://nexus.sonatype.org/> For details, see [Section B.1, “Install the JBoss Enterprise Application Platform Repository Using Nexus”](#).

[Report a bug](#)

3.4. CONFIGURE THE MAVEN REPOSITORY

To configure the installed Red Hat JBoss Data Grid Maven repository, edit the `settings.xml` file. This file may be configured in one of two locations:

1. User level - Maven user settings are located in the `${user.home}/.m2/` directory:
 - For Linux or Mac environments this is typically `~/.m2/`
 - For Windows environments this is typically `\Documents and Settings\.m2\` or `\Users\.m2\`
2. Global level - Settings for all users on a machine, assuming they are all using the same Maven installation, is typically provided in `${maven.home}/conf/settings.xml`

See [Section B.2, “Maven Repository Configuration Example”](#) to view sample Maven configurations, and refer to the [Maven Documentation](#) for more information about configuring Maven.

[Report a bug](#)

3.4.1. Configuring the JBoss Data Grid Maven Repository in an Offline Environment

In certain environments it is preferred to have the Maven repository available offline. To accomplish this configuration perform the following steps:

Prerequisites:

- The JBoss Data Grid Maven repository has been downloaded to the internal network where it will be referenced.
- An internal repository, such as Sonatype Nexus or Apache Archiva, is available on the network that contains Maven dependencies.

Procedure 3.2. Configure the JBoss Data Grid Maven Repository for Offline Usage

1. Install the downloaded JBoss Data Grid Maven repository locally, following the instructions in [Section 3.3.1, “Local File System Repository Installation”](#).

2. Update the `settings.xml` to point to the locally extracted repository, as seen in [Section 3.4, “Configure the Maven Repository”](#). A sample configuration is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
  http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <profiles>
    <profile>
      <id>jboss-datagrid-repository</id>
      <repositories>
        <repository>
          <id>jboss-datagrid-repository</id>
          <name>JBoss Data Grid Maven Repository</name>
          <url>file:///path/to/jboss-datagrid-7.0.0.GA-maven-
repository/maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-datagrid-repository</id>
          <name>JBoss Data Grid Maven Repository</name>
          <url>file:///path/to/jboss-datagrid-7.0.0.GA-maven-
repository/maven-repository</url>
          <layout>default</layout>
          <releases>
            <enabled>>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <!-- make the profile active by default -->
    <activeProfile>jboss-datagrid-repository</activeProfile>
  </activeProfiles>

</settings>
```

3. Ensure that projects may now be built locally.

[Report a bug](#)

3.4.2. Next Steps

After the newest available version of Red Hat JBoss Data Grid is installed and Maven is set up and configured, see [Section 9.1, “Create a New Red Hat JBoss Data Grid Project”](#) to learn how to use JBoss Data Grid for the first time.

[Report a bug](#)

PART III. SUPPORTED CONTAINERS FOR JBOSS DATA GRID

CHAPTER 4. USING JBOSS DATA GRID WITH SUPPORTED CONTAINERS

Red Hat JBoss Data Grid can be used in the following runtimes:

- Java SE, started by your application.
- As a standalone JBoss Data Grid server.
- Bundled as a library in your application, deployed to an application server, and started by your application. For example, JBoss Data Grid can be used with Tomcat or Weblogic.
- Inside an OSGi runtime environment, in this case, Apache Karaf.

For a list of containers supported with Red Hat JBoss Data Grid, see the *Release Notes* or the support information here: <https://access.redhat.com/knowledge/articles/115883>

[Report a bug](#)

4.1. DEPLOY JBOSS DATA GRID IN JBOSS EAP (LIBRARY MODE)

Red Hat JBoss Data Grid provides a set of modules for Red Hat JBoss Enterprise Application Platform 6.x. Using these modules means that JBoss Data Grid libraries do not need to be included in the user deployment. To avoid conflicts with the Infinispan modules that are already included with JBoss EAP, the JBoss Data Grid modules are placed within a separate slot and identified by the JBoss Data Grid version (*major.minor*).



NOTE

The JBoss EAP modules are not included in JBoss EAP. Instead, navigate to the Customer Support Portal at <http://access.redhat.com> to download these modules from the Red Hat JBoss Data Grid downloads page.

To deploy JBoss Data Grid in JBoss EAP, add dependencies from the JBoss Data Grid module to the application's classpath (the JBoss EAP deployer) in one of the following ways:

- Add a dependency to the **jboss-deployment-structure.xml** file.
- Add a dependency to the **MANIFEST.MF** file.
- Generate the **MANIFEST.MF** file via Maven.

Add a Dependency to the **jboss-deployment-structure.xml** File

Add the following configuration to the **jboss-deployment-structure.xml** file:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <deployment>
    <dependencies>
      <module name="org.infinispan" slot="jdg-7.0"
services="export"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

```

    </dependencies>
  </deployment>
</jboss-deployment-structure>

```



NOTE

For details about the `jboss-deployment-structure.xml` file, see the Red Hat JBoss Enterprise Application Platform documentation.

Add a Dependency to the MANIFEST.MF File.

Add a dependency to the `MANIFEST.MF` files as follows:

Example 4.1. Example MANIFEST.MF File

```

Manifest-Version: 1.0
Dependencies: org.infinispan:jdg-7.0 services

```

The first line remains the same as the example. Depending on the dependency required, add one of the following to the second line of the file:

- JBoss Data Grid Core:

```
Dependencies: org.infinispan:jdg-7.0 services
```

- Embedded Query:

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.query:jdg-7.0 services, org.infinispan.query.dsl:jdg-
7.0 services
```

- JDBC Cache Store:

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.persistence.jdbc:jdg-7.0 services
```

- JPA Cache Store:

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.persistence.jpa:jdg-7.0 services
```

- LevelDB Cache Store:

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.persistence.leveldb:jdg-7.0 services
```

- CDI:

```
Dependencies: org.infinispan:jdg-7.0 services,
org.infinispan.cdi:jdg-7.0 meta-inf
```

Generate the MANIFEST.MF file via Maven

The **MANIFEST.MF** file is generated during the build (specifically during the JAR or WAR process). As an alternative to adding a dependency to the **MANIFEST.MF** file, configure the dependency directly in Maven by adding the following to the **pom.xml** file:

```
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <version>2.4</version>
  <configuration>
    <failOnMissingWebXml>>false</failOnMissingWebXml>
    <archive>
      <manifestEntries>
        <Dependencies>org.infinispan:jdg-7.0 services</Dependencies>
      </manifestEntries>
    </archive>
  </configuration>
</plugin>
```

[Report a bug](#)

4.2. DEPLOY JBOSS DATA GRID IN JBOSS EAP (REMOTE CLIENT-SERVER MODE)

Red Hat JBoss Data Grid provides a set of modules for Red Hat JBoss Enterprise Application Platform 6.x. Using these modules means that JBoss Data Grid libraries do not need to be included in the user deployment. To avoid conflicts with the Infinispan modules that are already included with JBoss EAP, the JBoss Data Grid modules are placed within a separate slot and identified by the JBoss Data Grid version (*major.minor*).



NOTE

The JBoss EAP modules are not included in JBoss EAP. Instead, navigate to the Customer Support Portal at <http://access.redhat.com> to download these modules from the Red Hat JBoss Data Grid downloads page.

To deploy JBoss Data grid in JBoss EAP, add dependencies from the JBoss Data Grid module to the application's classpath (the JBoss EAP deployer) in one of the following ways:

- Add a dependency to the **jboss-deployment-structure.xml** file.
- Add a dependency to the **MANIFEST.MF** file.

Add a Dependency to the jboss-deployment-structure.xml File

Add the following configuration to the **jboss-deployment-structure.xml** file:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <deployment>
    <dependencies>
      <module name="org.infinispan.commons" slot="jdg-7.0"
services="export"/>
      <module name="org.infinispan.client.hotrod" slot="jdg-7.0"
services="export"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

```

    </dependencies>
  </deployment>
</jboss-deployment-structure>

```



NOTE

For details about the `jboss-deployment-structure.xml` file, see the Red Hat JBoss Enterprise Application Platform documentation.

Add a Dependency to the MANIFEST.MF File.

Add a dependency to the `MANIFEST.MF` files as follows:

Example 4.2. Example MANIFEST.MF File

```

Manifest-Version: 1.0
Dependencies: org.infinispan.commons:jdg-7.0 services,
org.infinispan.client.hotrod:jdg-7.0 services

```

The first line remains the same as the example. Depending on the dependency required, add one of the following to the second line of the file:

- Basic Hot Rod client:

```

org.infinispan.commons:jdg-7.0 services,
org.infinispan.client.hotrod:jdg-7.0 services

```

- Hot Rod client with Remote Query functionality:

```

org.infinispan.commons:jdg-7.0 services,
org.infinispan.client.hotrod:jdg-7.0 services,
org.infinispan.query.dsl:jdg-7.0 services, org.jboss.remoting3

```

[Report a bug](#)

4.2.1. Using Custom Classes with the Hot Rod client

Either of the following two methods may be used to use custom classes with the Hot Rod client:

- **Option 1:** Reference the deployment's class loader in the configuration builder for the Hot Rod client, as seen in the below example:

Example 4.3. Referencing the custom class loader in the ConfigurationBuilder instance

```

import
org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
[...]
ConfigurationBuilder config = new ConfigurationBuilder();
config.marshaller(new
GenericJBossMarshaller(Thread.currentThread().getContextClassLoader()));

```

- **Option 2:** Install the custom classes as their own module within JBoss EAP, and add a dependency on the newly created module should be added to the JBoss Data Grid module at `${EAP_HOME}/modules/system/layers/base/org/infinispan/commons/jdg-6.x/module.xml`.

[Report a bug](#)

4.3. DEPLOY JBOSS DATA GRID IN JBOSS ENTERPRISE WEB SERVER

Red Hat JBoss Data Grid supports JBoss Enterprise Web Server in Library and Remote Client Server mode. To use JBoss Data Grid with JBoss Enterprise Web Server, bundle the JDG libraries in a web application and deploy the application on the server.

For further information on how to deploy JBoss Data Grid on JBoss Enterprise Web Server, see ([Section 15.4, “The \(Non-transactional\) CarMart Quickstart Using JBoss Enterprise Web Server”](#) and [Section 15.6, “The \(Non-Transactional\) CarMart Quickstart in Remote Client-Server Mode \(JBoss Enterprise Web Server\)”](#)).

[Report a bug](#)

4.4. DEPLOY WEB APPLICATIONS ON WEBLOGIC SERVER (LIBRARY MODE)

Red Hat JBoss Data Grid supports the WebLogic 12c application server in Library mode. The following procedure describes how to deploy web applications on a WebLogic server.

Prerequisites

The prerequisites to deploy the web applications are as follows:

1. WebLogic Server 12c.
2. JBoss Data Grid Library (Embedded) mode.

Procedure 4.1. Deploying Web Applications on a WebLogic Server

1. Create Web Applications

Create a web application and add the libraries in the **WEB-INF** folder.

2. Create a `weblogic.xml` Deployment Descriptor

Create a `weblogic.xml` deployment descriptor with the following elements in it:

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app
  xmlns="http://www.bea.com/ns/weblogic/90"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
http://www.bea.com/ns/weblogic/90/weblogic-web-app.xsd">
  <container-descriptor>
```



```
<prefer-web-inf-classes>true</prefer-web-inf-classes>
</container-descriptor>
</weblogic-web-app>
```

**NOTE**

The **prefer-web-inf-classes** class indicates that the libraries and the classes in the **WEB-INF/lib** folder is preferred over the default libraries bundled in the WebLogic server. For example, the **commons-pool.jar** file in the WebLogic server has version 1.4 and is automatically loaded by the classloader, but the Hot Rod client uses a newer version of this library.

3. Pack the Web Application into a Web Archive File

Create a web application archive (WAR) file of the web application and verify that the JBoss Data Grid libraries are in the **WEB-INF** folder along with the WebLogic deployment descriptor file.

4. Deploy the Application onto WebLogic Server

To deploy the web application using the Infinispan CDI module, stop the WebLogic server if it is running, apply the patch on it (Patch file **p17424706_121200_Generic.zip**) and restart the WebLogic server. If the Infinispan CDI module is not being used, deploy the web application normally.

For more information about applying patch to the WebLogic Server, see the *Oracle patch database* on the Oracle website.

[Report a bug](#)

4.5. DEPLOY WEB APPLICATIONS ON WEBLOGIC SERVER (REMOTE CLIENT-SERVER MODE)

Red Hat JBoss Data Grid supports the WebLogic 12c application server in Remote Client-Server mode. The following procedure describes how to deploy web applications on a WebLogic server.

Procedure 4.2. Deploying Web Applications on a WebLogic Server

1. To install the WebLogic server, see http://docs.oracle.com/cd/E24329_01/doc.1211/e24492/toc.htm.
2. Configure JBoss Data Grid in Remote Client-Server mode, define cache, cache container, and endpoint configuration. After configuration, start JBoss Data Grid to confirm that the Hot Rod endpoint is listening on the configured port. For information about configuring JBoss Data Grid in Remote Client-Server, see [Chapter 7, Run Red Hat JBoss Data Grid in Remote Client-Server Mode](#)
3. Create a web application and add the **infinispan-remote** library as a dependency if Maven is used.
4. Create a **weblogic.xml** deployment descriptor with the following elements in it:

```
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-web-app
```

```

xmlns="http://www.bea.com/ns/weblogic/90"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
http://www.bea.com/ns/weblogic/90/weblogic-web-app.xsd">
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
</weblogic-web-app>

```



NOTE

The **prefer-web-inf-classes** class indicates that the libraries and classes in the **WEB-INF/Lib** folder are preferred over the default libraries bundled in the WebLogic server. For example, the **commons-pool.jar** file in the WebLogic server has version 1.4 and is automatically loaded by the classloader, however the Hot Rod client uses a newer version of this library.

5. Add deployment descriptor file in the **WEB-INF** folder.
6. Ensure that the **infinispan-remote** dependency is added to the **pom.xml** file, then use a Maven plugin to create a web archive.

Alternatively, create the web archive manually and add the library manually.

7. Deploy the application in the WebLogic server and verify that the Hot Rod client embedded inside the web application connects to the remote JBoss Data Grid server.

[Report a bug](#)

4.6. RUNNING RED HAT JBOSS DATA GRID IN KARAF (OSGI)

Apache Karaf is a powerful, lightweight OSGi-based runtime container into which components and applications are deployed. OSGi implements a dynamic component model that does not exist in standalone JVM environments. OSGi containers such as Karaf include a rich set of tools for managing the life cycle of an application.

All dependencies between individual modules, including version numbers, must be explicitly specified. Where more than one class of the same name exists, the strict rules of OSGi specify which of the classes will be used by your bundle.

[Report a bug](#)

4.6.1. Running a Deployment of JBoss Data Grid in Karaf (Remote Client-Server)

The Red Hat JBoss Data Grid Hot Rod client can be run in an OSGi-based container such as Karaf, allowing client applications deployed in Karaf to connect to pre-existing JBoss Data Grid servers.

Use the path in the JBoss Data Grid Maven repository to set up Karaf. Additionally, JBoss Data Grid requires a **features** file, located in **org/infinispan/infinispan-remote/\${VERSION}**. This file lists all dependencies for the Hot Rod client in OSGi, while

also making it simpler to install the feature into Karaf (version 2.3.3 or 3.0).

[Report a bug](#)

4.6.2. Installing the Hot Rod client feature in Karaf

Red Hat JBoss Data Grid's Hot Rod feature is installed in Karaf as follows:

Prerequisite

Configure the Red Hat JBoss Data Grid Maven Repository.

Procedure 4.3. Install the Hot Rod Feature in Karaf

1. Karaf 2.3.3

For Karaf 2.3.3 use the following commands:

- a.

```
karaf@root> features:addUrl mvn:org.infinispan/infinispan-remote/${VERSION}/xml/features
```
- b.

```
karaf@root> features:install infinispan-remote
```
- c. Verify that the feature was successfully installed as follows:

```
karaf@root> features:list
//output
```

2. Karaf 3.0.0

For Karaf use the following commands.

- a.

```
karaf@root> feature:repo-add mvn:org.infinispan/infinispan-remote/${VERSION}/xml/features
```
- b.

```
karaf@root> feature:install infinispan-remote
```
- c. Verify that the feature was successfully installed:

```
karaf@root> feature:list
```

Alternatively, use the **-i** command parameter to install the Hot Rod Client feature using the following:

```
karaf@root(>) feature:repo-add -i mvn:org.infinispan/infinispan-remote/${VERSION}/xml/features
```

[Report a bug](#)

4.6.3. Installing Red Hat JBoss Data Grid in Karaf (Library Mode)

The Red Hat JBoss Data Grid JAR files contain the required OSGi manifest headers and are used inside OSGi runtime environments as OSGi bundles. Additionally, the required third-party dependencies must be installed. These can be installed individually, or altogether via

the **features** file, which defines all required dependencies.

To install bundles using the **features** file:

- Register the feature repositories inside Karaf.
- Install the features contained in the repositories.

Procedure 4.4. Installing bundles using the features file

1. Start the Karaf console

Start the Karaf console using the following commands:

```
$ cd $APACHE_KARAF_HOME/bin
$ ./karaf
```

2. Register a feature repository

Register a feature repository as follows:

- For Karaf 2.3.3:

```
karaf@root(> features:addUrl mvn:org.infinispan/infinispan-embedded/${VERSION}/xml/features
```

```
karaf@root> features:install infinispan-embedded
```

- For Karaf 3.0.0:

```
karaf@root(> feature:repo-add mvn:org.infinispan/infinispan-embedded/${VERSION}/xml/features
```

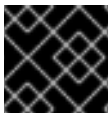
```
karaf@root> feature:install infinispan-embedded
```

Result

JBoss Data Grid runs in library mode using Karaf.

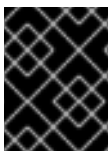
The URL for feature repositories is constructed from the Maven artifact coordinates using the following format:

```
mvn:<groupId>/<artifactId>/<version>/xml/features
```



IMPORTANT

The JPA Cache Store is not supported in Apache Karaf in JBoss Data Grid 7.0.



IMPORTANT

Querying in Library mode (which is covered in the JBoss Data Grid *Developer Guide*) is not supported in Apache Karaf in JBoss Data Grid 7.0.

[Report a bug](#)

CHAPTER 5. RUNNING RED HAT JBOSS DATA GRID WITH APACHE CAMEL

Apache Camel is an open source integration and routing system that allows transference of messages from various sources to different destinations, providing an integration framework that allows interaction with various systems using the same API, regardless of the protocol or data type. Using Camel with Red Hat JBoss Data Grid and Red Hat JBoss Fuse simplifies integration in large enterprise applications by providing a wide variety of transports and APIs that add connectivity.

JBoss Data Grid provides support for caching on Camel routes in JBoss Fuse, partially replacing Ehcache. JBoss Data Grid is supported as an embedded cache (local or clustered) or as a remote cache in a Camel route.

[Report a bug](#)

5.1. THE CAMEL-JBOSSDATAGRID COMPONENT

Red Hat JBoss Data Grid's `camel-jbossgdatagrid` component includes the following features:

- **Local Camel Consumer**

Receives cache change notifications and sends them to be processed. This can be done synchronously or asynchronously, and is also supported with a replicated or distributed cache.

- **Local Camel Producer**

A producer creates and sends messages to an endpoint. The `camel-jbossgdatagrid` producer uses **GET**, **PUT**, **REMOVE**, and **CLEAR** operations. The local producer is also supported with a replicated or distributed cache.

- **Remote Camel Producer**

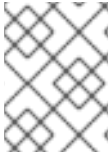
In Remote Client-Server mode, the Camel producer can send messages using Hot Rod.

- **Remote Camel Consumer**

In Client-Server mode, receives cache change notifications and sends them to be processed. The events are processed asynchronously.

The following `camel-jbossgdatagrid` dependency must be added to the `pom.xml` file to run JBoss Data Grid with Camel:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jbossgdatagrid</artifactId>
  <version>7.0.0.ER4-redhat-1</version>
  <!-- use the same version as your JBoss Data Grid version -->
</dependency>
```

**NOTE**

The **camel-jbossdatagrid** component ships with JBoss Data Grid, and is not included in the JBoss Fuse 6.1 or JBoss Fuse Service Works 6.0 distributions.

Camel components are the main extension point in Camel, and are associated with the name used in a URI, and act as a factory of endpoints. For example, a **FileComponent** is referred to in a URI as **file**, which creates **FileEndpoints**.

URI Format

The following URI format is used for **camel-jbossdatagrid**:

```
infinispan://hostname?[options]
```

URI Options

The producer can create and send messages to a local or remote JBoss Data Grid cache configured in the registry. If a **cacheContainer** is present, the cache will be either local or remote, depending on whether the **cacheContainer** instance is a **DefaultCacheManager** or **RemoteCacheManager**. If it is not present, the cache will try to connect to remote cache using the supplied hostname/port.

A consumer listens for events from the local JBoss Data Grid cache accessible from the registry.

Table 5.1. URI Options

Name	Default Value	Type	Context	Description
cacheContainer	null	CacheContainer	Shared	Reference to a org.infinispan.manager.CacheContainer in the Registry.
cacheName	null	String	Shared	The cache name to use. If not specified, the default cache is used.
command	PUT	String	Producer	The operation to perform. Only the PUT, GET, REMOVE, and CLEAR values are currently supported.

Name	Default Value	Type	Context	Description
eventTypes	null	Set<String>	Consumer	<p>A comma separated list of the event types to register. By default, this listens for all event types. Possible values are defined in org.infinisp an.notifications.cachelistener.event.Event.Type.</p> <p>Example:</p> <pre>...? eventTypes =CACHE_ENT RY_EXPIRED ,CACHE_ENT RY_EVICTED ,...</pre>
sync	true	Boolean	Consumer	<p>By default the consumer will receive notifications synchronously by the same thread that process the cache operation. Remote HotRod listeners support only asynchronous notification.</p>

Name	Default Value	Type	Context	Description
clustered	false	Boolean	Consumer	By default the consumer will only receive local events. By using this option, the consumer also listens to events originated on other nodes in the cluster. The only events available for clustered listeners are CACHE_ENTRY_CREATED , CACHE_ENTRY_REMOVED , and CACHE_ENTRY_MODIFIED .

Camel Operations

A list of all available operations, along with their header information, is found below:

Table 5.2. Put Operations

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationPut	Embedded / Remote	Puts a key/value pair in the cache, optionally with expiration	CamelInfinispanKey , CamelInfinispanValue	CamelInfinispanLifeSpanTime , CamelInfinispanLifeSpanTimeUnit , CamelInfinispanMaxIdleTime , CamelInfinispanMaxIdleTimeUnit , CamelInfinispanIgnoreReturnValues	CamelInfinispanOperationResult
CamelInfinispanOperationPutAsync		Asynchronously puts a key/value pair in the cache, optionally with expiration			

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationPutIfAbsent		Puts a key/value pair in the cache if it did not exist, optionally with expiration			
CamelInfinispanOperationPutIfAbsentAsync		Asynchronously puts a key/value pair in the cache if it did not exist, optionally with expiration			

Table 5.3. Put All Operations

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationPutAll	Embedded / Remote	Adds multiple entries to a cache, optionally with expiration	CamelInfinispanMap	CamelInfinispanLifespanTime, CamelInfinispanLifespanTimeUnit, CamelInfinispanMaxIdleTime, CamelInfinispanMaxIdleTimeUnit	
CamelInfinispanOperationPutAllAsync		Asynchronously adds multiple entries to a cache, optionally with expiration			

Table 5.4. Get Operation

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationGet	Embedded / Remote	Retrieves the value associated with a specific key from the cache	CamelInfinispanKey		

Table 5.5. Contains Key Operation

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationContainsKey	Embedded / Remote	Determines whether a cache contains a specific key	CamelInfinispanKey		CamelInfinispanOperationResult

Table 5.6. Contains Value Operation

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationContainsValue	Embedded / Remote	Determines whether a cache contains a specific value	CamelInfinispanKey		

Table 5.7. Remove Operations

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationRemove	Embedded / Remote	Removes an entry from a cache, optionally only if the value matches a given one	CamelInfinispanKey	CamelInfinispanValue	CamelInfinispanOperationResult

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationRemoveAsync		Asynchronously removes an entry from a cache, optionally only if the value matches a given one			

Table 5.8. Replace Operations

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationReplace	Embedded / Remote	Conditionally replaces an entry in the cache, optionally with expiration	CamelInfinispanKey , CamelInfinispanValue , CamelInfinispanOldValue	CamelInfinispanLifespanTime , CamelInfinispanLifespanTimeUnit , CamelInfinispanMaxIdleTime , CamelInfinispanMaxIdleTimeUnit , CamelInfinispanIgnoreReturnValues	CamelInfinispanOperationResult
CamelInfinispanOperationReplaceAsync		Asynchronously conditionally replaces an entry in the cache, optionally with expiration			

Table 5.9. Clear Operation

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationClear	Embedded / Remote	Clears the cache			

Table 5.10. Size Operation

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationSize	Embedded / Remote	Returns the number of entries in the cache			CamelInfinispanOperationResult

Table 5.11. Query Operation

Operation Name	Context	Description	Required Headers	Optional Headers	Result Header
CamelInfinispanOperationQuery	Remote	Executes a query on the cache	CamelInfinispanQueryBuilder		CamelInfinispanOperationResult

**NOTE**

Any operations that take **CamelInfinispanIgnoreReturnValues** will receive a null result.

Table 5.12. Message Headers

Name	Default Value	Type	Context	Description
CamelInfinispanCacheName	null	String	Shared	The cache participating in the operation or event.
CamelInfinispanMap	null	Map	Producer	A Map to use in case of the CamelInfinispanOperationPutAll operation.
CamelInfinispanKey	null	Object	Shared	The key to perform the operation to or the key generating the event.
CamelInfinispanValue	null	Object	Producer	The value to use for the operation.

Name	Default Value	Type	Context	Description
CamelInfinispan OperationResult	null	Object	Producer	The result of the operation.
CamelInfinispan EventType	null	String	Consumer	<p>For local cache listeners (non-clustered), one of the following values:</p> <p>CACHE_ENTRY_ACTIVATED, CACHE_ENTRY_PASSIVATED, CACHE_ENTRY_VISITED, CACHE_ENTRY_LOADED, CACHE_ENTRY_EVICTED, CACHE_ENTRY_CREATED, CACHE_ENTRY_REMOVED, CACHE_ENTRY_MODIFIED</p> <p>For remote HotRod listeners, one of the following values:</p> <p>CLIENT_CACHE_ENTRY_CREATED, CLIENT_CACHE_ENTRY_MODIFIED, CLIENT_CACHE_ENTRY_REMOVED, CLIENT_CACHE_FAILOVER.</p>

Name	Default Value	Type	Context	Description
CamellInfinispanIsPre	null	Boolean	Consumer	Infinispan fires two events for each operation when local non-clustered listener is used: one before and one after the operation. For clustered listeners and remote HotRod listeners, Infinispan fires only one event after the operation.
CamellInfinispanQueryBuilder	null	InfinispanQueryBuilder	Producer	An instance of InfinispanQueryBuilder that, in its build() , defines the query to be executed on the cache.
CamellInfinispanLifespanTime	null	long	Producer	The Lifespan time of a value inside the cache. Negative values are interpreted as infinity.
CamellInfinispanTimeUnit	null	String	Producer	The Time Unit of an entry Lifespan Time.
CamellInfinispanMaxIdleTime	null	long	Producer	The maximum amount of time an entry is allowed to be idle for before it is considered as expired.
CamellInfinispanMaxIdleTimeUnit	null	String	Producer	The Time Unit of an entry Max Idle Time.

[Report a bug](#)

5.2. ROUTING WITH CAMEL IN JBOSS DATA GRID

Camel routing is a chain of processors that move messages in the background. The following is an example of a route that retrieves a value from the cache for a specific key.

```
from("direct:start")
    .setHeader(InfinispanConstants.OPERATION,
constant(InfinispanConstants.GET))
    .setHeader(InfinispanConstants.KEY, constant("123"))
    .to("infinispan://localhost?cacheContainer=#cacheContainer");
```

Routing can also be performed using XML configuration. The following example demonstrates camel-jbossgdatagrid's **local-camel-producer**, a camel route that uses the **camel-jbossgdatagrid** component to send data to an embedded cache created by the **local-cache** module.

```
<camelContext id="local-producer"
xmlns="http://camel.apache.org/schema/blueprint">
    <route>
        <from uri="timer://local?fixedRate=true&period=5000"/>
            <setHeader headerName="CamelInfinispanKey">
                <constant>CamelTimerCounter</constant>
            </setHeader>
            <setHeader headerName="CamelInfinispanValue">
                <constant>CamelTimerCounter</constant>
            </setHeader>
            <to uri="infinispan://foo?cacheContainer=#cacheManager"/>
            <to uri="log:local-put?showAll=true"/>
        </route>
    </camelContext>
```

The provided example requires the **cacheManager** to be instantiated.

The **cacheManager** bean for Spring XML can be instantiated as follows:

```
<bean id="cacheManager" class="org.infinispan.manager.DefaultCacheManager"
init-method="start" destroy-method="stop">
    <constructor-arg type="java.lang.String" value="infinispan.xml"/>
</bean>
```

The following demonstrates how to instantiate the **cacheManager** bean using Blueprint XML.

```
<bean id="cacheManager" class="org.infinispan.manager.DefaultCacheManager"
init-method="start" destroy-method="stop">
    <argument value="infinispan.xml" />
</bean>
```

**NOTE**

Both the Spring XML and Blueprint XML examples use the configuration file **infinispan.xml** for configuration of the cache. This file must be present on the classpath.

[Report a bug](#)

5.3. REMOTE QUERY

When executing remote queries the `cacheManager` must be an instance of **RemoteCacheManager**, and an example configuration utilizing a **RemoteCacheManager** is found below for both Java and `blueprint.xml`:

Using only Java

```
from("direct:start")
    .setHeader(InfinispanConstants.OPERATION, InfinispanConstants.QUERY)
    .setHeader(InfinispanConstants.QUERY_BUILDER,
        new InfinispanQueryBuilder() {
            public Query build(QueryFactory<Query> queryFactory) {
                return
queryFactory.from(User.class).having("name").like("%abc%")
                    .toBuilder().build();
            }
        })
    .to("infinispan://localhost?
cacheContainer=#cacheManager&cacheName=remote_query_cache") ;
```

Using Blueprint and Java

Java **RemoteCacheManagerFactory** class:

```
public class RemoteCacheManagerFactory {
    ConfigurationBuilder clientBuilder;
    public RemoteCacheManagerFactory(String hostname, int port) {
        clientBuilder = new ConfigurationBuilder();
        clientBuilder.addServer()
            .host(hostname).port(port);
    }
    public RemoteCacheManager newRemoteCacheManager() {
        return new RemoteCacheManager(clientBuilder.build());
    }
}
```

Java **InfinispanQueryExample** class:

```
public class InfinispanQueryExample {
    public InfinispanQueryBuilder getBuilder() {
        return new InfinispanQueryBuilder() {
            public Query build(QueryFactory<Query> queryFactory) {
                return queryFactory.from(User.class)
                    .having("name")
                    .like("%abc%")
                    .toBuilder().build();
            }
        };
    }
}
```



```

    }
  }
}

```

blueprint.xml:

```

<bean id="remoteCacheManagerFactory"
class="com.jboss.datagrid.RemoteCacheManagerFactory">
  <argument value="localhost"/>
  <argument value="11222"/>
</bean>

<bean id="cacheManager"
  factory-ref="remoteCacheManagerFactory"
  factory-method="newRemoteCacheManager">
</bean>

<bean id="queryBuilder" class="org.example.com.InfinispanQueryExample"/>

<camelContext id="route" xmlns="http://camel.apache.org/schema/blueprint">
  <route>
    <from uri="direct:start"/>
    <setHeader headerName="CamelInfinispanOperation">
      <constant>CamelInfinispanOperationQuery</constant>
    </setHeader>
    <setHeader headerName="CamelInfinispanQueryBuilder">
      <method ref="queryBuilder" method="getBuilder"/>
    </setHeader>
    <to uri="infinispan://localhost?
cacheContainer=#cacheManager&cacheName=remote_query_cache"/>
  </route>
</camelContext>

```

The **remote_query_cache** is an arbitrary name for a cache that holds the data, and the results of the query will be a list of domain objects stored as a **CamelInfinispanOperationResult** header.

In addition, there are the following requirements:

- The **RemoteCacheManager** must be configured to use **ProtoStreamMarshaller**.
- The **ProtoStreamMarshaller** must be registered with the **RemoteCacheManager**'s serialization context.
- The .proto descriptors for domain objects must be registered with the remote JBoss Data Grid server.

For more details on how to setup a **RemoteCacheManager**, see the **Remote Querying** section of the *Red Hat JBoss Data Grid Infinispan Query Guide*

[Report a bug](#)

5.4. CUSTOM LISTENERS FOR EMBEDDED CACHE

Custom Listeners for an embedded cache can be registered through the *customListener* parameter as shown below:

Using Java

```
from("infinispan://?
cacheContainer=#myCustomContainer&cacheName=customCacheName&customListener
=#myCustomListener")
    .to("mock:result");
```

Using Blueprint

```
<bean id="myCustomContainer" org.infinispan.manager.DefaultCacheManager"
    init-method="start" destroy-method="stop">
    <argument value="infinispan.xml" />
</bean>

<bean id="myCustomListener" class="org.example.com.CustomListener"/>

<camelContext id="route" xmlns="http://camel.apache.org/schema/blueprint">
    <route>
        <from uri="infinispan://?
cacheContainer=#myCustomContainer&cacheName=customCacheName&customListener
=#myCustomListener"/>
        <to uri="mock:result"/>
    </route>
</camelContext>
```

The instance of **myCustomListener** must exist. Users are encouraged to extend the **org.apache.camel.component.infinispan.embedded.InfinispanEmbeddedCustomListener** and annotate the resulting class with the **@Listener** annotation from **org.infinispan.notifications**.



NOTE

Custom filters and converters for embedded caches are currently not supported.

[Report a bug](#)

5.5. CUSTOM LISTENERS FOR REMOTE CACHE

Custom listeners for a remote cache can be registered in the same way as an embedded cache, with the exception that **sync=false** must be present. For instance:

Using only Java

```
from(infinispan://?
cacheContainer=#cacheManager&sync=false&customListener=#myCustomListener")
    .to(mock:result);
```

Using Blueprint and Java

Java class:

```

public class RemoteCacheManagerFactory {
    ConfigurationBuilder clientBuilder;
    public RemoteCacheManagerFactory(String hostname, int port) {
        clientBuilder = new ConfigurationBuilder();
        clientBuilder.addServer()
            .host(hostname).port(port);
    }
    public RemoteCacheManager newRemoteCacheManager() {
        return new RemoteCacheManager(clientBuilder.build());
    }
}

```

blueprint.xml:

```

<bean id="remoteCacheManagerFactory"
class="com.jboss.datagrid.RemoteCacheManagerFactory">
    <argument value="localhost"/>
    <argument value="11222"/>
</bean>

<bean id="cacheManager"
    factory-ref="remoteCacheManagerFactory"
    factory-method="newRemoteCacheManager">
</bean>

<bean id="myCustomListener" class="org.example.com.CustomListener"/>

<camelContext id="route" xmlns="http://camel.apache.org/schema/blueprint">
    <route>
        <from uri="infinispan://?
cacheContainer=#cacheManager&sync=false&customListener=#myCustomListener"/
>
        <to uri="mock:result"/>
    </route>
</camelContext>

```

The instance of **myCustomListener** must exist. Users are encouraged to extend the **org.apache.camel.component.infinispan.remote.InfinispanRemoteCustomListener** class and annotate the resulting class with **@ClientListener**; this annotation is found in **org.infinispan.client.hotrod.annotation**.

Remote listeners may also be associated with custom filters and converters as shown below:

```

@ClientListener(includeCurrentState=true, filterFactoryName = "static-
filter-factory", converterFactoryName = "static-converter-factory")
private static class MyCustomListener extends
InfinispanRemoteCustomListener {
}

```

In order to use custom filters or converters classes annotated with **@NamedFactory** must be implemented. A skeleton that implements the necessary methods is shown below:

```

import org.infinispan.notifications.cachelistener.filter;

```

```

@NamedFactory(name = "static-converter-factory")
public static class StaticConverterFactory implements
CacheEventConverterFactory {
    @Override
    public CacheEventConverter<Integer, String, CustomEvent>
getConverter(Object[] params) {
        ...
    }

    static class StaticConverter implements CacheEventConverter<Integer,
String, CustomEvent>, Serializable {
        @Override
        public CustomEvent convert(Integer key, String previousValue, Metadata
previousMetadata,
                                String value, Metadata metadata, EventType
eventType) {
            ...
        }
    }
}

@NamedFactory(name = "static-filter-factory")
public static class StaticCacheEventFilterFactory implements
CacheEventFilterFactory {
    @Override
    public CacheEventFilter<Integer, String> getFilter(final Object[]
params) {
        ...
    }

    static class StaticCacheEventFilter implements CacheEventFilter<Integer,
String>, Serializable {
        @Override
        public boolean accept(Integer key, String previousValue, Metadata
previousMetadata,
                                String value, Metadata metadata, EventType
eventType) {
            ...
        }
    }
}

```

Custom filters and converters must be registered with the server. Registering these classes is documented in the **Remote Event Listeners** section of the *Red Hat JBoss Data Grid Developer Guide*.



NOTE

In order to listen for remote HotRod events the cacheManager must be of type **RemoteCacheManager** and instantiated.

[Report a bug](#)

5.6. RED HAT JBOSS DATA GRID AND RED HAT JBOSS FUSE

5.6.1. Installing camel-jbossgdatagrid for Red Hat JBoss Fuse

Red Hat JBoss Fuse is an OSGi container based on the Karaf container. To run Red Hat JBoss Data Grid and JBoss Fuse using **camel-jbossgdatagrid**, ensure that both JBoss Data Grid 7.0 and JBoss Fuse 6.1 (Full Installation) are installed.

Procedure 5.1. Installing JBoss Data Grid

- For information about installing JBoss Data Grid, see [Part II, “Download and Install Red Hat JBoss Data Grid”](#). Only the following JBoss Data Grid components are required to run the camel component in JBoss Fuse:
 - JBoss Data Grid Maven repository.
 - The JBoss Data Grid Server package (to use the Hot Rod client).

The **camel-jbossgdatagrid** library is also available in a separate distribution called **jbossgdatagrid-7.0.0-camel-library**.

Procedure 5.2. Installing JBoss Fuse

Prerequisites

Before attempting to install and use Red Hat JBoss Fuse, ensure your system meets the minimum requirements. For supported Platforms and recommended Java Runtime platforms, see the *Red Hat JBoss Fuse Installation Guide*

The following hardware is required for the JBoss Fuse 6.1 Full Installation.

- 700 MB of free disk space
- 2 GB of RAM

In addition to the disk space required for the base installation, a running system will require space for caching, persistent message stores, and other functions.

1. Download the JBoss Fuse Full Installation

You can download the Red Hat JBoss Fuse archive from the **Red Hat Customer Portal>Downloads>Red Hat JBoss Middleware>Downloads** page, after you register and login to your customer account.

When logged in:

- a. Select **Fuse**, listed under **Integrated Platforms** in the sidebar menu.
- b. Select **6.1.0** from the Version drop-down list on the **Software Downloads** page.
- c. Click the **Download** button next to the Red Hat JBoss Fuse 6.1.0 distribution file to download.

JBoss Fuse allows you to choose between installations that contain different feature sets. To run JBoss Data Grid with JBoss Fuse, the Full installation is required. The Full installation includes the following:

- Apache Karaf
- Apache Camel

- Apache ActiveAMQ
- Apache CXF
- Fuse Management
- Console (hawtio)
- JBI components

2. Unpacking the Archive

Red Hat JBoss Fuse is installed by unpacking an archive on a system. JBoss Fuse is packaged as a zip file. Using a suitable archive tool, unpack Red Hat JBoss Fuse into a directory to which you have full access.



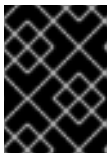
WARNING

Do not unpack the archive file into a folder that has spaces in its path name. For example, do not unpack into **C:\Documents and Settings\Greco Roman\Desktop\fusesrc**.

Additionally, do not unpack the archive file into a folder that has any of the following special characters in its path name: #, %, ^, ".

3. Adding a Remote Console User

The server's remote command console is not configured with a default user. Before remotely connecting to the server's console, add a user to the configuration.



IMPORTANT

The information in this file is unencrypted so it is not suitable for environments that require strict security.

To add a user:

- Open **InstallDir/etc/users.properties** in your favorite text editor.
- Locate the line **#admin=admin,admin**. This line specifies a user **admin** with the password **admin** and the role **admin**.
- Remove the leading **#** to uncomment the line.
- Replace the first **admin** with a name for the user.
- Replace the second **admin** with the password for the user.
- Leave the last **admin** as it is, and save the changes.

**NOTE**

To access the Fuse Management Console to monitor and manage your Camel routes, ActiveMQ brokers, Web applications, and so on, open a browser to `http://localhost:8181/hawtio`, after starting Red Hat JBoss Fuse.

4. Red Hat JBoss Fuse Maven Repositories

To use Maven to build projects, specify the location of the artifacts in a Maven `settings.xml` file.

The following JBoss Fuse Maven repository contains the required dependencies for Camel and must be added to the `settings.xml` file.

<https://repo.fusesource.com/nexus/content/groups/public/>

The JBoss Fuse repository runs alongside the JBoss Data Grid repository.

JBoss Data Grid includes a `features.xml` file for Karaf that deploys all artifacts required for the `camel-jbossgatagrid` component. This file is not included in the JBoss Fuse container distribution. The `features.xml` file is in `jboss-datagrid-7.0.0-maven-repository/org/apache/camel/camel-jbossgatagrid/${version}/`. No further configuration of the JBoss Data Grid repository is required.

For more information about installing and getting started with JBoss Fuse, see the Red Hat JBoss Fuse documentation on the Red Hat Customer Portal.

[Report a bug](#)

5.7. RED HAT JBOSS DATA GRID AND RED HAT JBOSS EAP**5.7.1. Installing camel-jbossgatagrid for Red Hat JBoss Enterprise Application Platform**

Red Hat JBoss Enterprise Application Platform 6 (JBoss EAP 6) is a middleware platform built on open standards and compliant with the Java Enterprise Edition 6 specification.

As Camel is only supported through Red Hat JBoss Fuse valid entitlements for all of the following products will be necessary:

- Red Hat JBoss EAP
- Red Hat JBoss Fuse
- Red Hat JBoss Data Grid

**NOTE**

Entitlement to Red Hat JBoss Fuse Service Works includes entitlements to Red Hat JBoss EAP and Red Hat JBoss Fuse.

The following variables are used instead of specific version numbers in the installation procedures below:

- **jdg.version** - the latest version of Red Hat JBoss Data Grid
- **fuse.version** - the latest version of Red Hat JBoss Fuse
- **infinispan.version** - the version of Infinispan that is included in the latest version of Red Hat JBoss Data Grid
- **camel.version** - the version of Apache Camel that is included in the latest version of Red Hat JBoss Fuse

For more information on tested integrations for **camel-jbossdatagrid**, please refer to <https://access.redhat.com/articles/115883>.

Procedure 5.3. Installing JBoss Data Grid

- For information about installing JBoss Data Grid, see [Part II, “Download and Install Red Hat JBoss Data Grid”](#). Only the following JBoss Data Grid components are required to run the camel component in JBoss EAP:
 - JBoss Data Grid Maven repository.
 - The JBoss Data Grid Server package (to use the Hot Rod client).

The **camel-jbossdatagrid** library is also available in a separate distribution called **jboss-datagrid-`{jdg.version}`-camel-library**.

Procedure 5.4. Installing JBoss EAP

1. Before attempting to install and use Red Hat JBoss EAP ensure your system meets the minimum requirements as documented in *Red Hat JBoss EAP Installation Guide*
2. **Unpacking the Archive**
Red Hat JBoss EAP is installed by unpacking an archive on a system, as JBoss EAP is packaged as a zip file; using a suitable archive tool, unpack Red Hat JBoss EAP into a directory to which you have full access.



WARNING

Do not unpack the archive file into a folder that has spaces in its path name. For example, do not unpack into **C:\Documents and Settings\Greco Roman\Desktop\JBoss**.

Additionally, do not unpack the archive file into a folder that has any of the following special characters in its path name: #, %, ^, ".
.

3. Once the archive has been extracted JBoss EAP will have been successfully installed. For more information on installation options refer to the *Red Hat JBoss EAP Installation Guide*.
4. If JBoss Data Grid is being used in Library mode then refer to [Section 4.1, “Deploy JBoss Data Grid in JBoss EAP \(Library Mode\)”](#) to ensure the necessary dependencies have been installed.
5. If JBoss Data Grid is being used in Remote Client-Server mode then refer to [Section 4.2, “Deploy JBoss Data Grid in JBoss EAP \(Remote Client-Server Mode\)”](#) to ensure the necessary dependencies have been installed.

[Report a bug](#)

5.7.2. Deploy Camel with EAP:

5.7.2.1. Add development and runtime dependencies

In order to compile your application the dependent libraries for Camel and JBoss Data Grid will have to be added to the `pom.xml` (if using Maven).

Procedure 5.5. Add Camel from Fuse

1. Ensure that the Fuse repository has been added to the `pom.xml`:

```
<repository>
  <id>fusesource</id>
  <name>FuseSource Release Repository</name>

  <url>https://repo.fusesource.com/nexus/content/groups/public/</url>
  <snapshots>
    <enabled>>false</enabled>
  </snapshots>
  <releases>
    <enabled>>true</enabled>
  </releases>
</repository>
```

2. Add the Camel components as a dependency in the `pom.xml`:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-core</artifactId>
  <version>${camel.version}</version>
</dependency>
```

Procedure 5.6. Add camel-jbossgatagrid to the deployment

1. Follow the instructions in [Chapter 3, Install and Use the Maven Repositories](#) to add the maven repository.
2. Add camel-jbossgatagrid as a dependency in the `pom.xml`:

```
<dependency>
```

```

<groupId>org.apache.camel</groupId>
<artifactId>camel-jbosmdatagrid</artifactId>
<version>${jdg.version}</version>
</dependency>

```

3. Add in the remaining JBoss Data Grid dependencies depending on which functionality is in use:

```

<!-- If Remote Camel Producer is used add the following dependency -
->
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-remote</artifactId>
  <version>${infinispan.version}</version>
</dependency>

<!-- If the Local Camel Producer or Local Camel Consumer are in use
add -->
<!-- the following dependency -->
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-embedded</artifactId>
  <version>${infinispan.version}</version>
</dependency>

```

[Report a bug](#)

5.7.2.2. Optionally: Add runtime dependencies as a JBoss EAP Module

Sometimes it is preferable to maintain other product libraries in JBoss EAP as modules. This requires some additional procedures to create these modules.



NOTE

When using modules for dependencies the **pom.xml** will need to have the **scope** set to "provided" for any dependencies provided as modules. For example:

```

<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-jbosmdatagrid</artifactId>
  <scope>provided</scope>
  <version>${jdg.version}</version>
</dependency>

```

Download **jboss-datagrid-\${jdg.version}-camel-library.zip** from the customer portal.

Extract the **apache-camel-\${camel.version}.zip** found in Red Hat JBoss Fuse's **sextras/** directory:

```

user@example modules] unzip /path/to/jboss-fuse-
${fuse.version}/extras/apache-camel-${camel.version}

```

Procedure 5.7. Add in the Camel components from JBoss Fuse

1. Create a directory under \$EAP_HOME/modules:

```
user@example jboss-eap-6.4] cd modules
user@example modules] mkdir -p org/apache/camel/core
```

2. Create a **main** subdirectory to store the jars:

```
user@example modules] mkdir org/apache/camel/core/main
```

3. Copy over the camel-core jar from **apache-camel-`{camel.version}`.zip** to the newly created **main** directory:

```
user@example modules] cp /path/to/jboss/fuse/extras/apache-camel-
{camel.version}/lib/camel-core-{camel.version}.jar
./org/apache/camel/core/main/
```

4. Create the **module.xml** descriptor by adding in the following text to **org/apache/camel/core/main/module.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.apache.camel.core">
  <resources>
    <resource-root path="camel-core-{camel.version}.jar"/>
  </resources>
</module>
```

5. Repeat the above steps to create a module for each dependency in use; note that modules may have dependencies on other modules as described in the *Red Hat JBoss Administration and Configuration Guide*.

Procedure 5.8. Add in Camel components from JBoss Data Grid

1. Create a **main** subdirectory for the JDG Camel components:

```
user@example jboss-eap-6.4] mkdir -p modules/org/apache/camel/main
```

2. Unzip **jboss-datagrid-`{jdg.version}`-camel-library.zip**.

3. Copy **camel-jbosmdatagrid-`{jdg.version}`.jar** into the newly created directory:

```
user@example jboss-eap-6.4] cp jboss-datagrid-{jdg.version}-camel-
library/camel-jbosmdatagrid-{jdg.version}.jar
modules/org/apache/camel/main/
```

4. Create a **module.xml** descriptor by adding in the following text to **org/apache/camel/main/module.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="org.apache.camel">
  <resources>
    <resource-root path="camel-jbosmdatagrid-{jdg.version}.jar"/>
  </resources>
```

```
<dependencies>
  <module name="org.apache.camel.core" />
</dependencies>
</module>
```

Create a **jboss-deployment-structure.xml** in the **WEB-INF** of the war, and add dependencies on the newly created module:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.apache.camel" meta-inf="import"/>

      <!-- Add the following lines if Library mode is used -->
      <module name="org.infinispan" slot="jdg-7.0" />
      <module name="org.jgroups" slot="jdg-7.0" />

      <!-- Add the following lines if Remote Client-Server mode is
used -->
      <module name="org.infinispan.client.hotrod" slot="jdg-7.0" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

[Report a bug](#)

PART IV. RUNNING RED HAT JBOSS DATA GRID

CHAPTER 6. RUN RED HAT JBOSS DATA GRID JAR FILES WITH MAVEN

6.1. RUN JBOSS DATA GRID (REMOTE CLIENT-SERVER MODE)

Use the following instructions to run Red Hat JBoss Data Grid JAR files with Maven in Remote Client-Server mode.

Hot Rod Client with Querying

Add the following dependencies to the `pom.xml` file:

1. Add **infinispan-remote** dependency:

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-remote</artifactId>
  <version>${infinispan.version}</version>
</dependency>
```

2. For instances where a **Remote Cache Store** is in use also add the **infinispan-embedded** dependency as shown below:

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-embedded</artifactId>
  <version>${infinispan.version}</version>
</dependency>
```

3. For instances where **JSR-107** is in use, ensure that the **cache-api** packages are available at runtime. Having these packages available can be accomplished by any of the following methods:

1. **Option 1:** If JBoss EAP is in use, then add the JBoss Data Grid modules to this instance as described in [Section 4.2, “Deploy JBoss Data Grid in JBoss EAP \(Remote Client-Server Mode\)”](#).

Add the `javax.cache.api` module to the application's **jboss-deployment-structure.xml**. An example is shown below:

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-structure:1.2">
  <deployment>
    <dependencies>
      <module name="javax.cache.api" slot="jdg-7.0"
services="export"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

2. **Option 2:** Download the **jboss-datagrid-`{jdg.version}`-library** file from the customer portal.

Extract the downloaded archive.

Embed the `jboss-datagrid-{jdg.version}-library/lib/cache-api-{jcache.version}.jar` file into the desired application.

3. **Option 3:** If the JBoss Data Grid Maven repository is available then add an explicit dependency to the `pom.xml` of the project as seen below:

```
<dependency>
  <groupId>javax.cache</groupId>
  <artifactId>cache-api</artifactId>
  <version>${jcache.version}</version>
</dependency>
```



WARNING

The Infinispan query API directly exposes the Hibernate Search and the Lucene APIs and cannot be embedded within the `infinispan-embedded-query.jar` file. Do not include other versions of Hibernate Search and Lucene in the same deployment as `infinispan-embedded-query`. This action will cause classpath conflicts and result in unexpected behavior.

[Report a bug](#)

6.2. RUN JBOSS DATA GRID (LIBRARY MODE)

Use the provided instructions to run Red Hat JBoss Data Grid with Maven in Library mode.



NOTE

To simplify embedding Red Hat JBoss Data Grid directly in your application, the distribution of JBoss Data Grid contains fewer, consolidated jars. For the list of supported jar files, see the Packaging Revisions in the *Release Notes*.

Infinispan Embedded without Querying

Add the following dependencies to the `pom.xml` file:

1. Add `infinispan-embedded` dependency:

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-embedded</artifactId>
  <version>${infinispan.version}</version>
</dependency>
```

Infinispan Embedded with Querying

Add the following dependencies to the `pom.xml` file:

1. Add **infinispan-embedded-query** dependency:

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-embedded-query</artifactId>
  <version>${infinispan.version}</version>
</dependency>
```



WARNING

The Infinispan query API directly exposes the Hibernate Search and the Lucene APIs and cannot be embedded within the **infinispan-embedded-query.jar** file. Do not include other versions of Hibernate Search and Lucene in the same deployment as **infinispan-embedded-query**. This action will cause classpath conflicts and result in unexpected behavior.

[Report a bug](#)

CHAPTER 7. RUN RED HAT JBOSS DATA GRID IN REMOTE CLIENT-SERVER MODE

7.1. PREREQUISITES

The following is a list of prerequisites to run Red Hat JBoss Data Grid in Remote Client-Server mode for the first time:

- Ensure an appropriate version of OpenJDK is installed. For more information, see [Section 2.3, “Install OpenJDK on Red Hat Enterprise Linux”](#)
- Download and install the latest version of JBoss Data Grid. For more information, see [Section 2.4.1, “Download Red Hat JBoss Data Grid”](#)

[Report a bug](#)

7.2. RUN RED HAT JBOSS DATA GRID IN STANDALONE MODE

Standalone mode refers to a single instance of Red Hat JBoss Data Grid operating in local mode. In local mode, JBoss Data Grid operates as a simple single node in-memory data cache.

Run the following script to start JBoss Data Grid in standalone mode:

```
$JDG_HOME/bin/standalone.sh
```

This command starts JBoss Data Grid using the default configuration information provided in the `$JDG_HOME/standalone/configuration/standalone.xml` file.

[Report a bug](#)

7.3. RUN RED HAT JBOSS DATA GRID IN CLUSTERED MODE

Clustered mode refers to a cluster made up of two or more Red Hat JBoss Data Grid standalone instances.

Run the following script to start JBoss Data Grid in clustered mode:

```
$JDG_HOME/bin/standalone.sh -c clustered.xml
```

This command starts JBoss Data Grid using the default configuration information provided in the `$JDG_HOME/standalone/configuration/clustered.xml` file.

[Report a bug](#)

7.4. RUN RED HAT JBOSS DATA GRID IN A MANAGED DOMAIN

A managed domain allows multiple server instances and groups to be centrally managed from the Administration Console of the domain controller.

Run the following script to start JBoss Data Grid in a managed domain:

```
$JDG_HOME/bin/domain.sh
```

This command starts JBoss Data Grid using the default configuration information provided in the `$JDG_HOME/domain/configuration/domain.xml` and `$JDG_HOME/domain/configuration/host.xml` files.

[Report a bug](#)

7.5. RUN RED HAT JBOSS DATA GRID WITH A CUSTOM CONFIGURATION

To run Red Hat JBoss Data Grid with a custom configuration, add a configuration file to the `$JDG_HOME/standalone/configuration` directory.

Use the following command to specify the created custom configuration file for standalone mode:

```
$JDG_HOME/bin/standalone.sh -c ${FILENAME}
```

The `-c` used for this script does not allow absolute paths, therefore the specified file must be available in the `$JDG_HOME/standalone/configuration` directory.

If the command is run without the `-c` parameter, JBoss Data Grid uses the default configuration.

As a managed domain is configured with two separate files, `domain.xml` and `host.xml` by default, there are two separate flags for specifying custom configuration files.

To define the custom configuration file for the server group profiles, use the `-c` parameter as described above, and demonstrated in the following command:

```
$JDG_HOME/bin/domain.sh -c ${FILENAME}
```

To define the custom configuration file for the servers use the `--host-config` parameter, as demonstrated in the following command:

```
$JDG_HOME/bin/domain.sh --host-config=${FILENAME}
```

[Report a bug](#)

7.6. SET AN IP ADDRESS TO RUN RED HAT JBOSS DATA GRID

For production use, the Red Hat JBoss Data Grid server must be bound to a specified IP address rather than to `127.0.0.1/localhost`. Use the `-b` parameter with the script to specify an IP address.

For standalone mode, set the IP address as follows:

```
$JDG_HOME/bin/standalone.sh -b ${IP_ADDRESS}
```

For domain mode, set the IP address for the host controller and any servers as follows:

```
$JDG_HOME/bin/domain.sh -b ${IP_ADDRESS}
```

■ [Report a bug](#)

7.7. RUNNING RED HAT JBOSS DATA GRID

JBoss Data Grid can be run in one of three ways:

- Use the following command to run JBoss Data Grid using the configuration defined in the **standalone.xml** file (located at **\$JDG_HOME/standalone/configuration**):

```
$JDG_HOME/bin/standalone.sh
```

- Use the following command with an appended **-c** followed by the configuration file name to run JBoss Data Grid with a non-default configuration file:

```
$JDG_HOME/bin/standalone.sh -c clustered.xml
```

- Use the following command to run JBoss Data Grid using the configuration defined in the **domain.xml** and **host.xml** files (located at **\$JDG_HOME/domain/configuration/**):

```
$JDG_HOME/bin/domain.sh
```

[Report a bug](#)

CHAPTER 8. RUN A RED HAT JBOSS DATA GRID AS A NODE WITHOUT ENDPOINTS

Services send messages using channels to communicate with each other. An endpoint is a communications point for these services and is used to send and receive the messages sent through the channels. As a result, a node with no endpoints can communicate with other nodes in the same cluster, but not with clients.

[Report a bug](#)

8.1. BENEFITS OF A NODE WITHOUT ENDPOINTS

The primary benefit for creating a node without endpoints in Red Hat JBoss Data Grid involves data replication.

A node without any endpoints cannot be accessed by the client directly. As a result, they are primarily used to replicate data from other nodes that can communicate with clients. The result is a node with a backup copy of the data that cannot be accessed by the client, which protects it from failure via an error sent by the client.

[Report a bug](#)

8.2. SAMPLE CONFIGURATION FOR A NODE WITHOUT ENDPOINTS

Red Hat JBoss Data Grid provides a sample configuration to configure a node without an endpoint:

Procedure 8.1. Find the JBoss Data Grid Sample Configuration for a Node Without Endpoints

1. **Extract the JBoss Data Grid ZIP**

1. Extract the ZIP file for JBoss Data Grid Remote Client-Server mode. This is named **jboss-datagrid-server-*{version}***. Add the relevant version to the file name.

2. **Navigate to the Appropriate Folder**

- In the extracted folder, navigate to the **`$JDG_HOME/docs/examples/config`** folder.

3. **Find the Configuration Sample File**

- View the **`clustered-storage-only.xml`** file, which contains the configuration for a node with no endpoints.

[Report a bug](#)

8.3. CONFIGURE A NODE WITH NO ENDPOINTS

A standard configuration, such as a standalone high availability configuration, can be changed into a configuration for a node with no endpoints using the following steps:

1. Remove the ***datagrid*** subsystem.
2. Remove the ***modcluster*** subsystem.
3. Remove the ***datasource*** definition.

4. Remove *socket-bindings* for *mod_cluster*, **Hot Rod** and **memcached**.

Removing the listed items ensure that all endpoints are removed from the configuration and that clustering is not possible. The resulting configuration is a node with no endpoints.

[Report a bug](#)

CHAPTER 9. RUN RED HAT JBOSS DATA GRID IN LIBRARY MODE

This part includes information about using Red Hat JBoss Data Grid in Library Mode.

- As a prerequisite for the subsequent chapters, set up a new project using the instructions in [Section 9.1, “Create a New Red Hat JBoss Data Grid Project”](#)
- Next, use JBoss Data Grid either as an embedded cache (see [Chapter 10, Run Red Hat JBoss Data Grid in Library Mode \(Single-Node Setup\)](#) for more information) or as a clustered cache (see [Chapter 11, Run Red Hat JBoss Data Grid in Library Mode \(Multi-Node Setup\)](#)). Each tutorial is based on an Infinispan quickstart.
- Finally, monitor Red Hat JBoss EAP applications using JBoss Data Grid using the instructions in [Chapter 12, Monitor Red Hat JBoss Data Grid Applications in Red Hat JBoss EAP](#)

[Report a bug](#)

9.1. CREATE A NEW RED HAT JBOSS DATA GRID PROJECT

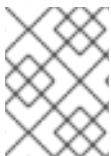
This chapter is a guide to creating a new Red Hat JBoss Data Grid project. The tasks prescribed are a prerequisite for the quickstart tasks provided in [Chapter 10, Run Red Hat JBoss Data Grid in Library Mode \(Single-Node Setup\)](#) and [Chapter 11, Run Red Hat JBoss Data Grid in Library Mode \(Multi-Node Setup\)](#)

[Report a bug](#)

9.2. ADD DEPENDENCIES TO YOUR PROJECT

Set up Red Hat JBoss Data Grid by adding dependencies to your project. If you are using Maven or other build systems that support Maven dependencies, add the following to your `pom.xml` file, located in the Maven repository folder:

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-embedded</artifactId>
  <version>${VERSION}</version>
</dependency>
```



NOTE

Replace the **version** value with the appropriate version of the libraries included in JBoss Data Grid.

[Report a bug](#)

9.3. ADD A PROFILE TO YOUR PROJECT

To enable the JBoss Maven repository for your project, add a profile to your `settings.xml` file in `$HOME/.m2/settings.xml` as follows:

Example 9.1. Adding a Profile

```

<profiles>

  <!-- Configure the JBoss GA Maven repository -->
  <profile>
    <id>jboss-ga-repository</id>
    <repositories>
      <repository>
        <id>jboss-ga-repository</id>
        <url>http://maven.repository.redhat.com/techpreview/all</url>
        <releases>
          <enabled>>true</enabled>
        </releases>
        <snapshots>
          <enabled>>false</enabled>
        </snapshots>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>jboss-ga-plugin-repository</id>
        <url>http://maven.repository.redhat.com/techpreview/all</url>
        <releases>
          <enabled>>true</enabled>
        </releases>
        <snapshots>
          <enabled>>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>

  <!-- Configure the JBoss Early Access Maven repository -->
  <profile>
    <id>jboss-earlyaccess-repository</id>
    <repositories>
      <repository>
        <id>jboss-earlyaccess-repository</id>
        <url>http://maven.repository.redhat.com/earlyaccess/all/</url>
        <releases>
          <enabled>>true</enabled>
        </releases>
        <snapshots>
          <enabled>>false</enabled>
        </snapshots>
      </repository>
    </repositories>
    <pluginRepositories>
      <pluginRepository>
        <id>jboss-earlyaccess-plugin-repository</id>
        <url>http://maven.repository.redhat.com/earlyaccess/all/</url>
        <releases>
          <enabled>>true</enabled>
        </releases>
      </pluginRepository>
    </pluginRepositories>
  </profile>

```

```
        <snapshots>
            <enabled>>false</enabled>
        </snapshots>
    </pluginRepository>
</pluginRepositories>
</profile>

</profiles>

<!-- Add active profiles information here -->
```

Enable the profile by ensuring the following is included in the **settings.xml** file:

Example 9.2. Enable the Profile

```
<activeProfiles>
  <!-- Optionally, make the repositories active by default -->
  <activeProfile>jboss-ga-repository</activeProfile>
  <activeProfile>jboss-earlyaccess-repository</activeProfile>
</activeProfiles>
```

If you are using a build system that does not support declarative dependency management, add the contents of the **client/java/** directory, included in the Red Hat JBoss Data Grid package to the build classpath.

[Report a bug](#)

CHAPTER 10. RUN RED HAT JBOSS DATA GRID IN LIBRARY MODE (SINGLE-NODE SETUP)

10.1. RUN THE QUICKSTART CLASS

Prerequisites

These quickstarts use the Infinispan quickstarts located at <https://github.com/infinispan/infinispan-quickstart>. The following procedure uses the `infinispan-quickstart/embedded-cache` quickstart.

Procedure 10.1. Run the Quickstart Class

1. **Open the Quickstart.java File**

Open the file called `Quickstart.java` in the `infinispan-quickstart/embedded-cache`.

2. **Add the Quickstart Class**

Examining the `Quickstart.java` file we can see that this class creates a `DefaultCacheManager` and then obtains a local reference to the `default` cache, as no cache name is specified:

```
package org.infinispan.quickstart.embeddedcache;

import org.infinispan.Cache;
import org.infinispan.manager.DefaultCacheManager;

public class Quickstart {

    public static void main(String args[]) throws Exception {
        Cache<Object, Object> c = new
        DefaultCacheManager().getCache();
    }

}
```

3. **Copy Dependencies and Compile Java Classes**

Use the following command to copy all project dependencies to a directory and compile the Java classes from your project:

```
$ mvn clean compile dependency:copy-dependencies -DstripVersion
```

4. **Run the Main Method**

Use the following command to run the main method:

```
$ java -cp target/classes/:target/dependency/* package
org.infinispan.quickstart.embeddedcache.Quickstart
```

[Report a bug](#)

10.2. USE THE DEFAULT CACHE

10.2.1. Add and Remove Data from the Cache

Red Hat JBoss Data Grid offers an interface that is similar to the proposed JSR-107 API to access and alter data stored in a cache.

The following procedure is an example that defines what each line entered into the `DefaultCacheQuickstart.java` file does:

Procedure 10.2. Add and Remove Data from the Cache

1. Add an entry, replacing *key* and *value* with the desired key and value:

```
cache.put("key", "value");
```

2. Confirm that the entry is present in the cache:

```
assertEquals(1, cache.size());  
assertTrue(cache.containsKey("key"));
```

3. Remove the entry from the cache:

```
Object v = cache.remove("key");
```

4. Confirm that the entry is no longer present in the cache:

```
assertEquals("value", v);  
assertTrue(cache.isEmpty());
```

[Report a bug](#)

10.2.2. Adding and Replacing a Key Value

Red Hat JBoss Data Grid offers a thread-safe data structure.

The following procedure is an example that defines what each line entered into the `DefaultCacheQuickstart.java` file does:

Procedure 10.3. Adding and Replacing a Key Value

- Add an entry **key** with **value** as the key's value.

```
cache.put("key", "value");
```

Procedure 10.4. Replacing a Key Value

1. The following code searches for keys (named **key** and **key2**). If the two specific keys beings searched for are not found, JBoss Data Grid creates two new keys with the specified key names and values.

```
cache.putIfAbsent("key", "newValue");  
cache.putIfAbsent("key2", "value2");
```

- The following code confirms that the value of the stored key equals the value we wanted to store.

```
assertEquals(cache.get("key"), "value");
assertEquals(cache.get("key2"), "value2");
```

See Also:

- [Section A.1, “About Key-Value Pairs”](#)

[Report a bug](#)

10.2.3. Removing Entries

Separate methods are used to remove entries depending on how JBoss Data Grid is used:

Library Mode

All of the following methods are found on **org.infinispan.Cache** and its subclasses.

- remove(key)** - remove a single key from the cache.
- removeAsync(key)** - remove a single key from the cache, asynchronously.
- clear()** - removes all of the mappings from the cache, leaving it empty once the call completes.
- clearAsync()** - asynchronously remove all of the mappings from the cache, leaving it empty once the call completes.
- cache.evict(key)** - remove the entry from the cache, moving it to the cache store if one is defined. With no cache store defined the entry is removed from the cache and is lost.

Remote Client-Server Mode

All of the following methods are found on **org.infinispan.client.hotrod.RemoteCache** and its subclasses.

- remove(key)** - remove a single key from the cache.
- removeAsync(key)** - remove a single key from the cache, asynchronously.
- clear()** - removes all of the mappings from the cache, leaving it empty once the call completes.
- clearAsync()** - asynchronously remove all of the mappings from the cache, leaving it empty once the call completes.
- removeWithVersion(key, version)** - remove a single key from the cache only if its current version matches the supplied version.
- removeWithVersionAsync(key, value)** - asynchronously remove a single key from the cache only if its current version matches the supplied version.

For additional information on any of the above methods refer to the *API Documentation*.

[Report a bug](#)

10.2.4. Placing and Retrieving Sets of Data

The **AdvancedCache** and **RemoteCache** interfaces include methods to either put or get a **Map** of existing data in bulk. These operations are typically much more efficient than an equivalent sequence of individual operations, especially when using them in server-client mode, as a single network operation occurs as opposed to multiple transactions.

When using the bulk operations the memory overhead is higher during the operation itself, as the **get** or **put** operation must accommodate for the full **Map** in a single execution.

The methods for each class are found below:

- **AdvancedCache:**
 - **Map<K, V> getAll(Set<?> keys)** - returns a **Map** containing the values associated with the set of keys requested.
 - **void putAll(Map<? extends K, ? extends V> map, Metadata metadata)** - copies all of the mappings from the specified map to this cache, which takes an instance of **Metadata** to provide metadata information such as the lifespan, version, etc. on the entries being stored.
- **RemoteCache:**
 - **Map<K, V> getAll(Set<? extends K> keys)** - returns a **Map** containing the values associated with the set of keys requested.
 - **void putAll(Map<? extends K, ? extends V> map)** - copies all of the mappings from the specified map to this cache.
 - **void putAll(Map<? extends K, ? extends V> map, long lifespan, TimeUnit unit)** - copies all of the mappings from the specified map to this cache, along with a lifespan before the entry is expired.
 - **void putAll(Map<? extends K, ? extends V> map, long lifespan, TimeUnit lifespanUnit, long maxIdleTime, TimeUnit maxIdleTimeUnit)** - copies all of the mappings from the specified map to this cache, along with both a timespan before the entries are expired and the maximum amount of time the entry is allowed to be idle before it is considered to be expired.

[Report a bug](#)

10.2.5. Adjust Data Life

Red Hat JBoss Data Grid entries are immortal by default, but these settings can be altered.

The following procedure is an example that defines what each line entered into the **DefaultCacheQuickstart.java** file does:

Procedure 10.5. Adjust the Data Life

1. Alter the key's *lifespan* value:

```
cache.put("key", "value", 5, TimeUnit.SECONDS);
```

2. Check if the cache contains the key:

```
assertTrue(cache.containsKey("key"));
```

3. After the allocated ***Lifespan*** time has expired, the key is no longer in the cache:

```
Thread.sleep(10000);  
assertFalse(cache.containsKey("key"));
```

[Report a bug](#)

10.2.6. 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](#)

10.2.7. Register the Named Cache Using XML

To configure the named cache declaratively (using XML) rather than programmatically, configure the **`infinispan.xml`** file.

An example **`infinispan.xml`** file is located in <https://github.com/jboss-developer/jboss-jdg-quickstarts/> within the **`secure-embedded-cache/src/main/resources/`** folder, and the full schema is available in the **`docs/schema/`** directory of the *Red Hat JBoss Data Grid Library* distribution.

[Report a bug](#)

CHAPTER 11. RUN RED HAT JBOSS DATA GRID IN LIBRARY MODE (MULTI-NODE SETUP)

11.1. SHARING JGROUP CHANNELS

Red Hat JBoss Data Grid offers an easy to use form of clustering using JGroups as the network transport. As a result, JGroups manages the initial operations required to form a cluster for JBoss Data Grid.

All caches created from a single CacheManager share the same JGroups channel by default. This JGroups channel is used to multiplex replication/distribution messages.

In the following example, all three caches used the same JGroups channel:

Example 11.1. Shared JGroups Channel

```
EmbeddedCacheManager cm = $LOCATION
Cache<Object, Object> cache1 = cm.getCache("replSyncCache");
Cache<Object, Object> cache2 = cm.getCache("replAsyncCache");
Cache<Object, Object> cache3 = cm.getCache("invalidationSyncCache");
```

Substitute `$LOCATION` with the CacheManager's location.

[Report a bug](#)

11.2. RUN RED HAT JBOSS DATA GRID IN A CLUSTER

The clustered quickstarts for Red Hat JBoss Data Grid are based on the quickstarts found in <https://github.com/infinispan/infinispan-quickstart/tree/master/clustered-cache>.

[Report a bug](#)

11.2.1. Compile the Project

Use Maven to compile your project with the following command:

```
$ mvn clean compile dependency:copy-dependencies -DstripVersion
```

[Report a bug](#)

11.2.2. Run the Clustered Cache with Replication Mode

To run Red Hat JBoss Data Grid's replication mode example of a clustered cache, launch two nodes from different consoles.

Procedure 11.1. Run the Clustered Cache with Replication Mode

1. Use the following command to launch the first node:

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.replication.Node0
```

-
- 2. Use the following command to launch the second node:

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.replication.Node1
```

Result

JGroups and JBoss Data Grid initialized on both nodes. After approximately fifteen seconds, the cache entry log message appears on the console of the first node.

[Report a bug](#)

11.2.3. Run the Clustered Cache with Distribution Mode

To run Red Hat JBoss Data Grid's distribution mode example of a clustered cache, launch three nodes from different consoles.

Procedure 11.2. Run the Clustered Cache with Distribution Mode

1. Use the following command to launch the first node:

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.distribution.Node0
```

2. Use the following command to launch the second node:

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.distribution.Node1
```

3. Use the following command to launch the third node:

```
$ java -cp target/classes/:target/dependency/*
org.infinispan.quickstart.clusteredcache.distribution.Node2
```

Result

JGroups and JBoss Data Grid initialized on the three nodes. After approximately fifteen seconds, the ten entries added by the third node are visible as they are distributed to the first and second nodes.

[Report a bug](#)

11.2.4. Configure the Cluster

Use the following steps to add and configure your cluster:

Procedure 11.3. Configure the Cluster

1. Add the default configuration for a new cluster.
2. Customize the default cluster configuration according to the requirements of your network. This is done declaratively (using XML) or programmatically.

3. Configure the replicated or distributed data grid.

[Report a bug](#)

11.2.4.1. Add the Default Cluster Configuration

Add a cluster configuration to ensure that Red Hat JBoss Data Grid is aware that a cluster exists and is defined. The following is a default configuration that serves this purpose:

Example 11.2. Default Configuration

```
new ConfigurationBuilder()
    .clustering().cacheMode(CacheMode.REPL_SYNC)
    .build()
```



NOTE

Use the new `GlobalConfigurationBuilder().clusteredDefault()` to quickly create a preconfigured and cluster-aware `GlobalConfiguration` for clusters. This configuration can also be customized.

[Report a bug](#)

11.2.4.2. Customize the Default Cluster Configuration

Depending on the network requirements, you may need to customize your JGroups configuration.

Programmatic Configuration:

Use the following `GlobalConfiguration` code to specify the name of the file to use for JGroups configuration:

```
new
GlobalConfigurationBuilder().transport().addProperty("configurationFile",
"jgroups.xml")
    .build()
```

Replace `jgroups.xml` with the desired file name.

The `jgroups.xml` file is located at `Infinispan-Quickstart/clustered-cache/src/main/resources/`.



NOTE

To bind JGroups solely to your loopback interface (to avoid any configured firewalls), use the system property `-Djgroups.bind_addr="127.0.0.1"`. This is particularly useful to test a cluster where all nodes are on a single machine.

Declarative Configuration:

Use the following XML snippet in the `infinispan.xml` file to configure the JGroups properties to use Red Hat JBoss Data Grid's XML configuration:

```
<global>
  <transport>
    <properties>
      <property name="configurationFile" value="jgroups.xml"/>
    </properties>
  </transport>
</global>
```

[Report a bug](#)

11.2.4.3. Configure the Replicated Data Grid

Red Hat JBoss Data Grid's replicated mode ensures that every entry is replicated on every node in the data grid.

This mode offers security against data loss due to node failures and excellent data availability. These benefits are at the cost of limiting the storage capacity to the amount of storage available on the node with the least memory.

Programmatic Configuration:

Use the following code snippet to programmatically configure the cache for replication mode (either synchronous or asynchronous):

```
private static EmbeddedCacheManager createCacheManagerProgramatically() {
    return new DefaultCacheManager(
        new GlobalConfigurationBuilder()
            .transport().addProperty("configurationFile", "jgroups.xml")
            .build(),
        new ConfigurationBuilder()
            .clustering().cacheMode(CacheMode.REPL_SYNC)
            .build()
    );
}
```

Declarative Configuration:

Edit the `infinispan.xml` file to include the following XML code to declaratively configure the cache for replication mode (either synchronous or asynchronous):

```
<infinispan xsi:schemaLocation="urn:infinispan:config:8.3
http://www.infinispan.org/schemas/infinispan-config-8.3.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:infinispan:config:8.3">
  <global>
    <transport>
      <properties>
        <property name="configurationFile" value="jgroups.xml"/>
      </properties>
    </transport>
  </global>
  <default>
    <clustering mode="replication">
```

```

        <sync/>
    </clustering>
</default>
</infinispan>

```

Use the following code to initialize and return a DefaultCacheManager with the XML configuration file:

```

private static EmbeddedCacheManager createCacheManagerFromXml() throws
IOException {
    return new DefaultCacheManager("infinispan.xml");}

```

NOTE

JBoss EAP includes its own underlying JMX. This can cause a collision when using the sample code with JBoss EAP and display an error such as **org.infinispan.jmx.JmxDomainConflictException: Domain already registered org.infinispan.**

To avoid this, configure global configuration as follows:

```

GlobalConfiguration glob = new GlobalConfigurationBuilder()
    .clusteredDefault()
    .globalJmxStatistics()
    .allowDuplicateDomains(true)
    .enable()
    .build();

```

[Report a bug](#)

11.2.4.4. Configure the Distributed Data Grid

Red Hat JBoss Data Grid's distributed mode ensures that each entry is stored on a subset of the total nodes in the data grid. The number of nodes in the subset is controlled by the **numOwners** parameter, which sets how many owners each entry has.

Distributed mode offers increased storage capacity but also results in increased access times and less durability (protection against node failures). Adjust the **numOwners** value to set the desired trade off between space, durability and availability. Durability is further improved by JBoss Data Grid's topology aware consistent hash, which locates entry owners across a variety of data centers, racks and nodes.

Programmatic Configuration:

Programmatically configure the cache for distributed mode (either synchronous or asynchronous) as follows:

```

new ConfigurationBuilder()
    .clustering()
    .cacheMode(CacheMode.DIST_SYNC)
    .hash().numOwners(2)
    .build()

```

Declarative Configuration:

Edit the `infinispan.xml` file to include the following XML code to declaratively configure the cache for distributed mode (either synchronous or asynchronous):

```
<default>
  <clustering mode="distribution">
    <sync/>
    <hash numOwners="2"/>
  </clustering>
</default>
```

[Report a bug](#)

CHAPTER 12. MONITOR RED HAT JBOSS DATA GRID APPLICATIONS IN RED HAT JBOSS EAP

Red Hat JBoss Data Grid library applications (in the form of WAR or EAR files) can be deployed within JBoss Enterprise Application Server 6 (or better) and then monitored using JBoss Operations Network.

[Report a bug](#)

12.1. PREREQUISITES

The following are prerequisites to monitor a Red Hat JBoss Data Grid library application in JBoss Enterprise Application Platform:

- Install and configure JBoss Enterprise Application Platform 6 (or better).
- Install and configure JBoss Operations Network 3.2.2 (or better).
- Install and configure JBoss Data Grid (6.3 or better) Library mode plug-in.

[Report a bug](#)

12.2. MONITOR RED HAT JBOSS DATA GRID APPLICATIONS IN RED HAT JBOSS EAP

Ensure that all requirements outlined as prerequisites are met. Follow the listed steps to monitor Red Hat JBoss Data Grid applications in JBoss Enterprise Application Platform using JBoss Operations Network or RHQ.

Procedure 12.1. Monitor JBoss Data Grid Applications in JBoss Enterprise Application Platform

1. Configure RHQ/JBoss Operations Network

Add an RHQ/JBoss Operations Network specific property (named *org.rhq.resourceKey*) to the `/bin/standalone.conf` file as follows:

```
JAVA_OPTS="$JAVA_OPTS -Dorg.rhq.resourceKey=MyEAP"
```

This command adds the property to the JBoss Enterprise Application Platform's command line indirectly.

2. Check RHQ/JBoss Operations Network is Running Using a Full JDK

Ensure that the RHQ/JBoss Operations Network agent started using a full JDK instead of a JRE. This is because the agent requires access to the JDK's `tools.jar` file.

To configure your RHQ/JBoss Operations Network agent to use the JDK, follow the instructions relevant to your operating system:

- a. For Linux users, set the *RHQ_AGENT_JAVA_HOME* environment variable to the JDK home directory in the agent's `rhq-agent-env.sh` file.
- b. For Windows users, set the *RHQ_AGENT_JAVA_HOME* environment variable to the JDK home directory in the agent's `rhq-agent-env.bat` file.

3. **Ensure the Agent is Local to the JBoss Enterprise Application Platform Instance**

Ensure that the RHQ/JBoss Operations Network agent runs locally to and under the same user as the JBoss Application Platform instance. This is required for the Java Attach API to connect to the process.

4. **Import Resources to the Agent Inventory**

RHQ/JBoss Operations Network can now discover resources. These resources can subsequently be imported into the agent inventory.

When a JBoss Data Grid user deployment enables JMX statistics to expose JBoss Data Grid Cache Managers or caches, the resources appear as children resources of the JBoss Enterprise Application Platform instance.

[Report a bug](#)

PART V. SET UP A CACHE MANAGER

CHAPTER 13. CACHE MANAGERS

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

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

- it creates cache instances on demand.
- it retrieves existing cache instances (i.e. caches that have already been created).

[Report a bug](#)

13.1. TYPES OF CACHE MANAGERS

Red Hat JBoss Data Grid offers the following Cache Managers:

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

[Report a bug](#)

13.2. CREATING CACHEMANAGERS

13.2.1. Create a New RemoteCacheManager

Procedure 13.1. Configure a New RemoteCacheManager

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

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

1. Use the **ConfigurationBuilder()** constructor to create a new configuration builder. The **.addServer()** method adds a remote server, configured via the **.host(<hostname|ip>)** and **.port(<port>)** properties.
2. Create a new **RemoteCacheManager** using the supplied configuration.
3. Retrieve the default cache from the remote server.

[Report a bug](#)

13.2.2. Create a New Embedded Cache Manager

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

Procedure 13.2. Create a New Embedded Cache Manager

1. Create a configuration XML file. For example, create the `my-config-file.xml` file on the classpath (in the `resources/` folder) and add the configuration information in this file.
2. Use the following programmatic configuration to create a cache manager using the configuration file:

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

The outlined procedure creates a new `EmbeddedCacheManager` using the configuration specified in `my-config-file.xml`.

[Report a bug](#)

13.2.3. Create a New Embedded Cache Manager Using CDI

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

Procedure 13.3. Use CDI to Create a New EmbeddedCacheManager

1. Specify a default configuration:

```
public class Config
    @Produces
    public EmbeddedCacheManager defaultCacheManager() {
        ConfigurationBuilder builder = new ConfigurationBuilder();
        Configuration configuration =
builder.eviction().strategy(EvictionStrategy.LRU).maxEntries(100).build();
        return new DefaultCacheManager(configuration);
    }
}
```

2. Inject the default cache manager.

```
<!-- Additional configuration information here -->
@Inject
EmbeddedCacheManager cacheManager;
<!-- Additional configuration information here -->
```

[Report a bug](#)

13.3. MULTIPLE CACHE MANAGERS

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

[Report a bug](#)

13.3.1. Create Multiple Caches with a Single Cache Manager

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

[Report a bug](#)

13.3.2. Using Multiple Cache Managers

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

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

[Report a bug](#)

13.3.3. Create Multiple Cache Managers

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

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

[Report a bug](#)

PART VI. RED HAT JBOSS DATA GRID QUICKSTARTS

The following lists the quickstarts included in this document and provides information about which container and mode they are used in:

Table 14. Quickstarts Information

Quickstart Name	Container	JBoss Data Grid Mode	Link to Details
Hello World	JBoss EAP	Library mode	Chapter 14, <i>The Hello World Quickstart</i>
Carmart Non-Transactional	JBoss EAP and JBoss Enterprise Web Server	Library mode	Section 15.3, “The (Non-transactional) CarMart Quickstart Using JBoss EAP”
Carmart Non-Transactional	JBoss EAP and JBoss Enterprise Web Server	Remote Client-Server mode	Section 15.5, “The (Non-transactional) CarMart Quickstart in Remote Client-Server Mode (JBoss EAP)” and Section 15.6, “The (Non-Transactional) CarMart Quickstart in Remote Client-Server Mode (JBoss Enterprise Web Server)”
Carmart Transactional	JBoss EAP and JBoss Enterprise Web Server	Library mode	Section 15.7, “The (Transactional) CarMart Quickstart Using JBoss EAP” and Section 15.8, “The (Transactional) CarMart Quickstart Using JBoss Enterprise Web Server”
Football Application	No container	Remote Client-Server mode	Chapter 16, <i>The Football Quickstart Endpoint Examples</i>
Rapid Stock Market	No container	Remote Client-Server mode	Chapter 17, <i>The Rapid Stock Market Quickstart</i>

Quickstart Name	Container	JBoss Data Grid Mode	Link to Details
Cluster App	JBoss EAP	Library mode	Chapter 18, <i>The Cluster App Quickstart</i>
camel-jbossgdatagrid-fuse	JBoss Fuse	Library mode	Chapter 19, <i>The camel-jbossgdatagrid-fuse Quickstart</i>

[Report a bug](#)

CHAPTER 14. THE HELLO WORLD QUICKSTART

Hello World is a simple quickstart that illustrates how to store and retrieve data from a cache using Red Hat JBoss Data Grid. For this quickstart, users can access the cache in two ways:

- from a servlet.
- from a JSF page using request scoped beans.

All libraries (JAR files) bundles with the application are deployed to JBoss Enterprise Application Platform 7.x. JBoss Data Grid's Library mode only allows local access to a single node in a distributed cluster. This mode also allows the application to access the data grid functionality within a virtual machine in the target container.



IMPORTANT

The Hello World quickstart works only in JBoss Data Grid's Library mode.

Location

JBoss Data Grid's Hello World quickstart is available at the following location: **jboss-datagrid-{VERSION}-quickstarts/**

[Report a bug](#)

14.1. QUICKSTART PREREQUISITES

The prerequisites for this quickstart are as follows:

- Java 8.0 (Java SDK 1.8) or better
- JBoss Enterprise Application Platform 7.x or JBoss Enterprise Web Server 3.x
- Maven 3.0 or better
- Configure the Maven Repository. For details, see [Chapter 3, Install and Use the Maven Repositories](#)

[Report a bug](#)

14.2. START TWO APPLICATION SERVER INSTANCES

Before deploying the Hello World quickstart, start two instances of your application server (JBoss Enterprise Application Platform 7.x).

Procedure 14.1. Start the First Application Server Instance

1. Navigate to the Root Directory

In the command line terminal, navigate to the root for your JBoss server directory.

2. Start the First Application Server

Depending on your operating system, use the appropriate command from the following to start the first instance of your selected application server:

- a. For Linux:

```
$JBOSS_HOME/bin/standalone.sh
```

- b. For Windows:

```
$JBOSS_HOME\bin\standalone.bat
```

Procedure 14.2. Start the Second Application Server Instance

1. **Clone the Application Server**

Create a copy of the selected JBoss Server to create a second instance.

2. **Navigate to the Root Directory**

In the command line terminal, navigate to the root for your JBoss server directory.

3. **Start the Second Application Server**

Use the appropriate command for your operating system from the following commands. This command starts the server with the provided port offset to ensure that both the server instances run on the same host.

- a. For Linux:

```
$JBOSS_HOME2/bin/standalone.sh -Djboss.socket.binding.port-  
offset=100
```

- b. For Windows:

```
$JBOSS_HOME2\bin\standalone.bat -Djboss.socket.binding.port-  
offset=100
```

[Report a bug](#)

14.3. BUILD AND DEPLOY THE HELLO WORLD QUICKSTART

Before building and deploying the quickstart, ensure that all the listed prerequisites are met and that the two application server instances are running (see [Section 14.2, “Start Two Application Server Instances”](#) for details).

Procedure 14.3. Build and Deploy the Hello World Quickstart

1. **Navigate to the Required Directory**

In the command line terminal, navigate to the root directory of the quickstart on the command line interface.

2. **Build and Deploy to the First Application Server Instance**

Use the following command to build and deploy the quickstart to the first application server instance as follows:

```
# mvn clean package wildfly:deploy
```

This command deploys **target/jboss-helloworld-jdg.war** to the first running server instance.

3. **Build and Deploy to the Second Application Server Instance**

Use the following command to build and deploy the quickstart to the second application server instance with the specified ports as follows:

```
# mvn clean package wildfly:deploy -Dwildfly.port=10090
```

This command deploys **target/jboss-helloworld-jdg.war** to the second running server instance.

[Report a bug](#)

14.4. ACCESS THE RUNNING APPLICATION

The Hello World quickstart application runs on the following URLs:

- First Server Instance: <http://localhost:8080/jboss-helloworld-jdg>
- Second Server Instance: <http://localhost:8180/jboss-helloworld-jdg>

[Report a bug](#)

14.5. TEST REPLICATION ON THE APPLICATION

Use the following instructions to test that cache entries are replicating from the first server instance to the second as desired.

Procedure 14.4. Test Replication on the Application

1. **Access the First Server**

Access the first application server and enter the key and value.

- a. Access the first application server in a browser window using the following URL:

```
http://localhost:8080/jboss-helloworld-jdg
```

- b. Insert the key **foo**.
- c. Insert the value **bar**.

2. **Access the Second Server**

Access the second application server and enter the key and value.

- a. Access the second application server in a browser window using the following URL:

```
http://localhost:8180/jboss-helloworld-jdg
```

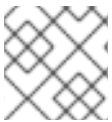
- b. Click **Get Some**.
- c. Get the key **foo**.

- d. Click **Put Some More**.
 - e. Insert the key **mykey**.
 - f. Insert the value **myvalue**.
3. **Get All Keys and Values**
Access the first server and request all keys.
- a. Access the first application server in a browser window using the following URL:


```
http://localhost:8080/jboss-helloworld-jdg
```
 - b. Click **Get Some**.
 - c. Click **Get All** to request all key and values.

Result

As the results of the last step show, all the data added at each server has been replicated to the other server.



NOTE

Entries expire after **60** seconds from the most recent update.

Directly Access Keys in the Cache

To interact with predefined servlets or to directly store and retrieve keys from the cache, use the following URLs:

```
http://localhost:8080/jboss-helloworld-jdg/TestServletPut
```

```
http://localhost:8180/jboss-helloworld-jdg/TestServletGet
```

[Report a bug](#)

14.6. REMOVE THE APPLICATION

Use the following procedure to remove the Hello World application:

Procedure 14.5. Remove the Application

1. **Start the Application Servers**
Ensure that both server instances are running.
2. **Navigate to the Root**
In the command line terminal, navigate to the root directory of the quickstart.
3. **Remove the Archive**
Use the following commands to remove the archive from both the server instances.
 - a. Remove the archive from the first server as follows:

```
mvn wildfly:undeploy
```

- b. Remove the archive from the second server as follows:

```
mvn wildfly:undeploy -Dwildfly.port=10090
```

[Report a bug](#)

CHAPTER 15. THE CARMART QUICKSTARTS

Red Hat JBoss Data Grid includes a transactional and non-transactional CarMart quickstart. The CarMart quickstart is a simple web application that uses JBoss Data Grid instead of a relational database. Information about each car is stored in a cache. Caches are configured declaratively or programmatically depending on the usage mode.

Features

The CarMart quickstart offers the following features:

- List all cars
- Add new cars
- Remove cars
- View statistics for caches, such as hits, stores, and retrievals

Usage Modes

The CarMart quickstart can be used in the following JBoss Data Grid usage modes:

- Remote Client-Server Mode, where the application includes the Hot Rod client to communicate with a remote JBoss Data Grid server.
- Library Mode, where all libraries are bundled with the application in the form of **jar** files.

Location

JBoss Data Grid's CarMart quickstart is available at the following location: **jboss-datagrid-{VERSION}-quickstarts/**

[Report a bug](#)

15.1. ABOUT THE CARMART TRANSACTIONAL QUICKSTART

The transactional version of the CarMart quickstart is a simple web application that uses Red Hat JBoss Data Grid instead of a relational database. Information about each car is stored in a cache. Caches are configured declaratively or programmatically (depending on the usage mode) and run in the same Java Virtual Machine (JVM) as the web application.

Features

The Transactional CarMart Quickstart offers the following features:

- List all cars
- Add new cars
- Add new cars with rollback
- Remove cars
- View statistics for caches, such as hits, stores, and retrievals

Usage Modes

The Transactional CarMart Quickstart can only be used in JBoss Data Grid's Library mode. A standalone transaction manager from JBoss Transactions is used when the Transactional CarMart Quickstart is run in Red Hat JBoss Enterprise Web Server 2.x.

Location

JBoss Data Grid's Transactional CarMart Quickstart can be found at the following location:
`jboss-datagrid-{VERSION}-quickstarts/carmart-tx`

[Report a bug](#)

15.2. DIFFERENCES BETWEEN THE CARMART AND TRANSACTIONAL QUICKSTARTS

Despite the similarity in steps to build, deploy and remove the transactional and non-transactional CarMart quickstarts, some differences must be noted. The following is a list of such differences:

- CarMart is available for both Remote Client-Server Mode and Library Mode. Transactional CarMart is only available in Library Mode because transactions are not available in Remote Client-Server Mode.
- The Transactional Quickstart also displays how a transaction rollback occurs. Use the **Add car with rollback** button to view the rollback. The CarMart example has a simple **Add car** button instead.

[Report a bug](#)

15.3. THE (NON-TRANSACTIONAL) CARMART QUICKSTART USING JBOSS EAP

The Carmart (non-transactional) quickstart is supported for JBoss Data Grid's Library Mode with the JBoss EAP container.

[Report a bug](#)

15.3.1. Quickstart Prerequisites

The prerequisites for this quickstart are as follows:

- Java 8.0 (Java SDK 1.8) or better
- JBoss Enterprise Application Platform 7.x or JBoss Enterprise Web Server 3.x
- Maven 3.0 or better
- Configure the Maven Repository. For details, see [Chapter 3, Install and Use the Maven Repositories](#)

[Report a bug](#)

15.3.2. Build and Deploy the CarMart Quickstart to JBoss EAP

The following procedure provides directions to build and deploy the CarMart application to JBoss EAP.

Prerequisites

Prerequisites for this procedure are as follows:

1. Obtain the supported JBoss Data Grid Library Mode distribution files.
2. Ensure that the JBoss Data Grid and JBoss Enterprise Application Platform Maven repositories are installed and configured. For details, see [Chapter 3, *Install and Use the Maven Repositories*](#)
3. Select a JBoss server to use (JBoss Enterprise Application Platform 7 or better).

Procedure 15.1. Build and Deploy CarMart to JBoss EAP

1. Start JBoss EAP

Depending on your operating system, use the appropriate command from the following to start the first instance of your selected application server:

For Linux users:

```
$JBOSS_HOME/bin/standalone.sh
```

For Windows users:

```
$JBOSS_HOME\bin\standalone.bat
```

2. Navigate to the Root Directory

Open a command line and navigate to the root directory of this quickstart.

3. Build and Deploy the Application

Use the following command to build and deploy the application using Maven:

```
$ mvn clean package wildfly:deploy
```

Result

The target war file (**target/jboss-carmart.war**) is deployed to the running instance of JBoss EAP.

[Report a bug](#)

15.3.3. View the CarMart Quickstart on JBoss EAP

The following procedure outlines how to view the CarMart quickstart on JBoss EAP:

Prerequisite

The CarMart quickstart must be built and deployed to be viewed.

Procedure 15.2. View the CarMart Quickstart on JBoss EAP

- To view the application, use your browser to navigate to the following link:

```
http://localhost:8080/jboss-carmart
```

[Report a bug](#)

15.3.4. Remove the CarMart Quickstart from JBoss EAP

The following procedure provides directions to remove a deployed application from JBoss EAP.

Procedure 15.3. Remove an Application from JBoss EAP

- To remove an application, use the following from the root directory of this quickstart:

```
$ mvn wildfly:undeploy
```

[Report a bug](#)

15.4. THE (NON-TRANSACTIONAL) CARMART QUICKSTART USING JBOSS ENTERPRISE WEB SERVER

The Carmart (non-transactional) quickstart is supported for JBoss Data Grid's Library Mode with the JBoss Enterprise Web Server container.

[Report a bug](#)

15.4.1. Build and Deploy the CarMart Quickstart to JBoss Enterprise Web Server

The following procedure provides directions to build and deploy the CarMart quickstart to the JBoss Enterprise Web Server.

Prerequisites

Prerequisites for this procedure are as follows:

1. Ensure that the JBoss Data Grid and JBoss Enterprise Application Platform Maven repositories are installed and configured. For details, see [Chapter 3, Install and Use the Maven Repositories](#)
2. Select JBoss Enterprise Web Server 3 (or better) for your application and install it.

Procedure 15.4. Build the CarMart Quickstart to the Server (Library Mode)

1. Start the Server

Depending on your operating system, use the appropriate command from the following to start the first instance of your selected application server:

For Linux users:

```
$JBOSS_EWS_HOME/tomcat8/bin/catalina.sh run
```

For Windows users:

```
$JBOSS_EWS_HOME\tomcat8\bin\catalina.bat run
```

2. Navigate to the Root Directory

Open a command line and navigate to the root directory of this quickstart.

3. Build and Deploy your Application

Use the following command to build and deploy your application using Maven:

```
$ mvn -Plibrary-tomcat clean package tomcat:deploy
```

Result

The target war file (**target/jboss-carmart.war**) is deployed to the running instance of the selected server.

[Report a bug](#)

15.4.2. View the CarMart Quickstart Using JBoss Enterprise Web Server

The following procedure outlines how to view the CarMart quickstart on the JBoss Enterprise Web Server:

Prerequisite

The CarMart quickstart must be built and deployed to be viewed.

Procedure 15.5. View the CarMart Quickstart

- To view the application, use your browser to navigate to the following link:

```
http://localhost:8080/jboss-carmart
```

[Report a bug](#)

15.4.3. Remove the CarMart Quickstart from JBoss Enterprise Web Server

The following procedure provides directions to remove an already deployed application from JBoss Enterprise Web Server.

Procedure 15.6. Remove an Application from JBoss Enterprise Web Server

- To remove an application, use the following command from the root directory of this quickstart:

```
$ mvn -Plibrary-tomcat tomcat:undeploy
```

[Report a bug](#)

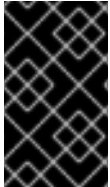
15.5. THE (NON-TRANSACTIONAL) CARMART QUICKSTART IN REMOTE CLIENT-SERVER MODE (JBoss EAP)

The Carmart (non-transactional) quickstart is supported for JBoss Data Grid's Remote Client-Server Mode with the JBoss Enterprise Application Platform container.

[Report a bug](#)

15.5.1. Build and Deploy the CarMart Quickstart in Remote Client-Server Mode

This quickstart accesses Red Hat JBoss Data Grid via Hot Rod. This feature is not available for the Transactional CarMart quickstart.



IMPORTANT

This quickstart deploys to JBoss Enterprise Application Platform. The application cannot be deployed to JBoss Data Grid because it does not support application deployment.

Prerequisites

Prerequisites for this procedure are as follows:

1. Obtain the most recent supported JBoss Data Grid Remote Client-Server Mode distribution files from [Red Hat](#).
2. Ensure that the JBoss Data Grid and JBoss Enterprise Application Platform Maven repositories are installed and configured. For details, see [Chapter 3, Install and Use the Maven Repositories](#)
3. Select a JBoss server to use (JBoss Enterprise Application Platform 7 or later). Navigate to the root of the JBoss server directory in a terminal window and enter the following command:

For Linux users:

```
$JBOSS_HOME/bin/standalone.sh
```

For Windows users:

```
$JBOSS_HOME\bin\standalone.bat
```

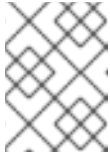
Procedure 15.7. Build and Deploy the CarMart Quickstart in Remote Client-Server Mode

1. Configure the Standalone File

Add the following configuration to the `standalone.xml` file located in the `$JDG_HOME/standalone/configuration/` directory.

- a. Add the following configuration within the `infinispan` subsystem tags, before the ending `cache-container` tag:

```
<local-cache name="carcache"
  start="EAGER"
  batching="false"
  statistics="true">
  <eviction strategy="LIRS"
    max-entries="4"/>
</local-cache>
```

**NOTE**

If the *carcache* element already exists in your configuration, replace it with the provided configuration.

2. Start the JBoss Data Grid Server

Run the following script to start the JBoss Data Grid Server:

```
$JDG_HOME/bin/standalone.sh -Djboss.socket.binding.port-offset=100
```

3. Start the JBoss Server

Run the following script to start the JBoss server instance where your application will deploy:

```
$JBOSS_HOME/bin/standalone.sh
```

4. Optional: Specify the Host and Port Address

The application uses the values in the `jboss-datagrid-{VERSION}-quickstarts/carmart/src/main/resources/META-INF/datagrid.properties` file to locate the JBoss Data Grid server. If your JBoss Data Grid server is not running using the default host and port values, edit the file and insert the correct host and port values, as follows:

```
datagrid.host=localhost
datagrid.hotrod.port=11322
```

5. Navigate to the Root Directory

Open a command line and navigate to the root directory of this quickstart.

6. Build and Deploy the Application

Use the following command to build and deploy your application using Maven:

```
$ mvn clean package wildfly:deploy -PreMOTE-eap
```

[Report a bug](#)

15.5.2. View the CarMart Quickstart in Remote Client-Server Mode

The following procedure outlines how to view the CarMart quickstart in Red Hat JBoss Data Grid's Remote Client-Server Mode:

Prerequisite

The CarMart quickstart must be built and deployed before viewed.

Procedure 15.8. View the CarMart Quickstart in Remote Client-Server Mode

- Visit the following link in a browser window to view the application:

```
http://localhost:8080/jboss-carmart
```

[Report a bug](#)

15.5.3. Remove the CarMart Quickstart in Remote Client-Server Mode

The following procedure provides directions to remove an already deployed application in Red Hat JBoss Data Grid's Remote Client-Server mode.

Procedure 15.9. Remove an Application in Remote Client-Server Mode

- To remove an application, use the following command from the root directory of this quickstart:

```
$ mvn wildfly:undeploy -PreMOTE-eap
```

[Report a bug](#)

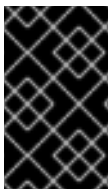
15.6. THE (NON-TRANSACTIONAL) CARMART QUICKSTART IN REMOTE CLIENT-SERVER MODE (JBOSS ENTERPRISE WEB SERVER)

The Carmart (non-transactional) quickstart is supported for JBoss Data Grid's Remote Client-Server Mode with the JBoss Enterprise Web Server container.

[Report a bug](#)

15.6.1. Build and Deploy the CarMart Quickstart in Remote Client-Server Mode

This quickstart accesses Red Hat JBoss Data Grid via Hot Rod. This feature is not available for the Transactional CarMart quickstart.



IMPORTANT

This quickstart deploys to JBoss Enterprise Web Server or Tomcat. The application cannot be deployed to JBoss Data Grid because it does not support application deployment.

Prerequisites

Prerequisites for this procedure are as follows:

1. Obtain the most recent supported JBoss Data Grid Remote Client-Server Mode distribution files from [Red Hat](#).
2. Ensure that the JBoss Data Grid and JBoss Enterprise Application Platform Maven repositories are installed and configured. For details, see [Chapter 3, Install and Use the Maven Repositories](#)
3. Add a **server** element to the `MavenSettings.xml` file. In the *id* elements within **server**, add the appropriate tomcat credentials.

```
<server>
  <id>tomcat</id>
  <username>admin</username>
```



```
<password>admin</password>
</server>
```

Procedure 15.10. Build and Deploy the CarMart Quickstart in Remote Client-Server Mode

1. Configure the Standalone File

Add the following configuration to the `standalone.xml` file located in the `$JDG_HOME/standalone/configuration/` directory.

- a. Add the following configuration within the `infinispan` subsystem tags:

```
<local-cache name="carcache"
  start="EAGER"
  batching="false"
  statistics="true">
  <eviction strategy="LIRS"
    max-entries="4"/>
</local-cache>
```



NOTE

If the `carcache` element already exists in your configuration, replace it with the provided configuration.

2. Start the Container

Start the JBoss server instance where your application will deploy.

For Linux:

```
$JBOSS_EWS_HOME/tomcat8/bin/catalina.sh run
```

For Windows:

```
$JBOSS_EWS_HOME\tomcat8\bin\catalina.bat run
```

3. Build the Application

Use the following command to build your application in the relevant directory:

```
$ mvn clean package -Premote-tomcat
```

4. Deploy the Application

Use the following command to deploy the application in the relevant directory:

```
mvn tomcat:deploy -Premote-tomcat
```

[Report a bug](#)

15.6.2. View the CarMart Quickstart in Remote Client-Server Mode

The following procedure outlines how to view the CarMart quickstart in Red Hat JBoss Data Grid's Remote Client-Server Mode:

Prerequisite

The CarMart quickstart must be built and deployed be viewed.

Procedure 15.11. View the CarMart Quickstart in Remote Client-Server Mode

- Visit the following link in a browser window to view the application:

```
http://localhost:8080/jboss-carmart
```

[Report a bug](#)

15.6.3. Remove the CarMart Quickstart in Remote Client-Server Mode

The following procedure provides directions to remove an already deployed application in Red Hat JBoss Data Grid's Remote Client-Server mode.

Procedure 15.12. Remove an Application in Remote Client-Server Mode

- To remove an application, use the following command from the root directory of this quickstart:

```
$ mvn tomcat:undeploy -Premote-tomcat
```

[Report a bug](#)

15.7. THE (TRANSACTIONAL) CARMART QUICKSTART USING JBOSS EAP

This CarMart Transactional quickstart requires JBoss Data Grid's Library mode with the JBoss Enterprise Application Platform container.

All required libraries (jar files) are bundled with the application and deployed to the server. Caches are configured programmatically and run in the same JVM as the web application.

All operations are transactional and are configured at **JBossASCacheContainerProvider/TomcatCacheContainerProvider** implementation classes for the **CacheContainerProvider** interface.

[Report a bug](#)

15.7.1. Quickstart Prerequisites

The prerequisites for this quickstart are as follows:

- Java 8.0 (Java SDK 1.8) or better
- JBoss Enterprise Application Platform 7.x or JBoss Enterprise Web Server 3.x
- Maven 3.0 or better
- Configure the Maven Repository. For details, see [Chapter 3, Install and Use the Maven Repositories](#)

[Report a bug](#)

15.7.2. Build and Deploy the Transactional CarMart Quickstart

Prerequisites

Ensure that the following prerequisites are met before building and deploying the CarMart quickstart.

1. Configure Maven (See [Section 15.7.1, “Quickstart Prerequisites”](#))
2. Start JBoss Enterprise Application Platform:
 1. In a command line terminal, navigate to the root of the JBoss EAP server directory.
 2. Use one of the following commands to start the server with a web profile:

For Linux:

```
$JBOSS_HOME/bin/standalone.sh
```

For Windows:

```
%JBOSS_HOME%\bin\standalone.bat
```

Procedure 15.13. Build and Deploy the Transactional Quickstart

1. Open a command line and navigate to the root directory of this quickstart.
2. Enter the following command to build and deploy the archive:

```
mvn clean package wildfly:deploy
```

3. The **target/jboss-carmart-tx.war** file is deployed to the running instance of the server.

[Report a bug](#)

15.7.3. View the Transactional CarMart Quickstart

The following procedure outlines how to view the CarMart quickstart:

Prerequisite

The CarMart quickstart must be built and deployed to be viewed.

Procedure 15.14. View the CarMart Quickstart

- To view the application, use your browser to navigate to the following link:

```
http://localhost:8080/jboss-carmart-tx
```

[Report a bug](#)

15.7.4. Undeploy The Transactional CarMart Quickstart

Undeploy the transactional CarMart quickstart as follows:

1. In a command line terminal, navigate to the root directory of the quickstart.
2. Undeploy the archive as follows:

```
mvn wildfly:undeploy
```

[Report a bug](#)

15.7.5. Test the Transactional CarMart Quickstart

The JBoss Data Grid quickstarts include Arquillian Selenium tests. To run these tests:

1. Stop JBoss EAP, if it is running.
2. In a command line terminal, navigate to root directory for the quickstart.
3. Build the quickstarts as follows:

```
mvn clean package
```

4. Run the tests as follows:

```
mvn test -Puitests-eap -Das7home=/path/to/server
```

[Report a bug](#)

15.8. THE (TRANSACTIONAL) CARMART QUICKSTART USING JBOSS ENTERPRISE WEB SERVER

This CarMart Transactional quickstart requires JBoss Data Grid's Library mode with the JBoss Enterprise Web Server container.

All the required libraries (jar files) are bundled with the application and deployed to the server. Caches are configured programatically and run in the same JVM as the web application for this quickstart.

Additionally, all operations are transactional. The standalone transaction manager from JBoss Transactions is used to run the application in JBoss Enterprise Web Server.

Only the **library-tomcat** profile can be used when running this quickstart with JBoss Enterprise Web Server because this profile enables library mode.

[Report a bug](#)

15.8.1. Quickstart Prerequisites

The prerequisites for this quickstart are as follows:

- Java 8.0 (Java SDK 1.8) or better

- JBoss Enterprise Application Platform 7.x or JBoss Enterprise Web Server 3.x
- Maven 3.0 or better
- Configure the Maven Repository. For details, see [Chapter 3, *Install and Use the Maven Repositories*](#)

[Report a bug](#)

15.8.2. Build and Deploy the Transactional CarMart Quickstart

Prerequisites

Ensure that the following prerequisites are met before building and deploying the CarMart quickstart.

1. Configure Maven (See [Section 15.8.1, “Quickstart Prerequisites”](#))
2. To configure JBoss Enterprise Web Server, add the following lines to the `conf/tomcat-users.xml` file:

```
<role rolename="manager-script"/>
  <user username="admin" password="admin" roles="manager-script"/>
```

3. Configure Maven by adding the following configuration information to the Maven `settings.xml` file with the appropriate credentials:

```
<server>
  <id>tomcat</id>
  <username>admin</username>
  <password>admin</password>
</server>
```

4. Start JBoss Enterprise Web Server:
 1. In a command line terminal, navigate to the root of the JBoss Enterprise Web Server directory.
 2. Use one of the following commands to start the server with a web profile:

For Linux:

```
$TOMCAT_HOME/bin/catalina.sh run
```

For Windows:

```
%TOMCAT_HOME%\bin\catalina.bat run
```

Procedure 15.15. Build and Deploy the Transactional CarMart Quickstart

1. In a command line terminal, navigate to the root directory for the quickstart.
2. Enter the following command to build and deploy archive:

```
mvn -Plibrary-tomcat clean package tomcat:deploy
```

3. The **target/jboss-carmart-tx.war** file is deployed to the running instance of JBoss Enterprise Web Server.

[Report a bug](#)

15.8.3. View the Transactional CarMart Quickstart

The following procedure outlines how to view the CarMart quickstart:

Prerequisite

The CarMart quickstart must be built and deployed to be viewed.

Procedure 15.16. View the CarMart Quickstart

- To view the application, use your browser to navigate to the following link:

```
http://localhost:8080/jboss-carmart-tx
```

[Report a bug](#)

15.8.4. Undeploy The Transactional CarMart Quickstart

Undeploy the transactional CarMart quickstart as follows:

1. In a command line terminal, navigate to the root directory of the quickstart.
2. Undeploy the archive as follows:

```
mvn -Plibrary-tomcat tomcat:undeploy
```

[Report a bug](#)

15.8.5. Test the Transactional CarMart Quickstart

The JBoss Data Grid quickstarts include Arquillian Selenium tests. To run these tests:

1. Undeploy the archive (see [Section 15.8.4, “Undeploy The Transactional CarMart Quickstart”](#))
2. Stop JBoss Enterprise Web Server, if it is running.
3. In a command line terminal, navigate to root directory for the quickstart.
4. Build the quickstarts as follows:

```
mvn clean package
```

5. Run the tests as follows:

```
mvn test -Puitests-tomcat -Das7home=/path/to/server
```

[Report a bug](#)

CHAPTER 16. THE FOOTBALL QUICKSTART ENDPOINT EXAMPLES

The Football application is a simple example to illustrate the use of Red Hat JBoss Data Grid endpoints, namely Hot Rod, REST, and Memcached. Each example shows one of these protocols used to connect to JBoss Data Grid to remotely store, retrieve, and remove data from caches.

Each application is a variation of a simple football team manager utility as a console application.

Features

The following features are available with the example Football Manager application:

- Add a team
- Add players
- Remove all entities (teams and players)
- Listing all teams and players

Location

JBoss Data Grid's Football quickstart can be found at the following locations:

- `jboss-datagrid-{VERSION}-quickstarts/rest-endpoint`
- `jboss-datagrid-{VERSION}-quickstarts/hotrod-endpoint`
- `jboss-datagrid-{VERSION}-quickstarts/memcached-endpoint`

[Report a bug](#)

16.1. QUICKSTART PREREQUISITES

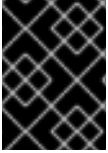
The prerequisites for this quickstart are as follows:

- Java 8.0 (Java SDK 1.8) or better
- JBoss Enterprise Application Platform 7.x or JBoss Enterprise Web Server 3.x
- Maven 3.0 or better
- Configure the Maven Repository. For details, see [Chapter 3, Install and Use the Maven Repositories](#)

[Report a bug](#)

16.2. BUILD THE FOOTBALL APPLICATION

The following procedure outlines the steps to build a football manager application as an example of REST, Hot Rod and memcached endpoints in Red Hat JBoss Data Grid.



IMPORTANT

JBoss Data Grid does not support deploying applications, therefore this quickstart cannot be installed as a deployment.

Prerequisites

Prerequisites for this procedure are as follows:

1. Obtain the most recent supported JBoss Data Grid Remote Client-Server Mode distribution files from [Red Hat](#).
2. Ensure that the JBoss Data Grid and JBoss Enterprise Application Platform Maven repositories are installed and configured. For details, see [Chapter 3, Install and Use the Maven Repositories](#).

Procedure 16.1. Build the Football Application

1. Add Configurations

Edit the `standalone.xml` file (located at `$JDG_HOME/standalone/configuration/`) to add definitions for the datasource and infinispan subsystems.

- a. Add the following subsystem definition for the datasource:

```
<subsystem xmlns="urn:jboss:domain:datasources:4.0">
  <!-- Define this Datasource with jndi
  name java:jboss/datasources/ExampleDS -->
  <datasources>
    <datasource jndi-name="java:jboss/datasources/ExampleDS"
      pool-name="ExampleDS"
      enabled="true"
      use-java-context="true">
      <!-- The connection URL uses H2 Database
      Engine with in-memory database called test -->
      <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1</connection-url>
      <!-- JDBC driver name -->
      <driver>h2</driver>
      <!-- Credentials -->
      <security>
        <user-name>sa</user-name>
        <password>sa</password>
      </security>
    </datasource>
    <!-- Define the JDBC driver called 'h2' -->
    <drivers>
      <driver name="h2" module="com.h2database.h2">
        <xa-datasource-
class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
```

```

        </driver>
    </drivers>

</datasources>
</subsystem>

```

- b. Add the following subsystem definition for infinispn:

```

<subsystem xmlns="urn:infinispn:server:core:8.0"
    default-cache-container="local">

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

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

            <locking isolation="NONE"
                acquire-timeout="30000"
                concurrency-level="1000"
                striping="false"/>

            <transaction mode="NONE"/>

        </local-cache>

        <local-cache name="memcachedCache"
            start="EAGER"
            statistics="true">

            <locking isolation="NONE"
                acquire-timeout="30000"
                concurrency-level="1000"
                striping="false"/>

            <transaction mode="NONE"/>

        </local-cache>

        <local-cache name="namedCache"
            start="EAGER"
            statistics="true"/>

        <!-- ADD a local cache called 'teams' -->

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

            <!-- Disable transactions for this cache -->
            <transaction mode="NONE" />

            <!-- Define the JdbcBinaryStores
            to point to the ExampleDS previously

```

```

defined -->

<string-keyed-jdbc-store
  datasource="java:jboss/datasources/ExampleDS"
  passivation="false"
  preload="false"
  purge="false">

      <!-- Define the database dialect -->
      <property name="databaseType">H2</property>

      <!-- specifies information about
      database table/column names
      and data types -->

      <string-keyed-table prefix="JDG">
          <id-column name="id"
            type="VARCHAR"/>
          <data-column name="datum"
            type="BINARY"/>
          <timestamp-column name="version"
            type="BIGINT"/>
        </string-keyed-table>
      </string-keyed-jdbc-store>

      </local-cache>

      <!-- End of local cache called 'teams' definition -->

</cache-container>
</subsystem>

```



NOTE

The Hot Rod and REST endpoints use the cache named **teams** and memcached endpoint uses **memcachedCache** as a default.

2. Disable REST Security

As a default, the **standalone.xml** configuration file protects the REST endpoint with **BASIC** authentication. This quickstart cannot perform authentication, therefore the REST authentication must be disabled in the REST connector by removing the **security-domain** and **auth-method** parameters. The resulting configuration (with REST authentication disabled) is as follows:

```

<rest-connector virtual-server="default-host"
  cache-container="local" />

```

For more details about security, see the REST Authentication Chapter in JBoss Data Grid's *Developer Guide*.

3. Edit the Submodule Configuration File

Each submodule (specifically **hotrod-endpoint**, **rest-endpoint** and **memcached-endpoint**) contains a configuration file (located at

`$JDG_QUICKSTART/src/main/resources/jdg.properties`). Modify the default values in the configuration file to set the values required for your specific JBoss Data Grid installation.

4. **Navigate to the Root Directory**

Open a command line and navigate to the root directory of this quickstart.

5. **Build the Application**

Use the following command to build the example application in its directory:

```
mvn clean package
```

This step results in the use of Maven's shade plugin, which bundles all dependencies into a single jar file for ease of use. This file is named `{PROTOCOL}-endpoint-quickstart.jar`, for example `rest-endpoint-quickstart.jar` for the REST version.

6. **Start JBoss Data Grid**

Run the following script to run JBoss Data Grid:

```
$JDG_HOME/bin/standalone.sh
```

7. **Run the Application**

Run the example application in its directory with the following command:

```
mvn exec:java
```

[Report a bug](#)

CHAPTER 17. THE RAPID STOCK MARKET QUICKSTART

The Rapid Stock Market quickstart demonstrates how JBoss Data Grid's compatibility mode works with a Hot Rod client (to store data) and a HTTP client using REST (to retrieve data). This quickstart is only available in JBoss Data Grid's Remote Client-Server mode and does not use any containers.

The Rapid Stock Market quickstart includes a server-side and a client-side application.

[Report a bug](#)

17.1. BUILD AND RUN THE RAPID STOCK MARKET QUICKSTART

The Rapid Stock Market quickstart requires the following configuration for the server and client sides of the application.

Procedure 17.1. Rapid Stock Market Quickstart Server-side Configuration

1. **Navigate to the Root Directory**

Open a command line and navigate to the root directory of this quickstart.

2. Build a server module for the JBoss Data Grid Server by packaging a class that is common for the client and server in a jar file:

```
$ mvn clean package -Pprepare-server-module
```

Place the new jar file in a directory structure that is similar to the server module.

3. Install the server module into the server.

- a. Copy the prepared module to the server:

```
$ cp -r target/modules ${JDG_SERVER_HOME}/
```

- b. Add the new module as a dependency of the **org.infinispan.commons** module by adding the following into the **modules/system/layers/base/org/infinispan/commons/main/module.xml** file:

```
<module name="org.infinispan.quickstart.compatibility.common"/>
```

4. Build the application:

```
$ mvn clean package
```

5. Configure the JBoss Data Grid to use the appropriate configuration file.

- a. Copy the example configuration file for compatibility mode to a location where the JBoss Data Grid Server can locate and use it:

```
$ cp ${JDG_SERVER_HOME}/docs/examples/configs/standalone-  
compatibility-mode.xml  
${JDG_SERVER_HOME}/standalone/configuration
```

- b. Remove the **security-domain** and **auth-method** attributes from the **rest-connector** element to disable REST security.
6. Start the JBoss Data Grid Server in compatibility mode:

```
$ ${JDG_SERVER_HOME}/bin/standalone.sh -c standalone-compatibility-  
mode.xml
```

Procedure 17.2. Rapid Stock Market Quickstart Client-side Configuration

1. In a new command line terminal window, start the client-side application:

```
$ mvn exec:java -Pclient
```

2. Use the instructions in the help menu for the client application.

[Report a bug](#)

CHAPTER 18. THE CLUSTER APP QUICKSTART

The EAP Cluster App quickstart demonstrates how to use a JBoss Data Grid cache in library mode.

There are three different applications which show:

- How to create and use a JBoss Data Grid clustered cache without having a JBoss EAP cluster.
- A configuration where the JBoss EAP cluster is independent of an Infinispan cluster.
- One JBoss EAP instance can use JBoss Data Grid caches which are members of a different JBoss Data Grid cluster.
- Programming cache configuration using the JBoss Data Grid API.
- File based configuration, seen in administration application App1Cache.
- Use CDI to inject the cache managers.

Each application contains an embedded JBoss Data Grid cache which is accessed by stateless EJB's in the same application. Different JBoss EAP servers, with the exception of AppTwo, are not clustered to demonstrate that an unclustered application can share the JBoss Data Grid cache. In addition, there are two cache managers, neither of which are members of the same JBoss Data Grid cluster. The AdminApp accesses and modified both of these caches. AppOne can only read the App1Cache, but it uses a clustered EJB invocation to AppTwo to read from App2Cache; while AppTwo is deployed as a clustered EJB application and only read App2Cache.

[Report a bug](#)

18.1. EAP CLUSTER APP PREREQUISITES

The prerequisites for the EAP Cluster App are as follows:

- Java 8.0 (Java SDK 1.8) or better
- JBoss Enterprise Application Platform 7.0 or later
- Maven 3.0 or better
- Configure the Maven repository. For details, see [Chapter 3, Install and Use the Maven Repositories](#).
- JBoss Data Grid modules installed in the JBoss EAP server.

[Report a bug](#)

18.2. START THE APPLICATION SERVER INSTANCES

Procedure 18.1. Configure and Start the Servers in standalone mode

1. Prepare an instance of JBoss EAP for copying:
 1. Unzip `jboss-datagrid-${version}-eap-modules-library.zip`.

2. Copy the modules to the server modules directory:

- For Linux:

```
cp -a jboss-datagrid-${version}-eap-modules-library/modules
EAP_HOME
```

For Windows:

```
xcopy /e/i/f jboss-datagrid-${version}-eap-modules-
library/modules EAP_HOME\modules
```

3. Add a user to each server for EJB access:

- For Linux: `EAP_HOME/bin/add-user.sh -a -u quickuser -p quick-123`
- For Windows: `EAP_HOME\bin\add-user.bat -a -u quickuser -p quick-123`

2. Copy the prepared JBoss EAP server to 4 different directories labeled EAP_HOME[1-4].

3. Open a command line for each of the 4 nodes and navigate to the root of the JBoss EAP server directory.

4. Start all four of the JBoss EAP servers using the following commands:

- For Linux:

```
EAP_HOME1/bin/standalone.sh -Djboss.node.name=node1
EAP_HOME2/bin/standalone.sh -Djboss.node.name=node2 -
Djboss.socket.binding.port-offset=100
EAP_HOME3/bin/standalone.sh -Djboss.node.name=node3 -
Djboss.socket.binding.port-offset=200 -c standalone-ha.xml
EAP_HOME4/bin/standalone.sh -Djboss.node.name=node4 -
Djboss.socket.binding.port-offset=300 -c standalone-ha.xml
```

- For Windows:

```
EAP_HOME1\bin\standalone.bat -Djboss.node.name=node1
EAP_HOME2\bin\standalone.bat -Djboss.node.name=node2 -
Djboss.socket.binding.port-offset=100
EAP_HOME3\bin\standalone.bat -Djboss.node.name=node3 -
Djboss.socket.binding.port-offset=200 -c standalone-ha.xml
EAP_HOME4\bin\standalone.bat -Djboss.node.name=node4 -
Djboss.socket.binding.port-offset=300 -c standalone-ha.xml
```

5. Add the configuration for node2 (AppOne) to use EJB server-to-server invocation:

- For Linux:

```
EAP_HOME2/bin/jboss-cli.sh -c --controller=localhost:10099 --
file=QUICKSTART_HOME/install-appOne-standalone.cli
```

For Windows:


```
EAP_HOME2\bin\jboss-cli.bat -c --controller=localhost:10099 --
file=QUICKSTART_HOME/install-appOne-standalone.cli
```

Procedure 18.2. Configure and Start the Servers in domain mode

1. Copy a fresh JBoss EAP installation to EAP_HOME:

1. Unzip **jboss-datagrid-`{version}`-eap-modules-library.zip**.

2. Copy the modules to the server modules directory:

- For Linux:

```
cp -a jboss-datagrid-{version}-eap-modules-library/modules
EAP_HOME
```

- For Windows:

```
xcopy /e/i/f jboss-datagrid-{version}-eap-modules-
library/modules EAP_HOME\modules
```

2. Open a command line and navigate to the root of JBoss EAP.

3. Add a user:

- For Linux:

```
EAP_HOME/bin/add-user.sh -a -u quickuser -p quick-123
```

- For Windows:

```
EAP_HOME\bin\add-user.bat -a -u quickuser -p quick-123
```

4. The following shows the command to start the domain:

- For Linux:

```
EAP_HOME/bin/domain.sh
```

- For Windows:

```
EAP_HOME\bin\domain.bat
```

5. Apply the configuration for the quickstart, the domain will contain 4 nodes:

- For Linux:

```
EAP_HOME/bin/jboss-cli.sh -c --file=QUICKSTART_HOME/install-
domain.cli
```

- For Windows:

```
EAP_HOME\bin\jboss-cli.bat -c --file=QUICKSTART_HOME/install-domain.cli
```

[Report a bug](#)

18.3. BUILD THE APPLICATION

Procedure 18.3. Build the Application

1. Open a command line and navigate to the root directory of this quickstart.
2. Execute the following command to build the archive:

```
mvn clean install
```

3. Copy the application to the appropriate server:

- For Linux:

```
cp adminApp/ear/target/jboss-eap-application-adminApp.ear
EAP_HOME1/standalone/deployments
cp appOne/ear/target/jboss-eap-application-AppOne.ear
EAP_HOME2/standalone/deployments
cp appTwo/ear/target/jboss-eap-application-AppTwo.ear
EAP_HOME3/standalone/deployments
cp appTwo/ear/target/jboss-eap-application-AppTwo.ear
EAP_HOME4/standalone/deployments
```

- For Windows:

```
copy adminApp\ear\target\jboss-eap-application-adminApp.ear
EAP_HOME1\standalone\deployments
copy appOne\ear\target\jboss-eap-application-AppOne.ear
EAP_HOME2\standalone\deployments
copy appTwo\ear\target\jboss-eap-application-AppTwo.ear
EAP_HOME3\standalone\deployments
copy appTwo\ear\target\jboss-eap-application-AppTwo.ear
EAP_HOME4\standalone\deployments
```

4. When domain mode is used, deploy the applications in the following way:

- For Linux:

```
EAP_HOME/bin/jboss-cli.sh -c --file=QUICKSTART_HOME/deploy-domain.cli
```

- For Windows:

```
EAP_HOME\bin\jboss-cli.bat -c --file=QUICKSTART_HOME/deploy-domain.cli
```

[Report a bug](#)

18.4. RUN THE APPLICATION

Once built the deployed applications may be accessed using the included classes.

Using the AdminClient

In this example neither the **AdminServer** nor the **AppOneServer** are clustered at the JBoss EAP level; only the JBoss Data Grid instances are clustered as configured by the application. This application will perform the following:

- Add values to the **App1** cache, using **AdminApp**, and validated they are replicated to the server instance of **AppOne**.
- Add a value to the **App2** cache, then rollback the transaction and confirm that it is not added to the cache after the rollback.

Procedure 18.4. Execute the AdminClient application

1. Open a command line and navigate to the `$QUICKSTART_HOME/client/` directory.
2. Execute the following command:

```
mvn -
  Dexec.mainClass=org.jboss.as.quickstarts.datagrid.eap.app.AdminClient
  t exec:java
```



NOTE

By default the application will attempt to access **localhost:4447** and **localhost:4547** for the **AdminHost** and **AppOneHost**. This may be changed by adding the following parameter to the **mvn** command:

```
-Dexec.args="AdminHost AdminPort AppOneHost AppOnePort"
```

3. Confirm that the following text appears in the console; any unexpected results will raise an Exception:

```
Add a value to App1Cache with the AdminApp and check on the same
instance that the value is correct added
  success
Check the previous added value of App1Cache by accessing the AppOne
Server
  success
Add a value to App2Cache and check on the same instance that the
value is correct added
  success
Check whether changes to a cache are rolledback if the transaction
fail
  The cache App2 work as expected on rollback
```

Using the AppOneClient

In this example both JBoss EAP and the JBoss Data Grid caches are clustered; however, the purpose of this example is to demonstrate that the clusters are independent of one other. The JBoss Data Grid cluster is able to use a different JGroups implementation compared to

the JBoss EAP servers, and this example will:

- Add values to the **App2** cache using the **AdminApp** and access **AppOne** to show that the EJB invocation is clustered and both **AppTwo** instances are used.

Procedure 18.5. Execute the AppOneClient

1. Open a command line and navigate to the `$QUICKSTART_HOME/client/` directory.
2. Execute the following command:

```
mvn -
Dexec.mainClass=org.jboss.as.quickstarts.datagrid.eap.app.AppOneClient
exec:java
```

3. Confirm that the following text appears in the console; any unexpected results will raise an Exception:

```
Add a value to App2Cache with the AdminApp
Access the App2Cache from the AppOneServer by using the clustered
EJB@AppTwoServer
    success : received the following node names for EJB invocation :
[node3, node4]
```

[Report a bug](#)

18.5. DEBUG THE APPLICATION

To debug, or examine the source code, of the quickstart or any of its associated libraries run either of the following commands to pull them into the local repository:

```
mvn dependency:sources
mvn dependency:resolve -Dclassifier=javadoc
```

[Report a bug](#)

CHAPTER 19. THE CAMEL-JBOSSDATAGRID-FUSE QUICKSTART

This quickstart shows how to use the component described in [Section 5.1, “The camel-jbossgdatagrid Component”](#) on JBoss Fuse to interact with JBoss Data Grid.

This quickstart will deploy two bundles, **local_cache_producer** and **local_cache_consumer**, on Fuse, one on each container **child1** and **child2** respectively. Below is a description of each of the bundles:

- **local_cache_producer**: Scans a folder (/tmp/incoming) for incoming CSV files of the format "id, firstName, lastName, age". If a file is dropped with entries in the given format, each entry is read and transformed into a Person POJO and stored in the data grid.
- **local_cache_consumer**: Lets you query for a POJO using a RESTful interface and receive a JSON representation of the Person POJO stored in the data grid for the given key

The bundles reside in two different containers; the consumer is able to extract what the producer has put in due to the same configuration being used in the **infinispan.xml** and **jgroups.xml** files. The **infinispan.xml** file defines a **REPL** (replicated) cache named **camel-cache**, and both the consumer and producer interact with this cache.

[Report a bug](#)

19.1. QUICKSTART PREREQUISITES

The prerequisites for this quickstart are as follows:

- Java 7.0 (Java SDK 1.7) or better
- Maven 3.0 or better
- JBoss Fuse 6.2.0 or better
- Configure the Maven Repository. For details, see [Chapter 3, Install and Use the Maven Repositories](#)

[Report a bug](#)

19.2. SETUP

1. Download the Fuse Full Install binary from the Red Hat Customer Portal.
2. Export the path to the folder where the CSV files will be placed by running the command:

```
export incomingFolderPath=[Full path to the CSV folder]
```

3. Run the following command in the root directory of the quickstart:

```
mvn clean install -DincomingFolderPath=$incomingFolderPath
```

4. Set the **FUSE_INSTALL_PATH** and **FUSE_BINARY_PATH** variables in the same shell:

```
export FUSE_INSTALL_PATH = [Full path to the folder where Fuse will
be installed]
export FUSE_BINARY_PATH = [Full path to the Fuse zip file downloaded
in step 1]
```

5. If running the camel-jbossdatagrid-fuse quickstart in JBoss Fuse 6.2.1 the following changes to the **setupEverythingOnFuse.sh** script must be made; otherwise proceed to the next step:

1. Change the version of Fuse being exported to reference the 6.2.1 component.
2. Update the container name to **child1**.

```
# Original line with old version
# export FUSE_VERSION=jboss-fuse-6.2.0.redhat-133
# Updated line for Fuse 6.2.1:
export FUSE_VERSION=jboss-fuse-6.2.1.redhat-084

[...]

# Original line exporting the profile
#sh client -r 2 -d 10 "fabric:container-add-profile child demo-
local_producer" > /dev/null 2>&1
# Updated line for Fuse 6.2.1:
sh client -r 2 -d 10 "fabric:container-add-profile child1 demo-
local_producer" > /dev/null 2>&1
```

6. Once the environment variables are set run the following from the root directory of the quickstart:

```
./setupEverythingOnFuse.sh
```

7. After the script completes confirm that the Fuse Hawtio Console may be accessed without error. This console, by default, runs at **http://127.0.0.1:8181/hawtio/index.html#/login**; the username and password are both **admin**.
8. Confirm that both the **child1** and **child2** containers were created by accessing Fuse Fabric at **http://127.0.0.1:8181/hawtio/index.html#/fabric/containers**. Both containers should be highlighted in green to indicate they are ready.

[Report a bug](#)

19.3. TESTING THE CAMEL-JBOSSDATAGRID-FUSE QUICKSTART

To test the **local_cache_producer** create a CSV file in the **incomingFolderPath**, previously specified. The following command will generate a file with a single entry:

```
echo "1,Bill,Gates,59" > $incomingFolderPath/sample.csv
```

Once the file has been removed from the directory then the producer has successfully parsed the file. Proceed to testing the consumer.

To test the **local_cache_consumer** navigate to **http://127.0.0.1:8282/cache/get/1** in a web browser. This will query the cache for the entry with an Id of **1**, which was specified above. The following JSON of the created POJO should be returned:

```
┌ {"id":1,"firstName":"Bill","lastName":"Gates","age":59}
```

[Report a bug](#)

PART VII. UNINSTALL RED HAT JBOSS DATA GRID

CHAPTER 20. REMOVE RED HAT JBOSS DATA GRID

20.1. REMOVE RED HAT JBOSS DATA GRID FROM YOUR LINUX SYSTEM

The following procedures contain instructions to remove Red Hat JBoss Data Grid from your Linux system.



WARNING

Once deleted, all JBoss Data Grid configuration and settings are permanently lost.

Procedure 20.1. Remove JBoss Data Grid from Your Linux System

1. **Shut Down Server**

Ensure that the JBoss Data Grid server is shut down.

2. **Navigate to the JBoss Data Grid Home Directory**

Use the command line to change into the level above the `$JDG_HOME` folder.

3. **Delete the JBoss Data Grid Home Directory**

Enter the following command in the terminal to remove JBoss Data Grid, replacing `$JDG_HOME` with the name of your JBoss Data Grid home directory:

```
$ rm -Rf $JDG_HOME
```

[Report a bug](#)

20.2. REMOVE RED HAT JBOSS DATA GRID FROM YOUR WINDOWS SYSTEM

The following procedures contain instructions to remove Red Hat JBoss Data Grid from your Microsoft Windows system.



WARNING

Once deleted, all JBoss Data Grid configuration and settings are permanently lost.

Procedure 20.2. Remove JBoss Data Grid from Your Windows System

1. **Shut Down Server**

Ensure that the JBoss Data Grid server is shut down.

2. **Navigate to the JBoss Data Grid Home Directory**

Use the Windows Explorer to navigate to the directory in which the **\$JDG_HOME** folder is located.

3. **Delete the JBoss Data Grid Home Directory**

Select the **\$JDG_HOME** folder and delete it.

[Report a bug](#)

APPENDIX A. REFERENCES

A.1. 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. It consists of entry data attributes from the related entry.
- A value is the data assigned to and identified by the key.

[Report a bug](#)

APPENDIX B. MAVEN CONFIGURATION INFORMATION

B.1. INSTALL THE JBOSS ENTERPRISE APPLICATION PLATFORM REPOSITORY USING NEXUS

This example outlines the steps to install the JBoss Enterprise Application Platform 6 Maven Repository using Sonatype Nexus Maven Repository Manager. For further instructions, see <http://www.sonatype.org/nexus/>.

Procedure B.1. Download the JBoss Enterprise Application Platform 6 Maven Repository ZIP archive

1. Open a web browser and access the following URL:
<https://access.redhat.com/jbosnetwork/restricted/listSoftware.html?product=appplatform>.
2. Find **Application Platform 6 Maven Repository** in the list.
3. Click **Download** to download a ZIP file that contains the repository.
4. Unzip the files into the desired target directory.

Procedure B.2. Add the JBoss Enterprise Application Platform 6 Maven Repository using Nexus Maven Repository Manager

1. Log into Nexus as an Administrator.
2. Select the **Repositories** section from the **Views** → **Repositories** menu to the left of your repository manager.
3. Click the **Add...** drop-down menu, then select **Hosted Repository**.
4. Provide a name and ID for the new repository.
5. Enter the unzipped repository path in the **Override Local Storage Location** field.
6. Continue if the artifact must be available in a repository group. If not, do not continue with this procedure.
7. Select the repository group.
8. Click on the **Configure** tab.
9. Drag the new JBoss Maven repository from the **Available Repositories** list to the **Ordered Group Repositories** list on the left.



NOTE

The order of this list determines the priority for searching Maven artifacts.

Result

The repository is configured using Nexus Maven Repository Manager.

[Report a bug](#)

B.2. MAVEN REPOSITORY CONFIGURATION EXAMPLE

A sample Maven repository file named **example-settings.xml** is available in the root directory of the Maven repository folder after it is unzipped. The following is an excerpt that contains the relevant parts of the **example-settings.xml** file:

Example B.1. Sample Maven Repository Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
  http://maven.apache.org/xsd/settings-1.0.0.xsd">

  <profiles>
    <profile>
      <id>jboss-datagrid-repository</id>
      <repositories>
        <repository>
          <id>jboss-datagrid-repository</id>
          <name>JBoss Data Grid Maven Repository</name>
          <url>JDG_REPOSITORY_URL</url>
          <layout>default</layout>
          <releases>
            <enabled>>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>jboss-datagrid-repository</id>
          <name>JBoss Data Grid Maven Repository</name>
          <url>JDG_REPOSITORY_URL</url>
          <layout>default</layout>
          <releases>
            <enabled>>true</enabled>
            <updatePolicy>never</updatePolicy>
          </releases>
          <snapshots>
            <enabled>>false</enabled>
            <updatePolicy>never</updatePolicy>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
```

```
<!-- make the profile active by default -->
<activeProfile>jboss-datagrid-repository</activeProfile>
</activeProfiles>

</settings>
```

The `JDG_REPOSITORY_URL` may be found by following the instructions in [Section B.3, “Determining the URL of the JBoss Data Grid Repository”](#).

[Report a bug](#)

B.3. DETERMINING THE URL OF THE JBOSS DATA GRID REPOSITORY

The repository URL depends on where the repository is located. You can configure Maven to use any of the following repository locations:

- To use the online JBoss Data Grid Maven repository, specify the following URL:
`https://maven.repository.redhat.com/ga/`
- To use a JBoss Data Grid Maven repository installed on the local file system, you must download the repository and then use the local file path for the URL. For example: **`file:///path/to/repo/jboss-datagrid-7.0.0-maven-repository/maven-repository/`**
- If you install the JBoss Data Grid Maven repository using the Nexus Repository Manager, the URL will look something like the following:
`https://intranet.acme.com/nexus/content/repositories/jboss-datagrid-7.0.0-maven-repository/maven-repository/`

[Report a bug](#)

APPENDIX C. REVISION HISTORY

Revision 7.0.0-4 Updated quickstarts.	Wed 9 Nov 2016	Christian Huffman
Revision 7.0.0-3 Updated java version requirements.	Fri 22 Jul 2016	Christian Huffman
Revision 7.0.0-2 Included instructions for domain mode. Updated for GA release.	Mon 18 Jul 2016	Christian Huffman
Revision 7.0.0-0 Initial draft for 7.0.0.	Tue 19 Apr 2016	Christian Huffman