



Red Hat JBoss Data Grid 7.1

Administration and Configuration Guide

For use with Red Hat JBoss Data Grid 7.1

Red Hat JBoss Data Grid 7.1 Administration and Configuration Guide

For use with Red Hat JBoss Data Grid 7.1

Legal Notice

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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 presents information about the administration and configuration of Red Hat JBoss Data Grid 7.1

Table of Contents

PART I. INTRODUCTION	16
CHAPTER 1. SETTING UP RED HAT JBOSS DATA GRID	17
1.1. PREREQUISITES	17
1.2. STEPS TO SET UP RED HAT JBOSS DATA GRID	17
PART II. SET UP JVM MEMORY MANAGEMENT	20
CHAPTER 2. SET UP EVICTION	21
2.1. ABOUT EVICTION	21
2.2. EVICTION STRATEGIES	21
2.2.1. Eviction Strategies	21
2.2.2. LRU Eviction Algorithm Limitations	22
2.3. USING EVICTION	22
2.3.1. Using Eviction	22
2.3.2. Initialize Eviction	22
2.3.3. Eviction Configuration Examples	23
2.3.4. Utilizing Memory Based Eviction	23
2.3.5. Eviction and Passivation	24
CHAPTER 3. SET UP EXPIRATION	25
3.1. ABOUT EXPIRATION	25
3.2. EXPIRATION OPERATIONS	25
3.3. EVICTION AND EXPIRATION COMPARISON	25
3.4. CACHE ENTRY EXPIRATION BEHAVIOR	26
3.5. CONFIGURE EXPIRATION	26
3.6. TROUBLESHOOTING EXPIRATION	26
PART III. MONITOR YOUR CACHE	28
CHAPTER 4. SET UP LOGGING	29
4.1. ABOUT LOGGING	29
4.2. SUPPORTED APPLICATION LOGGING FRAMEWORKS	29
4.2.1. Supported Application Logging Frameworks	29
4.2.2. About JBoss Logging	29
4.2.3. JBoss Logging Features	29
4.3. BOOT LOGGING	30
4.3.1. Boot Logging	30
4.3.2. Configure Boot Logging	30
4.3.3. Default Log File Locations	30
4.4. LOGGING ATTRIBUTES	30
4.4.1. About Log Levels	30
4.4.2. Supported Log Levels	31
4.4.3. About Log Categories	32
4.4.4. About the Root Logger	32
4.4.5. About Log Handlers	32
4.4.6. Log Handler Types	33
4.4.7. Selecting Log Handlers	34
4.4.8. About Log Formatters	35
4.5. LOGGING SAMPLE CONFIGURATIONS	35
4.5.1. Logging Sample Configuration Location	35
4.5.2. Sample XML Configuration for the Root Logger	35
4.5.3. Sample XML Configuration for a Log Category	36

4.5.4. Sample XML Configuration for a Console Log Handler	36
4.5.5. Sample XML Configuration for a File Log Handler	37
4.5.6. Sample XML Configuration for a Periodic Log Handler	38
4.5.7. Sample XML Configuration for a Size Log Handler	39
4.5.8. Sample XML Configuration for a Async Log Handler	40
PART IV. SET UP CACHE MODES	42
CHAPTER 5. CACHE MODES	43
5.1. CACHE MODES	43
5.2. ABOUT CACHE CONTAINERS	43
5.3. LOCAL MODE	44
5.3.1. Local Mode	44
5.3.2. Configure Local Mode	44
5.4. CLUSTERED MODES	45
5.4.1. Clustered Modes	45
5.4.2. Asynchronous and Synchronous Operations	45
5.4.3. About Asynchronous Communications	45
5.4.4. Cache Mode Troubleshooting	45
5.4.4.1. Invalid Data in ReadExternal	46
5.4.4.2. Cluster Physical Address Retrieval	46
CHAPTER 6. SET UP DISTRIBUTION MODE	47
6.1. ABOUT DISTRIBUTION MODE	47
6.2. DISTRIBUTION MODE'S CONSISTENT HASH ALGORITHM	47
6.3. LOCATING ENTRIES IN DISTRIBUTION MODE	47
6.4. RETURN VALUES IN DISTRIBUTION MODE	47
6.5. CONFIGURE DISTRIBUTION MODE	48
6.6. SYNCHRONOUS AND ASYNCHRONOUS DISTRIBUTION	48
CHAPTER 7. SET UP REPLICATION MODE	49
7.1. ABOUT REPLICATION MODE	49
7.2. OPTIMIZED REPLICATION MODE USAGE	49
7.3. CONFIGURE REPLICATION MODE	49
7.4. SYNCHRONOUS AND ASYNCHRONOUS REPLICATION	50
7.4.1. Synchronous and Asynchronous Replication	50
7.4.2. Troubleshooting Asynchronous Replication Behavior	50
7.5. THE REPLICATION QUEUE	50
7.5.1. The Replication Queue	50
7.5.2. Replication Queue Usage	51
7.6. ABOUT REPLICATION GUARANTEES	51
7.7. REPLICATION TRAFFIC ON INTERNAL NETWORKS	51
CHAPTER 8. SET UP INVALIDATION MODE	53
8.1. ABOUT INVALIDATION MODE	53
8.2. CONFIGURE INVALIDATION MODE	53
8.3. SYNCHRONOUS/ASYNCHRONOUS INVALIDATION	53
8.4. THE L1 CACHE AND INVALIDATION	54
CHAPTER 9. STATE TRANSFER	55
9.1. STATE TRANSFER	55
9.2. NON-BLOCKING STATE TRANSFER	55
9.3. SUPPRESS STATE TRANSFER VIA JMX	56
9.4. THE REBALANCINGENABLED ATTRIBUTE	56

PART V. ENABLING APIS	57
CHAPTER 10. ENABLING APIS DECLARATIVELY	58
10.1. ENABLING APIS DECLARATIVELY	58
10.2. BATCHING API	58
10.3. GROUPING API	58
10.4. EXTERNALIZABLE API	59
10.4.1. The Externalizable API	59
10.4.2. Register the Advanced Externalizer (Declaratively)	59
10.4.3. Custom Externalizer ID Values	60
10.4.3.1. Custom Externalizer ID Values	60
10.4.3.2. Customize the Externalizer ID (Declaratively)	60
CHAPTER 11. SET UP AND CONFIGURE THE INFINISPAN QUERY API	62
11.1. SET UP INFINISPAN QUERY	62
11.1.1. Infinispan Query Dependencies in Library Mode	62
11.2. INDEXING MODES	62
11.2.1. Managing Indexes	62
11.2.2. Managing the Index in Local Mode	62
11.2.3. Managing the Index in Replicated Mode	63
11.2.4. Managing the Index in Distribution Mode	63
11.2.5. Managing the Index in Invalidation Mode	64
11.3. DIRECTORY PROVIDERS	64
11.3.1. Directory Providers	64
11.3.2. RAM Directory Provider	64
11.3.3. Filesystem Directory Provider	65
11.3.4. Infinispan Directory Provider	65
11.4. CONFIGURE INDEXING	66
11.4.1. Configure the Index in Remote Client-Server Mode	66
11.4.2. Rebuilding the Index	67
11.5. TUNING THE INDEX	67
11.5.1. Near-Realtime Index Manager	67
11.5.2. Tuning Infinispan Directory	67
11.5.3. Per-Index Configuration	68
CHAPTER 12. THE HEALTH CHECK API	69
12.1. THE HEALTH CHECK API	69
12.2. ACCESSING THE HEALTH API USING JMX	69
12.3. ACCESSING THE HEALTH CHECK API USING THE CLI	70
12.4. ACCESSING THE HEALTH CHECK API USING THE MANAGEMENT REST INTERFACE	71
PART VI. REMOTE CLIENT-SERVER MODE INTERFACES	74
CHAPTER 13. REMOTE CLIENT-SERVER MODE INTERFACES	75
CHAPTER 14. THE HOT ROD INTERFACE	76
14.1. ABOUT HOT ROD	76
14.2. THE BENEFITS OF USING HOT ROD OVER MEMCACHED	76
14.3. HOT ROD HASH FUNCTIONS	77
14.4. THE HOT ROD INTERFACE CONNECTOR	77
14.4.1. The Hot Rod Interface Connector	77
14.4.2. Configure Hot Rod Connectors	77
CHAPTER 15. THE REST INTERFACE	80
15.1. THE REST INTERFACE	80

15.2. THE REST INTERFACE CONNECTOR	80
15.2.1. The REST Interface Connector	80
15.2.2. Configure REST Connectors	80
CHAPTER 16. THE MEMCACHED INTERFACE	81
16.1. THE MEMCACHED INTERFACE	81
16.2. ABOUT MEMCACHED SERVERS	81
16.3. MEMCACHED STATISTICS	81
16.4. THE MEMCACHED INTERFACE CONNECTOR	83
16.4.1. The Memcached Interface Connector	83
16.4.2. Configure Memcached Connectors	83
PART VII. SET UP LOCKING FOR THE CACHE	85
CHAPTER 17. LOCKING	86
17.1. LOCKING	86
17.2. CONFIGURE LOCKING (REMOTE CLIENT-SERVER MODE)	86
17.3. CONFIGURE LOCKING (LIBRARY MODE)	86
17.4. LOCKING TYPES	87
17.4.1. About Optimistic Locking	87
17.4.2. About Pessimistic Locking	87
17.4.3. Pessimistic Locking Types	87
17.4.4. Explicit Pessimistic Locking Example	88
17.4.5. Implicit Pessimistic Locking Example	88
17.4.6. Configure Locking Mode (Remote Client-Server Mode)	88
17.4.7. Configure Locking Mode (Library Mode)	89
17.5. LOCKING OPERATIONS	89
17.5.1. About the LockManager	89
17.5.2. About Lock Acquisition	89
17.5.3. About Concurrency Levels	89
CHAPTER 18. SET UP LOCK STRIPING	90
18.1. ABOUT LOCK STRIPING	90
18.2. CONFIGURE LOCK STRIPING (REMOTE CLIENT-SERVER MODE)	90
18.3. CONFIGURE LOCK STRIPING (LIBRARY MODE)	90
CHAPTER 19. SET UP ISOLATION LEVELS	92
19.1. ABOUT ISOLATION LEVELS	92
19.2. ABOUT READ_COMMITTED	92
19.3. ABOUT REPEATABLE_READ	92
PART VIII. SET UP AND CONFIGURE A CACHE STORE	94
CHAPTER 20. CACHE STORES	95
20.1. CACHE STORES	95
20.2. CACHE LOADERS AND CACHE WRITERS	95
20.3. CACHE STORE CONFIGURATION	95
20.3.1. Configuring the Cache Store	95
20.3.2. Configure the Cache Store using XML (Library Mode)	95
20.3.3. About SKIP_CACHE_LOAD Flag	96
20.3.4. About the SKIP_CACHE_STORE Flag	96
20.3.5. About the SKIP_SHARED_CACHE_STORE Flag	96
20.4. SHARED CACHE STORES	96
20.4.1. Shared Cache Stores	96
20.4.2. Invalidation Mode and Shared Cache Stores	97

20.4.3. The Cache Store and Cache Passivation	97
20.4.4. Application Cachestore Registration	97
20.5. CONNECTION FACTORIES	97
20.5.1. Connection Factories	97
20.5.2. About ManagedConnectionFactory	97
20.5.3. About SimpleConnectionFactory	97
20.5.4. About PooledConnectionFactory	98
CHAPTER 21. CACHE STORE IMPLEMENTATIONS	99
21.1. CACHE STORES	99
21.2. CACHE STORE COMPARISON	99
21.3. CACHE STORE CONFIGURATION DETAILS (LIBRARY MODE)	99
21.4. CACHE STORE CONFIGURATION DETAILS (REMOTE CLIENT-SERVER MODE)	104
21.5. SINGLE FILE CACHE STORE	107
21.5.1. Single File Cache Store	107
21.5.2. Single File Store Configuration (Remote Client-Server Mode)	107
21.5.3. Single File Store Configuration (Library Mode)	107
21.5.4. Upgrade JBoss Data Grid Cache Stores	108
21.6. LEVELDB CACHE STORE	108
21.6.1. LevelDB Cache Store	108
21.6.2. Configuring LevelDB Cache Store (Remote Client-Server Mode)	108
21.6.3. LevelDB Cache Store Sample XML Configuration (Library Mode)	109
21.6.4. Configure a LevelDB Cache Store Using JBoss Operations Network	109
21.7. JDBC BASED CACHE STORES	112
21.7.1. JDBC Based Cache Stores	112
21.7.2. JdbcBinaryStores	113
21.7.2.1. JdbcBinaryStores	113
21.7.2.2. JdbcBinaryStore Configuration (Remote Client-Server Mode)	113
21.7.2.3. JdbcBinaryStore Configuration (Library Mode)	113
21.7.3. JdbcStringBasedStores	114
21.7.3.1. JdbcStringBasedStores	114
21.7.3.2. JdbcStringBasedStore Configuration (Remote Client-Server Mode)	114
21.7.3.3. JdbcStringBasedStore Configuration (Library Mode)	115
21.7.3.4. JdbcStringBasedStore Multiple Node Configuration (Remote Client-Server Mode)	116
21.7.4. JdbcMixedStores	116
21.7.4.1. JdbcMixedStores	116
21.7.4.2. JdbcMixedStore Configuration (Remote Client-Server Mode)	116
21.7.4.3. JdbcMixedStore Configuration (Library Mode)	117
21.7.5. Cache Store Troubleshooting	118
21.7.5.1. IOExceptions with JdbcStringBasedStore	118
21.8. THE REMOTE CACHE STORE	118
21.8.1. Remote Cache Stores	118
21.8.2. Remote Cache Store Configuration (Remote Client-Server Mode)	118
21.8.3. Remote Cache Store Configuration (Library Mode)	119
21.8.4. Define the Outbound Socket for the Remote Cache Store	119
21.9. JPA CACHE STORE	119
21.9.1. JPA Cache Stores	119
21.9.2. JPA Cache Store Sample XML Configuration (Library Mode)	120
21.9.3. Storing Metadata in the Database	120
21.9.4. Deploying JPA Cache Stores in Various Containers	121
21.10. CASSANDRA CACHE STORE	122
21.10.1. Cassandra Cache Store	122
21.10.2. Enabling the Cassandra Cache Store	122

21.10.3. Cassandra Cache Store Sample XML Configuration (Remote Client-Server Mode)	123
21.10.4. Cassandra Cache Store Sample XML Configuration (Library Mode)	123
21.10.5. Cassandra Configuration Parameters	124
21.11. CUSTOM CACHE STORES	125
21.11.1. Custom Cache Stores	125
21.11.2. Custom Cache Store Maven Archetype	126
21.11.3. Custom Cache Store Configuration (Remote Client-Server Mode)	126
21.11.3.1. Custom Cache Store Configuration (Remote Client-Server Mode)	126
21.11.3.2. Option 1: Add Custom Cache Store using deployments (Remote Client-Server Mode)	127
21.11.3.3. Option 2: Add Custom Cache Store using the CLI (Remote Client-Server Mode)	127
21.11.3.4. Option 3: Add Custom Cache Store using JON (Remote Client-Server Mode)	128
21.11.4. Custom Cache Store Configuration (Library Mode)	129
PART IX. SET UP PASSIVATION	130
CHAPTER 22. ACTIVATION AND PASSIVATION MODES	131
22.1. ACTIVATION AND PASSIVATION MODES	131
22.2. PASSIVATION MODE BENEFITS	131
22.3. CONFIGURE PASSIVATION	131
22.4. EVICTION AND PASSIVATION	131
22.4.1. Eviction and Passivation	131
22.4.2. Eviction and Passivation Usage	132
22.4.3. Eviction Example when Passivation is Disabled	132
22.4.4. Eviction Example when Passivation is Enabled	132
PART X. SET UP CACHE WRITING	134
CHAPTER 23. CACHE WRITING MODES	135
23.1. CACHE WRITING MODES	135
23.2. WRITE-THROUGH CACHING	135
23.2.1. Write-Through Caching	135
23.2.2. Write-Through Caching Benefits and Disadvantages	135
23.2.3. Write-Through Caching Configuration (Library Mode)	135
23.3. WRITE-BEHIND CACHING	136
23.3.1. Write-Behind Caching	136
23.3.2. About Unscheduled Write-Behind Strategy	136
23.3.3. Unscheduled Write-Behind Strategy Configuration (Remote Client-Server Mode)	136
23.3.4. Unscheduled Write-Behind Strategy Configuration (Library Mode)	137
PART XI. MONITOR CACHES AND CACHE MANAGERS	138
CHAPTER 24. SET UP JAVA MANAGEMENT EXTENSIONS (JMX)	139
24.1. ABOUT JAVA MANAGEMENT EXTENSIONS (JMX)	139
24.2. USING JMX WITH RED HAT JBOSS DATA GRID	139
24.3. JMX STATISTIC LEVELS	139
24.4. ENABLE JMX FOR CACHE INSTANCES	139
24.5. ENABLE JMX FOR CACHEMANAGERS	139
24.6. DISABLING THE CACHESTORE VIA JMX WHEN USING ROLLING UPGRADES	140
24.7. MULTIPLE JMX DOMAINS	140
24.8. MBEANS	140
24.8.1. MBeans	140
24.8.2. Understanding MBeans	141
24.8.3. Registering MBeans in Non-Default MBean Servers	141
CHAPTER 25. SET UP JBOSS OPERATIONS NETWORK (JON)	143

25.1. ABOUT JBOSS OPERATIONS NETWORK (JON)	143
25.2. DOWNLOAD JBOSS OPERATIONS NETWORK (JON)	143
25.2.1. Prerequisites for Installing JBoss Operations Network (JON)	143
25.2.2. Download JBoss Operations Network	143
25.2.3. Remote JMX Port Values	144
25.2.4. Download JBoss Operations Network (JON) Plugin	144
25.3. JBOSS OPERATIONS NETWORK SERVER INSTALLATION	144
25.4. JBOSS OPERATIONS NETWORK AGENT	145
25.5. JBOSS OPERATIONS NETWORK FOR REMOTE CLIENT-SERVER MODE	145
25.5.1. JBoss Operations Network for Remote Client-Server Mode	145
25.5.2. Installing the JBoss Operations Network Plug-in (Remote Client-Server Mode)	145
25.6. JBOSS OPERATIONS NETWORK REMOTE-CLIENT SERVER PLUGIN	146
25.6.1. JBoss Operations Network Plugin Metrics	146
25.6.2. JBoss Operations Network Plugin Operations	150
25.6.3. JBoss Operations Network Plugin Attributes	151
25.6.4. Create a New Cache Using JBoss Operations Network (JON)	151
25.7. JBOSS OPERATIONS NETWORK FOR LIBRARY MODE	152
25.7.1. JBoss Operations Network for Library Mode	152
25.7.2. Installing the JBoss Operations Network Plug-in (Library Mode)	152
25.7.3. Monitoring of JBoss Data Grid Instances in Library Mode	155
25.7.3.1. Prerequisites	155
25.7.3.2. Manually Adding JBoss Data Grid Instances in Library Mode	156
25.7.3.3. Monitor Custom Applications Using Library Mode Deployed On JBoss Enterprise Application Platform	161
25.7.3.3.1. Monitor an Application Deployed in Standalone Mode	161
25.7.3.3.2. Monitor an Application Deployed in Domain Mode	162
25.8. JBOSS OPERATIONS NETWORK PLUG-IN QUICKSTART	163
25.9. OTHER MANAGEMENT TOOLS AND OPERATIONS	163
25.9.1. Other Management Tools and Operations	163
25.9.2. Accessing Data via URLs	163
25.9.3. Limitations of Map Methods	163
PART XII. RED HAT JBOSS DATA GRID WEB ADMINISTRATION	165
CHAPTER 26. RED HAT JBOSS DATA GRID ADMINISTRATION CONSOLE	166
26.1. ABOUT JBOSS DATA GRID ADMINISTRATION CONSOLE	166
26.2. RED HAT JBOSS DATA GRID ADMINISTRATION CONSOLE PREREQUISITES	166
26.3. RED HAT JBOSS DATA GRID ADMINISTRATION CONSOLE GETTING STARTED	166
26.3.1. Red Hat JBoss Data Grid Administration Console Getting Started	166
26.3.2. Adding Management User	166
26.3.3. Logging in the JBoss Data Grid Administration Console	167
26.4. DASHBOARD VIEW	167
26.4.1. Dashboard View	167
26.4.2. Cache Containers View	167
26.4.3. Clusters View	168
26.4.4. Status Events View	168
26.5. CACHE ADMINISTRATION	169
26.5.1. Adding a New Cache	169
26.5.2. Editing Cache Configuration	171
26.5.3. Cache Statistics and Properties View	174
26.5.4. Enable and Disable Caches	177
26.5.5. Cache Flush and Clear	180
Flushing a Cache	180

Clearing a Cache	182
26.5.6. Server Tasks Execution	184
26.5.7. Server Tasks	184
26.5.7.1. New Server Task	184
26.5.7.2. Server Tasks View	185
26.6. CACHE CONTAINER CONFIGURATION	186
26.6.1. Cache Container Configuration	186
26.6.2. Defining Protocol Buffer Schema	187
26.6.3. Transport Setting	188
26.6.4. Defining Thread Pools	189
26.6.5. Adding New Security Role	192
26.6.6. Creating Cache Configuration Template	193
26.7. CLUSTER ADMINISTRATION	195
26.7.1. Cluster Nodes View	195
26.7.2. Cluster Nodes Mismatch	196
26.7.3. Cluster Rebalancing	196
26.7.4. Cluster Partition Handling	200
26.7.5. Cluster Events	201
26.7.6. Adding Node	202
26.7.7. Node Statistics and Properties View	205
26.7.8. Node Performance Metrics View	205
26.7.9. Disabling a Node	206
26.7.10. Cluster Shutdown and Restart	207
26.7.10.1. Cluster Shutdown	207
26.7.10.2. Cluster Start	208
PART XIII. SECURING DATA IN RED HAT JBOSS DATA GRID	210
CHAPTER 27. INTRODUCTION	211
27.1. SECURING DATA IN RED HAT JBOSS DATA GRID	211
CHAPTER 28. RED HAT JBOSS DATA GRID SECURITY: AUTHORIZATION AND AUTHENTICATION	212
28.1. RED HAT JBOSS DATA GRID SECURITY: AUTHORIZATION AND AUTHENTICATION	212
28.2. PERMISSIONS	212
28.3. ROLE MAPPING	214
28.4. CONFIGURING AUTHENTICATION AND ROLE MAPPING USING LOGIN MODULES	215
28.5. CONFIGURING RED HAT JBOSS DATA GRID FOR AUTHORIZATION	216
28.6. AUTHORIZATION USING A SECURITYMANAGER	217
28.7. SECURITYMANAGER IN JAVA	220
28.7.1. About the Java Security Manager	220
28.7.2. About Java Security Manager Policies	220
28.7.3. Write a Java Security Manager Policy	221
28.7.4. Run Red Hat JBoss Data Grid Server Within the Java Security Manager	221
28.8. DATA SECURITY FOR REMOTE CLIENT SERVER MODE	222
28.8.1. About Security Realms	222
28.8.2. Add a New Security Realm	223
28.8.3. Add a User to a Security Realm	224
28.8.4. Configuring Security Realms Declaratively	224
28.8.5. Loading Roles from LDAP for Authorization (Remote Client-Server Mode)	225
username-to-dn	226
The Group Search	227
General Group Searching	229
28.9. SECURING INTERFACES	230
28.9.1. Hot Rod Interface Security	230

28.9.1.1. Publish Hot Rod Endpoints as a Public Interface	231
28.9.1.2. Encryption of communication between Hot Rod Server and Hot Rod client	231
28.9.1.3. Securing Hot Rod to LDAP Server using SSL	232
28.9.1.4. User Authentication over Hot Rod Using SASL	232
28.9.1.4.1. User Authentication over Hot Rod Using SASL	233
28.9.1.4.2. Configure Hot Rod Authentication (GSSAPI/Kerberos)	233
28.9.1.4.3. Configure Hot Rod Authentication (MD5)	234
28.9.1.4.4. Configure Hot Rod Using LDAP/Active Directory	235
28.9.1.4.5. Configure Hot Rod Authentication (X.509)	236
28.9.2. REST Interface Security	237
28.9.2.1. Publish REST Endpoints as a Public Interface	237
28.9.2.2. Enable Security for the REST Endpoint	237
28.9.3. Memcached Interface Security	239
28.9.3.1. Publish Memcached Endpoints as a Public Interface	239
28.10. ACTIVE DIRECTORY AUTHENTICATION (NON-KERBEROS)	239
28.11. ACTIVE DIRECTORY AUTHENTICATION USING KERBEROS (GSSAPI)	239
28.12. THE SECURITY AUDIT LOGGER	241
28.12.1. The Security Audit Logger	241
28.12.2. Configure the Security Audit Logger (Library Mode)	241
28.12.3. Configure the Security Audit Logger (Remote Client-Server Mode)	241
28.12.4. Custom Audit Loggers	242
CHAPTER 29. SECURITY FOR CLUSTER TRAFFIC	243
29.1. NODE AUTHENTICATION AND AUTHORIZATION (REMOTE CLIENT-SERVER MODE)	243
29.1.1. Node Authentication and Authorization (Remote Client-Server Mode)	243
29.1.2. Configure Node Authentication for Cluster Security (DIGEST-MD5)	244
29.1.3. Configure Node Authentication for Cluster Security (GSSAPI/Kerberos)	245
29.2. CONFIGURE NODE SECURITY IN LIBRARY MODE	246
29.2.1. Configure Node Security in Library Mode	246
29.2.2. Simple Authorizing Callback Handler	247
29.2.3. Configure Node Authentication for Library Mode (DIGEST-MD5)	248
29.2.4. Configure Node Authentication for Library Mode (GSSAPI)	248
29.3. JGROUPS ENCRYPTION	249
29.3.1. JGroups Encryption	249
29.3.2. Configuring JGroups Encryption Protocols	249
29.3.3. SYM_ENCRYPT: Using a Key Store	250
29.3.4. ASYM_ENCRYPT: Configured with Algorithms and Key Sizes	250
29.3.5. JGroups Encryption Configuration Parameters	251
PART XIV. COMMAND LINE TOOLS	253
CHAPTER 30. INTRODUCTION	254
30.1. COMMAND LINE TOOLS	254
CHAPTER 31. RED HAT JBOSS DATA GRID CLIS	255
31.1. JBOSS DATA GRID CLIS	255
31.2. RED HAT JBOSS DATA GRID LIBRARY MODE CLI	255
31.2.1. Red Hat JBoss Data Grid Library Mode CLI	255
31.2.2. Start the Library Mode CLI (Server)	255
31.2.3. Start the Library Mode CLI (Client)	255
31.2.4. CLI Client Switches for the Command Line	255
31.2.5. Connect to the Application	256
31.3. RED HAT JBOSS DATA GRID SERVER CLI	256
31.3.1. Red Hat Data Grid Server Mode CLI	256

31.3.2. Start the Server Mode CLI	257
31.4. CLI COMMANDS	257
31.4.1. CLI Commands	257
31.4.2. The abort Command	257
31.4.3. The begin Command	257
31.4.4. The cache Command	257
31.4.5. The clearcache Command	258
31.4.6. The commit Command	258
31.4.7. The container Command	258
31.4.8. The create Command	258
31.4.9. The deny Command	258
31.4.10. The disconnect Command	259
31.4.11. The encoding Command	259
31.4.12. The end Command	259
31.4.13. The evict Command	259
31.4.14. The get Command	260
31.4.15. The grant Command	260
31.4.16. The info Command	260
31.4.17. The locate Command	261
31.4.18. The put Command	261
31.4.19. The replace Command	261
31.4.20. The roles command	261
31.4.21. The rollback Command	262
31.4.22. The site Command	262
31.4.23. The start Command	262
31.4.24. The stats Command	262
31.4.25. The upgrade Command	263
31.4.26. The version Command	263
PART XV. OTHER RED HAT JBOSS DATA GRID FUNCTIONS	264
CHAPTER 32. SET UP THE L1 CACHE	265
32.1. ABOUT THE L1 CACHE	265
32.2. L1 CACHE CONFIGURATION	265
32.2.1. L1 Cache Configuration (Library Mode)	265
32.2.2. L1 Cache Configuration (Remote Client-Server Mode)	265
CHAPTER 33. SET UP TRANSACTIONS	267
33.1. ABOUT TRANSACTIONS	267
33.1.1. About Transactions	267
33.1.2. About the Transaction Manager	267
33.1.3. XA Resources and Synchronizations	267
33.1.4. Optimistic and Pessimistic Transactions	267
33.1.5. Write Skew Checks	268
33.1.6. Transactions Spanning Multiple Cache Instances	268
33.2. CONFIGURE TRANSACTIONS	268
33.2.1. Configure Transactions (Library Mode)	268
33.2.2. Configure Transactions (Remote Client-Server Mode)	269
33.3. TRANSACTION RECOVERY	270
33.3.1. Transaction Recovery	270
33.3.2. Transaction Recovery Process	270
33.3.3. Transaction Recovery Example	270
33.4. DEADLOCK DETECTION	271
33.4.1. Deadlock Detection	271

33.4.2. Enable Deadlock Detection	271
CHAPTER 34. CONFIGURE JGROUPS	272
34.1. ABOUT JGROUPS	272
34.2. CONFIGURE RED HAT JBOSS DATA GRID INTERFACE BINDING (REMOTE CLIENT-SERVER MODE)	272
34.2.1. Interfaces	272
34.2.2. Binding Sockets	273
34.2.2.1. Binding Sockets	273
34.2.2.2. Binding a Single Socket Example	273
34.2.2.3. Binding a Group of Sockets Example	273
34.2.3. Configure JGroups Socket Binding	273
34.3. CONFIGURE JGROUPS (LIBRARY MODE)	275
34.3.1. Configure JGroups for Clustered Modes	275
34.3.2. JGroups Transport Protocols	275
34.3.2.1. JGroups Transport Protocols	275
34.3.2.2. The UDP Transport Protocol	275
34.3.2.3. The TCP Transport Protocol	276
34.3.2.4. Using the TCPPing Protocol	276
34.3.3. Pre-Configured JGroups Files	276
34.3.3.1. Pre-Configured JGroups Files	276
34.3.3.2. default-jgroups-udp.xml	276
34.3.3.3. default-jgroups-tcp.xml	277
34.3.3.4. default-jgroups-ec2.xml	278
34.3.3.5. default-jgroups-google.xml	280
34.4. TEST MULTICAST USING JGROUPS	281
34.4.1. Test Multicast Using JGroups	281
34.4.2. Testing With Different Red Hat JBoss Data Grid Versions	281
34.4.3. Testing Multicast Using JGroups	285
CHAPTER 35. USE RED HAT JBOSS DATA GRID WITH AMAZON WEB SERVICES	286
35.1. THE S3_PING JGROUPS DISCOVERY PROTOCOL	286
35.2. S3_PING CONFIGURATION OPTIONS	286
35.2.1. S3_PING Configuration Options	286
35.2.2. Using Private S3 Buckets	286
35.2.3. Using Pre-Signed URLs	287
35.2.3.1. Using Pre-Signed URLs	287
35.2.3.2. Generating Pre-Signed URLs	287
35.2.3.3. Set Pre-Signed URLs Using the Command Line	288
35.2.4. Using Public S3 Buckets	289
35.3. UTILIZING AN ELASTIC IP ADDRESS	289
CHAPTER 36. USE RED HAT JBOSS DATA GRID WITH GOOGLE COMPUTE ENGINE	290
36.1. THE GOOGLE_PING PROTOCOL	290
36.2. GOOGLE_PING CONFIGURATION	290
36.2.1. GOOGLE_PING Configuration	290
36.2.2. Starting the Server in Google Compute Engine	290
36.3. UTILIZING A STATIC IP ADDRESS	291
CHAPTER 37. INTEGRATION WITH THE SPRING FRAMEWORK	292
37.1. INTEGRATION WITH THE SPRING FRAMEWORK	292
37.2. ENABLING SPRING CACHE SUPPORT DECLARATIVELY (LIBRARY MODE)	292
37.3. ENABLING SPRING CACHE SUPPORT DECLARATIVELY (REMOTE CLIENT-SERVER MODE)	292

CHAPTER 38. HIGH AVAILABILITY USING SERVER HINTING	294
38.1. HIGH AVAILABILITY USING SERVER HINTING	294
38.2. ESTABLISHING SERVER HINTING WITH JGROUPS	294
38.3. CONFIGURE SERVER HINTING (REMOTE CLIENT-SERVER MODE)	294
38.4. CONFIGURE SERVER HINTING (LIBRARY MODE)	295
CHAPTER 39. SET UP CROSS-DATACENTER REPLICATION	296
39.1. CROSS-DATACENTER REPLICATION	296
39.2. CROSS-DATACENTER REPLICATION OPERATIONS	296
39.3. CONFIGURE CROSS-DATACENTER REPLICATION	298
39.3.1. Configure Cross-Datcenter Replication (Remote Client-Server Mode)	298
39.3.2. Configure Cross-Datcenter Replication (Library Mode)	299
39.3.2.1. Configure Cross-Datcenter Replication Declaratively	299
39.4. TAKING A SITE OFFLINE	301
39.4.1. Taking a Site Offline	301
39.4.2. Taking a Site Offline	302
39.4.3. Taking a Site Offline via JBoss Operations Network (JON)	302
39.4.4. Taking a Site Offline via the CLI	302
39.4.5. Bring a Site Back Online	303
39.5. STATE TRANSFER BETWEEN SITES	303
39.5.1. State Transfer Between Sites	303
39.5.2. Active-Passive State Transfer	304
39.5.3. Active-Active State Transfer	305
39.5.4. State Transfer Configuration	305
39.6. CONFIGURE MULTIPLE SITE MASTERS	306
39.6.1. Configure Multiple Site Masters	306
39.6.2. Multiple Site Master Operations	306
39.6.3. Configure Multiple Site Masters (Remote Client-Server Mode)	306
39.6.4. Configure Multiple Site Masters (Library Mode)	307
39.7. CROSS-DATACENTER REPLICATION CONCERNS	307
CHAPTER 40. ROLLING UPGRADES	309
40.1. ROLLING UPGRADES	309
40.2. ROLLING UPGRADES USING HOT ROD	309
40.3. ROLLING UPGRADES USING REST	312
40.4. ROLLINGUPGRADEMANAGER OPERATIONS	313
40.5. REMOTECACHESTORE PARAMETERS FOR ROLLING UPGRADES	314
40.5.1. rawValues and RemoteCacheStore	314
40.5.2. hotRodWrapping	314
CHAPTER 41. EXTERNALIZE SESSIONS	315
41.1. EXTERNALIZE SESSIONS	315
41.2. EXTERNALIZE HTTP SESSION FROM JBOSS EAP TO JBOSS DATA GRID	315
41.3. EXTERNALIZE HTTP SESSIONS FROM JBOSS WEB SERVER (JWS) TO JBOSS DATA GRID	317
41.3.1. Externalize HTTP Session from JBoss Web Server (JWS) to JBoss Data Grid	317
41.3.2. Prerequisites	317
41.3.3. Installation	317
41.3.4. Session Management Details	317
41.3.5. Configure the JBoss Web Server Session Manager	318
CHAPTER 42. DATA INTEROPERABILITY	320
42.1. PROTOCOL INTEROPERABILITY	320
42.1.1. Enabling Compatibility Mode	320

CHAPTER 43. HANDLING NETWORK PARTITIONS (SPLIT BRAIN)	321
43.1. NETWORK PARTITION RECOVERY	321
43.2. DETECTING AND RECOVERING FROM A SPLIT-BRAIN PROBLEM	321
43.3. SPLIT BRAIN TIMING: DETECTING A SPLIT	324
43.4. SPLIT BRAIN TIMING: RECOVERING FROM A SPLIT	324
43.5. DETECTING AND RECOVERING FROM SUCCESSIVE CRASHED NODES	325
43.6. NETWORK PARTITION RECOVERY EXAMPLES	325
43.6.1. Network Partition Recovery Examples	325
43.6.2. Distributed 4-Node Cache Example With 3 Owners	326
43.6.3. Distributed 4-Node Cache Example With 2 Owners	327
43.6.4. Distributed 5-Node Cache Example With 3 Owners	328
43.6.5. Replicated 4-Node Cache Example With 4 Owners	331
43.6.6. Replicated 5-Node Cache Example With 5 Owners	332
43.6.7. Replicated 8-Node Cache Example With 8 Owners	334
43.7. CONFIGURE PARTITION HANDLING	340
APPENDIX A. RECOMMENDED JGROUPS VALUES FOR JBOSS DATA GRID	342
A.1. SUPPORTED JGROUPS PROTOCOLS	342
A.2. TCP DEFAULT AND RECOMMENDED VALUES	347
A.3. UDP DEFAULT AND RECOMMENDED VALUES	353
A.4. THE TCPGOSSIP JGROUPS PROTOCOL	359
A.5. TCPGOSSIP CONFIGURATION OPTIONS	360
A.6. JBOSS DATA GRID JGROUPS CONFIGURATION FILES	361
APPENDIX B. HOTROD.PROPERTIES	362
B.1. HOTROD.PROPERTIES	362
APPENDIX C. CONNECTING WITH JCONSOLE	365
C.1. CONNECT TO JDG VIA JCONSOLE	365
APPENDIX D. JMX MBEANS IN RED HAT JBOSS DATA GRID	369
D.1. ACTIVATION	369
D.2. CACHE	369
D.3. CACHECONTAINERSTATS	370
D.4. CACHELOADER	371
D.5. CACHEMANAGER	372
D.6. CACHESTORE	374
D.7. CLUSTERCACHESTATS	374
D.8. DEADLOCKDETECTINGLOCKMANAGER	376
D.9. DISTRIBUTIONMANAGER	377
D.10. INTERPRETER	378
D.11. INVALIDATION	378
D.12. LOCKMANAGER	379
D.13. LOCALTOPOLOGYMANAGER	379
D.14. MASSINDEXER	380
D.15. PASSIVATION	380
D.16. RECOVERYADMIN	381
D.17. ROLLINGUPGRADEMANAGER	382
D.18. RPCMANAGER	382
D.19. STATETRANFERMANAGER	383
D.20. STATISTICS	384
D.21. TRANSACTIONS	385
D.22. TRANSPORT	386
D.23. XSITEADMIN	387

APPENDIX E. CONFIGURATION RECOMMENDATIONS	389
E.1. TIMEOUT VALUES	389
APPENDIX F. PERFORMANCE RECOMMENDATIONS	390
F.1. CONCURRENT STARTUP FOR LARGE CLUSTERS	390
APPENDIX G. REFERENCES	391
G.1. ABOUT CONSISTENCY	391
G.2. ABOUT CONSISTENCY GUARANTEE	391
G.3. ABOUT JBOSS CACHE	391
G.4. ABOUT RELAY2	391
G.5. ABOUT RETURN VALUES	391
G.6. ABOUT RUNNABLE INTERFACES	392
G.7. ABOUT TWO PHASE COMMIT (2PC)	392
G.8. ABOUT KEY-VALUE PAIRS	392
G.9. REQUESTING A FULL BYTE ARRAY	392

PART I. INTRODUCTION

CHAPTER 1. SETTING UP RED HAT JBOSS DATA GRID

1.1. PREREQUISITES

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

1.2. STEPS TO SET UP RED HAT JBOSS DATA GRID

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

Set Up JBoss Data Grid

1. Set Up the Cache Manager

The foundation of a JBoss Data Grid configuration is a cache manager. Cache managers can retrieve cache instances and create cache instances quickly and easily using previously specified configuration templates. For details about setting up a cache manager, refer to the **Cache Manager** section in the JBoss Data Grid *Getting Started Guide*.

2. Set Up JVM Memory Management

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

a. Set Up Eviction

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

b. Set Up Expiration

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

3. Monitor Your Cache

JBoss Data Grid uses logging via JBossLogging to help users monitor their caches.

a. Set Up Logging

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

4. Set Up Cache Modes

Cache modes are used to specify whether a cache is local (simple, in-memory cache) or a clustered cache (replicates state changes over a small subset of nodes). Additionally, if a cache is clustered, either replication, distribution or invalidation mode must be applied to determine how the changes propagate across the subset of nodes. For details, see [Set Up Cache Modes](#).

5. Set Up Locking for the Cache

When replication or distribution is in effect, copies of entries are accessible across multiple nodes. As a result, copies of the data can be accessed or modified concurrently by different threads. To maintain consistency for all copies across nodes, configure locking. For details, see [Set Up Locking for the Cache](#) and [Set Up Isolation Levels](#).

6. Set Up and Configure a Cache Store

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

a. Set Up a Cache Store

The cache store serves as a connection to the persistent store. Cache stores are primarily used to fetch entries from the persistent store and to push changes back to the persistent store. For details, see [Set Up and Configure a Cache Store](#).

b. Set Up Passivation

Passivation stores entries evicted from memory in a cache store. This feature allows entries to remain available despite not being present in memory and prevents potentially expensive write operations to the persistent cache. For details, see [Set Up Passivation](#).

c. Set Up a Cache Writing Strategy

If passivation is disabled, every attempt to write to the cache results in writing to the cache store. This is the default Write-Through cache writing strategy. Set the cache writing strategy to determine whether these cache store writes occur synchronously or asynchronously. For details, see [Set Up Cache Writing](#).

7. Monitor Caches and Cache Managers

JBoss Data Grid includes three primary tools to monitor the cache and cache managers once the data grid is up and running.

a. Set Up JMX

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

b. Access the Administration Console

Red Hat JBoss Data Grid 7.1.0 introduces an Administration Console, allowing for web-based monitoring and management of caches and cache managers. For usage details refer to [Red Hat JBoss Data Grid Administration Console Getting Started](#).

c. Set Up Red Hat JBoss Operations Network (JON)

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



NOTE

The JON plugin has been deprecated in JBoss Data Grid 7.1 and is expected to be removed in a subsequent version.

8. Introduce Topology Information

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

a. Set Up Server Hinting

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

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

PART II. SET UP JVM MEMORY MANAGEMENT

CHAPTER 2. SET UP EVICTION

2.1. ABOUT EVICTION

Eviction is the process of removing entries from memory to prevent running out of memory. Entries that are evicted from memory remain in configured cache stores and the rest of the cluster to prevent permanent data loss. If no cache store is configured, and eviction is enabled, data loss is possible.

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

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

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

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

See also: [Eviction and Expiration Comparison](#)

2.2. EVICTION STRATEGIES

2.2.1. Eviction Strategies

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

Table 2.1. Eviction Strategies

Strategy Name	Operations	Details
EvictionStrategy.NONE	No eviction occurs.	This is the default eviction strategy in Red Hat JBoss Data Grid.
EvictionStrategy.LRU	Least Recently Used eviction strategy. This strategy evicts entries that have not been used for the longest period. This ensures that entries that are reused periodically remain in memory.	

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

2.2.2. LRU Eviction Algorithm Limitations

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

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

2.3. USING EVICTION

2.3.1. Using Eviction

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

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

2.3.2. Initialize Eviction

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

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

Initialize Eviction

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

```
<eviction />
```

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

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

3. Set the Maximum Size to use for Eviction
Set the maximum number of entries allowed in memory by defining the **size** element. The default value is **-1** for unlimited entries. The following demonstrates this step:

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

2.3.3. Eviction Configuration Examples

Eviction may be configured in Red Hat JBoss Data Grid programmatically or via the XML file. Eviction configuration is done on a per-cache basis.

A sample XML configuration for is as follows:

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

2.3.4. Utilizing Memory Based Eviction

Red Hat JBoss Data Grid 7 introduced memory based eviction, allowing eviction of entries based on memory usage of the entries instead of the number of entries. This can be particularly useful if the entries vary in size.

Key/Value Limitations

Only keys and values that are stored as primitives, primitive wrappers (such as **java.lang.Integer**), **java.lang.String** instances, or an **Array** of these values may be used with memory based eviction.

Due to this limitation if custom classes are used then either **store-as-binary** must be enabled on the cache, or the data from the custom class may be serialized, storing it in a byte array.

Compatibility mode prevents serialization into byte arrays, and as such these two features are mutually exclusive.

Eviction Strategy Limitations

Memory based eviction is only supported with the **LRU** eviction strategy.

Enabling Memory Based Eviction

This eviction method may be used by defining **MEMORY** as the eviction type, as seen in the following example:

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

```
<eviction size="10000000000" strategy="LRU" type="MEMORY"/>
</local-cache>
```

2.3.5. Eviction and Passivation

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

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

CHAPTER 3. SET UP EXPIRATION

3.1. ABOUT EXPIRATION

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

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

Expiration can be specified on a per-entry or per-cache basis and the per-entry configuration overrides per-cache configurations. If expiration is configured at the cache level, then the expiration defaults apply to all entries which do not explicitly specify a **lifespan** or **max-idle** value.

If expiration is not configured at the cache level, cache entries are created immortal (i.e. they will never expire) by default. Any entries that have **lifespan** or **max-idle** defined are mortal, as they will eventually be removed from the cache once one of these conditions are met.

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

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

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



IMPORTANT

It is not recommended to use **max-idle** in clustered mode. This is because max-idle is not updated on all nodes at the same time, and thus JBoss Data Grid can't reliably expire entries across the cluster. For more information on the limitations of using **max-idle** in clustered mode see this [knowledgebase solution](#).

3.2. EXPIRATION OPERATIONS

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

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

3.3. EVICTION AND EXPIRATION COMPARISON

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

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

3.4. CACHE ENTRY EXPIRATION BEHAVIOR

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

- An entry is passivated/overflowed to disk and is discovered to have expired.
- The expiration maintenance thread discovers that an entry it has found is expired.

If a user requests an entry that is expired but not yet removed, a null value is sent to the user. This mechanism ensures that the user never receives an expired entry. The entry is eventually removed by the expiration thread.

3.5. CONFIGURE EXPIRATION

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

Configure Expiration

1. Add the Expiration Tag

Add the expiration tag to your project's cache tags as follows:

```
<expiration />
```

2. Set the Expiration Lifespan

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

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

3. Set the Maximum Idle Time

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

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

3.6. TROUBLESHOOTING EXPIRATION

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

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

- An entry is passivated/overflowed to disk and is discovered to have expired.
- The expiration maintenance thread discovers that an entry it has found is expired.

Any attempt to use `get()` or `containsKey()` for the expired entry causes JBoss Data Grid to return a null value. The expired entry is later removed by the expiration thread.

PART III. MONITOR YOUR CACHE

CHAPTER 4. SET UP LOGGING

4.1. ABOUT LOGGING

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

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

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

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

4.2. SUPPORTED APPLICATION LOGGING FRAMEWORKS

4.2.1. Supported Application Logging Frameworks

Red Hat JBoss LogManager supports the following logging frameworks:

- JBoss Logging, which is included with Red Hat JBoss Data Grid 7.
- [Apache Commons Logging](#)
- [Simple Logging Facade for Java \(SLF4J\)](#)
- [Apache log4j](#)
- [Java SE Logging \(java.util.logging\)](#)

4.2.2. About JBoss Logging

JBoss Logging is the application logging framework that is included in JBoss Enterprise Application Platform 7. As a result of this inclusion, Red Hat JBoss Data Grid 7 also uses JBoss Logging.

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

4.2.3. JBoss Logging Features

JBossLogging includes the following features:

- Provides an innovative, easy to use *typed* logger.
- Full support for internationalization and localization. Translators work with message bundles in properties files while developers can work with interfaces and annotations.

- Build-time tooling to generate typed loggers for production, and runtime generation of typed loggers for development.

4.3. BOOT LOGGING

4.3.1. Boot Logging

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

4.3.2. Configure Boot Logging

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

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

4.3.3. Default Log File Locations

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

Table 4.1. Default Log File Locations

Log File	Location	Description
<i>boot.log</i>	<i>\$JDG_HOME/standalone/log/</i>	<p>The Server Boot Log. Contains log messages related to the start up of the server.</p> <p>By default this file is prepended to the <i>server.log</i>. This file may be created independently of the <i>server.log</i> by defining the org.jboss.boot.log property in <i>logging.properties</i>.</p>
<i>server.log</i>	<i>\$JDG_HOME/standalone/log/</i>	The Server Log. Contains all log messages once the server has launched.

4.4. LOGGING ATTRIBUTES

4.4.1. About Log Levels

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

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

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

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

4.4.2. Supported Log Levels

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

Table 4.2. Supported Log Levels

Log Level	Value	Description
FINEST	300	-
FINER	400	-
TRACE	400	Used for messages that provide detailed information about the running state of an application. TRACE level log messages are captured when the server runs with the TRACE level enabled.
DEBUG	500	Used for messages that indicate the progress of individual requests or activities of an application. DEBUG level log messages are captured when the server runs with the DEBUG level enabled.
FINE	500	-
CONFIG	700	-

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

4.4.3. About Log Categories

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

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

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

4.4.4. About the Root Logger

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

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

4.4.5. About Log Handlers

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

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

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

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

4.4.6. Log Handler Types

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

Table 4.3. Log Handler Types

Log Handler Type	Description	Use Case
Console	Console log handlers write log messages to either the host operating system's standard out (stdout) or standard error (stderr) stream. These messages are displayed when JBoss Data Grid is run from a command line prompt.	The Console log handler is preferred when JBoss Data Grid is administered using the command line. In such a case, the messages from a Console log handler are not saved unless the operating system is configured to capture the standard out or standard error stream.
File	File log handlers are the simplest log handlers. Their primary use is to write log messages to a specified file.	File log handlers are most useful if the requirement is to store all log entries according to the time in one place.

Log Handler Type	Description	Use Case
Periodic	Periodic file handlers write log messages to a named file until a specified period of time has elapsed. Once the time period has elapsed, the specified time stamp is appended to the file name. The handler then continues to write into the newly created log file with the original name.	The Periodic file handler can be used to accumulate log messages on a weekly, daily, hourly or other basis depending on the requirements of the environment.
Size	Size log handlers write log messages to a named file until the file reaches a specified size. When the file reaches a specified size, it is renamed with a numeric prefix and the handler continues to write into a newly created log file with the original name. Each size log handler must specify the maximum number of files to be kept in this fashion.	The Size handler is best suited to an environment where the log file size must be consistent.
Async	Async log handlers are wrapper log handlers that provide asynchronous behavior for one or more other log handlers. These are useful for log handlers that have high latency or other performance problems such as writing a log file to a network file system.	The Async log handlers are best suited to an environment where high latency is a problem or when writing to a network file system.
Custom	Custom log handlers enable you to configure new types of log handlers that have been implemented. A custom handler must be implemented as a Java class that extends <code>java.util.logging.Handler</code> and be contained in a module.	Custom log handlers create customized log handler types and are recommended for advanced users.

4.4.7. Selecting Log Handlers

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

- The **Console** log handler is preferred when JBoss Data Grid is administered using the command line. In such a case, errors and log messages appear on the console window and are not saved unless separately configured to do so.

- The **File** log handler is used to direct log entries into a specified file. This simplicity is useful if the requirement is to store all log entries according to the time in one place.
- The **Periodic** log handler is similar to the **File** handler but creates files according to the specified period. As an example, this handler can be used to accumulate log messages on a weekly, daily, hourly or other basis depending on the requirements of the environment.
- The **Size** log handler also writes log messages to a specified file, but only while the log file size is within a specified limit. Once the file size reaches the specified limit, log files are written to a new log file. This handler is best suited to an environment where the log file size must be consistent.
- The **Async** log handler is a wrapper that forces other log handlers to operate asynchronously. This is best suited to an environment where high latency is a problem or when writing to a network file system.
- The **Custom** log handler creates new, customized types of log handlers. This is an advanced log handler.

4.4.8. About Log Formatters

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

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

4.5. LOGGING SAMPLE CONFIGURATIONS

4.5.1. Logging Sample Configuration Location

All of the sample configurations presented in this section should be placed inside the server's configuration file, typically either *standalone.xml* or *clustered.xml* for standalone instances, or *domain.xml* for managed domain instances.

4.5.2. Sample XML Configuration for the Root Logger

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

Procedure: Configure the Root Logger

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

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
  <root-logger>
    <level name="INFO"/>
  </root-logger>
</subsystem>
```

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

```
<subsystem xmlns="urn:jboss:domain:logging:3.0">
  <root-logger>
    <level name="INFO"/>
    <handlers>
  </root-logger>
</subsystem>
```

```

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

```

4.5.3. Sample XML Configuration for a Log Category

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

Configure a Log Category

```

<subsystem xmlns="urn:jboss:domain:logging:3.0">
  <logger category="com.company.accounts.rec" use-parent-handlers="true">
    <level name="WARN"/>
    <handlers>
      <handler name="accounts-rec"/>
    </handlers>
  </logger>
</subsystem>

```

1. Use the **category** property to specify the log category from which log messages will be captured.
The **use-parent-handlers** is set to **"true"** by default. When set to **"true"**, this category will use the log handlers of the root logger in addition to any other assigned handlers.
2. Use the **level** property to set the maximum level of log message that the log category records.
3. The **handlers** element contains a list of log handlers.

4.5.4. Sample XML Configuration for a Console Log Handler

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

Configure the Console Log Handler

```

<subsystem xmlns="urn:jboss:domain:logging:3.0">
  <console-handler name="CONSOLE" autoflush="true">
    <level name="INFO"/>
    <encoding value="UTF-8"/>
    <target name="System.out"/>
    <filter-spec value="not(match(&quot;;JBAS.*&quot;;))"/>
    <formatter>
      <pattern-formatter pattern="%K{level}%d{HH:mm:ss,SSS} %-5p [%c]
(%t) %s%E%n"/>
    </formatter>
  </console-handler>
</subsystem>

```

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

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

2. Set the **level** Property
The **level** property sets the maximum level of log messages recorded.
3. Set the **encoding** Output
Use **encoding** to set the character encoding scheme to be used for the output.
4. Define the **target** Value
The **target** property defines the system output stream where the output of the log handler goes. This can be **System.err** for the system error stream, or **System.out** for the standard out stream.
5. Define the **filter-spec** Property
The **filter-spec** property is an expression value that defines a filter. The example provided defines a filter that does not match a pattern: **not(match("JBAS.*"))**.
6. Specify the **formatter**
Use **formatter** to list the log formatter used by the log handler.

4.5.5. Sample XML Configuration for a File Log Handler

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

Configure the File Log Handler

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

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

When **autoflush** is set to **"true"** the log messages will be sent to the handler's target immediately upon request.
2. Set the **level** Property
The **level** property sets the maximum level of log message that the root logger records.
3. Set the **encoding** Output
Use **encoding** to set the character encoding scheme to be used for the output.
4. Set the **file** Object
The **file** object represents the file where the output of this log handler is written to. It has two configuration properties: **relative-to** and **path**.

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

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

5. Specify the **formatter**

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

6. Set the **append** Property

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

4.5.6. Sample XML Configuration for a Periodic Log Handler

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

Configure the Periodic Log Handler

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

1. Add the Periodic Log Handler Identifier Information

The **name** property sets the unique identifier for this log handler.

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

2. Set the **level** Property

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

3. Set the **encoding** Output

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

4. Specify the **formatter**

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

5. Set the **file** Object

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

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

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

6. Set the **suffix** Value

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

7. Set the **append** Property

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

4.5.7. Sample XML Configuration for a Size Log Handler

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

Configure the Size Log Handler

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

1. Add the Size Log Handler Identifier Information

The **name** property sets the unique identifier for this log handler.

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

2. Set the **level** Property

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

3. Set the **encoding** Object

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

4. Set the **file** Object

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

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

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

5. Specify the **rotate-size** Value

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

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

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

7. Specify the **formatter**

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

8. Set the **append** Property

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

4.5.8. Sample XML Configuration for a Async Log Handler

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

Configure the Async Log Handler

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

1. The **name** property sets the unique identifier for this log handler.
2. The **level** property sets the maximum level of log message that the root logger records.
3. The **queue-length** defines the maximum number of log messages that will be held by this handler while waiting for sub-handlers to respond.
4. The **overflow-action** defines how this handler responds when its queue length is exceeded. This can be set to **BLOCK** or **DISCARD**. **BLOCK** makes the logging application wait until there is available space in the queue. This is the same behavior as an non-async log handler. **DISCARD**

allows the logging application to continue but the log message is deleted.

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

PART IV. SET UP CACHE MODES

CHAPTER 5. CACHE MODES

5.1. CACHE MODES

Red Hat JBoss Data Grid provides two modes:

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

5.2. ABOUT CACHE CONTAINERS

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

The following procedure demonstrates a sample cache container configuration:

How to Configure the Cache Container

```
<subsystem xmlns="urn:infinispan:server:core:8.4"
  default-cache-container="local">
  <cache-container name="local"
    default-cache="default"
    statistics="true"
    start="EAGER">
    <local-cache name="default"
      statistics="false">
      <!-- Additional configuration information here -->
    </local-cache>
  </cache-container>
</subsystem>
```

1. Configure the Cache Container

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

- a. The **name** parameter defines the name of the cache container.
- b. The **default-cache** parameter defines the name of the default cache used with the cache container.
- c. The **statistics** attribute is optional and is **true** by default. Statistics are useful in monitoring JBoss Data Grid via JMX or JBoss Operations Network, however they adversely affect performance. Disable this attribute by setting it to **false** if it is not required.

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

2. Configure Per-cache Statistics

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

5.3. LOCAL MODE

5.3.1. Local Mode

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

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

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

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

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

5.3.2. Configure Local Mode

A local cache can be added to any cache container in both Library Mode and Remote Client-Server Mode. The following example demonstrates how to add the **local-cache** element.

The **local-cache** Element

```
<cache-container name="local"
    default-cache="default"
    statistics="true">
  <local-cache name="default"
    statistics="true">
    <!-- Additional configuration information here -->
  </local-cache>
</cache-container>
```

The **local-cache** element specifies information about the local cache used with the cache container using the following parameters: . The **name** parameter specifies the name of the local cache to use. . If **statistics** are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the **statistics** attribute to **false**.

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

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

5.4. CLUSTERED MODES

5.4.1. Clustered Modes

Red Hat JBoss Data Grid offers the following clustered modes:

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

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

5.4.2. Asynchronous and Synchronous Operations

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

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

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

By default the cluster is configured for synchronous operations.

5.4.3. About Asynchronous Communications

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

Asynchronous Communications Example Configuration

```
<replicated-cache name="default"
    statistics="true">
    <!-- Additional configuration information here -->
</replicated-cache>
```



NOTE

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

5.4.4. Cache Mode Troubleshooting

5.4.4.1. Invalid Data in ReadExternal

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

5.4.4.2. Cluster Physical Address Retrieval

How can the physical addresses of the cluster be retrieved?

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

CHAPTER 6. SET UP DISTRIBUTION MODE

6.1. ABOUT DISTRIBUTION MODE

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

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

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

6.2. DISTRIBUTION MODE'S CONSISTENT HASH ALGORITHM

The hashing algorithm in Red Hat JBoss Data Grid is based on consistent hashing. The term consistent hashing is still used for this implementation, despite some divergence from a traditional consistent hash.

Distribution mode uses a consistent hash algorithm to select a node from the cluster to store entries upon. The consistent hash algorithm is configured with the number of copies of each cache entry to be maintained within the cluster. Unlike generic consistent hashing, the implementation used in JBoss Data Grid splits the key space into fixed segments. The number of segments is configurable using **numSegments** and cannot be changed without restarting the cluster. The mapping of keys to segments is also fixed — a key maps to the same segment, regardless of how the topology of the cluster changes.

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

Each hash segment is mapped to a list of nodes called owners. The order is important because the first owner (also known as the primary owner) has a special role in many cache operations (for example, locking). The other owners are called backup owners. There is no rule about mapping segments to owners, although the hashing algorithms simultaneously balance the number of segments allocated to each node and minimize the number of segments that have to move after a node joins or leaves the cluster.

6.3. LOCATING ENTRIES IN DISTRIBUTION MODE

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

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

6.4. RETURN VALUES IN DISTRIBUTION MODE

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

6.5. CONFIGURE DISTRIBUTION MODE

Distribution mode is a clustered mode in Red Hat JBoss Data Grid. Distribution mode can be added to any cache container, in both Library Mode and Remote Client-Server Mode, using the following procedure:

The `distributed-cache` Element

```
<cache-container name="clustered"
  default-cache="default"
  statistics="true">
  <!-- Additional configuration information here -->
  <distributed-cache name="default"
    statistics="true">
    <!-- Additional configuration information here -->
  </distributed-cache>
</cache-container>
```

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

1. The `name` parameter provides a unique identifier for the cache.
2. If `statistics` are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the `statistics` attribute to `false`.



IMPORTANT

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

6.6. SYNCHRONOUS AND ASYNCHRONOUS DISTRIBUTION

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

Communication Mode example

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

CHAPTER 7. SET UP REPLICATION MODE

7.1. ABOUT REPLICATION MODE

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

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

7.2. OPTIMIZED REPLICATION MODE USAGE

Replication mode is used for state sharing across a cluster; however, if you have a replicated cache and a large number of nodes are in use then there will be many writes to the replicated cache to keep all of the nodes synchronized. The amount of work performed will depend on many factors and on the specific use case, and for this reason it is recommended to ensure that each workload is tested thoroughly to determine if replication mode will be beneficial with the number of planned nodes. For many situations replication mode is not recommended once there are ten servers; however, in some workloads, such as if load read is important, this mode may be beneficial.

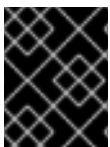
Red Hat JBoss Data Grid can be configured to use UDP multicast, which improves performance to a limited degree for larger clusters.

7.3. CONFIGURE REPLICATION MODE

Replication mode is a clustered cache mode in Red Hat JBoss Data Grid. Replication mode can be added to any cache container, in both Library Mode and Remote Client-Server Mode, using the following procedure.

The `replicated-cache` Element

```
<cache-container name="clustered"
  default-cache="default"
  statistics="true">
  <!-- Additional configuration information here -->
  <replicated-cache name="default"
    statistics="true">
    <!-- Additional configuration information here -->
  </replicated-cache>
</cache-container>
```



IMPORTANT

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

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

1. The `name` parameter provides a unique identifier for the cache.

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

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

7.4. SYNCHRONOUS AND ASYNCHRONOUS REPLICATION

7.4.1. Synchronous and Asynchronous Replication

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

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

7.4.2. Troubleshooting Asynchronous Replication Behavior

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

Use one of the following to remedy this problem:

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

7.5. THE REPLICATION QUEUE

7.5.1. The Replication Queue

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

- Previously set intervals.
- The queue size exceeding the number of elements.

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

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

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

A replication queue is used in conjunction with asynchronous mode.

7.5.2. Replication Queue Usage

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

- Disable asynchronous marshalling.
- Set the **max-threads** count value to **1** for the **executor** attribute of the **transport** element. The **executor** is only available in Library Mode, and is therefore defined in its configuration file as follows:

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

To implement either of these solutions, the replication queue must be in use in asynchronous mode. Asynchronous mode can be set by defining **mode="ASYNC"**, as seen in the following example:

Replication Queue in Asynchronous Mode

```
<replicated-cache name="asyncCache"
    mode="ASYNC"
    statistics="true"
    <!-- Additional configuration information here -->
</replicated-cache>
```

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

7.6. ABOUT REPLICATION GUARANTEES

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

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

7.7. REPLICATION TRAFFIC ON INTERNAL NETWORKS

Some cloud providers charge less for traffic over internal *IP* addresses than for traffic over public *IP* addresses, or do not charge at all for internal network traffic (for example,). To take advantage of lower rates, you can configure Red Hat JBoss Data Grid to transfer replication traffic using the internal network.

With such a configuration, it is difficult to know the internal *IP* address you are assigned. JBoss Data Grid uses JGroups interfaces to solve this problem.

CHAPTER 8. SET UP INVALIDATION MODE

8.1. ABOUT INVALIDATION MODE

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

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

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

8.2. CONFIGURE INVALIDATION MODE

Invalidation mode is a clustered mode in Red Hat JBoss Data Grid. Invalidation mode can be added to any cache container, in both Library Mode and Remote Client-Server Mode, using the following procedure:

The `invalidation-cache` Element

```
<cache-container name="local"
  default-cache="default"
  statistics="true">
  <invalidation-cache name="default"
    statistics="true">
    <!-- Additional configuration information here -->
  </invalidation-cache>
</cache-container>
```

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

1. The **name** parameter provides a unique identifier for the cache.
2. If **statistics** are enabled at the container level, per-cache statistics can be selectively disabled for caches that do not require monitoring by setting the **statistics** attribute to **false**.



IMPORTANT

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

For details about the `cache-container` see the appropriate chapter.

8.3. SYNCHRONOUS/ASYNCHRONOUS INVALIDATION

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

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

8.4. THE L1 CACHE AND INVALIDATION

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

CHAPTER 9. STATE TRANSFER

9.1. STATE TRANSFER

State transfer is a basic data grid or clustered cache functionality. Without state transfer, data would be lost as nodes are added to or removed from the cluster.

State transfer adjusts the cache's internal state in response to a change in a cache membership. The change can be when a node joins or leaves, when two or more cluster partitions merge, or a combination of joins, leaves, and merges. State transfer occurs automatically in Red Hat JBoss Data Grid whenever a node joins or leaves the cluster.

In Red Hat JBoss Data Grid's replication mode, a new node joining the cache receives the entire cache state from the existing nodes. In distribution mode, the new node receives only a part of the state from the existing nodes, and the existing nodes remove some of their state in order to keep **owners** copies of each key in the cache (as determined through consistent hashing). In invalidation mode the initial state transfer is similar to replication mode, the only difference being that the nodes are not guaranteed to have the same state. When a node leaves, a replicated mode or invalidation mode cache does not perform any state transfer. A distributed cache needs to make additional copies of the keys that were stored on the leaving nodes, again to keep **owners** copies of each key.

A State Transfer transfers both in-memory and persistent state by default, but both can be disabled in the configuration. When State Transfer is disabled a **ClusterLoader** must be configured, otherwise a node will become the owner or backup owner of a key without the data being loaded into its cache. In addition, if State Transfer is disabled in distributed mode then a key will occasionally have less than **owners** owners.

9.2. NON-BLOCKING STATE TRANSFER

Non-Blocking State Transfer in Red Hat JBoss Data Grid minimizes the time in which a cluster or node is unable to respond due to a state transfer in progress. Non-blocking state transfer is a core architectural improvement with the following goals:

- Minimize the interval(s) where the entire cluster cannot respond to requests because of a state transfer in progress.
- Minimize the interval(s) where an existing member stops responding to requests because of a state transfer in progress.
- Allow state transfer to occur with a drop in the performance of the cluster. However, the drop in the performance during the state transfer does not throw any exception, and allows processes to continue.
- Allows a **GET** operation to successfully retrieve a key from another node without returning a null value during a progressive state transfer.

For simplicity, the total order-based commit protocol uses a blocking version of the currently implemented state transfer mechanism. The main differences between the regular state transfer and the total order state transfer are:

- The blocking protocol queues the transaction delivery during the state transfer.
- State transfer control messages (such as `CacheTopologyControlCommand`) are sent according to the total order information.

The total order-based commit protocol works with the assumption that all the transactions are delivered in the same order and they see the same data set. So, no transactions are validated during the state transfer because all the nodes must have the most recent key or values in memory.

Using the state transfer and blocking protocol in this manner allows the state transfer and transaction delivery on all on the nodes to be synchronized. However, transactions that are already involved in a state transfer (sent before the state transfer began and delivered after it concludes) must be resent. When resent, these transactions are treated as new joiners and assigned a new total order value.

9.3. SUPPRESS STATE TRANSFER VIA JMX

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

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

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

9.4. THE REBALANCINGENABLED ATTRIBUTE

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

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

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

```
<state-transfer enabled="true" await-initial-transfer="false"/>
```

PART V. ENABLING APIS

CHAPTER 10. ENABLING APIS DECLARATIVELY

10.1. ENABLING APIS DECLARATIVELY

The various APIs that JBoss Data Grid provides are fully documented in the JBoss Data Grid *Developer Guide*; however, Administrators can enable these declaratively by adding elements to the configuration file. The following sections discuss methods on implementing the various APIs.

10.2. BATCHING API

Batching allows atomicity and some characteristics of a transaction, but does not allow full-blown JTA or XA capabilities. Batching is typically lighter and cheaper than a full-blown transaction, and should be used whenever the only participant in the transaction is the JBoss Data Grid cluster. If the transaction involves multiple systems then JTA Transactions should be used. For example, consider a transaction which transfers money from one bank account to another. If both accounts are stored within the JBoss Data Grid cluster then batching could be used; however, if only one account is inside the cluster, with the second being in an external database, then distributed transactions are required.



NOTE

Transaction batching is only available in JBoss Data Grid's Library Mode.

Enabling the Batching API

Batching may be enabled on a per-cache basis by defining a transaction mode of **BATCH**. The following example demonstrates this:

```
<local-cache name="batchingCache">  
  <transaction mode="BATCH"/>  
</local-cache>
```

By default invocation batching is disabled; in addition, a transaction manager is not required to use batching.

10.3. GROUPING API

The grouping API allows a group of entries to be co-located on the same node, instead of the default behavior of having each entry being stored on a node corresponding to a calculated hash code of the entry. By default JBoss Data Grid will take a hash code of each key when it is stored and map that key to a hash segment; this allows an algorithm to be used to determine the node that contains the key, allowing each node in the cluster to know which node contains the key without distributing ownership information. This behavior reduces overhead and improves redundancy as the ownership information does not need to be replicated should a node fail.

By enabling the grouping API the hash of the key is ignored when deciding which node to store the entry on. Instead, a hash of the group is obtained and used in its place, while the hash of the key is used internally to prevent performance degradation. When the group API is in use every node can still determine the owners of the key, and due to this reason the group may not be manually specified. A group may either be intrinsic to the entry, generated by the key class, or extrinsic to the entry, generated by an external function.

Enabling the Grouping API

The grouping API may be enabled on a per-cache basis by adding the **groups** element as seen in the following example:

```
<distributed-cache name="groupingCache">
  <groups enabled="true"/>
</distributed-cache>
```

Defining an Extrinsic Group

Assuming a custom **Grouper** exists it may be defined by passing in the classname as seen below:

```
<distributed-cache name="groupingCache">
  <groups enabled="true">
    <grouper class="com.acme.KXGrouper" />
  </groups>
</distributed-cache>
```

10.4. EXTERNALIZABLE API

10.4.1. The Externalizable API

An **Externalizer** is a class that can:

- Marshall a given object type to a byte array.
- Unmarshall the contents of a byte array into an instance of the object type.

Externalizers are used by Red Hat JBoss Data Grid and allow users to specify how their object types are serialized. The marshalling infrastructure used in Red Hat JBoss Data Grid builds upon JBoss Marshalling and provides efficient payload delivery and allows the stream to be cached. The stream caching allows data to be accessed multiple times, whereas normally a stream can only be read once.

The Externalizable interface uses and extends serialization. This interface is used to control serialization and deserialization in Red Hat JBoss Data Grid.

10.4.2. Register the Advanced Externalizer (Declaratively)

After the advanced externalizer is set up, register it for use with Red Hat JBoss Data Grid. This registration is done declaratively (via XML) as follows:

Register the Advanced Externalizer

```
<infinispan>
  <cache-container>
    <serialization>
      <advanced-externalizer class="Book$BookExternalizer" />
    </serialization>
  </cache-container>
</infinispan>
```

1. Add the **serialization** element to the **cache-container** element.

2. Add the **advanced-externalizer** element, defining the custom Externalizer with the **class** attribute. Replace the **Book\$BookExternalizer** values as required.

10.4.3. Custom Externalizer ID Values

10.4.3.1. Custom Externalizer ID Values

Advanced externalizers can be assigned custom IDs if desired. Some ID ranges are reserved for other modules or frameworks and must be avoided:

Table 10.1. Reserved Externalizer ID Ranges

ID Range	Reserved For
1000-1099	The Infinispan Tree Module
1100-1199	Red Hat JBoss Data Grid Server modules
1200-1299	Hibernate Infinispan Second Level Cache
1300-1399	JBoss Data Grid Lucene Directory
1400-1499	Hibernate OGM
1500-1599	Hibernate Search
1600-1699	Infinispan Query Module
1700-1799	Infinispan Remote Query Module
1800-1849	JBoss Data Grid Scripting Module
1850-1899	JBoss Data Grid Server Event Logger Module
1900-1999	JBoss Data Grid Remote Store

10.4.3.2. Customize the Externalizer ID (Declaratively)

Customize the advanced externalizer ID declaratively (via XML) as follows:

Customizing the Externalizer ID (Declaratively)

```
<infinispan>
  <cache-container>
    <serialization>
      <advanced-externalizer id="123"
                           class="Book$BookExternalizer"/>
    </serialization>
  </cache-container>
</infinispan>
```


1. Add the **serialization** element to the **cache-container** element.
2. Add the **advanced-externalizer** element to add information about the new advanced externalizer.
3. Define the externalizer ID using the **id** attribute. Ensure that the selected ID is not from the range of IDs reserved for other modules.
4. Define the externalizer class using the **class** attribute. Replace the **Book\$BookExternalizer** values as required.

CHAPTER 11. SET UP AND CONFIGURE THE INFINISPAN QUERY API

11.1. SET UP INFINISPAN QUERY

11.1.1. Infinispan Query Dependencies in Library Mode

To use the JBoss Data Grid Infinispan Query via Maven, add the following dependencies:

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

Non-Maven users must install all of the *infinispan-embedded-query.jar* and *infinispan-embedded.jar* files from the JBoss Data Grid distribution.



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.

11.2. INDEXING MODES

11.2.1. Managing Indexes

In Red Hat JBoss Data Grid's Query Module there are two options for storing indexes:

1. Each node can maintain an individual copy of the global index.
2. The index can be shared across all nodes.

When the indexes are stored locally, by setting **indexLocalOnly** to **true**, each write to cache must be forwarded to all other nodes so that they can update their indexes. If the index is shared, by setting **indexLocalOnly** to **false**, only the node where the write originates is required to update the shared index.

Lucene provides an abstraction of the directory structure called **directory provider**, which is used to store the index. The index can be stored, for example, as in-memory, on filesystem, or in distributed cache.

11.2.2. Managing the Index in Local Mode

In local mode, any Lucene Directory implementation may be used. The `indexLocalOnly` option is meaningless in local mode.

11.2.3. Managing the Index in Replicated Mode

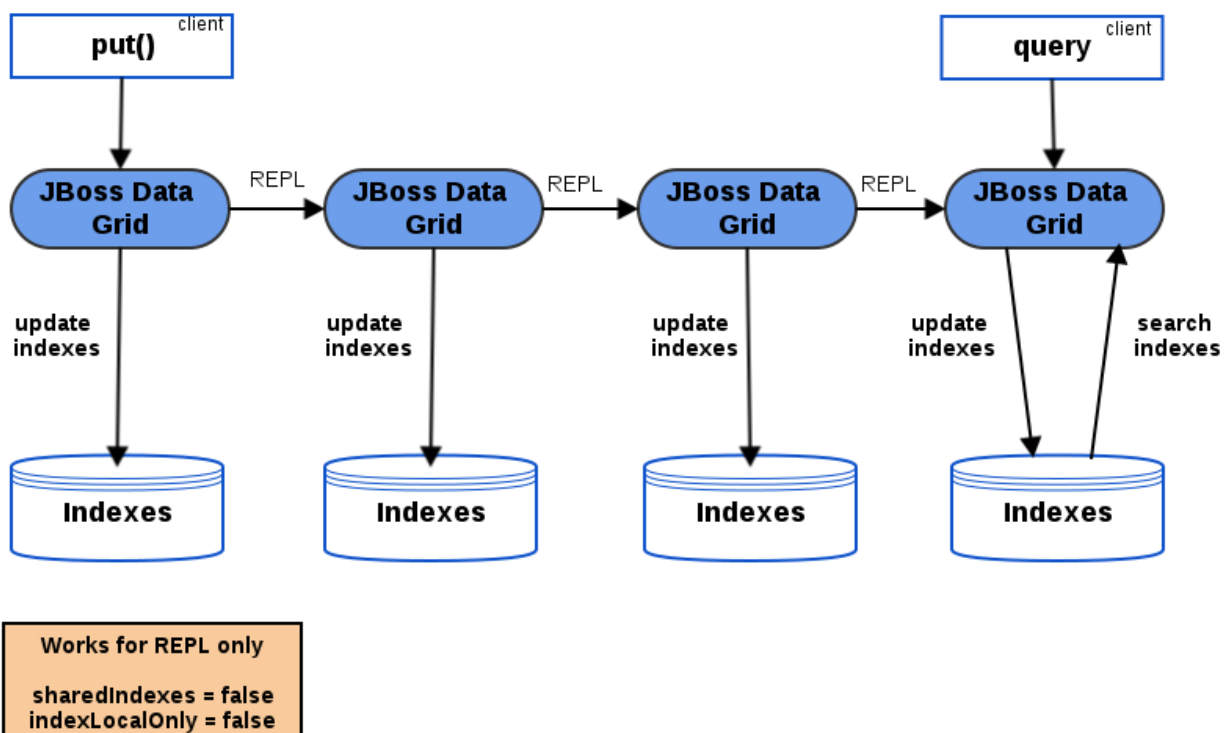
In replication mode, each node can store its own local copy of the index. To store indexes locally on each node, set `indexLocalOnly` to `false`, so that each node will apply the required updates it receives from other nodes in addition to the updates started locally.

Any Directory implementation can be used. When a new node is started it must receive an up to date copy of the index. Usually this can be done via resync, however being an external operation, this may result in a slightly out of sync index, particularly where updates are frequent.

Alternatively, if a shared storage for indexes is used (see [Infinispan Directory Provider](#)), `indexLocalOnly` must be set to `true` so that each node will only apply the changes originated locally. While there is no risk of having an out of sync index, this causes contention on the node used for updating the index.

The following diagram demonstrates a replicated deployment where each node has a local index.

Figure 11.1. Replicated Cache Querying

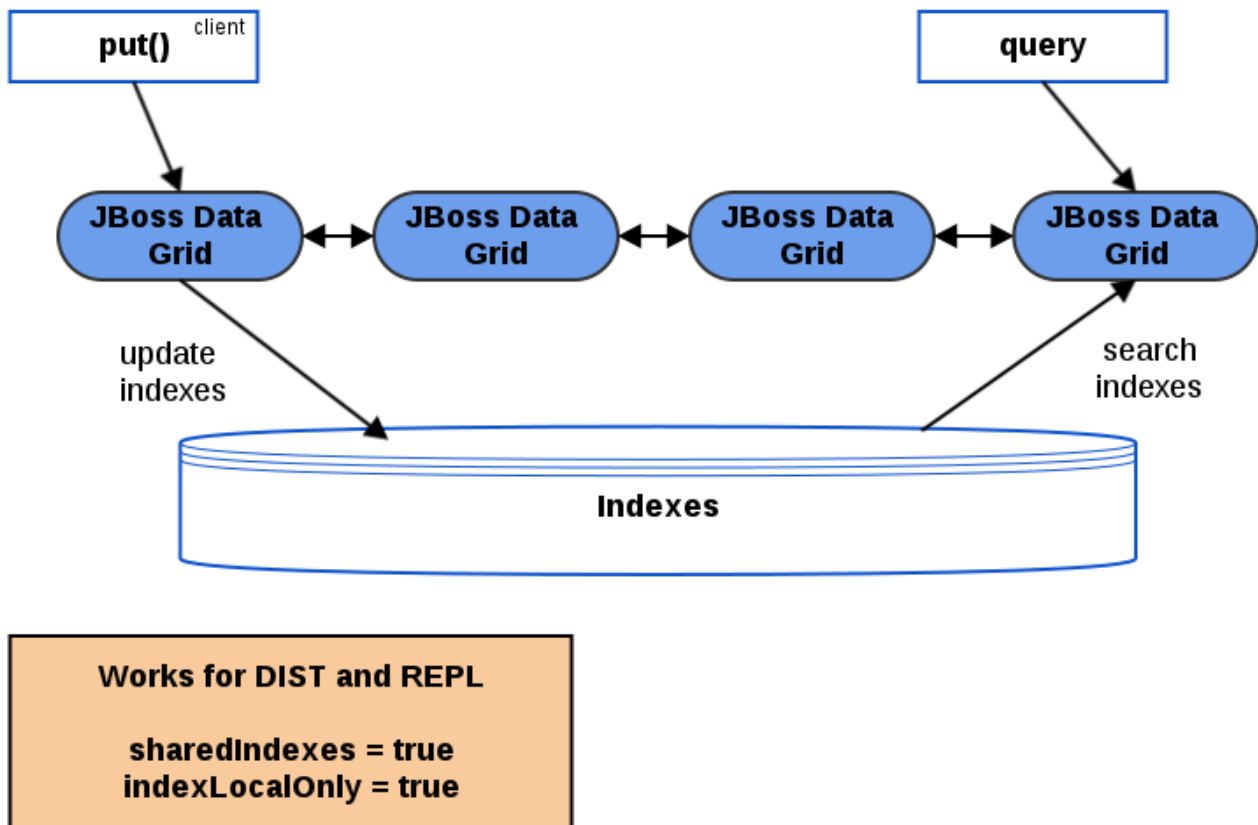


11.2.4. Managing the Index in Distribution Mode

In both Distribution modes, the shared index must be used, with the `indexLocalOnly` set to `true`.

The following diagram shows a deployment with a shared index.

Figure 11.2. Querying with a Shared Index



11.2.5. Managing the Index in Invalidation Mode

Indexing and searching of elements in Invalidation mode is not supported.

11.3. DIRECTORY PROVIDERS

11.3.1. Directory Providers

The following directory providers are supported in Infinispan Query:

- RAM Directory Provider
- Filesystem Directory Provider
- Infinispan Directory Provider

11.3.2. RAM Directory Provider

Storing the global index locally in Red Hat JBoss Data Grid's Query Module allows each node to

- maintain its own index.
- use **Lucene's** in-memory or filesystem-based index directory.

The following example demonstrates an in-memory, RAM-based index store:

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

```
<indexing index="LOCAL">
  <property name="default.directory_provider">ram</property>
</indexing>
</local-cache>
```

11.3.3. Filesystem Directory Provider

To configure the storage of indexes, set the appropriate properties when enabling indexing in the JBoss Data Grid configuration.

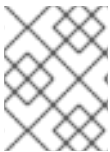
This example shows a disk-based index store:

Disk-based Index Store

```
<local-cache name="indexesInInfinispan">
  <indexing index="ALL">
    <property name="default.directory_provider">filesystem</property>
    <property name="default.indexBase">/tmp/ispn_index</property>
  </indexing>
</local-cache>
```

11.3.4. Infinispan Directory Provider

In addition to the **Lucene** directory implementations, Red Hat JBoss Data Grid also ships with an **infinispan-directory** module.

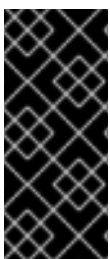


NOTE

Red Hat JBoss Data Grid only supports **infinispan-directory** in the context of the Querying feature, not as a standalone feature.

The **infinispan-directory** allows **Lucene** to store indexes within the distributed data grid. This allows the indexes to be distributed, stored in-memory, and optionally written to disk using the cache store for durability.

Sharing the same index instance using the Infinispan **Directory Provider** introduces a write contention point, as only one instance can write on the same index at the same time.



IMPORTANT

By default the **exclusive_index_use** is set to **true**, as this provides major performance increases; however, if external applications access the same index in use by Infinispan this property must be set to **false**. The default value is recommended for the majority of applications and use cases due to the performance increases, so only change this if absolutely necessary.

InfinispanIndexManager provides a default back end that sends all updates to master node which later applies the updates to the index. In case of master node failure, the update can be lost, therefore keeping the cache and index non-synchronized. Non-default back ends are not supported.

Enable Shared Indexes

```

<local-cache name="indexesInInfinispan">
  <indexing index="ALL">
    <property name="default.directory_provider">infinispan</property>
    <property
name="default.indexmanager">org.infinispan.query.indexmanager.InfinispanIn
dexManager</property>
  </indexing>
</local-cache>

```

When using an indexed, clustered cache ensure that the caches containing the index data are also clustered, as described in [Tuning Infinispan Directory](#).

11.4. CONFIGURE INDEXING

11.4.1. Configure the Index in Remote Client-Server Mode

In Remote Client-Server Mode, index configuration depends on the provider and its configuration. The indexing mode depends on the provider and whether or not it is local or distributed. The following indexing modes are supported:

- NONE
- LOCAL = `indexLocalOnly="true"`
- ALL = `indexLocalOnly="false"`

Index configuration in Remote Client-Server Mode is as follows:

Configuration in Remote Client-Server Mode

```

<indexing index="LOCAL">
  <property name="default.directory_provider">ram</property>
  <!-- Additional configuration information here -->
</indexing>

```

Configure Lucene Caches

By default the Lucene caches will be created as local caches; however, with this configuration the Lucene search results are not shared between nodes in the cluster. To prevent this define the caches required by Lucene in a clustered mode, as seen in the following configuration snippet:

Configuring the Lucene cache in Remote Client-Server Mode

```

<cache-container name="clustered" default-cache="repltestcache">
  [...]
  <replicated-cache name="LuceneIndexesMetadata" />
  <distributed-cache name="LuceneIndexesData" />
  <replicated-cache name="LuceneIndexesLocking" />
  [...]
</cache-container>

```

These caches are discussed in further detail at in the Red Hat JBoss Data Grid *Developer Guide*.

11.4.2. Rebuilding the Index

The Lucene index can be rebuilt, if required, by reconstructing it from the data store in the cache.

The index must be rebuilt if:

- The definition of what is indexed in the types has changed.
- A parameter affecting how the index is defined, such as the **Analyser** changes.
- The index is destroyed or corrupted, possibly due to a system administration error.

Rebuilding the index may be performed by executing the **Start** operation on the **MassIndexer** MBean.

This operation reprocesses all data in the grid, and therefore may take some time.

11.5. TUNING THE INDEX

11.5.1. Near-Realtime Index Manager

By default, each update is immediately flushed into the index. In order to achieve better throughput, the updates can be batched. However, this can result in a lag between the update and query — the query can see outdated data. If this is acceptable, you can use the Near-Realtime Index Manager by setting the following.

```
<property name="default.indexmanager">near-real-time</property>
```

11.5.2. Tuning Infinispan Directory

Lucene directory uses three caches to store the index:

- Data cache
- Metadata cache
- Locking cache

Configuration for these caches can be set explicitly, specifying the cache names as in the example below, and configuring those caches as usual. All of these caches must be clustered unless Infinispan Directory is used in local mode.

Tuning the Infinispan Directory

```
<distributed-cache name="indexedCache" >
  <indexing index="LOCAL">
    <property
name="default.indexmanager">org.infinispan.query.indexmanager.InfinispanIn
dexManager</property>
    <property
name="default.metadata_cachename">lucene_metadata_repl</property>
    <property
name="default.data_cachename">lucene_data_dist</property>
    <property
name="default.locking_cachename">lucene_locking_repl</property>
  </indexing>
</distributed-cache>
```

```
</distributed-cache>  
  
<replicated-cache name="lucene_metadata_repl" />  
  
<distributed-cache name="lucene_data_dist" />  
  
<replicated-cache name="lucene_locking_repl" />
```

11.5.3. Per-Index Configuration

The indexing properties in examples above apply for all indices - this is because we use the **default.** prefix for each property. To specify different configuration for each index, replace **default** with the index name. By default, this is the full class name of the indexed object, however you can override the index name in the **@Indexed** annotation.

CHAPTER 12. THE HEALTH CHECK API

12.1. THE HEALTH CHECK API

The Health Check API allows users to monitor the health of the cluster, and the caches contained within. This information is particularly important when working in a cloud environment, as it provides a method of querying to report the status of the cluster or cache.

This API exposes the following information:

- The name of the cluster.
- The number of machines in the cluster.
- The overall status of the cluster or cache, represented in one of three values:
 - Healthy - The entity is healthy.
 - Unhealthy - The entity is unhealthy. This value indicates that one or more caches are in a degraded state.
 - Rebalancing - The entity is operational, but a rebalance is in progress. Cluster nodes should not be adjusted when this value is reported.
- The status of each cache.
- A tail of the server log.

For information on using the Health Check API programmatically, refer to the JBoss Data Grid **Developer Guide**.

12.2. ACCESSING THE HEALTH API USING JMX

The Health Check API may be accessed through JMX, as seen in the following steps:

1. Connect to the JBoss Data Grid node using JMX, such as by [Connecting to JDG via JConsole](#).
2. Expand **jboss.datagrid-infinispan**.
3. Expand **CacheManager**.
4. Select the desired cache manager. By default the cache manager will be named **local**, if the server was started in local mode, or **clustered**, if the server was started in a clustered mode.
5. Expand the **CacheContainerHealth** object.
6. The Health Check API attributes are now available to be viewed.

An example of this using JConsole is seen below:

pid: 3362 jboss-modules.jar -mp /opt/jdg/jdgserver-er6/modules org.jboss.as.standalone -Djboss.home.dir=/opt/jdg/jdgserver...

Overview Memory Threads Classes VM Summary MBeans JBoss EAP CLI

- JMImplementation
- com.sun.management
- java.lang
- java.nio
- java.util.logging
- jboss.as
- jboss.as.expr
- jboss.datagrid-infinispan
 - Cache
 - CacheManager
 - "clustered"
 - CacheContainerHealth
 - Attributes
 - freeMemoryKb
 - numberOfNodes
 - totalMemoryKb
 - cacheHealth
 - clusterHealth
 - numberOfCpus
 - clusterName
 - CacheContainerStats
 - CacheManager
 - GlobalXSiteAdminOperations
 - Interpreter
 - LocalTopologyManager
 - RemoteQuery
 - Server
 - channel

- jboss.jta

Name	Value
cacheHealth	java.lang.String[10]
clusterHealth	HEALTHY
clusterName	clustered
freeMemoryKb	1078803
numberOfCpus	8
numberOfNodes	1
totalMemoryKb	1280000

Refresh

pid: 3362 jboss-modules.jar -mp /opt/jdg/jdgserver-er6/modules...

12.3. ACCESSING THE HEALTH CHECK API USING THE CLI

The Health Check API may be accessed using the included CLI. Once connected to the server use the following command, substituting the desired cache container for *CONTAINERNAME*:

```
/subsystem=datagrid-infinispan/cache-
container=CONTAINERNAME/health=HEALTH:read-resource(include-runtime=true)
```

The following demonstrates sample output from the above command, using the **clustered** cache-container:

```
[standalone@localhost:9990 health=HEALTH] /subsystem=datagrid-
infinispan/cache-container=clustered/health=HEALTH:read-resource(include-
runtime=true)
{
  "outcome" => "success",
  "result" => {
    "cache-health" => [
      "default",
      "HEALTHY",
      "_protobuf_metadata", "HEALTHY", "memcachedCache", "HEALTHY",
      "repl", "HEALTHY", "_script_cache",
      "HEALTHY"
    ]
  }
}
```

```

    ],
    "cluster-health" => "HEALTHY",
    "cluster-name" => "clustered",
    "free-memory" => 936823L,
    "log-tail" => [
        "2017-03-04 16:22:28,138 INFO
[org.infinispan.server.endpoint] (MSC service thread 1-7) DGENDPT10001:
MemcachedServer listening on 127.0.0.1:11211",
        "2017-03-04 16:22:28,146 INFO
[org.infinispan.server.endpoint] (MSC service thread 1-3) DGENDPT10000:
REST starting",
        "2017-03-04 16:22:28,188 INFO
[org.jboss.as.clustering.infinispan] (MSC service thread 1-3) DGISPN0001:
Started _protobuf_metadata cache from clustered container", "2017-03-04
16:22:28,195 INFO [org.jboss.as.clustering.infinispan] (MSC service thread
1-3) DGISPN0001: Started _script_cache cache from clustered container",
        "2017-03-04 16:22:28,515 INFO
[org.jboss.as.clustering.infinispan] (MSC service thread 1-4) DGISPN0001:
Started ___hotRodTopologyCache cache from clustered container",
        "2017-03-04 16:22:28,552 INFO
[org.infinispan.rest.NettyRestServer] (MSC service thread 1-3) ISPN012003:
REST server starting, listening on 127.0.0.1:8080",
        "2017-03-04 16:22:28,552 INFO
[org.infinispan.server.endpoint] (MSC service thread 1-3) DGENDPT10002:
REST mapped to /rest",
        "2017-03-04 16:22:28,613 INFO [org.jboss.as] (Controller Boot
Thread) WFLYSRV0060: Http management interface listening on
http://127.0.0.1:9990/management",
        "2017-03-04 16:22:28,613 INFO [org.jboss.as] (Controller Boot
Thread) WFLYSRV0051: Admin console listening on http://127.0.0.1:9990",
        "2017-03-04 16:22:28,613 INFO [org.jboss.as] (Controller Boot
Thread) WFLYSRV0025: Data Grid 7.1.0 (WildFly Core 2.1.10.Final-redhat-1)
started in 7608ms - Started 196 of 235 services (119 services are lazy,
passive or on-demand)"
    ],
    "number-of-cpus" => 8,
    "number-of-nodes" => 1,
    "total-memory" => 1280000L
}
}

```

12.4. ACCESSING THE HEALTH CHECK API USING THE MANAGEMENT REST INTERFACE

The Health Check API is integrated into the Management's REST interface as Metrics (read-only runtime resources).



IMPORTANT

Due to the metrics being exposed in runtime a HTTP POST method must be used instead of the typical HTTP GET.

To access these Metrics a HTTP POST method must be sent that contains valid user credentials. The following command demonstrates one such request:

```
curl --digest -L -D -
"http://JDGADDRESS:_JDGPORT_/management/subsystem/datagrid-
infinispan/cache-container/CONTAINERNAME/health/HEALTH?
operation=resource&include-runtime=true&json.pretty=1" --header "Content-
Type: application/json" -u username:password
```

The following properties should be substituted from the above command:

- **JDGADDRESS** - This should be the hostname or IP address where the JBoss Data Grid server is located.
- **JDGPORT** - This should be the port where the management interface is listening. By default this is **9990**.
- **CONTAINERNAME** - This should be the name of the cache container to query. By default the cache manager will be named **local**, if the server was started in local mode, or **clustered**, if the server was started in a clustered mode.
- **username** - The username for accessing the Administration Console.
- **password** - The associated password for accessing the Administration Console.

If successful, a 200 response should be received along with the health status, such as seen below:

```
HTTP/1.1 401 Unauthorized
Connection: keep-alive
WWW-Authenticate: Digest
realm="ManagementRealm", domain="/management", nonce="n1btFIY4yugNMTQ40DY2ND
Y3NjUxMy4utKorhon/y+zSHie9V58=", opaque="00000000000000000000000000000000",
algorithm=MD5, qop="auth"
X-Frame-Options: SAMEORIGIN
Content-Length: 77
Content-Type: text/html
Date: Sat, 04 Mar 2017 21:57:56 GMT

HTTP/1.1 200 OK
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Authentication-Info:
nextnonce="n1btFIY4yugNMTQ40DY2NDY3NjUxMy4utKorhon/y+zSHie9V58=", qop="auth
", rspauth="09ab5888ea71413b56dd724c13825a08", cnonce="Mz dj0TMyZWQ20Tk5Y2Q0N
mNlYzcxYzE2Zjg5NzdjZDE=", nc=00000001
Content-Type: application/json; charset=utf-8
Content-Length: 2108
Date: Sat, 04 Mar 2017 21:57:56 GMT

{
  "cache-health" : [
    "default",
    "HEALTHY",
    "__protobuf_metadata",
    "HEALTHY",
    "memcachedCache",
    "HEALTHY",
    "repl",
    "HEALTHY",
```

```

    "__script_cache",
    "HEALTHY"
  ],
  "cluster-health" : "HEALTHY",
  "cluster-name" : "clustered",
  "free-memory" : 1198983,
  "log-tail" : [
    "2017-03-04 16:22:28,138 INFO [org.infinispan.server.endpoint]
(MSC service thread 1-7) DGENDPT10001: MemcachedServer listening on
127.0.0.1:11211",
    "2017-03-04 16:22:28,146 INFO [org.infinispan.server.endpoint]
(MSC service thread 1-3) DGENDPT10000: REST starting",
    "2017-03-04 16:22:28,188 INFO
[org.jboss.as.clustering.infinispan] (MSC service thread 1-3) DGISPN0001:
Started __protobuf_metadata cache from clustered container",
    "2017-03-04 16:22:28,195 INFO
[org.jboss.as.clustering.infinispan] (MSC service thread 1-3) DGISPN0001:
Started __script_cache cache from clustered container",
    "2017-03-04 16:22:28,515 INFO
[org.jboss.as.clustering.infinispan] (MSC service thread 1-4) DGISPN0001:
Started __hotRodTopologyCache cache from clustered container",
    "2017-03-04 16:22:28,552 INFO
[org.infinispan.rest.NettyRestServer] (MSC service thread 1-3) ISPN012003:
REST server starting, listening on 127.0.0.1:8080",
    "2017-03-04 16:22:28,552 INFO [org.infinispan.server.endpoint]
(MSC service thread 1-3) DGENDPT10002: REST mapped to /rest",
    "2017-03-04 16:22:28,613 INFO [org.jboss.as] (Controller Boot
Thread) WFLYSRV0060: Http management interface listening on
http://127.0.0.1:9990/management",
    "2017-03-04 16:22:28,613 INFO [org.jboss.as] (Controller Boot
Thread) WFLYSRV0051: Admin console listening on http://127.0.0.1:9990",
    "2017-03-04 16:22:28,613 INFO [org.jboss.as] (Controller Boot
Thread) WFLYSRV0025: Data Grid 7.1.0 (WildFly Core 2.1.10.Final-redhat-1)
started in 7608ms - Started 196 of 235 services (119 services are lazy,
passive or on-demand)"
  ],
  "number-of-cpus" : 8,
  "number-of-nodes" : 1,
  "total-memory" : 1280000

```

PART VI. REMOTE CLIENT-SERVER MODE INTERFACES

CHAPTER 13. REMOTE CLIENT-SERVER MODE INTERFACES

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

- The Hot Rod Interface, including the RemoteCache API
- 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

CHAPTER 14. THE HOT ROD INTERFACE

14.1. ABOUT HOT ROD

Hot Rod is a binary TCP client-server protocol used in Red Hat JBoss Data Grid. It was created to overcome deficiencies in other client/server protocols, such as Memcached.

Hot Rod will failover on a server cluster that undergoes a topology change. Hot Rod achieves this by providing regular updates to clients about the cluster topology.

Hot Rod enables clients to do smart routing of requests in partitioned or distributed Red Hat JBoss Data Grid server clusters. To do this, Hot Rod allows clients to determine the partition that houses a key and then communicate directly with the server that has the key. This functionality relies on Hot Rod updating the cluster topology with clients, and that the clients use the same consistent hash algorithm as the servers.

Red Hat JBoss Data Grid contains a server module that implements the Hot Rod protocol. The Hot Rod protocol facilitates faster client and server interactions in comparison to other text-based protocols and allows clients to make decisions about load balancing, failover and data location operations.

14.2. THE BENEFITS OF USING HOT ROD OVER MEMCACHED

Red Hat JBoss Data Grid offers a choice of protocols for allowing clients to interact with the server in a Remote Client-Server environment. When deciding between using memcached or Hot Rod, the following should be considered.

Memcached

The memcached protocol causes the server endpoint to use the **memcached text wire protocol**. The **memcached wire protocol** has the benefit of being commonly used, and is available for almost any platform. All of JBoss Data Grid's functions, including clustering, state sharing for scalability, and high availability, are available when using memcached.

However the memcached protocol lacks dynamicity, resulting in the need to manually update the list of server nodes on your clients in the event one of the nodes in a cluster fails. Also, memcached clients are not aware of the location of the data in the cluster. This means that they will request data from a non-owner node, incurring the penalty of an additional request from that node to the actual owner, before being able to return the data to the client. This is where the Hot Rod protocol is able to provide greater performance than memcached.

Hot Rod

JBoss Data Grid's Hot Rod protocol is a binary wire protocol that offers all the capabilities of memcached, while also providing better scaling, durability, and elasticity.

The Hot Rod protocol does not need the hostnames and ports of each node in the remote cache, whereas memcached requires these parameters to be specified. Hot Rod clients automatically detect changes in the topology of clustered Hot Rod servers; when new nodes join or leave the cluster, clients update their Hot Rod server topology view. Consequently, Hot Rod provides ease of configuration and maintenance, with the advantage of dynamic load balancing and failover.

Additionally, the Hot Rod wire protocol uses smart routing when connecting to a distributed cache. This involves sharing a consistent hash algorithm between the server nodes and clients, resulting in faster read and writing capabilities than memcached.



WARNING

When using JCache over Hot Rod it is not possible to create remote clustered caches, as the operation is executed on a single node as opposed to the entire cluster; however, once a cache has been created on the cluster it may be obtained using the `cacheManager.getCache` method.

It is recommended to create caches using either configuration files or the CLI.

14.3. HOT ROD HASH FUNCTIONS

Hot Rod uses the same algorithm as on the server. The Hot Rod client always connects to the primary owner of the key, which is the first node in the list of owners. For more information about consistent hashing in Red Hat JBoss Data Grid, see [Distribution Mode's Consistent Hash Algorithm](#).

14.4. THE HOT ROD INTERFACE CONNECTOR

14.4.1. The Hot Rod Interface Connector

The following enables a Hot Rod server using the `hotrod` socket binding.

```
<hotrod-connector socket-binding="hotrod"
  cache-container="local" />
```

The connector creates a supporting topology cache with default settings. These settings can be tuned by adding the `<topology-state-transfer />` child element to the connector as follows:

```
<hotrod-connector socket-binding="hotrod"
  cache-container="local">
  <topology-state-transfer lazy-retrieval="false"
    lock-timeout="1000"
    replication-timeout="5000" />
</hotrod-connector>
```

The Hot Rod connector can be tuned with additional settings. See [Configure Hot Rod Connectors](#) for more information on how to configure the Hot Rod connector.



NOTE

The Hot Rod connector can be secured using SSL. See the *Hot Rod Authentication Using SASL* section of the *Developer Guide* for more information.

14.4.2. Configure Hot Rod Connectors

The following procedure describes the attributes used to configure the Hot Rod connector in Red Hat JBoss Data Grid's Remote Client-Server Mode. Both the `hotrod-connector` and `topology-state-transfer` elements must be configured based on the following procedure.

Configuring Hot Rod Connectors for Remote Client-Server Mode

```
<subsystem xmlns="urn:infinispan:server:endpoint:8.1">
  <hotrod-connector socket-binding="hotrod"
    cache-container="local"
    worker-threads="${VALUE}"
    idle-timeout="${SECONDS}"
    tcp-nodelay="${TRUE/FALSE}"
    send-buffer-size="${VALUE}"
    receive-buffer-size="${VALUE}" >
    <topology-state-transfer lock-timeout="${MILLISECONDS}"
      replication-timeout="${MILLISECONDS}"
      external-host="${HOSTNAME}"
      external-port="${PORT}"
      lazy-retrieval="${TRUE/FALSE}" />
  </hotrod-connector>
</subsystem>
```

1. The **hotrod-connector** element defines the configuration elements for use with Hot Rod.
 - The **socket-binding** parameter specifies the socket binding port used by the Hot Rod connector. This is a mandatory parameter.
 - The **cache-container** parameter names the cache container used by the Hot Rod connector. This is a mandatory parameter.
 - The **worker-threads** parameter specifies the number of worker threads available for the Hot Rod connector. The default value for this parameter is **160**. This is an optional parameter.
 - The **idle-timeout** parameter specifies the time, in seconds, that the connector can remain idle before the connection times out. The default value for this parameter is **0**, which means that no timeout period is set. This is an optional parameter.
 - The **tcp-nodelay** parameter specifies whether TCP packets will be delayed and sent out in batches. Valid values for this parameter are **true** and **false**. The default value for this parameter is **true**. This is an optional parameter.
 - The **send-buffer-size** parameter indicates the size of the send buffer for the Hot Rod connector. The default value for this parameter is the size of the TCP stack buffer. This is an optional parameter.
 - The **receive-buffer-size** parameter indicates the size of the receive buffer for the Hot Rod connector. The default value for this parameter is the size of the TCP stack buffer. This is an optional parameter.
2. The **topology-state-transfer** element specifies the topology state transfer configurations for the Hot Rod connector. This element can only occur once within a **hotrod-connector** element.
 - The **lock-timeout** parameter specifies the time (in milliseconds) after which the operation attempting to obtain a lock times out. The default value for this parameter is **10** seconds. This is an optional parameter.

- The **replication-timeout** parameter specifies the time (in milliseconds) after which the replication operation times out. The default value for this parameter is **10** seconds. This is an optional parameter.
- The **external-host** parameter specifies the hostname sent by the Hot Rod server to clients listed in the topology information. The default value for this parameter is the host address. This is an optional parameter.
- The **external-port** parameter specifies the port sent by the Hot Rod server to clients listed in the topology information. The default value for this parameter is the configured port. This is an optional parameter.
- The **lazy-retrieval** parameter indicates whether the Hot Rod connector will carry out retrieval operations lazily. The default value for this parameter is **true**. This is an optional parameter.

CHAPTER 15. THE REST INTERFACE

15.1. THE REST INTERFACE

Red Hat JBoss Data Grid provides a REST interface, allowing for loose coupling between the client and server. Its primary benefit is interoperability with existing HTTP clients, along with providing a connection for php clients. In addition, the need for specific versions of client libraries and bindings is eliminated.

The REST API introduces an overhead, and requires a REST client or custom code to understand and create REST calls. It is recommended to use the Hot Rod client where performance is a concern.

To interact with Red Hat JBoss Data Grid's REST API only a HTTP client library is required. For Java, this may be the Apache HTTP Commons Client, or the java.net API.



IMPORTANT

The following examples assume that REST security is disabled on the REST connector. To disable REST security remove the **authentication** and **encryption** elements from the connector.

15.2. THE REST INTERFACE CONNECTOR

15.2.1. The REST Interface Connector

The REST connector differs from the Hot Rod and Memcached connectors because it requires a web subsystem. Therefore configurations such as socket-binding, worker threads, timeouts, etc, must be performed on the web subsystem.

Once the REST interface has been enabled on the server it may be used normally for adding, removing, and retrieving data. For information on these processes refer to the JBoss Data Grid *Developer Guide*.

15.2.2. Configure REST Connectors

Use the following procedure to configure the **rest-connector** element in Red Hat JBoss Data Grid's Remote Client-Server mode.

Configuring REST Connectors for Remote Client-Server Mode

```
<subsystem xmlns="urn:infinispan:server:endpoint:8.1">
  <rest-connector cache-container="local"
                 context-path="{CONTEXT_PATH}"/>
</subsystem>
```

The **rest-connector** element specifies the configuration information for the REST connector.

1. The **cache-container** parameter names the cache container used by the REST connector. This is a mandatory parameter.
2. The **context-path** parameter specifies the context path for the REST connector. The default value for this parameter is an empty string (""). This is an optional parameter.

CHAPTER 16. THE MEMCACHED INTERFACE

16.1. THE MEMCACHED INTERFACE

Memcached is an in-memory caching system used to improve response and operation times for database-driven websites. The Memcached caching system defines a text based protocol called the Memcached protocol. The Memcached protocol uses in-memory objects or (as a last resort) passes to a persistent store such as a special memcached database.

Red Hat JBoss Data Grid offers a server that uses the Memcached protocol, removing the necessity to use Memcached separately with JBoss Data Grid. Additionally, due to JBoss Data Grid's clustering features, its data failover capabilities surpass those provided by Memcached.

16.2. ABOUT MEMCACHED SERVERS

Red Hat JBoss Data Grid contains a server module that implements the memcached protocol. This allows memcached clients to interact with one or multiple JBoss Data Grid based memcached servers.

The servers can be either:

- Standalone, where each server acts independently without communication with any other memcached servers.
- Clustered, where servers replicate and distribute data to other memcached servers.

16.3. MEMCACHED STATISTICS

The following table contains a list of valid statistics available using the memcached protocol in Red Hat JBoss Data Grid.

Table 16.1. Memcached Statistics

Statistic	Data Type	Details
uptime	32-bit unsigned integer.	Contains the time (in seconds) that the memcached instance has been available and running.
time	32-bit unsigned integer.	Contains the current time.
version	String	Contains the current version.
curr_items	32-bit unsigned integer.	Contains the number of items currently stored by the instance.
total_items	32-bit unsigned integer.	Contains the total number of items stored by the instance during its lifetime.

Statistic	Data Type	Details
cmd_get	64-bit unsigned integer	Contains the total number of get operation requests (requests to retrieve data).
cmd_set	64-bit unsigned integer	Contains the total number of set operation requests (requests to store data).
get_hits	64-bit unsigned integer	Contains the number of keys that are present from the keys requested.
get_misses	64-bit unsigned integer	Contains the number of keys that were not found from the keys requested.
delete_hits	64-bit unsigned integer	Contains the number of keys to be deleted that were located and successfully deleted.
delete_misses	64-bit unsigned integer	Contains the number of keys to be deleted that were not located and therefore could not be deleted.
incr_hits	64-bit unsigned integer	Contains the number of keys to be incremented that were located and successfully incremented
incr_misses	64-bit unsigned integer	Contains the number of keys to be incremented that were not located and therefore could not be incremented.
decr_hits	64-bit unsigned integer	Contains the number of keys to be decremented that were located and successfully decremented.
decr_misses	64-bit unsigned integer	Contains the number of keys to be decremented that were not located and therefore could not be decremented.
cas_hits	64-bit unsigned integer	Contains the number of keys to be compared and swapped that were found and successfully compared and swapped.

Statistic	Data Type	Details
cas_misses	64-bit unsigned integer	Contains the number of keys to be compared and swapped that were not found and therefore not compared and swapped.
cas_badval	64-bit unsigned integer	Contains the number of keys where a compare and swap occurred but the original value did not match the supplied value.
evictions	64-bit unsigned integer	Contains the number of eviction calls performed.
bytes_read	64-bit unsigned integer	Contains the total number of bytes read by the server from the network.
bytes_written	64-bit unsigned integer	Contains the total number of bytes written by the server to the network.

16.4. THE MEMCACHED INTERFACE CONNECTOR

16.4.1. The Memcached Interface Connector

The following enables a Memcached server using the **memcached** socket binding, and exposes the **memcachedCache** cache declared in the **local** container, using defaults for all other settings.

```
<memcached-connector socket-binding="memcached"
    cache-container="local"/>
```

Due to the limitations in the Memcached protocol, only one cache can be exposed by a connector. To expose more than one cache, declare additional memcached-connectors on different socket-bindings. See [Configure Memcached Connectors](#).

16.4.2. Configure Memcached Connectors

The following procedure describes the attributes used to configure the memcached connector within the **connectors** element in Red Hat JBoss Data Grid's Remote Client-Server Mode.

Configuring the Memcached Connector in Remote Client-Server Mode

The **memcached-connector** element defines the configuration elements for use with memcached.

```
<subsystem xmlns="urn:infinispan:server:endpoint:8.1">
  <memcached-connector socket-binding="memcached"
    cache-container="local"
    worker-threads="${VALUE}"
    idle-timeout="{SECONDS}"
```

```
tcp-nodelay="{TRUE/FALSE}"  
send-buffer-size="{VALUE}"  
receive-buffer-size="{VALUE}" />  
</subsystem>
```

1. The **socket-binding** parameter specifies the socket binding port used by the memcached connector. This is a mandatory parameter.
2. The **cache-container** parameter names the cache container used by the memcached connector. This is a mandatory parameter.
3. The **worker-threads** parameter specifies the number of worker threads available for the memcached connector. The default value for this parameter is 160. This is an optional parameter.
4. The **idle-timeout** parameter specifies the time, in seconds, that the connector can remain idle before the connection times out. The default value for this parameter is **0**, which means that no timeout period is set. This is an optional parameter.
5. The **tcp-nodelay** parameter specifies whether TCP packets will be delayed and sent out in batches. Valid values for this parameter are **true** and **false**. The default value for this parameter is **true**. This is an optional parameter.
6. The **send-buffer-size** parameter indicates the size of the send buffer for the memcached connector. The default value for this parameter is the size of the TCP stack buffer. This is an optional parameter.
7. The **receive-buffer-size** parameter indicates the size of the receive buffer for the memcached connector. The default value for this parameter is the size of the TCP stack buffer. This is an optional parameter.

PART VII. SET UP LOCKING FOR THE CACHE

CHAPTER 17. LOCKING

17.1. LOCKING

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

17.2. CONFIGURE LOCKING (REMOTE CLIENT-SERVER MODE)

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



NOTE

The default isolation mode for the Remote Client-Server mode configuration is **READ_COMMITTED**. If the **isolation** attribute is included to explicitly specify an isolation mode, it is ignored, a warning is thrown, and the default value is used instead.

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

Configure Locking (Remote Client-Server Mode)

```
<distributed-cache name="distributedCache">
  <locking acquire-timeout="30000"
    concurrency-level="1000"
    striping="false" />
  <!-- Additional configuration here -->
</distributed-cache>
```

1. The **acquire-timeout** parameter specifies the number of milliseconds after which lock acquisition will time out.
2. The **concurrency-level** parameter defines the number of lock stripes used by the LockManager.
3. The **striping** parameter specifies whether lock striping will be used for the local cache.

17.3. CONFIGURE LOCKING (LIBRARY MODE)

For Library mode, the **locking** element and its parameters are set within the **default** element found within cache element. An example of this configuration on a local cache is below:

Configure Locking (Library Mode)

```
<local-cache name="default">
  <locking concurrency-level="{VALUE}"
    isolation="{LEVEL}"
    acquire-timeout="{TIME}"
```

```

striping="${TRUE/FALSE}"
write-skew="${TRUE/FALSE}" />
</local-cache>

```

1. The **concurrency-level** parameter specifies the concurrency level for the lock container. Set this value according to the number of concurrent threads interacting with the data grid.
2. The **isolation** parameter specifies the cache's isolation level. Valid isolation levels are **READ_COMMITTED** and **REPEATABLE_READ**. For details about isolation levels, see [About Isolation Levels](#).
3. The **acquire-timeout** parameter specifies time (in milliseconds) after which a lock acquisition attempt times out.
4. The **striping** parameter specifies whether a pool of shared locks are maintained for all entries that require locks. If set to **FALSE**, locks are created for each entry in the cache. For details, see [About Lock Striping](#).
5. The **write-skew** parameter is only valid if the **isolation** is set to **REPEATABLE_READ**. If this parameter is set to **FALSE**, a disparity between a working entry and the underlying entry at write time results in the working entry overwriting the underlying entry. If the parameter is set to **TRUE**, such conflicts (namely write skews) throw an exception. The **write-skew** parameter can be only used with **OPTIMISTIC** transactions and it requires entry versioning to be enabled, with **SIMPLE** versioning scheme.

17.4. LOCKING TYPES

17.4.1. About Optimistic Locking

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

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

17.4.2. About Pessimistic Locking

Pessimistic locking is also known as eager locking.

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

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

17.4.3. Pessimistic Locking Types

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

- Explicit Pessimistic Locking, which uses the JBoss Data Grid Lock API to allow cache users to

explicitly lock cache keys for the duration of a transaction. The Lock call attempts to obtain locks on specified cache keys across all nodes in a cluster. This attempt either fails or succeeds for all specified cache keys. All locks are released during the commit or rollback phase.

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

17.4.4. Explicit Pessimistic Locking Example

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

Transaction with Explicit Pessimistic Locking

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

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

17.4.5. Implicit Pessimistic Locking Example

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

Transaction with Implicit Pessimistic locking

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

1. When the line **cache.put(K, V)** executes, a cluster-wide lock is acquired on **K**.
2. When the line **cache.put(K2, V2)** executes, a cluster-wide lock is acquired on **K2**.
3. When the line **cache.put(K, V5)** executes, the lock acquisition is non operational because a cluster-wide lock for **K** has been previously acquired. The **put** operation will still occur.
4. When the line **tx.commit()** executes, all locks held for this transaction are released.

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

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

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

17.4.7. Configure Locking Mode (Library Mode)

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

```
<transaction transaction-manager-lookup="{TransactionManagerLookupClass}"
  mode="{NONE, BATCH, NON_XA, NON_DURABLE_XA, FULL_XA}"
  locking="{OPTIMISTIC,PESSIMISTIC}">
</transaction>
```

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

17.5. LOCKING OPERATIONS

17.5.1. About the LockManager

The **LockManager** component is responsible for locking an entry before a write process initiates. The **LockManager** uses a **LockContainer** to locate, hold and create locks. There are two types of **LockContainers** JBoss Data Grid uses internally and their choice is dependent on the **useLockStriping** setting. The first type offers support for lock striping while the second type supports one lock per entry.

See Also: [Set Up Lock Striping](#)

17.5.2. About Lock Acquisition

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

17.5.3. About Concurrency Levels

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

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

CHAPTER 18. SET UP LOCK STRIPING

18.1. ABOUT LOCK STRIPING

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

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

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

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

Lock Striping (Remote Client-Server Mode)

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



NOTE

The default isolation mode for the Remote Client-Server mode configuration is **READ_COMMITTED**. If the **isolation** attribute is included to explicitly specify an isolation mode, it is ignored, a warning is thrown, and the default value is used instead.

The **locking** element uses the following attributes:

- The **acquire-timeout** attribute specifies the maximum time to attempt a lock acquisition. The default value for this attribute is **10000** milliseconds.
- The **concurrency-level** attribute specifies the concurrency level for lock containers. Adjust this value according to the number of concurrent threads interacting with JBoss Data Grid. The default value for this attribute is **32**.
- The **striping** attribute specifies whether a shared pool of locks is maintained for all entries that require locking (**true**). If set to **false**, a lock is created for each entry. Lock striping controls the memory footprint but can reduce concurrency in the system. The default value for this attribute is **false**.

18.3. CONFIGURE LOCK STRIPING (LIBRARY MODE)

Lock striping is disabled by default in Red Hat JBoss Data Grid. Configure lock striping in JBoss Data Grid's Library mode using the **striping** parameter as demonstrated in the following procedure.

Configure Lock Striping (Library Mode)

```
<local-cache>
```

```
<locking concurrency-level="{VALUE}"
  isolation="{LEVEL}"
  acquire-timeout="{TIME}"
  striping="{TRUE/FALSE}"
  write-skew="{TRUE/FALSE}" />
</local-cache>
```

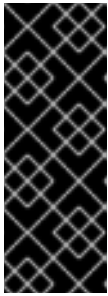
1. The **concurrency-level** is used to specify the size of the shared lock collection use when lock striping is enabled.
2. The **isolation** parameter specifies the cache's isolation level. Valid isolation levels are **READ_COMMITTED** and **REPEATABLE_READ**.
3. The **acquire-timeout** parameter specifies time (in milliseconds) after which a lock acquisition attempt times out.
4. The **striping** parameter specifies whether a pool of shared locks are maintained for all entries that require locks. If set to **FALSE**, locks are created for each entry in the cache. If set to **TRUE**, lock striping is enabled and shared locks are used as required from the pool.
5. The **write-skew** check determines if a modification to the entry from a different transaction should roll back the transaction. Write skew set to true requires **isolation_level** set to **REPEATABLE_READ**. The default value for **write-skew** and **isolation_level** are **FALSE** and **READ_COMMITTED** respectively. The **write-skew** parameter can be only used with **OPTIMISTIC** transactions and it requires entry versioning to be enabled, with **SIMPLE** versioning scheme.

CHAPTER 19. SET UP ISOLATION LEVELS

19.1. ABOUT ISOLATION LEVELS

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

- **READ_COMMITTED**. This isolation level is applicable to a wide variety of requirements. This is the default value in Remote Client-Server and Library modes.
- **REPEATABLE_READ**.



IMPORTANT

The only valid value for locks in Remote Client-Server mode is the default **READ_COMMITTED** value. The value explicitly specified with the **isolation** value is ignored.

If the **locking** element is not present in the configuration, the default isolation value is **READ_COMMITTED**.

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

- See [Configure Lock Striping \(Remote Client-Server Mode\)](#) for a Remote Client-Server mode configuration sample.
- See [Configure Lock Striping \(Library Mode\)](#) for a Library mode configuration sample.

19.2. ABOUT READ_COMMITTED

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

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

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

19.3. ABOUT REPEATABLE_READ

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

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

JBoss Data Grid's **REPEATABLE_READ** isolation mode preserves the value of an entry before a

modification occurs. As a result, the "non-repeatable read" phenomenon is avoided because a second read operation on the same entry retrieves the preserved value rather than the new modified value. As a result, the two values retrieved by the two read operations in a single transaction will always match, even if a write operation occurs in a different transaction between the two reads.

PART VIII. SET UP AND CONFIGURE A CACHE STORE

CHAPTER 20. CACHE STORES

20.1. CACHE STORES

The cache store connects Red Hat JBoss Data Grid to the persistent data store. Cache stores are associated with individual caches. Different caches attached to the same cache manager can have different cache store configurations.



NOTE

If a clustered cache is configured with an unshared cache store (where **shared** is set to **false**), on node join, stale entries which might have been removed from the cluster might still be present in the stores and can reappear.

20.2. CACHE LOADERS AND CACHE WRITERS

Integration with the persistent store is done through the following SPIs located in `org.infinispan.persistence.spi`:

- `CacheLoader`
- `CacheWriter`
- `AdvancedCacheLoader`
- `AdvancedCacheWriter`

`CacheLoader` and `CacheWriter` provide basic methods for reading and writing to a store.

`CacheLoader` retrieves data from a data store when the required data is not present in the cache, and `CacheWriter` is used to enforce entry passivation and activation on eviction in a cache.

`AdvancedCacheLoader` and `AdvancedCacheWriter` provide operations to manipulate the underlying storage in bulk: parallel iteration and purging of expired entries, clear and size.

The `org.infinispan.persistence.file.SingleFileStore` is a good starting point to write your own store implementation.



NOTE

Previously, JBoss Data Grid used the old API (`CacheLoader`, extended by `CacheStore`), which is also still available.

20.3. CACHE STORE CONFIGURATION

20.3.1. Configuring the Cache Store

Cache stores can be configured in a chain. Cache read operations checks each cache store in the order configured until a valid non-null element of data has been located. Write operations affect all cache stores unless the `ignoreModifications` element has been set to `"true"` for a specific cache store.

20.3.2. Configure the Cache Store using XML (Library Mode)

The following example demonstrates cache store configuration using XML in JBoss Data Grid's Library mode:

```
<persistence passivation="false">
  <file-store shared="false"
    preload="true"
    fetch-state="true"
    purge-startup="false"
    singleton="true"
    location="{java.io.tmpdir}" >
    <write-behind enabled="true"
      flush-lock-timeout="15000"
      thread-pool-size="5" />
  </singleFile>
</persistence>
```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).

20.3.3. About SKIP_CACHE_LOAD Flag

In Red Hat JBoss Data Grid's Remote Client-Server mode, when the cache is preloaded from a cache store and eviction is disabled, read requests go to the memory. If the entry is not found in a memory during a read request, it accesses the cache store which may impact the read performance.

To avoid referring to the cache store when a key is not found in the memory, use the **SKIP_CACHE_LOAD** flag.

20.3.4. About the SKIP_CACHE_STORE Flag

When the **SKIP_CACHE_STORE** Flag is used then the cache store will not be considered for the specified cache operations. This flag can be useful to place an entry in the cache without having it included in the configured cache store, along with determining if an entry is found within a cache without retrieving it from the associated cache store.

20.3.5. About the SKIP_SHARED_CACHE_STORE Flag

When the **SKIP_SHARED_CACHE_STORE** Flag is enabled then any shared cache store will not be considered for the specified cache operations. This flag can be useful to place an entry in the cache without having it included in the shared cache store, along with determining if an entry is found within a cache without retrieving it from the shared cache store.

20.4. SHARED CACHE STORES

20.4.1. Shared Cache Stores

A shared cache store is a cache store that is shared by multiple cache instances.

A shared cache store is useful when all instances in a cluster communicate with the same remote, shared database using the same **JDBC** settings. In such an instance, configuring a shared cache store prevents the unnecessary repeated write operations that occur when various cache instances attempt to write the same data to the cache store.

20.4.2. Invalidation Mode and Shared Cache Stores

When used in conjunction with a shared cache store, Red Hat JBoss Data Grid's invalidation mode causes remote caches to see the shared cache store to retrieve modified data.

The benefits of using invalidation mode in conjunction with shared cache stores include the following:

- Compared to replication messages, which contain the updated data, invalidation messages are much smaller and result in reduced network traffic.
- The remaining cluster caches look up modified data from the shared cache store lazily and only when required to do so, resulting in further reduced network traffic.

20.4.3. The Cache Store and Cache Passivation

In Red Hat JBoss Data Grid, a cache store can be used to enforce the passivation of entries and to activate eviction in a cache. Whether passivation mode or activation mode are used, the configured cache store both reads from and writes to the data store.

When passivation is disabled in JBoss Data Grid, after the modification, addition or removal of an element is carried out the cache store steps in to persist the changes in the store.

20.4.4. Application Cachestore Registration

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

20.5. CONNECTION FACTORIES

20.5.1. Connection Factories

In Red Hat JBoss Data Grid, all **JDBC** cache stores rely on a **ConnectionFactory** implementation to obtain a database connection. This process is also known as connection management or pooling.

A connection factory can be specified using the **ConnectionFactoryClass** configuration attribute. JBoss Data Grid includes the following **ConnectionFactory** implementations:

- `ManagedConnectionFactory`
- `SimpleConnectionFactory`.
- `PooledConnectionFactory`.

20.5.2. About ManagedConnectionFactory

ManagedConnectionFactory is a connection factory that is ideal for use within managed environments such as application servers. This connection factory can explore a configured location in the **JNDI** tree and delegate connection management to the **DataSource**.

20.5.3. About SimpleConnectionFactory

SimpleConnectionFactory is a connection factory that creates database connections on a per invocation basis. This connection factory is not designed for use in a production environment.

20.5.4. About PooledConnectionFactory

PooledConnectionFactory is a connection factory based on **C3P0**, and is typically recommended for standalone deployments as opposed to deployments utilizing a servlet container, such as JBoss EAP. This connection factory functions by allowing the user to define a set of parameters which may be used for all **DataSource** instances generated by the factory.

CHAPTER 21. CACHE STORE IMPLEMENTATIONS

21.1. CACHE STORES

The cache store connects Red Hat JBoss Data Grid to the persistent data store. Cache stores are associated with individual caches. Different caches attached to the same cache manager can have different cache store configurations.



NOTE

If a clustered cache is configured with an unshared cache store (where **shared** is set to **false**), on node join, stale entries which might have been removed from the cluster might still be present in the stores and can reappear.

21.2. CACHE STORE COMPARISON

Select a cache store based on your requirements. The following is a summary of high level differences between the cache stores available in Red Hat JBoss Data Grid:

- The Single File Cache Store is a local file cache store. It persists data locally for each node of the clustered cache. The Single File Cache Store provides superior read and write performance, but keeps keys in memory which limits its use when persisting large data sets at each node. See [Single File Cache Store](#) for details.
- The LevelDB file cache store is a local file cache store which provides high read and write performance. It does not have the limitation of Single File Cache Store of keeping keys in memory. See [LevelDB Cache Store](#) for details.
- The JDBC cache store is a cache store that may be shared, if required. When using it, all nodes of a clustered cache persist to a single database or a local JDBC database for every node in the cluster. The shared cache store lacks the scalability and performance of a local cache store such as the LevelDB cache store, but it provides a single location for persisted data. The JDBC cache store persists entries as binary blobs, which are not readable outside JBoss Data Grid. See [JDBC Based Cache Stores](#) for details.
- The JPA Cache Store (supported in Library mode only) is a shared cache store like JDBC cache store, but preserves schema information when persisting to the database. Therefore, the persisted entries can be read outside JBoss Data Grid. See [JPA Cache Store](#) for details.

21.3. CACHE STORE CONFIGURATION DETAILS (LIBRARY MODE)

The following lists contain details about the configuration elements and parameters for cache store elements in JBoss Data Grid's Library mode. The following list is meant to highlight certain parameters on each element, and a full list may be found in the schemas.

The persistence Element

- The **passivation** parameter affects the way in which Red Hat JBoss Data Grid interacts with stores. When an object is evicted from in-memory cache, passivation writes it to a secondary data store, such as a system or a database. Valid values for this parameter are **true** and **false** but **passivation** is set to **false** by default.

The file-store Element

- The **shared** parameter indicates that the cache store is shared by different cache instances. For example, where all instances in a cluster use the same JDBC settings to talk to the same remote, shared database. **shared** is **false** by default. When set to **true**, it prevents duplicate data being written to the cache store by different cache instances. For the LevelDB cache stores, this parameter must be excluded from the configuration, or set to **false** because sharing this cache store is not supported.
- The **preload** parameter is set to **false** by default. When set to **true** the data stored in the cache store is preloaded into the memory when the cache starts. This allows data in the cache store to be available immediately after startup and avoids cache operations delays as a result of loading data lazily. Preloaded data is only stored locally on the node, and there is no replication or distribution of the preloaded data. Red Hat JBoss Data Grid will only preload up to the maximum configured number of entries in eviction.
- The **fetch-state** parameter determines whether or not to fetch the persistent state of a cache and apply it to the local cache store when joining the cluster. If the cache store is shared the fetch persistent state is ignored, as caches access the same cache store. A configuration exception will be thrown when starting the cache service if more than one cache store has this property set to **true**. The **fetch-state** property is **false** by default.
- In order to speed up lookups, the single file cache store keeps an index of keys and their corresponding position in the file. To avoid this index resulting in memory consumption problems, this cache store can be bounded by a maximum number of entries that it stores, defined by the **max-entries** parameter. If this limit is exceeded, entries are removed permanently using the LRU algorithm both from the in-memory index and the underlying file based cache store. The default value is **-1**, allowing unlimited entries.
- The **singleton** parameter enables a singleton store cache store. SingletonStore is a delegating cache store used when only one instance in a cluster can interact with the underlying store; however, **singleton** parameter is not recommended for **file-store**. The default value is **false**.
- The **purge** parameter controls whether cache store is purged when it starts up.
- The **location** configuration element sets a location on disk where the store can write.

The write-behind Element

The **write-behind** element contains parameters that configure various aspects of the cache store.

- The **thread-pool-size** parameter specifies the number of threads that concurrently apply modifications to the store. The default value for this parameter is **1**.
- The **flush-lock-timeout** parameter specifies the time to acquire the lock which guards the state to be flushed to the cache store periodically. The default value for this parameter is **1**.
- The **modification-queue-size** parameter specifies the size of the modification queue for the asynchronous store. If updates are made at a rate that is faster than the underlying cache store can process this queue, then the asynchronous store behaves like a synchronous store for that period, blocking until the queue can accept more elements. The default value for this parameter is **1024** elements.
- The **shutdown-timeout** parameter specifies maximum amount of time that can be taken to stop the cache store. Default value for this parameter is **25000** milliseconds.

The remote-store Element

- The **cache** attribute specifies the name of the remote cache to which it intends to connect in the remote Infinispan cluster. The default cache will be used if the remote cache name is unspecified.
- The **fetch-state** attribute, when set to **true**, ensures that the persistent state is fetched when the remote cache joins the cluster. If multiple cache stores are chained, only one cache store can have this property set to **true**. The default for this value is **false**.
- The **shared** attribute is set to **true** when multiple cache instances share a cache store, which prevents multiple cache instances writing the same modification individually. The default for this attribute is **false**.
- The **preload** attribute ensures that the cache store data is pre-loaded into memory and is immediately accessible after starting up. The disadvantage of setting this to **true** is that the start up time increases. The default value for this attribute is **false**.
- The **singleton** parameter enables the SingletonStore delegating cache store, used in situations when only one instance in a cluster should interact with the underlying store. The default value is **false**.
- The **purge** attribute ensures that the cache store is purged during the start up process. The default value for this attribute is **false**.
- The **tcp-no-delay** attribute triggers the *TCPNODELAY* stack. The default value for this attribute is **true**.
- The **ping-on-start** attribute sends a ping request to a back end server to fetch the cluster topology. The default value for this attribute is **true**.
- The **key-size-estimate** attribute provides an estimation of the key size. The default value for this attribute is **64**.
- The **value-size-estimate** attribute specifies the size of the byte buffers when serializing and deserializing values. The default value for this attribute is **512**.
- The **force-return-values** attribute sets whether **FORCE_RETURN_VALUE** is enabled for all calls. The default value for this attribute is **false**.

The remote-server Element

Create a **remote-server** element within the **remote-store** element to define the server information.

- The **host** attribute configures the host address.
- The **port** attribute configures the port used by the Remote Cache Store. This defaults to **11222**.

The connection-pool Element (Remote Store)

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

- The **max-idle** parameter indicates the maximum number of idle connections for each server at a time. The default value for this attribute is **-1** which indicates an infinite number of idle connections.
- The **max-total** parameter indicates the maximum number of persistent connections within the combined set of servers. The default setting for this attribute is **-1** which indicates an infinite number of connections.
- The **min-idle-time** parameter sets a target value for the minimum number of idle connections (per server) that should always be available. If this parameter is set to a positive number and **timeBetweenEvictionRunsMillis** 0, each time the idle connection eviction thread runs, it will try to create enough idle instances so that there will be **minIdle** idle instances available for each server. The default setting for this parameter is **1**.
- The **eviction-interval** parameter indicates how long the eviction thread should sleep before "runs" of examining idle connections. When non-positive, no eviction thread will be launched. The default setting for this parameter is **120000** milliseconds, or 2 minutes.
- The **min-evictable-idle-time** parameter specifies the minimum amount of time that a connection may sit idle in the pool before it is eligible for eviction due to idle time. When non-positive, no connection will be dropped from the pool due to idle time alone. This setting has no effect unless **timeBetweenEvictionRunsMillis** 0. The default setting for this parameter is **1800000**, or (30 minutes).
- The **test-idle** parameter indicates whether or not idle connections should be validated by sending an TCP packet to the server, during idle connection eviction runs. Connections that fail to validate will be dropped from the pool. This setting has no effect unless **timeBetweenEvictionRunsMillis** 0. The default setting for this parameter is **true**.

The leveldb-store Element

- The **relative-to** parameter specifies the base directory in which to store the cache state.
- The **path** parameter specifies the location within the **relative-to** parameter to store the cache state.
- The **shared** parameter specifies whether the cache store is shared. The only supported value for this parameter in the LevelDB cache store is **false**.
- The **preload** parameter specifies whether the cache store will be pre-loaded. Valid values are **true** and **false**.
- The **block-size** parameter defines the block size of the cache store.
- The **singleton** parameter enables the SingletonStore delegating cache store, used in situations when only one instance in a cluster should interact with the underlying store. The default value is **false**.
- The **cache-size** parameter defines the cache size of the cache store.
- The **clear-threshold** parameter defines the cache clear threshold of the cache store.

The jpa-store Element

- The **persistence-unit** attribute specifies the name of the JPA cache store.

- The **entity-class** attribute specifies the fully qualified class name of the JPA entity used to store the cache entry value.
- The **batch-size** (optional) attribute specifies the batch size for cache store streaming. The default value for this attribute is **100**.
- The **store-metadata** (optional) attribute specifies whether the cache store keeps the metadata (for example expiration and versioning information) with the entries. The default value for this attribute is **true**.
- The **singleton** parameter enables the SingletonStore delegating cache store, used in situations when only one instance in a cluster should interact with the underlying store. The default value is **false**.

The **binary-keyed-jdbc-store**, **string-keyed-jdbc-store**, and **mixed-keyed-jdbc-store** Elements

- The **fetch-state** parameter determines whether the persistent state is fetched when joining a cluster. Set this to **true** if using a replication and invalidation in a clustered environment. Additionally, if multiple cache stores are chained, only one cache store can have this property enabled. If a shared cache store is used, the cache does not allow a persistent state transfer despite this property being set to **true**. The **fetch-state** parameter is **false** by default.
- The **singleton** parameter enables the SingletonStore delegating cache store, used in situations when only one instance in a cluster should interact with the underlying store. The default value is **false**.
- The **purge** parameter specifies whether the cache store is purged when initially started.
- The **key-to-string-mapper** parameter specifies the class name used to map keys to strings for the database tables.

The **connection-pool** Element (JDBC Store)

- The **connection-url** parameter specifies the JDBC driver-specific connection URL.
- The **username** parameter contains the username used to connect via the **connection-url**.
- The **password** parameter contains the password to use when connecting via the **connection-url**.
- The **driver** parameter specifies the class name of the driver used to connect to the database.

The **binary-keyed-table** and **string-keyed-table** Elements

- The **prefix** attribute defines the string prepended to name of the target cache when composing the name of the cache bucket table.
- The **drop-on-exit** parameter specifies whether the database tables are dropped upon shutdown.
- The **create-on-start** parameter specifies whether the database tables are created by the store on startup.
- The **fetch-size** parameter specifies the size to use when querying from this table. Use this parameter to avoid heap memory exhaustion when the query is large.

- The **batch-size** parameter specifies the batch size used when modifying this table.

The **id-column**, **data-column**, and **timestamp-column** Elements

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

The **custom-store** Element

- The **class** parameter specifies the class name of the cache store implementation.
- The **preload** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are **true** and **false**.
- The **shared** parameter specifies whether the cache store is shared. This is used when multiple cache instances share a cache store. Valid values for this parameter are **true** and **false**.

The **property** Element

A property may be defined inside of a cache store, with the entry between the property tags being the stored value. For instance, in the below example a value of **1** is defined for **minOccurs**.

```
<property name="minOccurs">1</property>
```

- The **name** attribute specifies the name of the property.

21.4. CACHE STORE CONFIGURATION DETAILS (REMOTE CLIENT-SERVER MODE)

The following tables contain details about the configuration elements and parameters for cache store elements in JBoss Data Grid's Remote Client-Server mode. The following list is meant to highlight certain parameters on each element, and a full list may be found in the schemas.

The **local-cache** Element

- The **name** parameter of the **local-cache** attribute is used to specify a name for the cache.
- The **statistics** parameter specifies whether statistics are enabled at the container level. Enable or disable statistics on a per-cache basis by setting the **statistics** attribute to **false**.

The **file-store** Element

- The **name** parameter of the **file-store** element is used to specify a name for the file store.
- The **passivation** parameter determines whether entries in the cache are passivated (**true**) or if the cache store retains a copy of the contents in memory (**false**).
- The **purge** parameter specifies whether or not the cache store is purged when it is started. Valid values for this parameter are **true** and **false**.
- The **shared** parameter is used when multiple cache instances share a cache store. This parameter can be set to prevent multiple cache instances writing the same modification multiple times. Valid values for this parameter are **true** and **false**. However, the **shared** parameter is

not recommended for the LevelDB cache store because this cache store cannot be shared.

- The **relative-to** property is the directory where the **file-store** stores the data. It is used to define a named path.
- The **path** property is the name of the file where the data is stored. It is a relative path name that is appended to the value of the **relative-to** property to determine the complete path.
- The **max-entries** parameter provides maximum number of entries allowed. The default value is -1 for unlimited entries.
- The **fetch-state** parameter when set to true fetches the persistent state when joining a cluster. If multiple cache stores are chained, only one of them can have this property enabled. Persistent state transfer with a shared cache store does not make sense, as the same persistent store that provides the data will just end up receiving it. Therefore, if a shared cache store is used, the cache does not allow a persistent state transfer even if a cache store has this property set to **true**. It is recommended to set this property to true only in a clustered environment. The default value for this parameter is false.
- The **preload** parameter when set to true, loads the data stored in the cache store into memory when the cache starts. However, setting this parameter to true affects the performance as the startup time is increased. The default value for this parameter is false.
- The **singleton** parameter enables a singleton store cache store. SingletonStore is a delegating cache store used when only one instance in a cluster can interact with the underlying store; however, **singleton** parameter is not recommended for **file-store**. The default value is **false**.

The store Element

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

The property Element

- The **name** parameter specifies the name of the property.
- The **value** parameter specifies the value assigned to the property.

The remote-store Element

- The **cache** parameter defines the name for the remote cache. If left undefined, the default cache is used instead.
- The **socket-timeout** parameter sets whether the value defined in **SO_TIMEOUT** (in milliseconds) applies to remote Hot Rod servers on the specified timeout. A timeout value of 0 indicates an infinite timeout. The default value is 60,000 ms, or one minute.
- The **tcp-no-delay** sets whether **TCP_NODELAY** applies on socket connections to remote Hot Rod servers.
- The **hotrod-wrapping** sets whether a wrapper is required for Hot Rod on the remote store.
- The **singleton** parameter enables the SingletonStore delegating cache store, used in situations when only one instance in a cluster should interact with the underlying store. The default value is **false**.

The remote-server Element

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

The binary-keyed-jdbc-store, string-keyed-jdbc-store, and mixed-keyed-jdbc-store Elements

- The **datasource** parameter defines the name of a JNDI for the datasource.
- The **passivation** parameter determines whether entries in the cache are passivated (**true**) or if the cache store retains a copy of the contents in memory (**false**).
- The **preload** parameter specifies whether to load entries into the cache during start up. Valid values for this parameter are **true** and **false**.
- The **purge** parameter specifies whether or not the cache store is purged when it is started. Valid values for this parameter are **true** and **false**.
- The **shared** parameter is used when multiple cache instances share a cache store. This parameter can be set to prevent multiple cache instances writing the same modification multiple times. Valid values for this parameter are **true** and **false**.
- The **singleton** parameter enables a singleton store cache store. SingletonStore is a delegating cache store used when only one instance in a cluster can interact with the underlying store

The binary-keyed-table and string-keyed-table Elements

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

The id-column, data-column, and timestamp-column Elements

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

The leveldb-store Element

- The **relative-to** parameter specifies the base directory to store the cache state. This value defaults to **jboss.server.data.dir**.
- The **path** parameter defines where, within the directory specified in the **relative-to** parameter, the cache state is stored. If undefined, the path defaults to the cache container name.
- The **passivation** parameter specifies whether passivation is enabled for the LevelDB cache store. Valid values are **true** and **false**.
- The **singleton** parameter enables the SingletonStore delegating cache store, used in situations when only one instance in a cluster should interact with the underlying store. The default value is **false**.
- The **purge** parameter specifies whether the cache store is purged when it starts up. Valid values are **true** and **false**.

21.5. SINGLE FILE CACHE STORE

21.5.1. Single File Cache Store

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

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

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

To reduce memory consumption, the size of the cache store can be set to a fixed number of entries to store in the file; however, this works only when JBoss Data Grid is used as a cache. When JBoss Data Grid is used this way, data which is not present in the cache can be recomputed or re-retrieved from the authoritative data store and stored in the JBoss Data Grid cache. This limitation exists so that once the maximum number of entries is reached older data in the cache store is removed. If JBoss Data Grid were used as an authoritative data store in this scenario it would lead to potential data loss.

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

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

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

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

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Remote Client-Server Mode\)](#).

21.5.3. Single File Store Configuration (Library Mode)

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

```
<local-cache name="writeThroughToFile">
  <persistence passivation="false">
    <file-store fetch-state="true"
```

```

        purge="false"
        shared="false"
        preload="false"
        location="/tmp/Another-FileCacheStore-Location"
        max-entries="100">
    <write-behind enabled="true"
        threadPoolSize="500"
        flush-lock-timeout="1"
        modification-queue-size="1024"
        shutdown-timeout="25000"/>
    </singleFile>
</persistence>
</local-cache>

```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).

21.5.4. Upgrade JBoss Data Grid Cache Stores

Red Hat JBoss Data Grid 7 stores data in a different format than previous versions of JBoss Data Grid. As a result, the newer version of JBoss Data Grid cannot read data stored by older versions. Use rolling upgrades to upgrade persisted data from the format used by the old JBoss Data Grid to the new format. Additionally, the newer version of JBoss Data Grid also stores persistence configuration information in a different location.

Rolling upgrades is the process by which a JBoss Data Grid installation is upgraded without a service shutdown. For JBoss Data Grid servers, this procedure refers to the server side components. The upgrade can be due to either hardware or software change, such as upgrading JBoss Data Grid.

Rolling upgrades are only available in JBoss Data Grid's Remote Client-Server mode.

21.6. LEVELDB CACHE STORE

21.6.1. LevelDB Cache Store

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

The LevelDB Cache Store uses two filesystem directories. Each directory is configured for a LevelDB database. One directory stores the non-expired data and the second directory stores the keys pending to be purged permanently.

21.6.2. Configuring LevelDB Cache Store (Remote Client-Server Mode)

Procedure: To configure LevelDB Cache Store:

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

```

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

```


**NOTE**

Directories will be automatically created if they do not exist.

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Remote Client-Server Mode\)](#).

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

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

```
<local-cache name="vehicleCache">
  <persistence passivation="false">
    <leveldb-store xmlns="urn:infinispan:config:store:leveldb:8.0"
      relative-to="/path/to/leveldb/data"
      shared="false"
      preload="true"/>
  </persistence>
</local-cache>
```

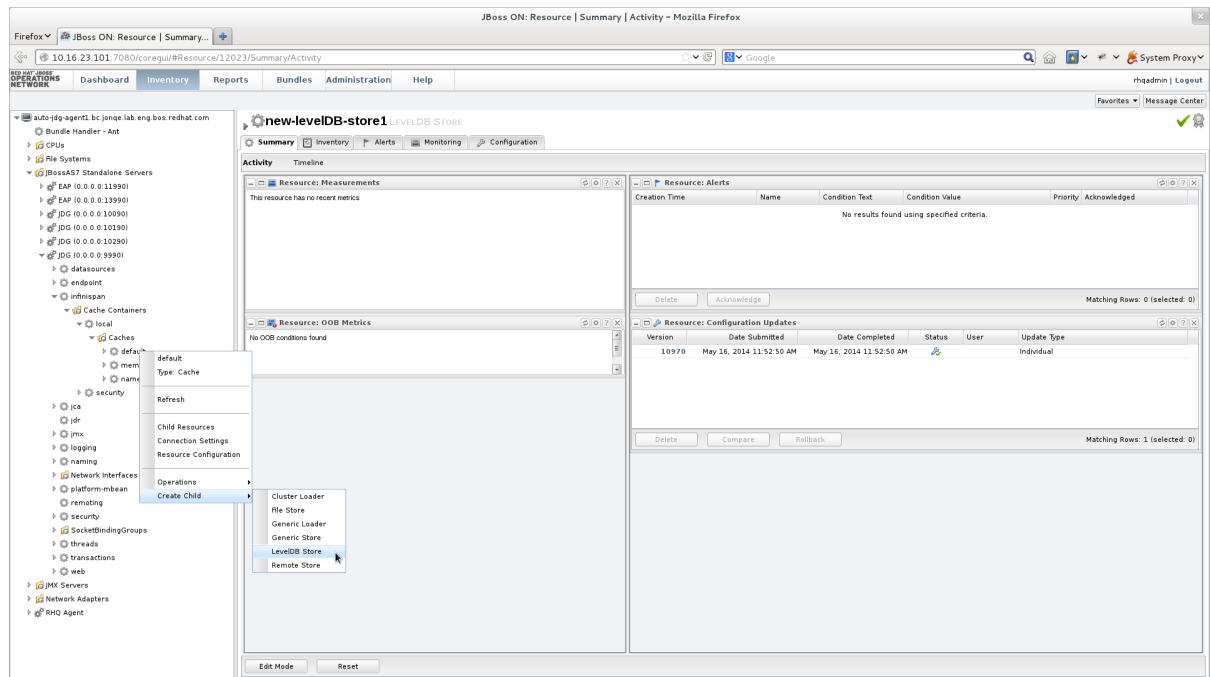
For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).

21.6.4. Configure a LevelDB Cache Store Using JBoss Operations Network

Use the following procedure to set up a new LevelDB cache store using the JBoss Operations Network.

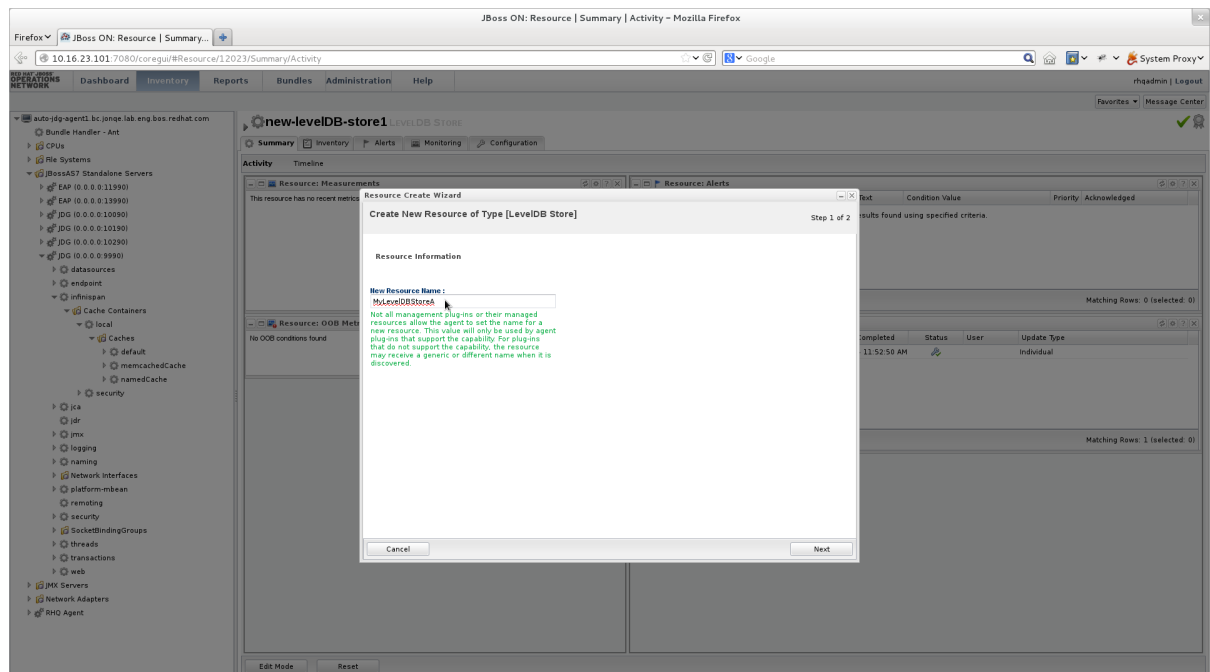
1. Ensure that Red Hat JBoss Operations Network 3.2 or higher is installed and started.
2. Install the Red Hat JBoss Data Grid Plugin Pack for JBoss Operations Network 3.2.0.
3. Ensure that JBoss Data Grid is installed and started.
4. Import JBoss Data Grid server into the inventory.
5. Configure the JBoss Data Grid connection settings.
6. Create a new LevelDB cache store as follows:

Figure 21.1. Create a new LevelDB Cache Store



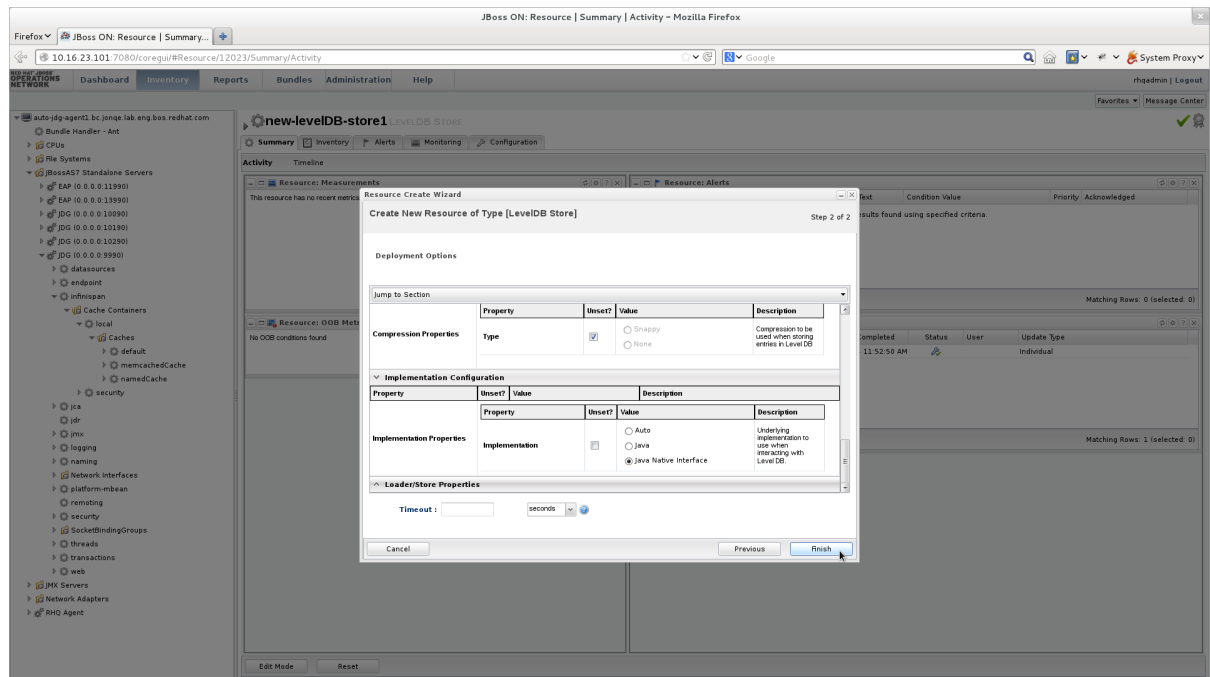
- a. Right-click the **default** cache.
 - b. In the menu, mouse over the option.
 - c. In the submenu, click menu:LevelDB Store[] .
7. Name the new LevelDB cache store as follows:

Figure 21.2. Name the new LevelDB Cache Store



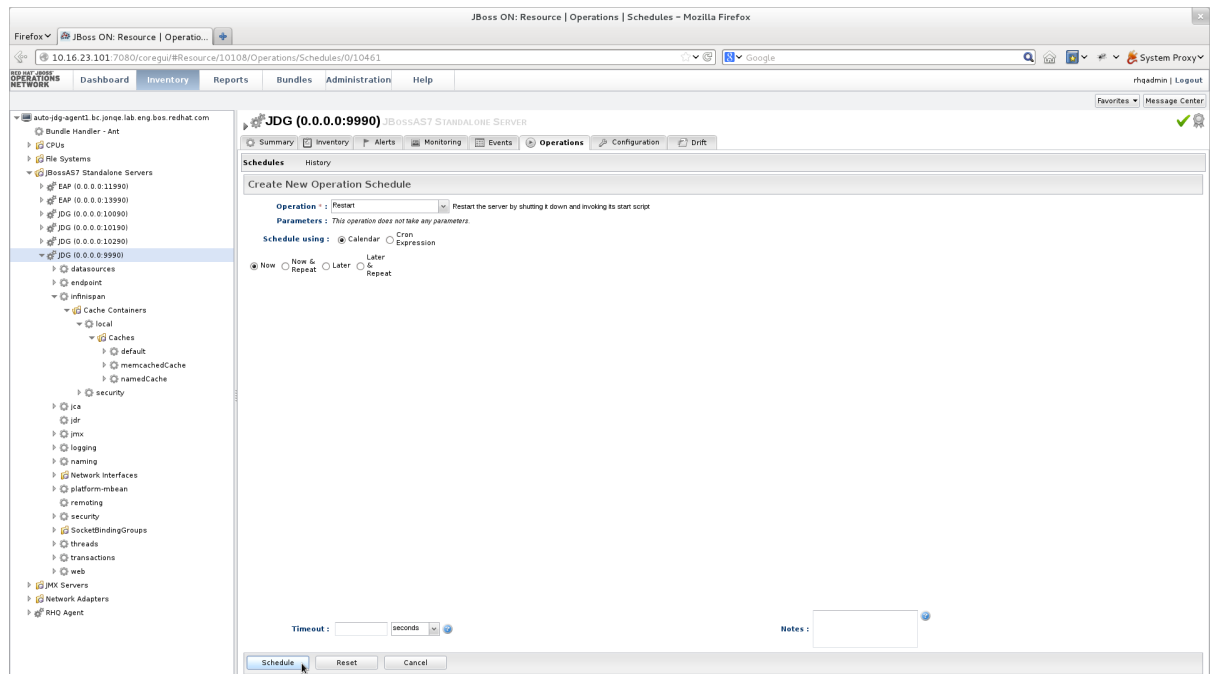
- a. In the Resource Create Wizard that appears, add a name for the new LevelDB Cache Store.
 - b. Click btn:[Next] to continue.
8. Configure the LevelDB Cache Store settings as follows:

Figure 21.3. Configure the LevelDB Cache Store Settings



- a. Use the options in the configuration window to configure a new LevelDB cache store.
 - b. Click menu:Finish[] to complete the configuration.
9. Schedule a restart operation as follows:

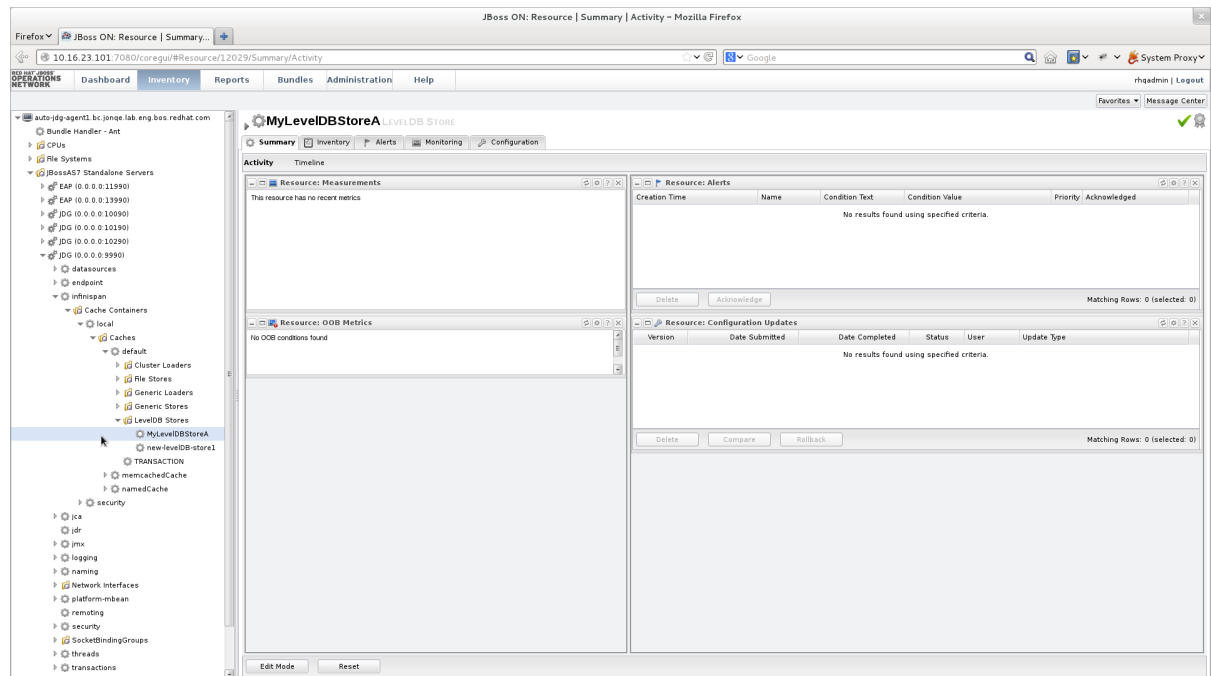
Figure 21.4. Schedule a Restart Operation



- a. In the screen's left panel, expand the JBoss AS7 Standalone Servers entry, if it is not currently expanded.
- b. Click JDG (0.0.0.0:9990) from the expanded menu items.
- c. In the screen's right panel, details about the selected server display. Click the menu:Operations[] tab.

- d. In the Operation drop-down box, select the Restart operation.
 - e. Select the radio button for the Now entry.
 - f. Click menu:Schedule[] to restart the server immediately.
10. Discover the new LevelDB cache store as follows:

Figure 21.5. Discover the New LevelDB Cache Store



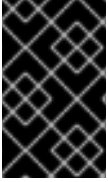
- a. In the screen's left panel, select each of the following items in the specified order to expand them: menu:JBoss AS7 Standalone Servers[JDG (0.0.0.0:9990) > infinispn > Cache Containers > local > Caches > default > LevelDB Stores]
- b. Click the name of your new LevelDB Cache Store to view its configuration information in the right panel.

21.7. JDBC BASED CACHE STORES

21.7.1. JDBC Based Cache Stores

Red Hat JBoss Data Grid offers several cache stores for use with common data storage formats. **JDBC** based cache stores are used with any cache store that exposes a **JDBC** driver. JBoss Data Grid offers the following **JDBC** based cache stores depending on the key to be persisted:

- **JdbcBinaryStore.**
- **JdbcStringBasedStore.**
- **JdbcMixedStore.**

**IMPORTANT**

Both Binary and Mixed JDBC stores are deprecated in JBoss Data Grid 7.1, and are not recommended for production use. It is recommended to utilize a String Based store instead.

21.7.2. JdbcBinaryStores**21.7.2.1. JdbcBinaryStores**

The **JdbcBinaryStore** supports all key types. It stores all keys with the same hash value (**hashCode** method on the key) in the same table row/blob. The hash value common to the included keys is set as the primary key for the table row/blob. As a result of this hash value, **JdbcBinaryStore** offers excellent flexibility but at the cost of concurrency and throughput.

As an example, if three keys (**k1**, **k2** and **k3**) have the same hash code, they are stored in the same table row. If three different threads attempt to concurrently update **k1**, **k2** and **k3**, they must do it sequentially because all three keys share the same row and therefore cannot be simultaneously updated.

**IMPORTANT**

Binary JDBC stores are deprecated in JBoss Data Grid 7.1, and are not recommended for production use. It is recommended to utilize a String Based store instead.

21.7.2.2. JdbcBinaryStore Configuration (Remote Client-Server Mode)

The following is a configuration for **JdbcBinaryStore** using Red Hat JBoss Data Grid's Remote Client-Server mode with Passivation enabled:

```
<local-cache name="customCache">
  <!-- Additional configuration elements here -->
  <binary-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
    passivation="${true/false}"
    preload="${true/false}"
    purge="${true/false}">
    <binary-keyed-table prefix="JDG">
      <id-column name="id"
        type="${id.column.type}"/>
      <data-column name="datum"
        type="${data.column.type}"/>
      <timestamp-column name="version"
        type="${timestamp.column.type}"/>
    </binary-keyed-table>
  </binary-keyed-jdbc-store>
</local-cache>
```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Remote Client-Server Mode\)](#).

21.7.2.3. JdbcBinaryStore Configuration (Library Mode)

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

■

```

<infinispan
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:infinispan:config:8.4
http://www.infinispan.org/schemas/infinispan-config-8.4.xsd
  urn:infinispan:config:store:jdbc:8.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
8.0.xsd"
  xmlns="urn:infinispan:config:8.4">
  <!-- Additional configuration elements here -->
  <persistence>
  <binary-keyed-jdbc-store xmlns="urn:infinispan:config:store:jdbc:8.0
  fetch-state="false"
  purge="false">
  <connection-pool connection-
url="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1"
  username="sa"
  driver="org.h2.Driver"/>
  <binary-keyed-table dropOnExit="true"
  createOnStart="true"
  prefix="ISPN_BUCKET_TABLE">
  <id-column name="ID_COLUMN"
  type="VARCHAR(255)" />
  <data-column name="DATA_COLUMN"
  type="BINARY" />
  <timestamp-column name="TIMESTAMP_COLUMN"
  type="BIGINT" />
  </binary-keyed-table>
  </binary-keyed-jdbc-store>
</persistence>

```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).

21.7.3. JdbcStringBasedStores

21.7.3.1. JdbcStringBasedStores

The **JdbcStringBasedStore** stores each entry in its own row in the table, instead of grouping multiple entries into each row, resulting in increased throughput under a concurrent load. It also uses a (pluggable) bijection that maps each key to a **String** object. The **key-to-string-mapper** interface defines the bijection.

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

21.7.3.2. JdbcStringBasedStore Configuration (Remote Client-Server Mode)

The following is a sample **JdbcStringBasedStore** for Red Hat JBoss Data Grid's Remote Client-Server mode:

```

<local-cache name="customCache">
  <!-- Additional configuration elements here -->
  <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
  passivation="true"
  preload="false"

```

```

    purge="false"
    shared="false"
    singleton="true">
    <string-keyed-table prefix="JDG">
        <id-column name="id"
            type="{id.column.type}"/>
        <data-column name="datum"
            type="{data.column.type}"/>
        <timestamp-column name="version"
            type="{timestamp.column.type}"/>
    </string-keyed-table>
</string-keyed-jdbc-store>
</local-cache>

```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Remote Client-Server Mode\)](#).

21.7.3.3. JdbcStringBasedStore Configuration (Library Mode)

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

```

<infinispan
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:infinispan:config:8.4
http://www.infinispan.org/schemas/infinispan-config-8.4.xsd
    urn:infinispan:config:store:jdbc:8.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
8.0.xsd"
    xmlns="urn:infinispan:config:8.4">
    <!-- Additional configuration elements here -->
    <persistence>
        <string-keyed-jdbc-store
xmlns="urn:infinispan:config:store:jdbc:8.0"
            fetch-state="false"
            purge="false"

key2StringMapper="org.infinispan.loaders.keymappers.DefaultTwoWayKey2StringMapper">
            <dataSource jndiUrl="java:jboss/datasources/JdbcDS"/>
            <string-keyed-table dropOnExit="true"
createOnStart="true"
prefix="ISPN_STRING_TABLE">
                <id-column name="ID_COLUMN"
                    type="VARCHAR(255)" />
                <data-column name="DATA_COLUMN"
                    type="BINARY" />
                <timestamp-column name="TIMESTAMP_COLUMN"
                    type="BIGINT" />
            </string-keyed-table>
        </string-keyed-jdbc-store>
    </persistence>

```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).

21.7.3.4. JdbcStringBasedStore Multiple Node Configuration (Remote Client-Server Mode)

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

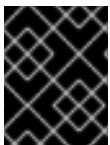
```
<subsystem xmlns="urn:infinispan:server:core:8.4" default-cache-
container="default">
  <cache-container <!-- Additional configuration information here --> >
    <!-- Additional configuration elements here -->
    <replicated-cache>
      <!-- Additional configuration elements here -->
      <string-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
        fetch-state="true"
        passivation="false"
        preload="false"
        purge="false"
        shared="false"
        singleton="true">
        <string-keyed-table prefix="JDG">
          <id-column name="id"
            type="${id.column.type}"/>
          <data-column name="datum"
            type="${data.column.type}"/>
          <timestamp-column name="version"
            type="${timestamp.column.type}"/>
        </string-keyed-table>
      </string-keyed-jdbc-store>
    </replicated-cache>
  </cache-container>
</subsystem>
```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Remote Client-Server Mode\)](#).

21.7.4. JdbcMixedStores

21.7.4.1. JdbcMixedStores

The **JdbcMixedStore** is a hybrid implementation that delegates keys based on their type to either the **JdbcBinaryStore** or **JdbcStringBasedStore**.



IMPORTANT

Mixed JDBC stores are deprecated in JBoss Data Grid 7.1, and are not recommended for production use. It is recommended to utilize a String Based store instead.

21.7.4.2. JdbcMixedStore Configuration (Remote Client-Server Mode)

The following is a configuration for a **JdbcMixedStore** for Red Hat JBoss Data Grid's Remote Client-Server mode:

```
<local-cache name="customCache">
  <mixed-keyed-jdbc-store datasource="java:jboss/datasources/JdbcDS"
```



```

    passivation="true"
    preload="false"
    purge="false">
<binary-keyed-table prefix="MIX_BKT2">
  <id-column name="id"
    type="{id.column.type}"/>
  <data-column name="datum"
    type="{data.column.type}"/>
  <timestamp-column name="version"
    type="{timestamp.column.type}"/>
</binary-keyed-table>
<string-keyed-table prefix="MIX_STR2">
  <id-column name="id"
    type="{id.column.type}"/>
  <data-column name="datum"
    type="{data.column.type}"/>
  <timestamp-column name="version"
    type="{timestamp.column.type}"/>
</string-keyed-table>
</mixed-keyed-jdbc-store>
</local-cache>

```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Remote Client-Server Mode\)](#).

21.7.4.3. JdbcMixedStore Configuration (Library Mode)

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

```

<infinispan
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:infinispan:config:8.4
http://www.infinispan.org/schemas/infinispan-config-8.4.xsd
  urn:infinispan:config:store:jdbc:8.0
http://www.infinispan.org/schemas/infinispan-cachestore-jdbc-config-
8.0.xsd"
  xmlns="urn:infinispan:config:8.4">
  <!-- Additional configuration elements here -->
  <persistence>
  <mixed-keyed-jdbc-store xmlns="urn:infinispan:config:store:jdbc:8.0"
    fetch-state="false"
    purge="false"
    key-to-string-
mapper="org.infinispan.persistence.keymappers.DefaultTwoWayKey2StringMapper">
    <connection-pool connection-
url="jdbc:h2:mem:infinispan_binary_based;DB_CLOSE_DELAY=-1"
    username="sa"
    driver="org.h2.Driver"/>
    <binary-keyed-table dropOnExit="true"
    createOnStart="true"
    prefix="ISPN_BUCKET_TABLE_BINARY">
    <id-column name="ID_COLUMN"
    type="VARCHAR(255)" />
    <data-column name="DATA_COLUMN"
    type="BINARY" />

```

```

    <timestamp-column name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </binary-keyed-table>
  <string-keyed-table dropOnExit="true"
    createOnStart="true"
    prefix="ISPN_BUCKET_TABLE_STRING">
    <id-column name="ID_COLUMN"
      type="VARCHAR(255)" />
    <data-column name="DATA_COLUMN"
      type="BINARY" />
    <timestamp-column name="TIMESTAMP_COLUMN"
      type="BIGINT" />
  </string-keyed-table>
</mixed-keyed-jdbc-store>
</persistence>

```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).

21.7.5. Cache Store Troubleshooting

21.7.5.1. IOExceptions with JdbcStringBasedStore

An `IOException` **Unsupported protocol version 48** error when using `JdbcStringBasedStore` indicates that your data column type is set to **VARCHAR**, **CLOB** or something similar instead of the correct type, **BLOB** or **VARBINARY**. Despite its name, `JdbcStringBasedStore` only requires that the keys are strings while the values can be any data type, so that they can be stored in a binary column.

21.8. THE REMOTE CACHE STORE

21.8.1. Remote Cache Stores

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

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

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

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

```

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

```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Remote Client-Server Mode\)](#).

21.8.3. Remote Cache Store Configuration (Library Mode)

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

```
<persistence passivation="false">
  <remote-store xmlns="urn:infinispan:config:store:remote:8.0"
    cache="default"
    fetch-state="false"
    shared="true"
    preload="false"
    purge="false"
    tcp-no-delay="true"
    key-size-estimate="62"
    value-size-estimate="512"
    force-return-values="false">
    <remote-server host="127.0.0.1"
      port="1971" />
    <connectionPool max-active="99"
      max-idle="97"
      max-total="98" />
  </remote-store>
</persistence>
```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).

21.8.4. Define the Outbound Socket for the Remote Cache Store

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

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

Define the Outbound Socket

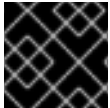
```
<server>
  <!-- Additional configuration elements here -->
  <socket-binding-group name="standard-sockets"
    default-interface="public"
    port-offset="{jboss.socket.binding.port-offset:0}">
    <!-- Additional configuration elements here -->
    <outbound-socket-binding name="remote-store-hotrod-server">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

21.9. JPA CACHE STORE

21.9.1. JPA Cache Stores

The JPA (Java Persistence API) Cache Store stores cache entries in the database using a formal schema, which allows other applications to read the persisted data and load data provided by other

applications into Red Hat JBoss Data Grid. The database should not be used by the other applications concurrently with JBoss Data Grid.



IMPORTANT

In Red Hat JBoss Data Grid, JPA cache stores are only supported in Library mode.

21.9.2. JPA Cache Store Sample XML Configuration (Library Mode)

To configure JPA Cache Stores using XML in Red Hat JBoss Data Grid, add the following configuration to the *infinispan.xml* file:

```
<local-cache name="users">
  <!-- Insert additional configuration elements here -->
  <persistence passivation="false">
    <jpa-store xmlns="urn:infinispan:config:store:jpa:8.0"
      shared="true"
      preload="true"
      persistence-unit="MyPersistenceUnit"
      entity-
class="org.infinispan.loaders.jpa.entity.User" />
  </persistence>
</local-cache>
```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).

21.9.3. Storing Metadata in the Database

When **storeMetadata** is set to **true** (default value), meta information about the entries such as expiration, creation and modification timestamps, and versioning is stored in the database. JBoss Data Grid stores the metadata in an additional table named *_ispn_metadata_* because the entity table has a fixed layout that cannot accommodate the metadata.

The structure of this table depends on the database in use. Enable the automatic creation of this table using the same database as the test environment and then transfer the structure to the production database.

Configure persistence.xml for Metadata Entities

1. Using Hibernate as the JPA implementation allows automatic creation of these tables using the property **hibernate.hbm2ddl.auto** in *persistence.xml* as follows:

```
<property name="hibernate.hbm2ddl.auto" value="update" />
```

2. Declare the metadata entity class to the JPA provider by adding the following to *persistence.xml*:

```
<class>org.infinispan.persistence.jpa.impl.MetadataEntity</class>
```

As outlined, metadata is always stored in a new table. If metadata information collection and storage is not required, set the **storeMetadata** attribute to **false** in the JPA Store configuration.

21.9.4. Deploying JPA Cache Stores in Various Containers

Red Hat JBoss Data Grid's JPA Cache Store implementations are deployed normally for all supported containers, except Red Hat JBoss Enterprise Application Platform. JBoss Data Grid's JBoss EAP modules contain the JPA cache store and related libraries (such as Hibernate). As a result, the relevant libraries are not packaged inside the application, but instead the application refers to the libraries in the JBoss EAP modules that have them installed.

These modules are not required for containers other than JBoss EAP. As a result, all the relevant libraries are packaged in the application's **WAR/EAR** file, such as with the following Maven dependency:

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-cachestore-jpa</artifactId>
  <version>8.4.0.Final-redhat-2</version>
</dependency>
```

Deploy JPA Cache Stores in JBoss EAP 6.3.x and earlier

- To add dependencies from the JBoss Data Grid modules to the application's classpath, provide the JBoss EAP deployer a list of dependencies in one of the following ways:
 - Add a dependency configuration to the *MANIFEST.MF* file:

```
Manifest-Version: 1.0
Dependencies: org.infinispan:jdg-7.1 services,
org.infinispan.persistence.jpa:jdg-7.1 services
```

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

```
<jboss-deployment-structure xmlns="urn:jboss:deployment-
structure:1.2">
  <deployment>
    <dependencies>
      <module name="org.infinispan.persistence.jpa"
slot="jdg-7.1" services="export"/>
      <module name="org.infinispan" slot="jdg-7.1"
services="export"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

Deploy JPA Cache Stores in JBoss EAP 6.4 and later

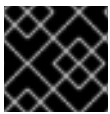
1. Add the following property in *persistence.xml*:

```
<persistence-unit>
  [...]
  <properties>
    <property name="jboss.as.jpa.providerModule" value="application"
/>
  </properties>
</persistence-unit>
```

2. Add the following dependencies to the *jboss-deployment-structure.xml*:

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.infinispan" slot="jdg-7.1"/>
      <module name="org.jgroups" slot="jdg-7.1"/>
      <module name="org.infinispan.persistence.jpa"
slot="jdg-7.1" services="export"/>
      <module name="org.hibernate"/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

3. Add any additional dependencies, such as additional JDG modules, are in use add these to the **dependencies** section in *jboss-deployment-structure.xml*.



IMPORTANT

JPA Cache Store is not supported in Apache Karaf in JBoss Data Grid 7.1.

21.10. CASSANDRA CACHE STORE

21.10.1. Cassandra Cache Store

Red Hat JBoss Data Grid allows Apache Cassandra to function as a Cache Store, leveraging their distributed database architecture to provide a virtually unlimited, horizontally scalable persistent store for cache entries.

In order to use the Cassandra Cache Store an appropriate keyspace must first be created on the Cassandra database. This may either be performed automatically or by enabling the **auto-create-keyspace** parameter in the cache store configuration. A sample keyspace creation is demonstrated below:

```
CREATE KEYSPACE IF NOT EXISTS Infinispan WITH replication =
{'class':'SimpleStrategy', 'replication_factor':1};
CREATE TABLE Infinispan.InfinispanEntries (key blob PRIMARY KEY, value
blob, metadata blob);
```

21.10.2. Enabling the Cassandra Cache Store

The Cassandra Cache Store is included based on the downloaded distribution. The following indicates where this is located, and steps to enable it if required:

- **Library Mode** - The *infinispan-cachestore-cassandra-8.4.0.Final-redhat-2-deployable.jar* is included in the *jboss-datagrid-`{jdg-version}`-library/* directory, and may be added to any projects that are using the Cassandra Cache Store.
- **Remote Client-Server Mode** - The Cassandra Cache Store is prepackaged in the *modules/* directory of the server, and may be used by default with no additional configuration necessary.

- **JBoss Data Grid modules for JBoss EAP** - The Cassandra Cache Store is included in the modules distributed, and may be added by using the `org.infinispan.persistence.cassandra` as the module name.

21.10.3. Cassandra Cache Store Sample XML Configuration (Remote Client-Server Mode)

In Remote Client-Server mode the Cassandra Cache Store is defined by using the class `org.infinispan.persistence.cassandra.CassandraStore` and defining the properties individually within the store.

The following configuration snippet provides an example on how to define a Cassandra Cache Store inside of an xml file:

```
<local-cache name="cassandrache">
  <locking acquire-timeout="30000" concurrency-level="1000"
striping="false"/>
  <transaction mode="NONE"/>
  <store name="casstore1"
    class="org.infinispan.persistence.cassandra.CassandraStore"
    shared="true"
    passivation="false">
    <property name="autoCreateKeyspace">true</property>
    <property name="keyspace">store1</property>
    <property name="entryTable">entries1</property>
    <property name="consistencyLevel">LOCAL_ONE</property>
    <property name="serialConsistencyLevel">SERIAL</property>
    <property
name="servers">127.0.0.1[9042],127.0.0.1[9041]</property>
    <property
name="connectionPool.heartbeatIntervalSeconds">30</property>
    <property name="connectionPool.idleTimeoutSeconds">120</property>
    <property name="connectionPool.poolTimeoutMillis">5</property>
  </store>
</local-cache>
```

21.10.4. Cassandra Cache Store Sample XML Configuration (Library Mode)

In Library Mode the Cassandra Cache Store may be configured using two different methods:

- **Option 1:** Using the same method discussed for Remote Client-Server Mode, found in [Cassandra Cache Store Sample XML Configuration \(Remote Client-Server Mode\)](#).
- **Option 2:** Using the `cassandra-store` schema. The following snippet shows an example configuration defining a Cassandra Cache Store:

```
<cache-container default-cache="cassandrache">
  <local-cache name="cassandrache">
    <persistence passivation="false">
      <cassandra-store
xmlns="urn:infinispan:config:store:cassandra:8.2"
      auto-create-keyspace="true"
      keyspace="Infinispan"
      entry-table="InfinispanEntries" shared="true">
      <cassandra-server host="127.0.0.1" port="9042" />
```

```

        <connection-pool heartbeat-interval-seconds="30"
            idle-timeout-seconds="120"
            pool-timeout-millis="5" />
    </cassandra-store>
</persistence>
</local-cache>
</cache-container>

```

21.10.5. Cassandra Configuration Parameters

When defining a backing Cassandra instance in Library Mode one or more **cassandra-server** elements may be specified in the configuration. Each of the elements has the following properties:

Table 21.1. Cassandra Server Configuration Parameters

Parameter Name	Description	Default Value
host	The hostname or ip address of a Cassandra server.	127.0.0.1
port	The port on which the server is listening.	9042

The following properties may be configured on the Cassandra Cache Store:

Table 21.2. Cassandra Configuration Parameter

Parameter Name	Description	Default Value
auto-create-keyspace	Determines whether the keyspace and entry table should be automatically created on startup.	true
keyspace	Name of the keyspace to use.	Infinispan
entry-table	Name of the table storing entries.	InfinispanEntries
consistency-level	Consistency level to use for the queries.	LOCAL_ONE
serial-consistency-level	Serial consistency level to use for the queries.	SERIAL

A **connection-pool** may also be defined with the following elements:

Table 21.3. Connection Pool Configuration Parameters

Parameter Name	Description	Default Value
----------------	-------------	---------------

Parameter Name	Description	Default Value
pool-timeout-millis	Time that the driver blocks when no connection from hosts pool is available. After this timeout, the driver will try the next host.	5
heartbeat-interval-seconds	Application-side heartbeat to avoid the connections being dropped when no activity is happening. Set to 0 to disable.	30
idle-timeout-seconds	Timeout before an idle connection is removed.	120

21.11. CUSTOM CACHE STORES

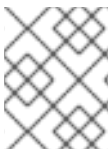
21.11.1. Custom Cache Stores

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

In order to create a custom cache store (or loader), implement all or a subset of the following interfaces based on the need:

- **CacheLoader**
- **CacheWriter**
- **AdvancedCacheLoader**
- **AdvancedCacheWriter**
- **ExternalStore**
- **AdvancedLoadWriteStore**

See [Cache Loaders and Cache Writers](#) for individual functions of the interfaces.



NOTE

If the **AdvancedCacheWriter** is not implemented, the expired entries cannot be purged or cleared using the given writer.



NOTE

If the **AdvancedCacheLoader** is not implemented, the entries stored in the given loader will not be used for preloading.

To migrate the existing cache store to the new API or to write a new store implementation, use **SingleFileStore** as an example. To view the **SingleFileStore** example code, download the JBoss Data Grid source code.

Use the following procedure to download **SingleFileStore** example code from the Customer Portal:

Download JBoss Data Grid Source Code

1. To access the Red Hat Customer Portal, navigate to <https://access.redhat.com/home> in a browser.
2. Click menu:Downloads[] .
3. In the section labeled JBoss Development and Management , click menu:Red Hat JBoss Data Grid[] .
4. Enter the relevant credentials in the Red Hat Login and Password fields and click menu:Log In[] .
5. From the list of downloadable files, locate Red Hat JBoss Data Grid 7 Source Code and click menu:Download[] . Save and unpack it in a desired location.
6. Locate the **SingleFileStore** source code by navigating through `jboss-datagrid-7.1.0-sources/infinispan-8.4.0.Final-redhat-2-src/core/src/main/java/org/infinispan/persistence/file/SingleFileStore.java` .

21.11.2. Custom Cache Store Maven Archetype

An easy way to get started with developing a Custom Cache Store is to use the Maven archetype; creating an archetype will generate a new Maven project with the correct directory layout and sample code.

Generate a Maven Archetype

1. Ensure the JBoss Data Grid Maven repository has been installed by following the instructions in the Red Hat JBoss Data Grid *Getting Started Guide* .
2. Open a command prompt and execute the following command to generate an archetype in the current directory:

```
mvn -Dmaven.repo.local="path/to/unzipped/jboss-datagrid-7.1.0-maven-repository/"
  archetype:generate
    -DarchetypeGroupId=org.infinispan
    -DarchetypeArtifactId=custom-cache-store-archetype
    -DarchetypeVersion=0.0.2.Final-redhat-2
```



NOTE

The above command has been broken into multiple lines for readability; however, when executed this command and all arguments must be on a single line.

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

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

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

Custom Cache Store Configuration

```
<distributed-cache name="cacheStore" mode="SYNC" segments="20" owners="2"
remote-timeout="30000">
  <store class="my.package.CustomCacheStore">
    <property name="customStoreProperty">10</property>
  </store>
</distributed-cache>
```

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Remote Client-Server Mode\)](#).

21.11.3.2. Option 1: Add Custom Cache Store using deployments (Remote Client-Server Mode)

Deploy Custom Cache Store .jar file to JDG server using deployments

1. Add the following Java service loader file **META-INF/services/org.infinispan.persistence.spi.AdvancedLoadWriteStore** to the module and add a reference to the Custom Cache Store Class, such as seen below:

```
my.package.CustomCacheStore
```

2. Copy the jar to the **\$JDG_HOME/standalone/deployments/** directory.
3. If the .jar file is available the server the following message will be displayed in the logs:

```
JBAS010287: Registering Deployed Cache Store service for store
'my.package.CustomCacheStore'
```

4. In the **infinispan-core** subsystem add an entry for the cache inside a **cache-container**, specifying the class that overrides one of the interfaces from [Custom Cache Stores](#):

```
<subsystem xmlns="urn:infinispan:server:core:8.4">
  [...]
  <distributed-cache name="cacheStore" mode="SYNC" segments="20"
owners="2" remote-timeout="30000">
    <store class="my.package.CustomCacheStore">
      <!-- If custom properties are included these may be specified
as below -->
      <property name="customStoreProperty">10</property>
    </store>
  </distributed-cache>
  [...]
</subsystem>
```

21.11.3.3. Option 2: Add Custom Cache Store using the CLI (Remote Client-Server Mode)

Deploying Custom Cache Store .jar file to JDG server using the CLI

1. Connect to the JDG server by running the below command:

```
[ $JDG_HOME ] $ bin/cli.sh --connect --controller=$IP:$PORT
```

2. Deploy the .jar file by executing the following command:

```
deploy /path/to/artifact.jar
```

21.11.3.4. Option 3: Add Custom Cache Store using JON (Remote Client-Server Mode)

Deploying Custom Cache Store .jar file to JDG server using JBoss Operation Network

1. Log into JON.
2. Navigate to **Bundles** along the upper bar.
3. Click the **New** button and choose the **Recipe** radio button.
4. Insert a deployment bundle file content that references the store, similar to the following example:

```
<?xml version="1.0"?>
<project name="cc-bundle" default="main"
xmlns:rhq="antlib:org.rhq.bundle">

  <rhq:bundle name="Mongo DB Custom Cache Store" version="1.0"
description="Custom Cache Store">
    <rhq:deployment-unit name="JDG" compliance="full">
      <rhq:file name="custom-store.jar"/>
    </rhq:deployment-unit>
  </rhq:bundle>

  <target name="main" />
</project>
```

5. Proceed with **Next** button to **Bundle Groups** configuration wizard page and proceed with **Next** button once again.
6. Locate custom cache store .jar file using file uploader and **Upload** the file.
7. Proceed with **Next** button to **Summary** configuration wizard page. Proceed with **Finish** button in order to finish bundle configuration.
8. Navigate back to the **Bundles** tab along the upper bar.
9. Select the newly created bundle and click **Deploy** button.
10. Enter **Destination Name** and choose the proper Resource Group; this group should only consist of JDG servers.
11. Choose **Install Directory** from **Base Location's** radio box group.
12. Enter **/standalone/deployments** in **Deployment Directory** text field below.

13. Proceed with the wizard using the default options.
14. Validate the deployment using the following command on the server's host:

```
find $JDG_HOME -name "custom-store.jar"
```

15. Confirm the bundle has been installed in `$JDG_HOME/standalone/deployments`.

Once the above steps are completed the .jar file will be successfully uploaded and registered by the JDG server.



NOTE

The JON plugin has been deprecated in JBoss Data Grid 7.1 and is expected to be removed in a subsequent version.

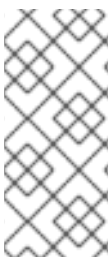
21.11.4. Custom Cache Store Configuration (Library Mode)

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

Custom Cache Store Configuration

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

For details about the elements and parameters used in this sample configuration, see [Cache Store Configuration Details \(Library Mode\)](#).



NOTE

The Custom Cache Store classes must be in the classpath where Red Hat JBoss Data Grid is used. Most often this is accomplished by packaging the Custom Cache Store in with the application; however, it may also be accomplished by defining the Custom Cache Store as a module to EAP and listed as a dependency, as discussed in the Red Hat JBoss Enterprise Application Platform *Administration and Configuration Guide*.

PART IX. SET UP PASSIVATION

CHAPTER 22. ACTIVATION AND PASSIVATION MODES

22.1. ACTIVATION AND PASSIVATION MODES

Activation is the process of loading an entry into memory and removing it from the cache store. Activation occurs when a thread attempts to access an entry that is in the store but not the memory (namely a passivated entry).

Passivation mode allows entries to be stored in the cache store after they are evicted from memory. Passivation prevents unnecessary and potentially expensive writes to the cache store. It is used for entries that are frequently used or referenced and therefore not evicted from memory.

While passivation is enabled, the cache store is used as an overflow tank, similar to virtual memory implementation in operating systems that swap memory pages to disk.

The passivation flag is used to toggle passivation mode, a mode that stores entries in the cache store only after they are evicted from memory.

22.2. PASSIVATION MODE BENEFITS

The primary benefit of passivation mode is that it prevents unnecessary and potentially expensive writes to the cache store. This is particularly useful if an entry is frequently used or referenced and therefore is not evicted from memory.

22.3. CONFIGURE PASSIVATION

In Red Hat JBoss Data Grid's Remote Client-Server mode, add the **passivation** parameter to the cache store element to toggle passivation for it:

Toggle Passivation in Remote Client-Server Mode

```
<local-cache name="customCache"/>
  <!-- Additional configuration elements for local-cache here -->
  <file-store passivation="true"
    <!-- Additional configuration elements for file-store here -->
</local-cache>
```

In Library mode, add the **passivation** parameter to the **persistence** element to toggle passivation:

Toggle Passivation in Library Mode

```
<persistence passivation="true">
  <!-- Additional configuration elements here -->
</persistence>
```

22.4. EVICTION AND PASSIVATION

22.4.1. Eviction and Passivation

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

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

22.4.2. Eviction and Passivation Usage

If the eviction policy caused the eviction of an entry from the cache while passivation is enabled, the following occur as a result:

- A notification regarding the passivated entry is emitted to the cache listeners.
- The evicted entry is stored.

When an attempt to retrieve an evicted entry is made, the entry is lazily loaded into memory from the cache loader. After the entry and its children are loaded, they are removed from the cache loader and a notification regarding the entry's activation is sent to the cache listeners.

22.4.3. Eviction Example when Passivation is Disabled

The following example indicates the state of the memory and the persistent store during eviction operations with passivation disabled.

Table 22.1. Eviction when Passivation is Disabled

Step	Key in Memory	Key on Disk
Insert keyOne	Memory: keyOne	Disk: keyOne
Insert keyTwo	Memory: keyOne , keyTwo	Disk: keyOne , keyTwo
Eviction thread runs, evicts keyOne	Memory: keyTwo	Disk: keyOne , keyTwo
Read keyOne	Memory: keyOne , keyTwo	Disk: keyOne , keyTwo
Eviction thread runs, evicts keyTwo	Memory: keyOne	Disk: keyOne , keyTwo
Remove keyTwo	Memory: keyOne	Disk: keyOne

22.4.4. Eviction Example when Passivation is Enabled

The following example indicates the state of the memory and the persistent store during eviction operations with passivation enabled.

Table 22.2. Eviction when Passivation is Enabled

Step	Key in Memory	Key on Disk
Insert keyOne	Memory: keyOne	Disk:

Step	Key in Memory	Key on Disk
Insert keyTwo	Memory: keyOne , keyTwo	Disk:
Eviction thread runs, evicts keyOne	Memory: keyTwo	Disk: keyOne
Read keyOne	Memory: keyOne , keyTwo	Disk:
Eviction thread runs, evicts keyTwo	Memory: keyOne	Disk: keyTwo
Remove keyTwo	Memory: keyOne	Disk:

PART X. SET UP CACHE WRITING

CHAPTER 23. CACHE WRITING MODES

23.1. CACHE WRITING MODES

Red Hat JBoss Data Grid presents configuration options with a single or multiple cache stores. This allows it to store data in a persistent location, for example a shared **JDBC** database or a local file system. JBoss Data Grid supports two caching modes:

- Write-Through (Synchronous)
- Write-Behind (Asynchronous)

23.2. WRITE-THROUGH CACHING

23.2.1. Write-Through Caching

The Write-Through (or Synchronous) mode in Red Hat JBoss Data Grid ensures that when clients update a cache entry (usually via a `Cache.put()` invocation), the call does not return until JBoss Data Grid has located and updated the underlying cache store. This feature allows updates to the cache store to be concluded within the client thread boundaries.

23.2.2. Write-Through Caching Benefits and Disadvantages

Write-Through Caching Benefits

The primary advantage of the Write-Through mode is that the cache and cache store are updated simultaneously, which ensures that the cache store remains consistent with the cache contents.

Write-Through Caching Disadvantages

Due to the cache store being updated simultaneously with the cache entry, there is a possibility of reduced performance for cache operations that occur concurrently with the cache store accesses and updates.

23.2.3. Write-Through Caching Configuration (Library Mode)

No specific configuration operations are required to configure a Write-Through or synchronous cache store. All cache stores are Write-Through or synchronous unless explicitly marked as Write-Behind or asynchronous. The following procedure demonstrates a sample configuration file of a Write-Through unshared local file cache store.

Configure a Write-Through Local File Cache Store

```
<local-cache name="persistentCache">
  <persistence>
    <file-store fetch-state="true"
      purge="false"
      shared="false"
      location="{java.io.tmpdir}"/>
  </persistence>
</local-cache>
```

1. The `name` parameter specifies the name of the `local-cache` to use.

2. The **fetch-state** parameter determines whether the persistent state is fetched when joining a cluster. Set this to **true** if using a replication and invalidation in a clustered environment. Additionally, if multiple cache stores are chained, only one cache store can have this property enabled. If a shared cache store is used, the cache does not allow a persistent state transfer despite this property being set to **true**. The **fetch-state** parameter is **false** by default.
3. The **purge** parameter specifies whether the cache is purged when initially started.
4. The **shared** parameter is used when multiple cache instances share a cache store and is now defined at the cache store level. This parameter can be set to prevent multiple cache instances writing the same modification multiple times. Valid values for this parameter are **true** and **false**.

23.3. WRITE-BEHIND CACHING

23.3.1. Write-Behind Caching

In Red Hat JBoss Data Grid's Write-Behind (Asynchronous) mode, cache updates are asynchronously written to the cache store. Asynchronous updates ensure that cache store updates are carried out by a thread different from the client thread interacting with the cache.

One of the foremost advantages of the Write-Behind mode is that the cache operation performance is not affected by the underlying store update. However, because of the asynchronous updates, for a brief period the cache store contains stale data compared to the cache.

23.3.2. About Unscheduled Write-Behind Strategy

In the Unscheduled Write-Behind Strategy mode, Red Hat JBoss Enterprise Data Grid attempts to store changes as quickly as possible by applying pending changes in parallel. This results in multiple threads waiting for modifications to conclude. Once these modifications are concluded, the threads become available and the modifications are applied to the underlying cache store.

This strategy is ideal for cache stores with low latency and low operational costs. An example of this is a local unshared file based cache store in which the cache store is local to the cache itself. Using this strategy the period of time where an inconsistency exists between the contents of the cache and the contents of the cache store is reduced to the shortest possible interval.

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

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

The write-behind Element

```
<file-store passivation="false"
  path="{PATH}"
  purge="true"
  shared="false">
  <write-behind modification-queue-size="1024"
    shutdown-timeout="25000"
    flush-lock-timeout="15000"
    thread-pool-size="5" />
</file-store>
```

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

1. The **modification-queue-size** parameter sets the modification queue size for the asynchronous store. If updates occur faster than the cache store can process the queue, the asynchronous store behaves like a synchronous store. The store behavior remains synchronous and blocks elements until the queue is able to accept them, after which the store behavior becomes asynchronous again.
2. The **shutdown-timeout** parameter specifies the time in milliseconds after which the cache store is shut down. When the store is stopped some modifications may still need to be applied. Setting a large timeout value will reduce the chance of data loss. The default value for this parameter is **25000**.
3. The **flush-lock-timeout** parameter specifies the time (in milliseconds) to acquire the lock that guards the state to be periodically flushed. The default value for this parameter is **15000**.
4. The **thread-pool-size** parameter specifies the size of the thread pool. The threads in this thread pool apply modifications to the cache store. The default value for this parameter is **5**.

23.3.4. Unscheduled Write-Behind Strategy Configuration (Library Mode)

To enable the write-behind strategy of the cache entries to a store, add the **async** element to the store configuration as follows:

The **async** Element

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

1. The **async** element uses the following configuration parameters: . The **modificationQueueSize** parameter sets the modification queue size for the asynchronous store. If updates occur faster than the cache store can process the queue, the asynchronous store behaves like a synchronous store. The store behavior remains synchronous and blocks elements until the queue is able to accept them, after which the store behavior becomes asynchronous again.
2. The **shutdownTimeout** parameter specifies the time in milliseconds after which the cache store is shut down. This provides time for the asynchronous writer to flush data to the store when a cache is shut down. The default value for this parameter is **25000**.
3. The **flushLockTimeout** parameter specifies the time (in milliseconds) to acquire the lock that guards the state to be periodically flushed. The default value for this parameter is **15000**.
4. The **threadPoolSize** parameter specifies the number of threads that concurrently apply modifications to the store. The default value for this parameter is **5**.

PART XI. MONITOR CACHES AND CACHE MANAGERS

CHAPTER 24. SET UP JAVA MANAGEMENT EXTENSIONS (JMX)

24.1. ABOUT JAVA MANAGEMENT EXTENSIONS (JMX)

Java Management Extension (JMX) is a Java based technology that provides tools to manage and monitor applications, devices, system objects, and service oriented networks. Each of these objects is managed, and monitored by **MBeans**.

JMX is the de facto standard for middleware management and administration. As a result, **JMX** is used in Red Hat JBoss Data Grid to expose management and statistical information.

24.2. USING JMX WITH RED HAT JBOSS DATA GRID

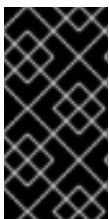
Management in Red Hat JBoss Data Grid instances aims to expose as much relevant statistical information as possible. This information allows administrators to view the state of each instance. While a single installation can comprise of tens or hundreds of such instances, it is essential to expose and present the statistical information for each of them in a clear and concise manner.

In JBoss Data Grid, JMX is used in conjunction with JBoss Operations Network (JON) to expose this information and present it in an orderly and relevant manner to the administrator.

24.3. JMX STATISTIC LEVELS

JMX statistics can be enabled at two levels:

- At the cache level, where management information is generated by individual cache instances.
- At the *CacheManager* level, where the *CacheManager* is the entity that governs all cache instances created from it. As a result, the management information is generated for all these cache instances instead of individual caches.



IMPORTANT

In Red Hat JBoss Data Grid, statistics are enabled by default in Remote Client-Server mode and disabled by default for Library mode. While statistics are useful in assessing the status of JBoss Data Grid, they adversely affect performance and must be disabled if they are not required.

24.4. ENABLE JMX FOR CACHE INSTANCES

At the Cache level, JMX statistics can be enabled either declaratively or programmatically, as follows.

Enable JMX Declaratively at the Cache Level

Add the following snippet within either the `<default>` element for the default cache instance, or under the target `<local-cache>` element for a specific cache:

```
<jmxStatistics enabled="true"/>
```

24.5. ENABLE JMX FOR CACHEMANAGERS

At the *CacheManager* level, JMX statistics can be enabled either declaratively or programmatically, as follows.

Enable JMX Declaratively at the CacheManager Level

Add the following in the <global> element to enable JMX declaratively at the *CacheManager* level:

```
<globalJmxStatistics enabled="true"/>
```

24.6. DISABLING THE CACHESTORE VIA JMX WHEN USING ROLLING UPGRADES

Red Hat JBoss Data Grid allows the *CacheStore* to be disabled via JMX by invoking the `disconnectSource` operation on the `RollingUpgradeManager` MBean.

See Also: [RollingUpgradeManager](#)

24.7. MULTIPLE JMX DOMAINS

Multiple JMX domains are used when multiple *CacheManager* instances exist on a single virtual machine, or if the names of cache instances in different *CacheManagers* clash.

To resolve this issue, name each *CacheManager* in manner that allows it to be easily identified and used by monitoring tools such as JMX and JBoss Operations Network.

Set a CacheManager Name Declaratively

Add the following snippet to the relevant *CacheManager* configuration:

```
<globalJmxStatistics enabled="true" cacheManagerName="Hibernate2LC"/>
```

24.8. MBEANS

24.8.1. MBeans

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

Red Hat JBoss Data Grid provides **MBeans** that monitor and manage multiple aspects. For example, **MBeans** that provide statistics on the transport layer are provided. If a JBoss Data Grid server is configured with **JMX** statistics, an **MBean** that provides information such as the hostname, port, bytes read, bytes written and the number of worker threads exists at the following location:

```
jboss.infinispan:type=Server,name=<Memcached|Hotrod>,component=Transport
```

MBeans are available under two **JMX** domains:

- `jboss.as` - these **MBeans** are created by the server subsystem.
- `jboss.infinispan` - these **MBeans** are symmetric to those created by embedded mode.

Only the **MBeans** under `jboss.infinispan` should be used for Red Hat JBoss Data Grid, as the ones under `jboss.as` are for Red Hat JBoss Enterprise Application Platform.

**NOTE**

A full list of available MBeans, their supported operations and attributes, is available in the Appendix

24.8.2. Understanding MBeans

When **JMX** reporting is enabled at either the Cache Manager or Cache level, use a standard **JMX** GUI such as JConsole or VisualVM to connect to a Java Virtual Machine running Red Hat JBoss Data Grid. When connected, the following **MBeans** are available:

- If Cache Manager-level **JMX** statistics are enabled, an **MBean** named **jboss.infinispan:type=CacheManager, name="DefaultCacheManager"** exists, with properties specified by the Cache Manager **MBean**.
- If the cache-level **JMX** statistics are enabled, multiple **MBeans** display depending on the configuration in use. For example, if a write behind cache store is configured, an **MBean** that exposes properties that belong to the cache store component is displayed. All cache-level **MBeans** use the same format:

```
jboss.infinispan:type=Cache, name="<name-of-cache>( <cache-
mode>)", manager="<name-of-cache-manager>", component=<component-name>
```

In this format:

- Specify the default name for the cache using the **cache-container** element's **default-cache** attribute.
- The **cache-mode** is replaced by the cache mode of the cache. The lower case version of the possible enumeration values represents the cache mode.
- The **component-name** is replaced by one of the **JMX** component names from the **JMX** reference documentation.

As an example, the cache store **JMX** component **MBean** for a default cache configured for synchronous distribution would be named as follows:

```
jboss.infinispan:type=Cache, name="default(dist_sync)",
manager="default", component=CacheStore
```

Each cache and cache manager name is within quotation marks to prevent the use of unsupported characters in these user-defined names.

24.8.3. Registering MBeans in Non-Default MBean Servers

The default location where all the MBeans used are registered is the standard JVM MBeanServer platform. Users can set up an alternative MBeanServer instance as well. Implement the MBeanServerLookup interface to ensure that the **getMBeanServer()** method returns the desired (non default) MBeanServer.

To set up a non default location to register your MBeans, create the implementation and then configure Red Hat JBoss Data Grid with the fully qualified name of the class. An example is as follows:

To Add the Fully Qualified Domain Name Declaratively

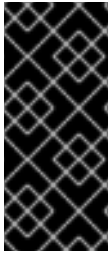
Add the following snippet:

```
<globalJmxStatistics enabled="true"  
mBeanServerLookup="com.acme.MyMBeanServerLookup"/>
```

CHAPTER 25. SET UP JBOSS OPERATIONS NETWORK (JON)

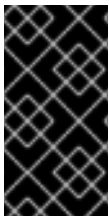
25.1. ABOUT JBOSS OPERATIONS NETWORK (JON)

The JBoss Operations Network (JON) is JBoss' administration and management platform used to develop, test, deploy and monitor the application life cycle. JBoss Operations Network is JBoss' enterprise management solution and is recommended for the management of multiple Red Hat JBoss Data Grid instances across servers. JBoss Operations Network's agent and auto discovery features facilitate monitoring the Cache Manager and Cache instances in JBoss Data Grid. JBoss Operations Network presents graphical views of key runtime parameters and statistics and allows administrators to set thresholds and be notified if usage exceeds or falls under the set thresholds.



IMPORTANT

In Red Hat JBoss Data Grid Remote Client-Server mode, statistics are enabled by default. While statistics are useful in assessing the status of JBoss Data Grid, they adversely affect performance and must be disabled if they are not required. In JBoss Data Grid Library mode, statistics are disabled by default and must be explicitly enabled when required.



IMPORTANT

To achieve full functionality of JBoss Operations Network library plugin for JBoss Data Grid's Library mode, upgrade to JBoss Operations Network 3.3.0 with patch *Update 04* or higher. For information on upgrading the JBoss Operations Network, see the *Upgrading JBoss ON* section in the JBoss Operations Network *Installation Guide*.



NOTE

JBoss Data Grid will support the JON plugin until its end of life in June 2019.

25.2. DOWNLOAD JBOSS OPERATIONS NETWORK (JON)

25.2.1. Prerequisites for Installing JBoss Operations Network (JON)

In order to install JBoss Operations Network in Red Hat JBoss Data Grid, the following is required:

- A Linux, Windows, or Mac OSX operating system, and an x86_64, i686, or ia64 processor.
- Java 6 or higher is required to run both the JBoss Operations Network Server and the JBoss Operations Network Agent.
- Synchronized clocks on JBoss Operations Network Servers and Agents.
- An external database must be installed.

25.2.2. Download JBoss Operations Network

Use the following procedure to download Red Hat JBoss Operations Network (JON) from the Customer Portal:

Download JBoss Operations Network

1. To access the Red Hat Customer Portal, navigate to <https://access.redhat.com/home> in a browser.
2. Click **Downloads**.
3. In the section labeled **JBoss Development and Management** , click **Red Hat JBoss Data Grid**.
4. Enter the relevant credentials in the **Red Hat Login** and **Password** fields and click **Log In**.
5. Select the appropriate version in the **Version** drop down menu list.
6. Click the **Download** button next to the desired download file.

25.2.3. Remote JMX Port Values

A port value must be provided to allow Red Hat JBoss Data Grid instances to be located. The value itself can be any available port.

Provide unique (and available) remote JMX ports to run multiple JBoss Data Grid instances on a single machine. A locally running JBoss Operations Network agent can discover each instance using the remote port values.

25.2.4. Download JBoss Operations Network (JON) Plugin

Download Installation Files

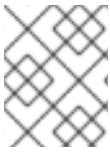
1. Open <http://access.redhat.com> in a web browser.
2. Click **Downloads** in the menu across the top of the page.
3. Click **Red Hat JBoss Operations Network** in the list under **JBoss Development and Management**.
4. Enter your login information.
You are taken to the Software Downloads page.
5. Download the JBoss Operations Network Plugin
If you intend to use the JBoss Operations Network plugin for JBoss Data Grid, select **JBoss ON for Data Grid** from either the **Product** drop-down box, or the menu on the left.

If you intend to use the JBoss Operations Network plugin for JBoss Enterprise Web Server, select **JBoss ON for Web Server** from either the **Product** drop-down box, or the menu on the left.

- a. Click the *Red Hat JBoss Operations Network VERSION Base Distribution* **Download** button.
- b. Repeat the steps to download the *Data Grid Management Plugin Pack for JBoss ON VERSION*

25.3. JBOSS OPERATIONS NETWORK SERVER INSTALLATION

The core of JBoss Operations Network is the server, which communicates with agents, maintains the inventory, manages resource settings, interacts with content providers, and provides a central management UI.



NOTE

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

25.4. JBOSS OPERATIONS NETWORK AGENT

The JBoss Operations Network Agent is a standalone Java application. Only one agent is required per machine, regardless of how many resources you require the agent to manage.

The JBoss Operations Network Agent does not ship fully configured. Once the agent has been installed and configured it can be run as a Windows service from a console, or run as a daemon or *init.d* script in a UNIX environment.

A JBoss Operations Network Agent must be installed on each of the machines being monitored in order to collect data.

The JBoss Operations Network Agent is typically installed on the same machine on which Red Hat JBoss Data Grid is running, however where there are multiple machines an agent must be installed on each machine.



NOTE

For more detailed information about configuring JBoss Operations Network agents, see the JBoss Operations Network *Installation Guide*.

25.5. JBOSS OPERATIONS NETWORK FOR REMOTE CLIENT-SERVER MODE

25.5.1. JBoss Operations Network for Remote Client-Server Mode

In Red Hat JBoss Data Grid's Remote Client-Server mode, the JBoss Operations Network plug-in is used to

- initiate and perform installation and configuration operations.
- monitor resources and their metrics.

In Remote Client-Server mode, the JBoss Operations Network plug-in uses JBoss Enterprise Application Platform's management protocol to obtain metrics and perform operations on the JBoss Data Grid server.

25.5.2. Installing the JBoss Operations Network Plug-in (Remote Client-Server Mode)

The following procedure details how to install the JBoss Operations Network plug-ins for Red Hat JBoss Data Grid's Remote Client-Server mode.

1. Install the plug-ins

- a. Copy the JBoss Data Grid server rhq plug-in to `$JON_SERVER_HOME/plugins`.
- b. Copy the JBoss Enterprise Application Platform plug-in to `$JON_SERVER_HOME/plugins`.

The server will automatically discover plug-ins here and deploy them. The plug-ins will be removed from the plug-ins directory after successful deployment.

2. Obtain plug-ins

Obtain all available plug-ins from the JBoss Operations Network server. To do this, type the following into the agent's console:

```
plugins update
```

3. List installed plug-ins

Ensure the JBoss Enterprise Application Platform plug-in and the JBoss Data Grid server rhq plug-in are installed correctly using the following:

```
plugins info
```

JBoss Operation Network can now discover running JBoss Data Grid servers.

25.6. JBOSS OPERATIONS NETWORK REMOTE-CLIENT SERVER PLUGIN

25.6.1. JBoss Operations Network Plugin Metrics

Table 25.1. JBoss Operations Network Traits for the Cache Container (Cache Manager)

Trait Name	Display Name	Description
cache-manager-status	Cache Container Status	The current runtime status of a cache container.
cluster-name	Cluster Name	The name of the cluster.
members	Cluster Members	The names of the members of the cluster.
coordinator-address	Coordinator Address	The coordinator node's address.
local-address	Local Address	The local node's address.
version	Version	The cache manager version.
defined-cache-names	Defined Cache Names	The caches that have been defined for this manager.

Table 25.2. JBoss Operations Network Metrics for the Cache Container (Cache Manager)

Metric Name	Display Name	Description
cluster-size	Cluster Size	How many members are in the cluster.
defined-cache-count	Defined Cache Count	How many caches that have been defined for this manager.
running-cache-count	Running Cache Count	How many caches are running under this manager.
created-cache-count	Created Cache Count	How many caches have actually been created under this manager.

Table 25.3. JBoss Operations Network Traits for the Cache

Trait Name	Display Name	Description
cache-status	Cache Status	The current runtime status of a cache.
cache-name	Cache Name	The current name of the cache.
version	Version	The cache version.

Table 25.4. JBoss Operations Network Metrics for the Cache

Metric Name	Display Name	Description
cache-status	Cache Status	The current runtime status of a cache.
number-of-locks-available	[LockManager] Number of locks available	The number of exclusive locks that are currently available.
concurrency-level	[LockManager] Concurrency level	The LockManager's configured concurrency level.
average-read-time	[Statistics] Average read time	Average number of milliseconds required for a read operation on the cache to complete.
hit-ratio	[Statistics] Hit ratio	The result (in percentage) when the number of hits (successful attempts) is divided by the total number of attempts.

Metric Name	Display Name	Description
elapsed-time	[Statistics] Seconds since cache started	The number of seconds since the cache started.
read-write-ratio	[Statistics] Read/write ratio	The read/write ratio (in percentage) for the cache.
average-write-time	[Statistics] Average write time	Average number of milliseconds a write operation on a cache requires to complete.
hits	[Statistics] Number of cache hits	Number of cache hits.
evictions	[Statistics] Number of cache evictions	Number of cache eviction operations.
remove-misses	[Statistics] Number of cache removal misses	Number of cache removals where the key was not found.
time-since-reset	[Statistics] Seconds since cache statistics were reset	Number of seconds since the last cache statistics reset.
number-of-entries	[Statistics] Number of current cache entries	Number of entries currently in the cache.
stores	[Statistics] Number of cache puts	Number of cache put operations
remove-hits	[Statistics] Number of cache removal hits	Number of cache removal operation hits.
misses	[Statistics] Number of cache misses	Number of cache misses.
success-ratio	[RpcManager] Successful replication ratio	Successful replications as a ratio of total replications in numeric double format.
replication-count	[RpcManager] Number of successful replications	Number of successful replications
replication-failures	[RpcManager] Number of failed replications	Number of failed replications
average-replication-time	[RpcManager] Average time spent in the transport layer	The average time (in milliseconds) spent in the transport layer.
commits	[Transactions] Commits	Number of transaction commits performed since the last reset.

Metric Name	Display Name	Description
prepares	[Transactions] Prepares	Number of transaction prepares performed since the last reset.
rollbacks	[Transactions] Rollbacks	Number of transaction rollbacks performed since the last reset.
invalidations	[Invalidation] Number of invalidations	Number of invalidations.
passivations	[Passivation] Number of cache passivations	Number of passivation events.
activations	[Activations] Number of cache entries activated	Number of activation events.
cache-loader-loads	[Activation] Number of cache store loads	Number of entries loaded from the cache store.
cache-loader-misses	[Activation] Number of cache store misses	Number of entries that did not exist in the cache store.
cache-loader-stores	[CacheStore] Number of cache store stores	Number of entries stored in the cache stores.

**NOTE**

Gathering of some of these statistics is disabled by default.

JBoss Operations Network Metrics for Connectors

The metrics provided by the JBoss Operations Network (JON) plugin for Red Hat JBoss Data Grid are for REST and Hot Rod endpoints only. For the REST protocol, the data must be taken from the Web subsystem metrics. For details about each of these endpoints, see the *Getting Started Guide*.

Table 25.5. JBoss Operations Network Metrics for the Connectors

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

**NOTE**

Gathering of these statistics is disabled by default.

25.6.2. JBoss Operations Network Plugin Operations

Table 25.6. JBoss ON Plugin Operations for the Cache

Operation Name	Description
Start Cache	Starts the cache.
Stop Cache	Stops the cache.
Clear Cache	Clears the cache contents.
Reset Statistics	Resets statistics gathered by the cache.
Reset Activation Statistics	Resets activation statistics gathered by the cache.
Reset Invalidation Statistics	Resets invalidations statistics gathered by the cache.
Reset Passivation Statistics	Resets passivation statistics gathered by the cache.
Reset Rpc Statistics	Resets replication statistics gathered by the cache.
Remove Cache	Removes the given cache from the cache-container.
Record Known Global Keyset	Records the global known keyset to a well-known key for retrieval by the upgrade process.
Synchronize Data	Synchronizes data from the old cluster to this using the specified migrator.
Disconnect Source	Disconnects the target cluster from the source cluster according to the specified migrator.

JBoss Operations Network Plugin Operations for the Cache Backups

The cache backups used for these operations are configured using cross-datacenter replication. In the JBoss Operations Network (JON) User Interface, each cache backup is the child of a cache. For more information about cross-datacenter replication, see [Set Up Cross-Datacenter Replication](#).

Table 25.7. JBoss Operations Network Plugin Operations for the Cache Backups

Operation Name	Description
status	Display the site status.
bring-site-online	Brings the site online.
take-site-offline	Takes the site offline.

Cache (Transactions)

Red Hat JBoss Data Grid does not support using Transactions in Remote Client-Server mode. As a result, none of the endpoints can use transactions.

25.6.3. JBoss Operations Network Plugin Attributes

Table 25.8. JBoss ON Plugin Attributes for the Cache (Transport)

Attribute Name	Type	Description
cluster	string	The name of the group communication cluster.
executor	string	The executor used for the transport.
lock-timeout	long	The timeout period for locks on the transport. The default value is 240000 .
machine	string	A machine identifier for the transport.
rack	string	A rack identifier for the transport.
site	string	A site identifier for the transport.
stack	string	The JGroups stack used for the transport.

25.6.4. Create a New Cache Using JBoss Operations Network (JON)

Use the following steps to create a new cache using JBoss Operations Network (JON) for Remote Client-Server mode.

Creating a new cache in Remote Client-Server mode

1. Log into the JBoss Operations Network Console.
 - a. From the JBoss Operations Network console, click **Inventory**.
 - b. Select **Servers** from the **Resources** list on the left of the console.
2. Select the specific Red Hat JBoss Data Grid server from the servers list.
 - a. Below the server name, click **infinispan** and then **Cache Containers**.
3. Select the desired cache container that will be parent for the newly created cache.
 - a. Right-click the selected cache container. For example, **clustered**.
 - b. In the context menu, navigate to **Create Child** and select **Cache**.

4. Create a new cache in the resource create wizard.
 - a. Enter the new cache name and click **Next**.
 - b. Set the cache attributes in the Deployment Options and click **Finish**.

**NOTE**

Refresh the view of caches in order to see newly added resource. It may take several minutes for the Resource to show up in the Inventory.

25.7. JBOSS OPERATIONS NETWORK FOR LIBRARY MODE

25.7.1. JBoss Operations Network for Library Mode

In Red Hat JBoss Data Grid's Library mode, the JBoss Operations Network plug-in is used to

- initiate and perform installation and configuration operations.
- monitor resources and their metrics.

In Library mode, the JBoss Operations Network plug-in uses **JMX** to obtain metrics and perform operations on an application using the JBoss Data Grid library.

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

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

Install JBoss Operations Network Library Mode Plug-in

1. Open the JBoss Operations Network Console
 - a. From the JBoss Operations Network console, select **Administration**.
 - b. Select **Agent Plugins** from the **Configuration** options on the left side of the console.

Figure 25.1. JBoss Operations Network Console for JBoss Data Grid

Name ^	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 1:35:12 PM	✓	✓
Ant Bundle Plugin		Sep 20, 2012 1:35:12 PM	✓	✓
Apache HTTP Server	Management of Apache web servers	Sep 20, 2012 1:35:12 PM	✓	✓
Generic JMX	Supports management of JMX MBean Servers via various remoting systems.	Sep 20, 2012 1:35:12 PM	✓	✓
IIS	Monitoring of Microsoft IIS Services	Sep 20, 2012 1:35:12 PM	✓	✓
Operating System Platforms	Management and monitoring of operating system level functions	Sep 20, 2012 1:35:12 PM	✓	✓
PostgreSQL Database	Management and monitoring of PostgreSQL versions 8.2 and higher	Sep 20, 2012 1:35:12 PM	✓	✓
RHQ Agent	Management and monitoring of the RHQ Agent	Sep 20, 2012 1:35:12 PM	✓	✓
Script	Manages resources that have command line executables or scripts as their management interfaces	Sep 20, 2012 1:35:12 PM	✓	✓

Scan For Updates Hide Deleted Upload Plugin : Browse... Upload ?

Enable Disable Delete Purge Refresh Total Rows: 10 (selected: 0)

2. Upload the Library Mode Plug-in

- a. Click **Browse**, locate the *InfinispanPlugin* on your local file system.
- b. Click **Upload** to add the plug-in to the JBoss Operations Network Server.

Figure 25.2. Upload the *InfinispanPlugin*.

The screenshot shows the JBoss Administration console interface. At the top, there is a navigation bar with tabs for Dashboard, Inventory, Reports, Bundles, Administration (selected), and Help. A user profile 'rhqadmin | Logout' is visible in the top right. A green notification bar at the top of the main content area displays the message 'File successfully uploaded'. Below this, a table lists various plugins. The table has columns for Name, Description, Last Updated, Enabled?, and Deployed?. The 'Agent Plugins' section in the left sidebar is highlighted. At the bottom of the interface, there are several buttons: 'Scan For Updates', 'Hide Deleted', 'Upload Plugin' (with a text input field and a 'Browse...' button), 'Upload' (with a green checkmark), 'Enable', 'Disable', 'Delete', 'Purge', and 'Refresh'. The status 'Total Rows: 10 (selected: 0)' is shown at the bottom right.

Name	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 1:35:12 PM	✓	✓
Ant Bundle Plugin		Sep 20, 2012 1:35:12 PM	✓	✓
Apache HTTP Server	Management of Apache web servers	Sep 20, 2012 1:35:12 PM	✓	✓
Generic JMX	Supports management of JMX MBean Servers via various remoting systems.	Sep 20, 2012 1:35:12 PM	✓	✓
IIS	Monitoring of Microsoft IIS Services	Sep 20, 2012 1:35:12 PM	✓	✓
Operating System Platforms	Management and monitoring of operating system level functions	Sep 20, 2012 1:35:12 PM	✓	✓
PostgreSQL Database	Management and monitoring of PostgreSQL versions 8.2 and higher	Sep 20, 2012 1:35:12 PM	✓	✓
RHQ Agent	Management and monitoring of the RHQ Agent	Sep 20, 2012 1:35:12 PM	✓	✓
Script	Manages resources that have command line executables or scripts as their management interfaces	Sep 20, 2012 1:35:12 PM	✓	✓

3. Scan for Updates

- a. Once the file has successfully uploaded, click **Scan For Updates** at the bottom of the screen.
- b. The *InfinispanPlugin* will now appear in the list of installed plug-ins.

Figure 25.3. Scan for Updated Plug-ins.

Name ^	Description	Last Updated	Enabled?	Deployed?
Abstract Augeas Plugin	An abstract plugin supporting concrete plugins that are based on Augeas, the open-source configuration library for Linux	Sep 20, 2012 1:35:12 PM	✓	✓
Abstract Database	Abstract plugin supporting concrete database plugins	Sep 20, 2012 1:35:12 PM	✓	✓
Ant Bundle Plugin		Sep 20, 2012 1:35:12 PM	✓	✓
Apache HTTP Server	Management of Apache web servers	Sep 20, 2012 1:35:12 PM	✓	✓
Generic JMX	Supports management of JMX MBean Servers via various remoting systems.	Sep 20, 2012 1:35:12 PM	✓	✓
IIS	Monitoring of Microsoft IIS Services	Sep 20, 2012 1:35:12 PM	✓	✓
InfinispanPlugin	Supports management and monitoring of Infinispan	Nov 12, 2012 11:39:25 AM	✓	✓
Operating System Platforms	Management and monitoring of operating system level functions	Sep 20, 2012 1:35:12 PM	✓	✓
PostgreSQL Database	Management and monitoring of PostgreSQL versions 8.2 and higher	Sep 20, 2012 1:35:12 PM	✓	✓
RHQ Agent	Management and monitoring of the RHQ Agent	Sep 20, 2012 1:35:12 PM	✓	✓
Script	Manages resources that have command line executables or scripts as their management interfaces	Sep 20, 2012 1:35:12 PM	✓	✓

25.7.3. Monitoring of JBoss Data Grid Instances in Library Mode

25.7.3.1. Prerequisites

The following is a list of common prerequisites for [Monitor an Application Deployed in Standalone Mode](#), [Monitor an Application Deployed in Domain Mode](#), and [Manually Adding JBoss Data Grid Instances in Library Mode](#).

- A correctly configured instance of JBoss Operations Network (JON) 3.2.0 with patch *Update 02* or higher version.
- A running instance of JON Agent on the server where the application will run. For more information, see [JBoss Operations Network Agent](#).
- An operational instance of the RHQ agent with a full JDK. Ensure that the agent has access to the *tools.jar* file from the JDK in particular. In the JON agent's environment file (*bin/rhq-env.sh*), set the value of the **RHQ_AGENT_JAVA_HOME** property to point to a full JDK home.
- The RHQ agent must have been initiated using the same user as the JBoss Enterprise Application Platform instance. As an example, running the JON agent as a user with root privileges and the JBoss Enterprise Application Platform process under a different user does not work as expected and must be avoided.
- An installed JON plugin for JBoss Data Grid Library Mode. For more information, see [Installing the JBoss Operations Network Plug-in \(Library Mode\)](#)
- *Generic JMX plugin* from JBoss Operation Networks 3.2.0 with patch *Update 02* or better version in use.

- A custom application using Red Hat JBoss Data Grid's Library mode with enabled JMX statistics for library mode caches in order to make statistics and monitoring working. For details how to enable JMX statistics for cache instances, see [Enable JMX for Cache Instances](#) and to enable JMX for cache managers see [Enable JMX for CacheManagers](#).
- The Java Virtual Machine (JVM) must be configured to expose the JMX MBean Server. For the Oracle/Sun JDK, see <http://docs.oracle.com/javase/1.5.0/docs/guide/management/agent.html>
- A correctly added and configured management user for JBoss Enterprise Application Platform.

25.7.3.2. Manually Adding JBoss Data Grid Instances in Library Mode

To add Red Hat JBoss Data Grid instances to JBoss Operations Network manually, use the following procedure in the JBoss Operations Network interface.

Add JBoss Data Grid Instances in Library Mode

1. Import the Platform
 - a. Navigate to the **Inventory** and select **Discovery Queue** from the **Resources** list on the left of the console.
 - b. Select the platform on which the application is running and click **Import** at the bottom of the screen.

Figure 25.4. Import the Platform from the menu:Discovery Queue[.].

The screenshot shows the JBoss Operations Network interface. The top navigation bar includes 'Dashboard', 'Inventory', 'Reports', 'Bundles', 'Administration', and 'Help'. The user is logged in as 'rhqadmin'. The left sidebar shows a tree view under 'Resources' with 'Discovery Queue' selected. The main content area displays a table titled 'Discovery Queue' with the following data:

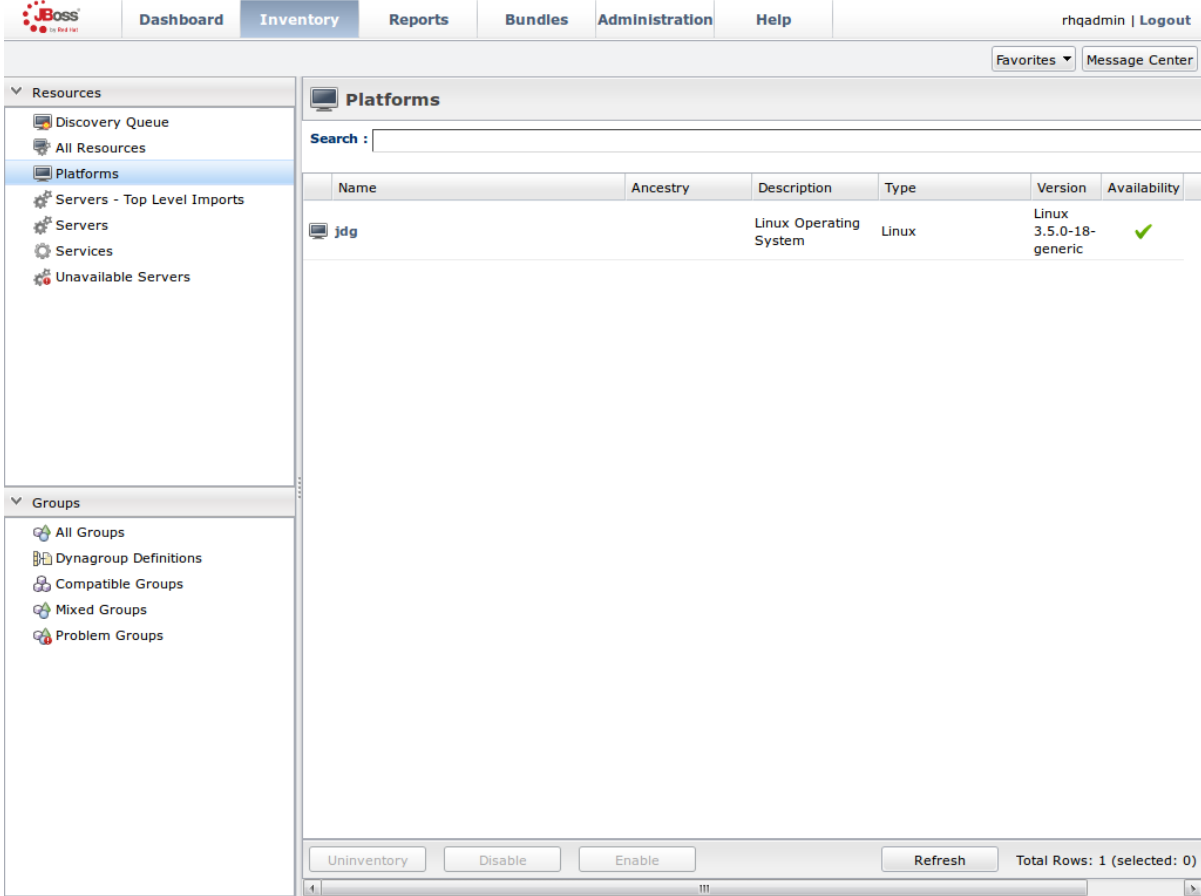
Resource Name	Resource Key	Resource Type	Description	Inventory Status	Discovery Time
<input checked="" type="checkbox"/> jdg	jdg	Linux	Linux Operating System	New	Nov 12, 2012 11:40:55 AM

At the bottom of the interface, there are buttons for 'Import', 'Ignore', and 'Unignore'. A 'Show' dropdown menu is set to 'New', and there are 'Select All' and 'Deselect All' buttons.

2. Access the Servers on the Platform
 - a. The **jdg** Platform now appears in the **Platforms** list.

- b. Click on the Platform to access the servers that are running on it.

Figure 25.5. Open the jdg Platform to view the list of servers.



The screenshot shows the JBoss Operations Network (JON) interface. The top navigation bar includes 'Dashboard', 'Inventory' (selected), 'Reports', 'Bundles', 'Administration', and 'Help'. The user is logged in as 'rhqadmin' and can click 'Logout'. On the left, there are two main sections: 'Resources' and 'Groups'. Under 'Resources', 'Platforms' is selected, showing a list of resources including 'Discovery Queue', 'All Resources', 'Platforms', 'Servers - Top Level Imports', 'Servers', 'Services', and 'Unavailable Servers'. Under 'Groups', there are 'All Groups', 'Dynagroup Definitions', 'Compatible Groups', 'Mixed Groups', and 'Problem Groups'. The main content area is titled 'Platforms' and contains a search bar and a table with the following data:

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

At the bottom of the interface, there are buttons for 'Uninventory', 'Disable', 'Enable', and 'Refresh'. The status bar indicates 'Total Rows: 1 (selected: 0)'.

3. Import the JMX Server

- a. From the **Inventory** tab, select **Child Resources**.
- b. Click the **Import button** at the bottom of the screen and select the **JMX Server**** option from the list.

Figure 25.6. Import the JMX Server

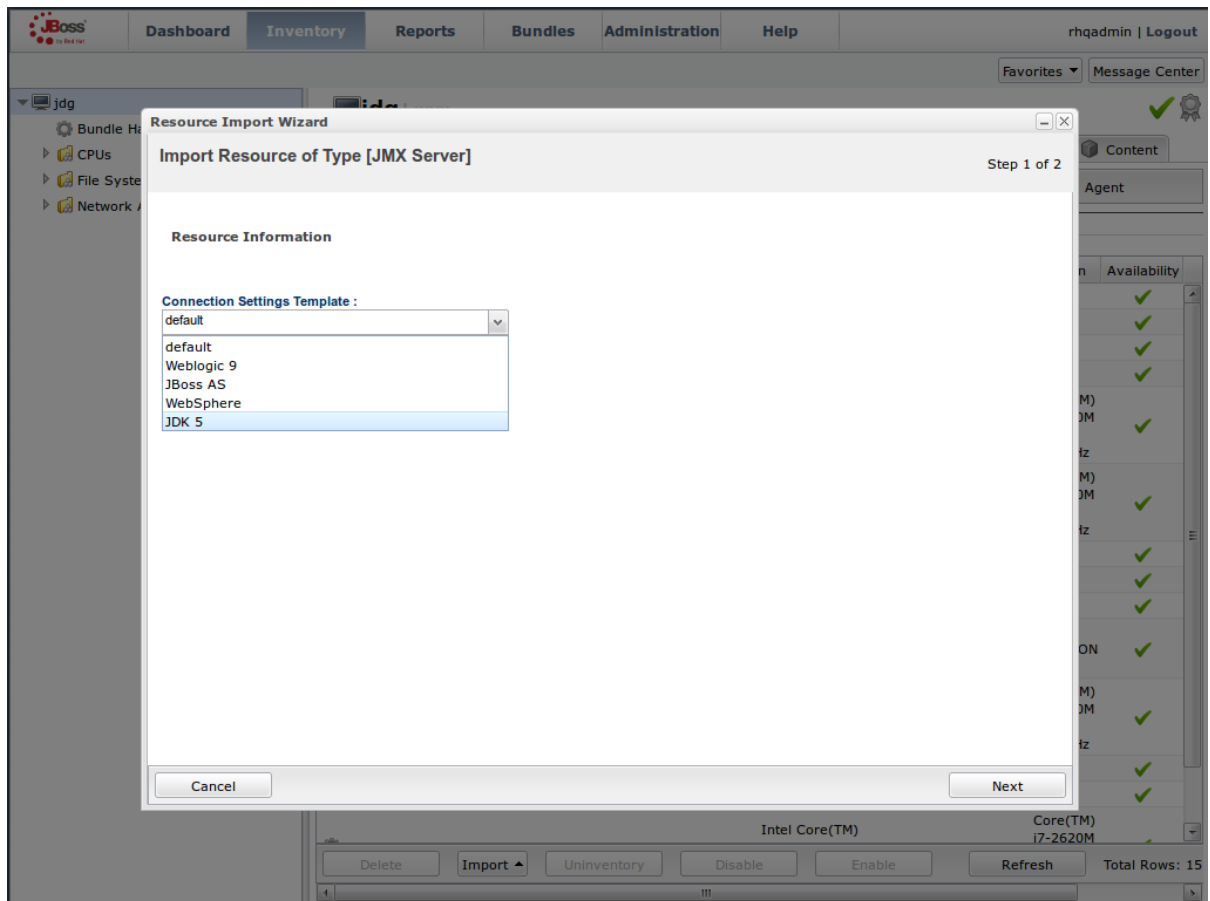
The screenshot shows the JBoss Data Grid Administration console. The 'Inventory' tab is active for the host 'jdg'. A context menu is open over the 'CPU 0' resource, with 'JMX Server' selected. The table below shows various system resources.

Name	Ancestry	Description	Type	Version	Availability
/run/user	jdg	none: /run/user	File System		✓
eth0	jdg	F0:DE:F1:7B:E9::	Network Adapter		✓
/run/lock	jdg	none: /run/lock	File System		✓
/run/shm	jdg	none: /run/shm	File System		✓
CPU 2	jdg	Intel Core(TM) i7-2620M CPU @ 2.70GHz	CPU	Core(TM) i7-2620M CPU @ 2.70GHz	✓
CPU 1	jdg	Intel Core(TM) i7-2620M CPU @ 2.70GHz	CPU	Core(TM) i7-2620M CPU @ 2.70GHz	✓
/run	jdg	tmpfs: /run	File System		✓
lo	jdg	00:00:00:00:00:00: Network Adapter	Network Adapter		✓
tun0	jdg	00:00:00:00:00:00: Network Adapter	Network Adapter		✓
Bundle Handler - Ant	jdg	For provisioning bundles with Ant script recipes	Ant Bundle Handler	4.4.0.JON	✓
CPU 0	jdg	Intel Core(TM) i7-2620M CPU @ 2.70GHz	CPU	Core(TM) i7-2620M CPU @ 2.70GHz	✓
/	jdg	/dev/sdb1: /	File System		✓
/data	jdg	/dev/sda1: /data	File System		✓
	jdg	Intel Core(TM) i7-2620M	CPU	Core(TM) i7-2620M	✓

4. Enable JDK Connection Settings

- a. In the **Resource Import Wizard** window, specify **JDK 8** from the list of **Connection Settings Template** options.

Figure 25.7. Select the JDK 5 Template.



5. Modify the Connector Address

- a. In the **Deployment Options** menu, modify the supplied **Connector Address** with the hostname and JMX port of the process containing the Infinispan Library.
- b. Enter the JMX connector address of the new JBoss Data Grid instance you want to monitor. For example:
Connector Address:

```
service:jmx:rmi:///127.0.0.1/jndi/rmi:///127.0.0.1:7997/jmxrmi
```



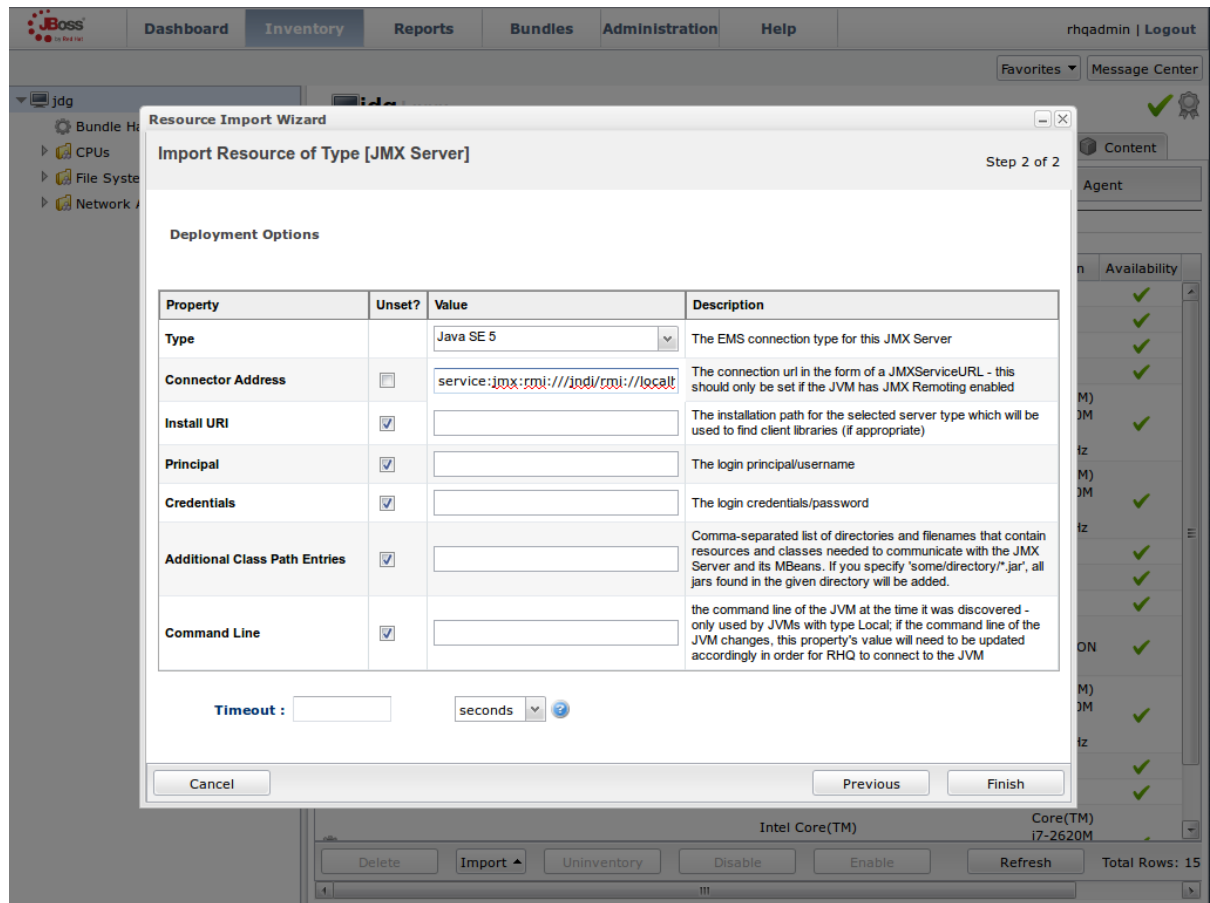
NOTE

The connector address varies depending on the host and the JMX port assigned to the new instance. In this case, instances require the following system properties at start up:

```
-Dcom.sun.management.jmxremote.port=7997 -  
Dcom.sun.management.jmxremote.ssl=false -  
Dcom.sun.management.jmxremote.authenticate=false
```

- c. Specify the **Principal** and **Credentials** information if required.
- d. Click **Finish**.

Figure 25.8. Modify the values in the Deployment Options screen.



6. View Cache Statistics and Operations

- a. Click **Refresh** to refresh the list of servers.
- b. The **JMX Servers** tree in the panel on the left side of the screen contains the **Infinispan Cache Managers** node, which contains the available cache managers. The available cache managers contain the available caches.
- c. Select a cache from the available caches to view metrics.
- d. Select the **Monitoring** tab.
- e. The **Tables** view shows statistics and metrics.
- f. The **Operations** tab provides access to the various operations that can be performed on the services.

Figure 25.9. Metrics and operational data relayed through JMX is now available in the JBoss Operations Network console.

The screenshot shows the JBoss Operations Network console interface. The main content area displays the monitoring page for the 'Default Cache(local)' component. The page has a navigation menu on the left with categories like 'Bundle Handler - Ant', 'CPUs', 'File Systems', 'JMX Servers', 'Logging', 'Memory Subsystem', 'Operating system informa', 'Threading', and 'Network Adapters'. The top navigation bar includes 'Dashboard', 'Inventory', 'Reports', 'Bundles', 'Administration', and 'Help'. The user profile 'rhqadmin | Logout' is visible in the top right corner. The main content area has a title 'Default Cache(local) INFINISPAN CACHE' and a sub-navigation bar with 'Summary', 'Inventory', 'Alerts', 'Monitoring', and 'Operations'. Below this is a table with columns 'Name', 'Alerts', 'Minimum', 'Maximum', 'Average', and 'Last'. The table lists various metrics for the cache, such as 'Average time spent in the transport layer', 'Number of failed replications', 'Hit ratio', and 'Number of cache hits'. The 'Alerts' column for all entries is 0. The 'Last' column shows values like 0, 5, 1, 2, 0, 2, 3, and 7376. At the bottom of the table, there is a 'Time Range - Previous' dropdown set to '4 minutes' and a 'Refresh' button. The total rows are 20, with 0 selected.

Name	Alerts	Minimum	Maximum	Average	Last
[RpcManager] Average time spent in the transport layer	0	NaN	NaN	NaN	NaN
[RpcManager] Number of failed replications	0	NaN	NaN	NaN	NaN
[RpcManager] Number of successful replications	0	NaN	NaN	NaN	NaN
[RpcManager] Successful replication ratio	0	NaN	NaN	NaN	NaN
[Statistics] Average read time	0	NaN	NaN	NaN	0
[Statistics] Average write time	0	NaN	NaN	NaN	0
[Statistics] Hit ratio	0	NaN	NaN	NaN	0.8333333333333333
[Statistics] Number of cache evictions	0	NaN	NaN	NaN	0
[Statistics] Number of cache hits	0	NaN	NaN	NaN	5
[Statistics] Number of cache misses	0	NaN	NaN	NaN	1
[Statistics] Number of cache puts	0	NaN	NaN	NaN	2
[Statistics] Number of cache removal hits	0	NaN	NaN	NaN	0
[Statistics] Number of cache removal misses	0	NaN	NaN	NaN	0
[Statistics] Number of current cache entries	0	NaN	NaN	NaN	2
[Statistics] Read/write ratio	0	NaN	NaN	NaN	3
[Statistics] Seconds since cache started	0	NaN	NaN	NaN	7376
[Statistics] Seconds since cache	0	NaN	NaN	NaN	7376

25.7.3.3. Monitor Custom Applications Using Library Mode Deployed On JBoss Enterprise Application Platform

25.7.3.3.1. Monitor an Application Deployed in Standalone Mode

Use the following instructions to monitor an application deployed in JBoss Enterprise Application Platform using its standalone mode:

Monitor an Application Deployed in Standalone Mode

1. Start the JBoss Enterprise Application Platform Instance
Start the JBoss Enterprise Application Platform instance as follows:
 - a. Enter the following command at the command line or change standalone configuration file (*/bin/standalone.conf*) respectively:

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

- b. Start the JBoss Enterprise Application Platform instance in standalone mode as follows:

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

2. Deploy the Red Hat JBoss Data Grid Application
Deploy the **WAR** file that contains the JBoss Data Grid Library mode application with **globalJmxStatistics** and **jmxStatistics** enabled.

3. Run JBoss Operations Network (JON) Discovery
Run the **discovery --full** command in the JBoss Operations Network (JON) agent.
4. Locate Application Server Process
In the JBoss Operations Network (JON) web interface, the JBoss Enterprise Application Platform process is listed as a JMX server.
5. Import the Process Into Inventory
Import the process into the JBoss Operations Network (JON) inventory.
6. Optional: Run Discovery Again
If required, run the **discovery --full** command again to discover the new resources.

25.7.3.3.2. Monitor an Application Deployed in Domain Mode

Use the following instructions to monitor an application deployed in JBoss Enterprise Application Platform 6 using its domain mode:

Monitor an Application Deployed in Domain Mode

1. Edit the Host Configuration
Edit the *domain/configuration/host.xml* file to replace the **server** element with the following configuration:

```
<servers>
  <server name="server-one" group="main-server-group">
    <jvm name="default">
      <jvm-options>
        <option value="-Dorg.rhq.resourceKey=EAP1"/>
      </jvm-options>
    </jvm>
  </server>
  <server name="server-two" group="main-server-group" auto-
start="true">
    <socket-bindings port-offset="150"/>
    <jvm name="default">
      <jvm-options>
        <option value="-Dorg.rhq.resourceKey=EAP2"/>
      </jvm-options>
    </jvm>
  </server>
</servers>
```

2. Start JBoss Enterprise Application Platform 6
Start JBoss Enterprise Application Platform 6 in domain mode:

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

3. Deploy the Red Hat JBoss Data Grid Application
Deploy the **WAR** file that contains the JBoss Data Grid Library mode application with **globalJmxStatistics** and **jmxStatistics** enabled.
4. Run Discovery in JBoss Operations Network (JON)

If required, run the **discovery --full** command for the JBoss Operations Network (JON) agent to discover the new resources.

25.8. JBOSS OPERATIONS NETWORK PLUG-IN QUICKSTART

For testing or demonstrative purposes with a single JBoss Operations Network agent, upload the plug-in to the server then type "plugins update" at the agent command line to force a retrieval of the latest plugins from the server.

25.9. OTHER MANAGEMENT TOOLS AND OPERATIONS

25.9.1. Other Management Tools and Operations

Managing Red Hat JBoss Data Grid instances requires exposing significant amounts of relevant statistical information. This information allows administrators to get a clear view of each JBoss Data Grid node's state. A single installation can comprise of tens or hundreds of JBoss Data Grid nodes and it is important to provide this information in a clear and concise manner. JBoss Operations Network is one example of a tool that provides runtime visibility. Other tools, such as **JConsole** can be used where **JMX** is enabled.

25.9.2. Accessing Data via URLs

Caches that have been configured with a REST interface have access to Red Hat JBoss Data Grid using RESTful HTTP access.

The RESTful service only requires a HTTP client library, eliminating the need for tightly coupled client libraries and bindings. For more information about how to retrieve data using the REST interface, refer to the JBoss Data Grid *Developer Guide*.

HTTP **put()** and **post()** methods place data in the cache, and the **URL** used determines the cache name and key(s) used. The data is the value placed into the cache, and is placed in the body of the request.

A Content-Type header must be set for these methods. **GET** and **HEAD** methods are used for data retrieval while other headers control cache settings and behavior.



NOTE

It is not possible to have conflicting server modules interact with the data grid. Caches must be configured with a compatible interface in order to have access to JBoss Data Grid.

25.9.3. Limitations of Map Methods

Specific Map methods, such as **size()**, **values()**, **keySet()** and **entrySet()**, can be used with certain limitations with Red Hat JBoss Data Grid as they are unreliable. These methods do not acquire locks (global or local) and concurrent modification, additions and removals are excluded from consideration in these calls.

The listed methods have a significant impact on performance. As a result, it is recommended that these methods are used for informational and debugging purposes only.

Performance Concerns

In JBoss Data Grid 7.1 the map methods `size()`, `values()`, `keySet()`, and `entrySet()` include entries in the cache loader by default. The cache loader in use will determine the performance of these commands; for instance, when using a database these methods will run a complete scan of the table where data is stored, which may result in slower processing. To not load entries from the cache loader, and avoid any potential performance hit, use

`Cache.getAdvancedCache().withFlags(Flag.SKIP_CACHE_LOAD)` before executing the desired method.

Understanding the `size()` Method (Embedded Caches)

In JBoss Data Grid 7.1 the `Cache.size()` method provides a count of all elements in both this cache and cache loader across the entire cluster. When using a loader or remote entries, only a subset of entries is held in memory at any given time to prevent possible memory issues, and the loading of all entries may be slow.

In this mode of operation, the result returned by the `size()` method is affected by the flags `org.infinispan.context.Flag#CACHE_MODE_LOCAL`, to force it to return the number of entries present on the local node, and `org.infinispan.context.Flag#SKIP_CACHE_LOAD`, to ignore any passivated entries. Either of these flags may be used to increase performance of this method, at the cost of not returning a count of all elements across the entire cluster.

Understanding the `size()` Method (Remote Caches)

In JBoss Data Grid 7.1 the Hot Rod protocol contain a dedicated `SIZE` operation, and the clients use this operation to calculate the size of all entries.

PART XII. RED HAT JBOSS DATA GRID WEB ADMINISTRATION

CHAPTER 26. RED HAT JBOSS DATA GRID ADMINISTRATION CONSOLE

26.1. ABOUT JBOSS DATA GRID ADMINISTRATION CONSOLE

The Red Hat JBoss Data Grid Administration Console allows administrators to monitor caches and JBoss Data Grid clusters, while providing a web interface for making dynamic changes to caches, cache-containers, and cluster nodes.

26.2. RED HAT JBOSS DATA GRID ADMINISTRATION CONSOLE PREREQUISITES

The Red Hat JBoss Data Grid Administration Console is only available in Remote Client-Server Mode.

26.3. RED HAT JBOSS DATA GRID ADMINISTRATION CONSOLE GETTING STARTED

26.3.1. Red Hat JBoss Data Grid Administration Console Getting Started

The Administration Console is started automatically when JBoss Data Grid is running in Remote Client-Server Mode. A management user must be added to the server instance, which will then be used to access the web console.

26.3.2. Adding Management User

In order to use the JBoss Data Grid Administration Console, a new management user must be created. To add a new user, execute the `add-user.sh` utility script within the `bin` folder of your JBoss Data Grid Server installation and enter the requested information.

The following procedure outlines the steps to add a new management user:

Adding a Management User

1. Run the `add-user` script within the `bin` folder as follows:

```
./add-user .sh
```

2. Select the option for the type of user to be added. For management user, select option **a**.
3. Set the Username and password as per the listed recommendations.
4. Enter the name of the group or groups in which the user has to be added. Leave blank for no group.



NOTE

See the *Download and Install JBoss Data Grid* section in the *Red Hat JBoss Data Grid Getting Started Guide* for download and installation details.

5. Confirm if you need the user to be used for Application Server process connection.

**NOTE**

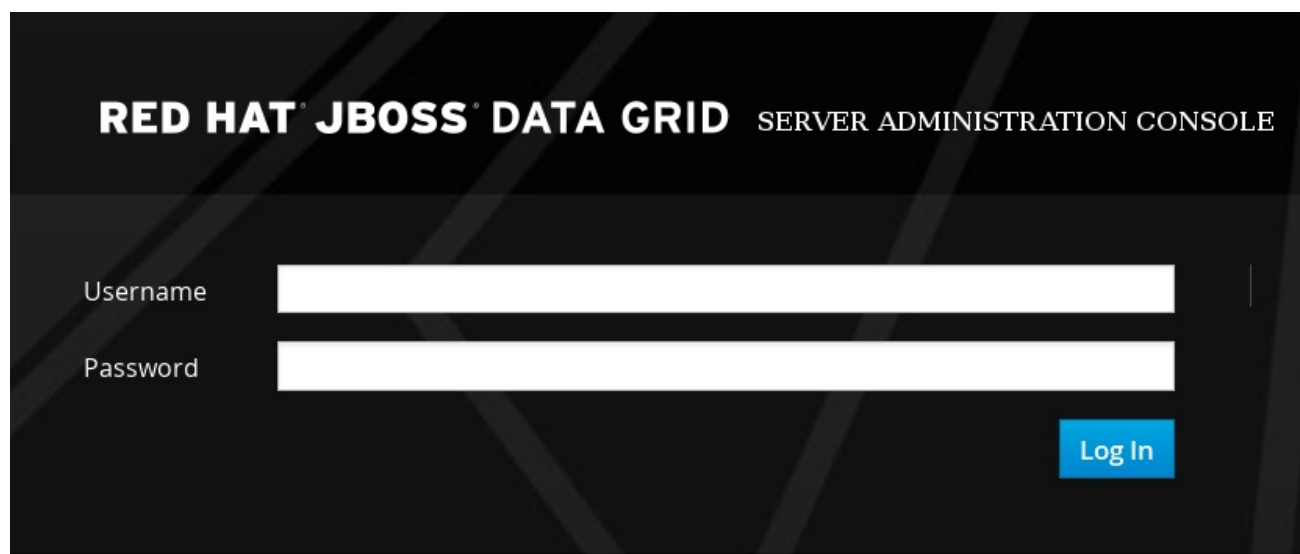
Before proceeding, make sure \$JBOSS_HOME is not set to a different installation. Otherwise, you may get unpredictable results.

26.3.3. Logging in the JBoss Data Grid Administration Console

Once the JBoss Data Grid server is running, in either domain or standalone mode, the JBoss Data Grid Administration Console may be accessed at the following login page:

```
http://${jboss.bind.address.management}:9990/console/index.html
```

Figure 26.1. JBoss Data Grid Administration Console Login Screen



Enter the user credentials to log in. After logging in, the cache container view is displayed.

26.4. DASHBOARD VIEW

26.4.1. Dashboard View

The Dashboard view is split into 3 tabs namely:

- Caches
- Clusters
- Status Events

**NOTE**

The **Clusters** and **Status Events** tabs are not available when running JBoss Data Grid in standalone non-clustered mode.

26.4.2. Cache Containers View

The first default view after logging in is the Cache Container list. A Cache Container is the primary mechanism for treating a cache instance and is used as a starting point for using a cache itself.

Cache centric view presents the list of configured caches. It is used for viewing and adding caches to clusters, adding and adjusting new cache configurations, adding and configuring endpoints and other cache related administrative tasks.

Figure 26.2. Cache Containers View

Name	Status	# Caches	Clustering info	Endpoints	Sites
clustered	AVAILABLE	2	cluster UDP	hotrod : 11222 memcached : 11211 rest : 8080	N/A

In this instance, there is one cache container with the name **clustered** with two caches deployed on the cluster group with UDP transport and three Endpoints attached to it. There are no remote sites configured for this cache container.

26.4.3. Clusters View

The Cluster tab presents the summary of the clusters along with the current status, number of hosts and number of nodes.

Figure 26.3. Clusters View

Name	Status	# Hosts	# Nodes
cluster	STARTED	1	2



NOTE

The Cluster view will not appear when the server is running in standalone non-clustered mode.

26.4.4. Status Events View

The JBoss Data Grid Administration Console displays the cluster wide events such as local rebalancing, cluster start and stop, cluster-split and cluster-merge events in a consolidated section. To view the detailed status events, navigate to the Status Events tab from the Dashboard.

Figure 26.4. Status Events View

Cache containers

Latest status events

Retrieve logs per cluster. Showing 55 events.

Type	Timestamp	Description
Info	2016-07-08 02:53:03 +0530	ISP100003: Finished local rebalance
Info	2016-07-08 02:53:03 +0530	ISP100003: Finished local rebalance
Info	2016-07-08 02:53:03 +0530	ISP100002: Started local rebalance
Info	2016-07-08 02:53:03 +0530	ISP100003: Finished local rebalance

The status events are displayed with the associated timestamp and the event description.



NOTE

The Status Events view will not appear when the server is running in standalone non-clustered mode.

26.5. CACHE ADMINISTRATION

26.5.1. Adding a New Cache

To add a new cache, follow these steps:

Adding a New Cache

1. In the Cache Containers view, click on the name of the cache container.

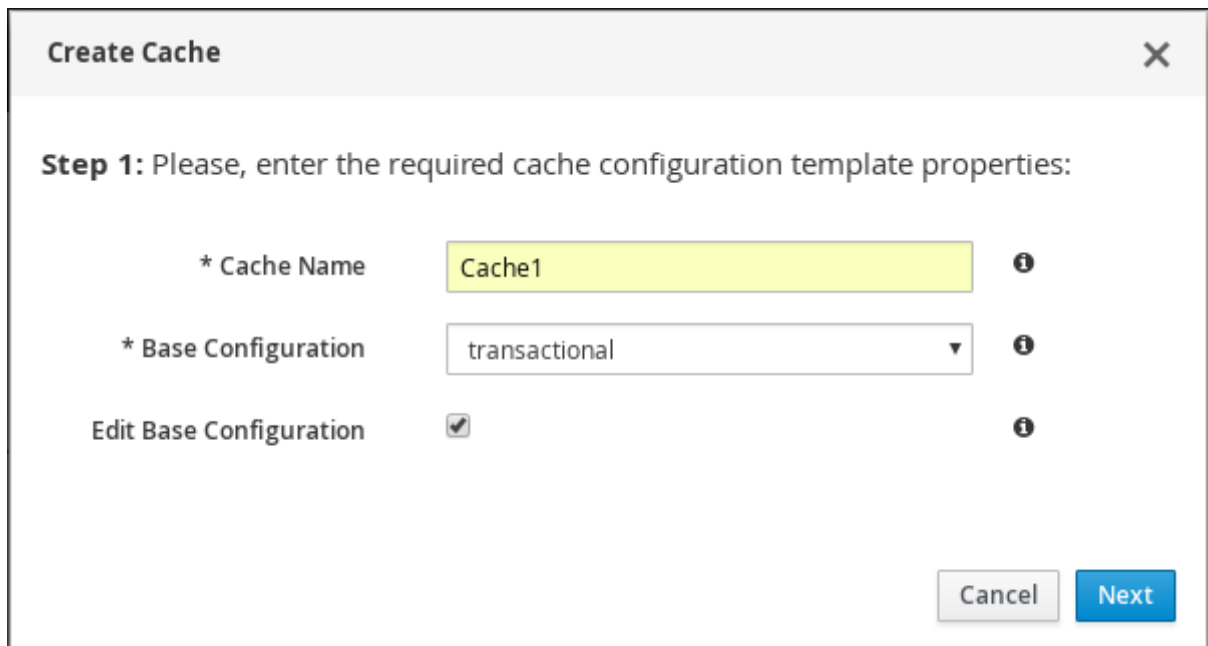
Figure 26.5. Cache Containers View

Cache containers

Name	Status	# Caches	Clustering info	Endpoints
clustered	AVAILABLE	2	cluster UDP	hotrod: 11222 memcached: 11211 rest: 8080

- The Caches view is displayed listing all the configured caches. Click **Add Cache** to add and configure a new cache. The new cache creation window is opened.

Figure 26.6. Add Cache



Create Cache [X]

Step 1: Please, enter the required cache configuration template properties:

* Cache Name: Cache1 ⓘ

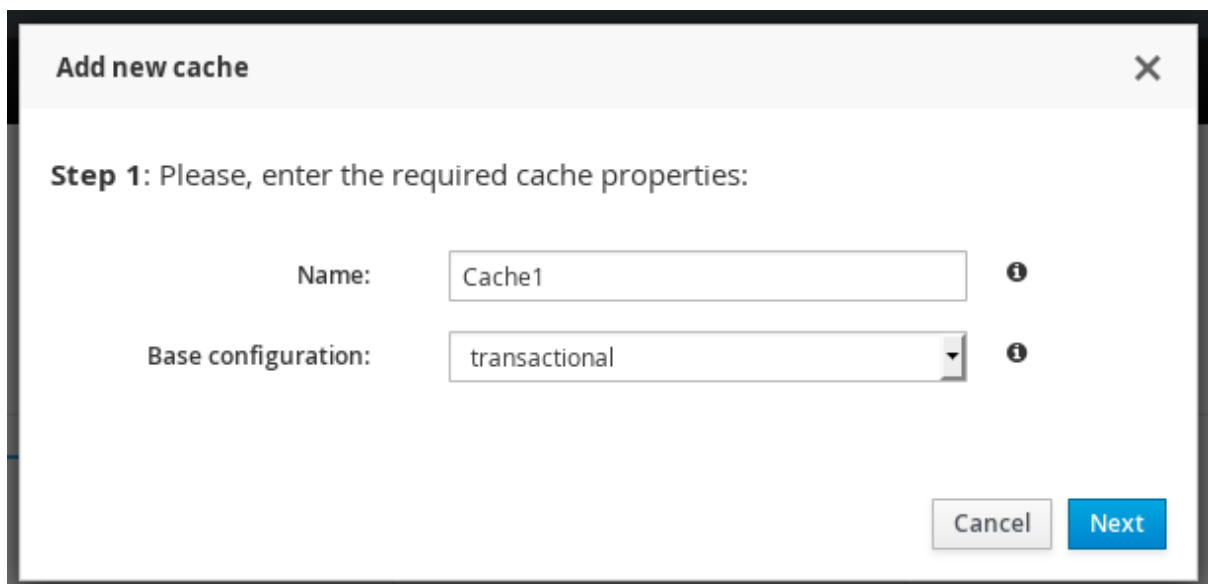
* Base Configuration: transactional ⓘ

Edit Base Configuration: ⓘ

Cancel Next

- Enter the new cache name, select the base configuration template from the drop-down menu, check the **Edit** button, and click **Next**. If the **Edit** button is not selected then the cache will be immediately created using the selected template.

Figure 26.7. Cache Properties



Add new cache [X]

Step 1: Please, enter the required cache properties:

Name: Cache1 ⓘ

Base configuration: transactional ⓘ

Cancel Next

- The cache configuration screen is displayed. Enter the cache parameters and click **Create**.

Figure 26.8. Cache Configuration

Main		
Type	distributed-cache	ⓘ
Template	transactional	ⓘ
Clustering		
Mode	SYNC	ⓘ
Owners		ⓘ
Segments		ⓘ
L1 Lifespan		ⓘ
Capacity Factor	34	ⓘ (Undo) Restart needed
Remote Timeout		ⓘ
Queue Size		ⓘ
Queue Flush Interval		ⓘ
Monitoring		
Statistics	<input type="checkbox"/>	ⓘ
Jndi Name	jndi/mon	ⓘ (Undo) Restart needed

5. A confirmation screen is displayed. Click **Create** to create the cache.

Figure 26.9. Cache Confirmation

Confirmation
✕

Create cache1 cache using transactional configuration template?

26.5.2. Editing Cache Configuration

The JBoss Data Grid Administration Console allows administrators to edit the configuration of an existing cache.

The following procedure outlines the steps to edit a cache configuration:

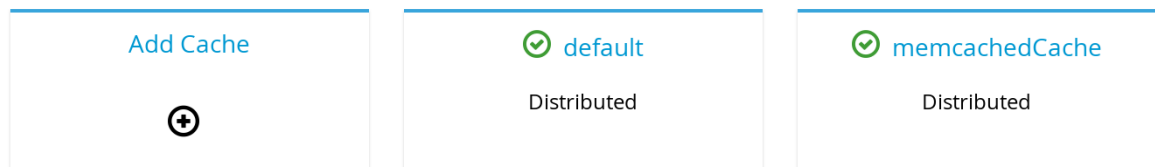
Editing Cache Configuration

1. Log into the JBoss Data Grid Administration Console and click on the cache container name.

Figure 26.10. Cache Containers

Caches Clusters Status events				
Cache containers				
Cache containers				
Cache containers				
Name	Status	# Caches	Clustering info	Endpoints
clustered	AVAILABLE	2	cluster UDP	hotrod : 11222 memcached : 11211 rest : 8080

- In the Caches view, click on the cache name.

Figure 26.11. Caches View

- The cache statistics and properties page is displayed. On the right hand side, click the **Configuration** tab.

Figure 26.12. Cache Configuration Button

- The edit cache configuration interface is opened. The editable cache properties are found in the cache properties menu at the left hand side.

Figure 26.13. Editing Cache Configuration Interface

memcachedCache

View / Edit cache configuration:

General	Main	
Locking	Type	distributed-cache ⓘ
Eviction	Template Name	memcachedCache ⓘ
Expiration	Clustering	
Indexing	Mode	SYNC ⓘ
Compatibility	Owners	2 ⓘ
Transactions	Segments	20 ⓘ
Security	L1 Lifespan	ⓘ
Partition handling	Remote Timeout	30000 ⓘ
State transfer		

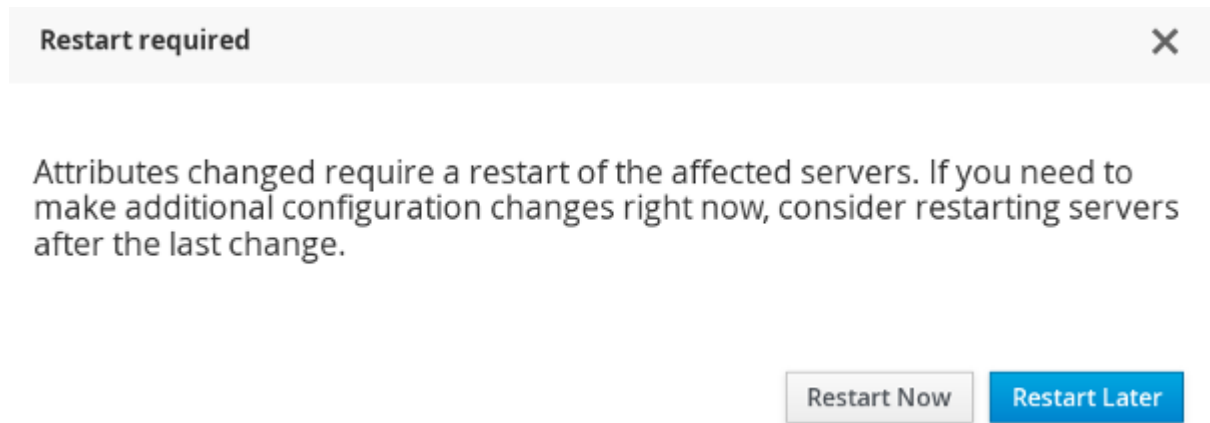
- Select the cache configuration property to be edited from the cache properties menu along the left-hand side. To get a description on the cache configuration parameters, hover the cursor over the information icon to the right of each field. The parameter description is presented in form of a tooltip.

Figure 26.14. Cache configuration paramaters

Type	distributed-cache	ⓘ
Template Name	memcachedCache	ⓘ

The cache configuration type The default value is .

- The **General** property is selected by default. Edit the required values in the given parameter input fields and click **Apply changes** below
- The restart dialogue box appears. Click **Restart Now** to apply the changes, or **Restart Later** to continue editing the cache properties.

Figure 26.15. Restart Dialogue Box**NOTE**

In standalone mode the dialog instead contains the following text:

Config changes will only be made available after you manually restart the server!

26.5.3. Cache Statistics and Properties View

The JBoss Data Grid Administration Console allows administrators to view all the cache statistics including the average time for reads, average times for writes, total number of entries, total number of reads, total number of failed reads and total number of writes.

To view the cache statistics, follow these steps:

Viewing Cache Statistics

1. Navigate to the list of caches by clicking on the name of the cache container in the Cache Container view.
2. Click on the name of the cache from the list of caches. Optionally you can use the cache filter on the left side to filter caches. The caches can be filtered by a keyword, substring or by selecting the type, the trait, and the status.

Figure 26.16. Caches View

- The next page displays the comprehensive cache statistics under the headings: **Cache content**, **Operations performance** and **Caching Activity**.

Figure 26.17. Cache Statistics

Operations performance		Caching activity	
Avg Reads	35 ms	# READ Hits	3.343
Avg Writes	55 ms	# READ misses	544
Avg Removes	65 ms	# REMOVE hits	4.454
		# REMOVE misses	4.454
		# PUTS	4.454

- Additional cache statistics are displayed under the headings: **Entries Lifecycle**, **Cache Loader** and **Locking**

Figure 26.18. Cache Statistics

Entries lifecycle		Cache loader		Locking	
# Activations	3.343	# Loads	3.343	# Locks available	3.343
# Evictions	544	# Misses	544	# Locks held	544
# Invalidation	4.454	# Stores	4.454		
# Passivations	4.454				

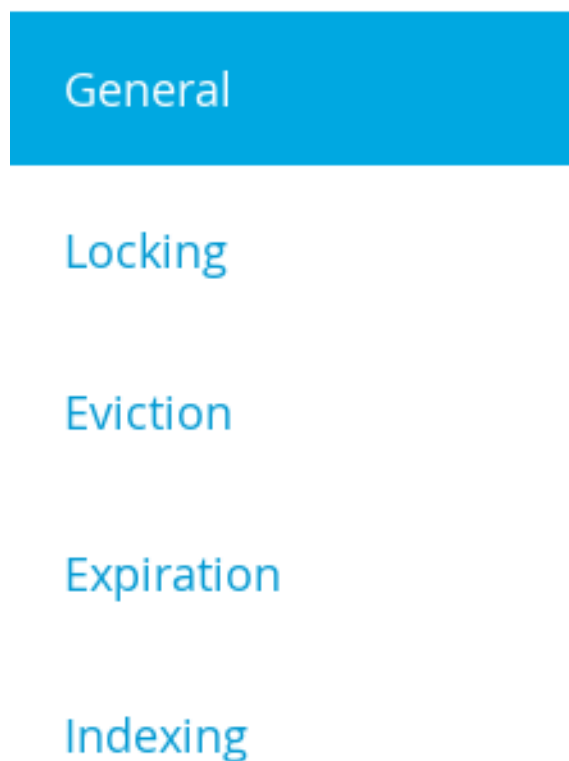
5. To view cache properties, click on **Configuration** at the right hand side.

Figure 26.19. Configuration Button



6. The cache properties menu is displayed at the left hand side.

Figure 26.20. Cache Properties Menu



To view on which node a cache resides, click on the **Nodes** tab next to the **General Status** tab on the cache statistics page.

Figure 26.21. General Status Tab

General status Nodes

Cache content

Entries	0
READ / WRITE ratio	0
HIT ratio	0

The name of the Node(s) is displayed along with the read-write statistics.

Figure 26.22. Cache Node Labels

Available memcachedCache Refresh Actions

General status Nodes

Name	Average reads	Average writes	Total entries	Total reads	Total failed reads	Total writes	Total failed writes
master/server-one	0	0	0	0	0	0	0
master/server-two	0	0	0	0	0	0	0

26.5.4. Enable and Disable Caches

The following procedure outlines the steps to disable a cache:

Disabling a Cache

1. Navigate to the caches view by clicking on the name of the cache container in the Cache Container view. Click on the name of the cache to be disabled.

Figure 26.23. Caches View

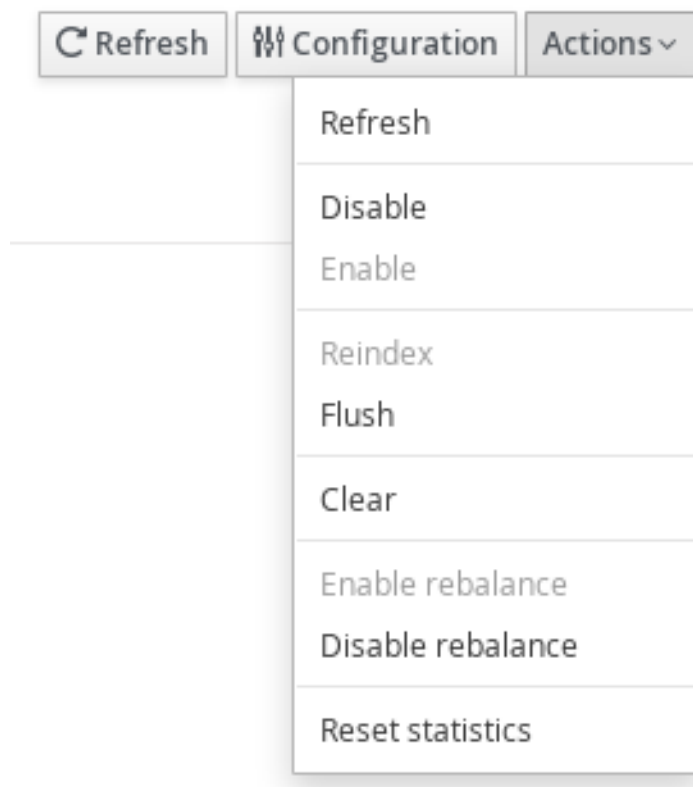
default Distributed

memcachedCache Distributed

MyCache Replicated

2. The cache statistics will be displayed. On the right hand side of the interface, click on the Actions tab and then click **Disable**.

Figure 26.24. Cache Disable



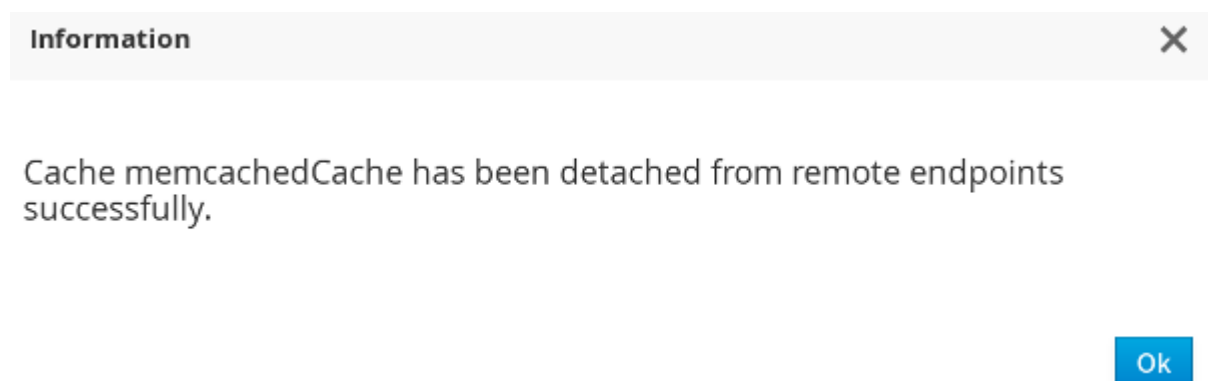
3. A confirmation dialogue box will appear. Click **Disable** to disable the cache.

Figure 26.25. Cache Disable Confirmation



4. A subsequent dialogue box appears. Click **Ok**.

Figure 26.26. Confirmation Box



- The selected cache is disabled successfully with a visual indicator **Disabled** next to the cache name label.

Figure 26.27. Disabled Cache

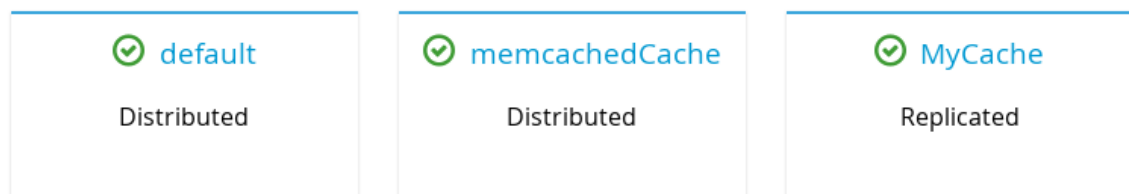


The following procedure outlines the steps to enable a cache:

Enabling a Cache

- To enable a cache, click on the specific disabled cache from the Cache view.

Figure 26.28. Caches View

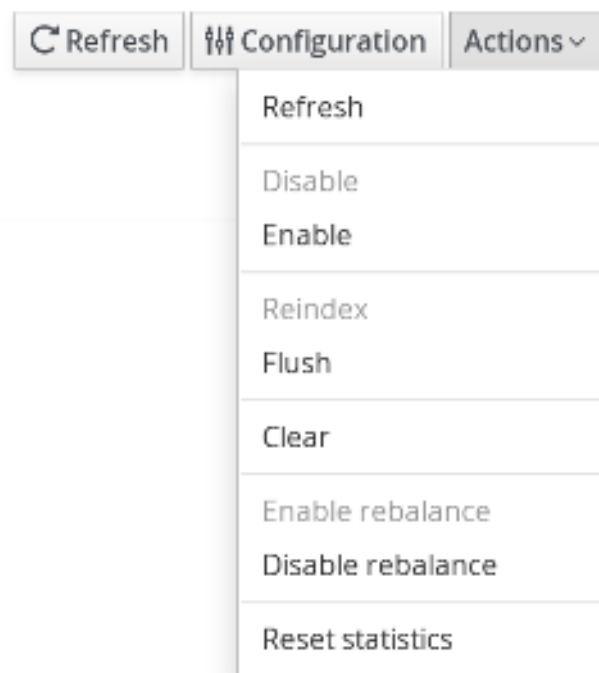


- On the right hand side of the interface, click on the **Actions** tab.

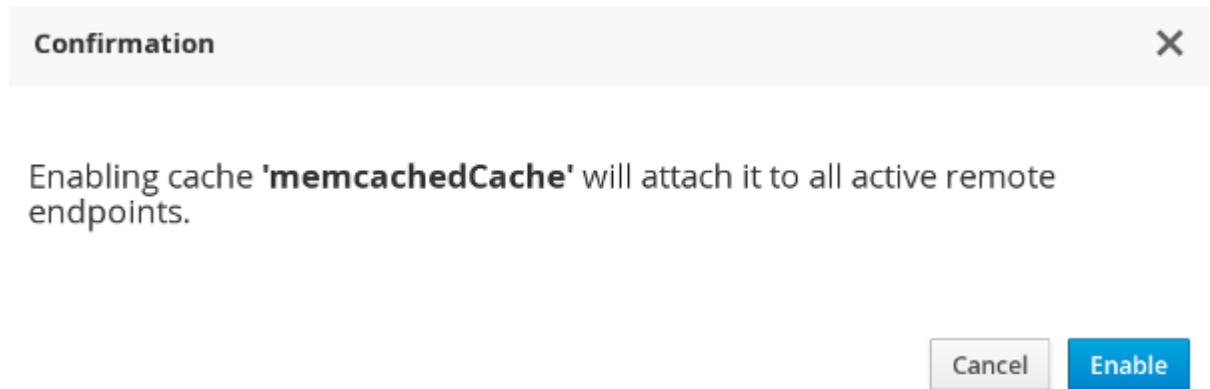


- From the Actions tab, click **Enable**

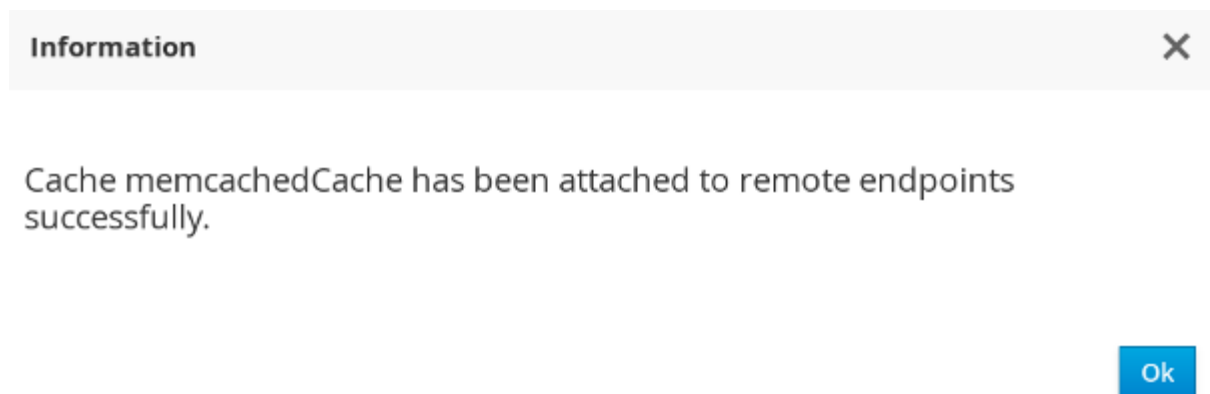
Figure 26.29. Actions Menu



- A confirmation dialogue box appears. Click **Enable**.

Figure 26.30. Confirmation Box

5. A subsequent dialogue box appears. Click **Ok**

Figure 26.31. Information Box

6. The selected cache is enabled successfully with a visual indicator Enabled next to the cache name label.

Figure 26.32. Cache Enabled

Available memcachedCache - Rebalancing completed - Enabled

26.5.5. Cache Flush and Clear

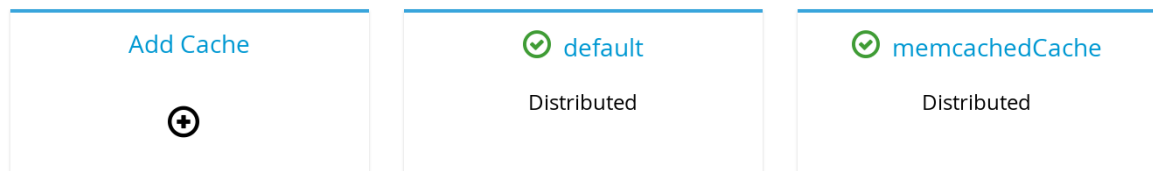
The JBoss Data Grid Administration Console allows administrators to remove all the entries from a cache and the cache stores through the cache Clear operation. The console also provides the Flush operation to store the entries from the cache memory to the cache store. These entries are not removed from the cache memory, as during a Clear operation.

Flushing a Cache

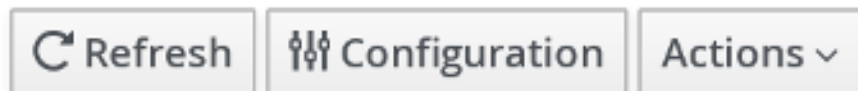
To flush a cache, follow these steps:

Flushing a Cache

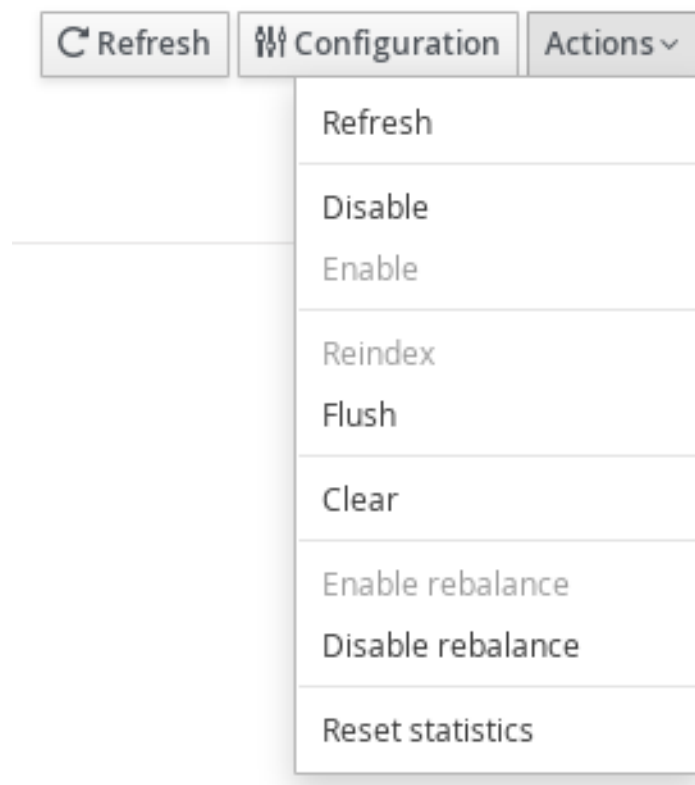
1. In the Cache Containers view, click on the name of the cache container.
2. The Caches view is displayed. Click on the cache to be flushed.

Figure 26.33. Caches View

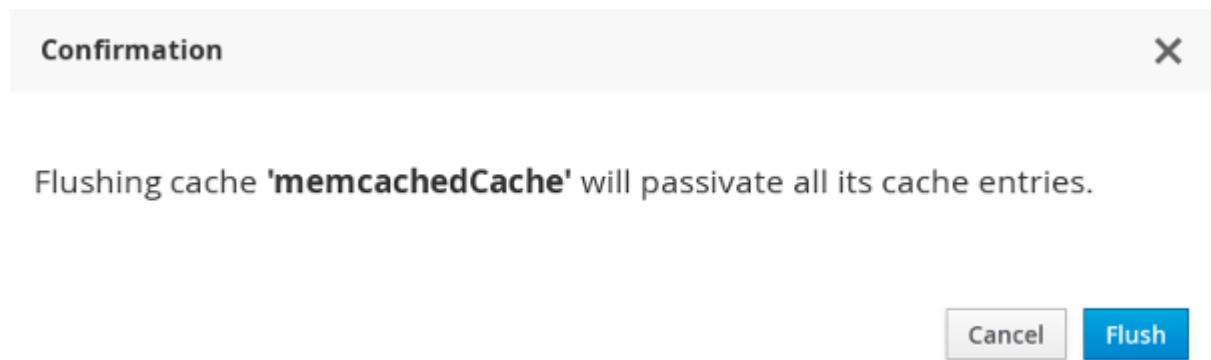
3. The cache statistics page is displayed. At the right hand side, click **Actions**.

Figure 26.34. Actions Button

4. From the **Actions** menu, click **Flush**.

Figure 26.35. Actions Menu

5. A confirmation dialogue box appears. Click **Flush**.

Figure 26.36. Cache Flush Confirmation Box

6. The cache is successfully flushed. Click **Ok**.

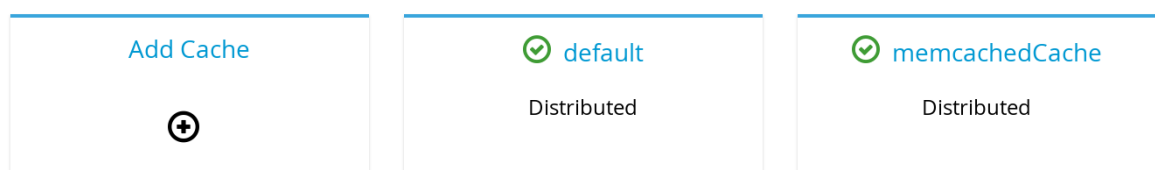
Figure 26.37. Cache Flush Information Box

Clearing a Cache

To clear a cache, follow these steps:

Clearing a Cache

1. In the Cache Containers view, click on the name of the cache container.
2. The Caches view is displayed. Click on the cache to be cleared.

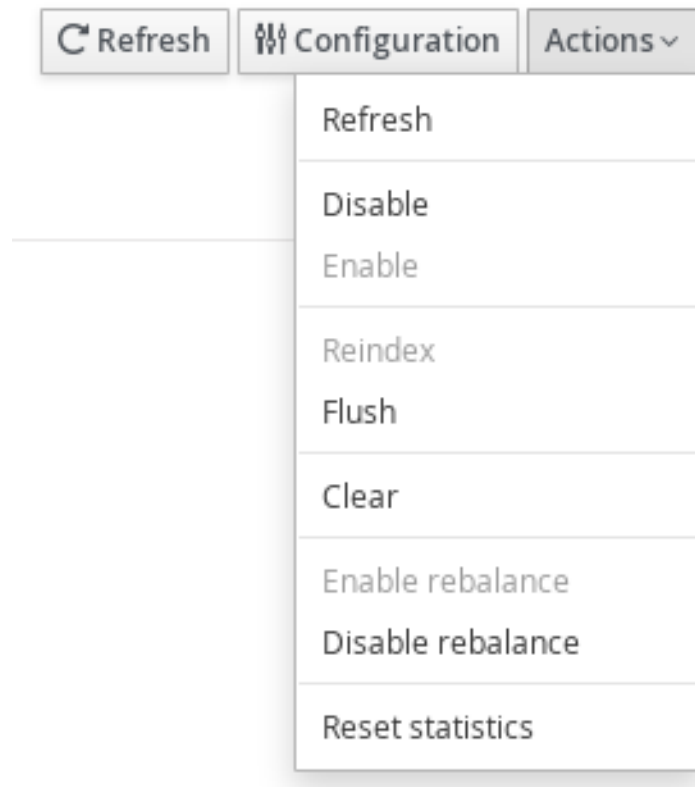
Figure 26.38. Caches View

3. On the cache statistics page, at the right hand side, click **Actions**.



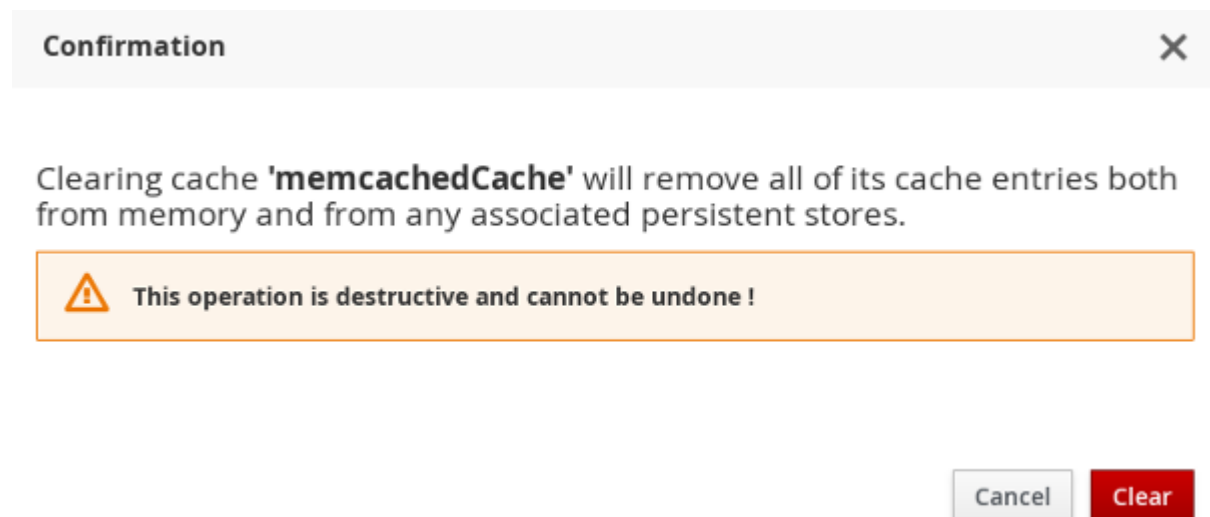
4. From the **Actions** menu, click **Clear**.

Figure 26.39. Clear Button

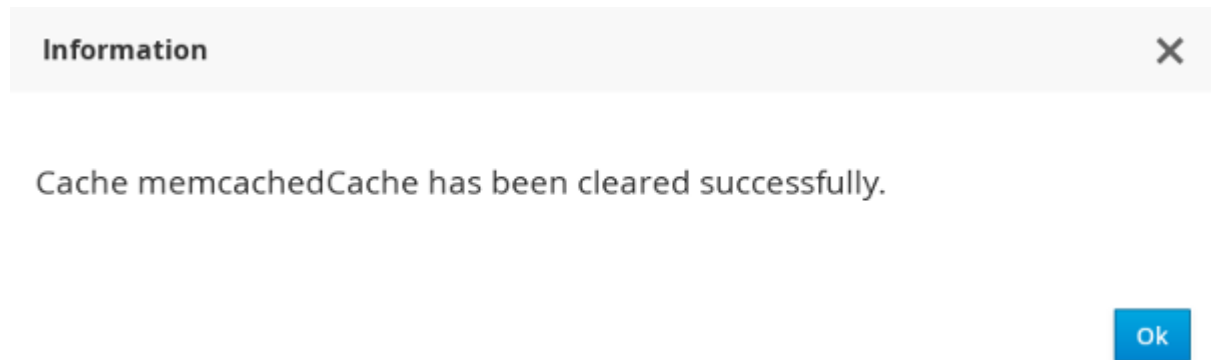


5. A confirmation dialogue box appears. Click **Clear**.

Figure 26.40. Confirmation Box



6. The cache is successfully cleared. Click **Ok**.

Figure 26.41. Information Box

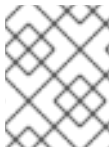
26.5.6. Server Tasks Execution

The JBoss Data Grid Administration Console allows administrators to start a server script job on the JBoss Data Grid cluster.

26.5.7. Server Tasks

26.5.7.1. New Server Task

The following procedure outlines the steps to launch a new server task:

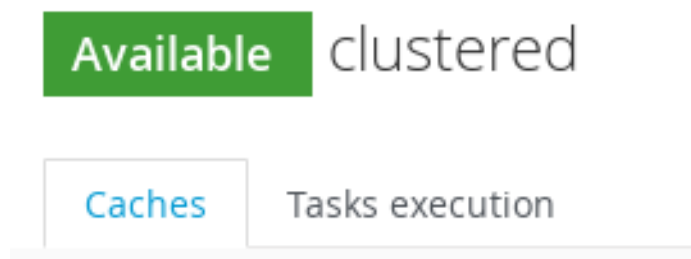


NOTE

Launching a new task is not supported if the server is running in standalone non-clustered mode.

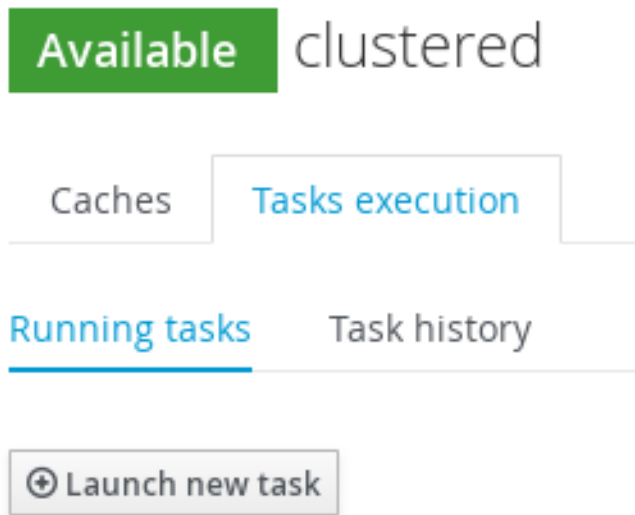
Launching a New Server Task

1. In the Cache Containers view of the JBoss Data Grid Administration Console, click on the name of the Cache container.
2. On the cache view page, click the **Task Execution** tab.

Figure 26.42. Task Execution

3. In the Tasks execution tab, click **Launch new task**.

Figure 26.43. Launch New Task



4. Enter the new task properties and click **Launch task**.

Figure 26.44. Task Properties

The screenshot shows a dialog box titled 'Launch new task'. It contains the following fields and controls:

- * Task:** A dropdown menu.
- * Cache:** A dropdown menu with the value 'memcachedCache'.
- * Originator Node:** A dropdown menu with the value 'master/Node1'.
- Parameter 1:** A text input field with 'MyParam1' and a corresponding value field with 'Value for param1'.
- Parameter 2:** A text input field with 'MyParam2' and a corresponding value field with 'Value for param2'.
- Parameter 3:** A text input field with 'Name for parameter 3' and a corresponding value field with 'Value for parameter 3'.
- Parameter 4:** A text input field with 'Name for parameter 4' and a corresponding value field with 'Value for parameter 4'.
- Parameter 5:** A text input field with 'Name for parameter 5' and a corresponding value field with 'Value for parameter 5'.
- Asynchronous task
- Launch task** button
- Close** button

26.5.7.2. Server Tasks View

After the server task is launched, it can be viewed in the Task execution tab along with the other running tasks. The set of completed server script jobs with the start time and end time can be viewed. Additionally, number of successful executions and number of failed executions can also be viewed.

Figure 26.45. Server Tasks View

Status	Cache name	Task name	Parameters
Error	My cache #1	Do some calculations	MyParam1 = 'Value for param1' AnotherParam2 = 'Another value' TheParam3 = 'Final value'
Success	My cache #2	Calculate aggregate stuff	-
Error	My cache #1	Do some calculations	MyParam1 = 'Value for param1' AnotherParam2 = 'Another value' TheParam3 = 'Final value'

Figure 26.46. Task Start/End Time

Start time	End time
Thu Sep 3 17:10:02 CEST 2015	Thu Sep 3 17:14:02 CEST 2015
Thu Sep 3 17:10:02 CEST 2015	Thu Sep 3 17:14:02 CEST 2015
Thu Sep 3 17:10:02 CEST 2015	Thu Sep 3 17:14:02 CEST 2015

26.6. CACHE CONTAINER CONFIGURATION

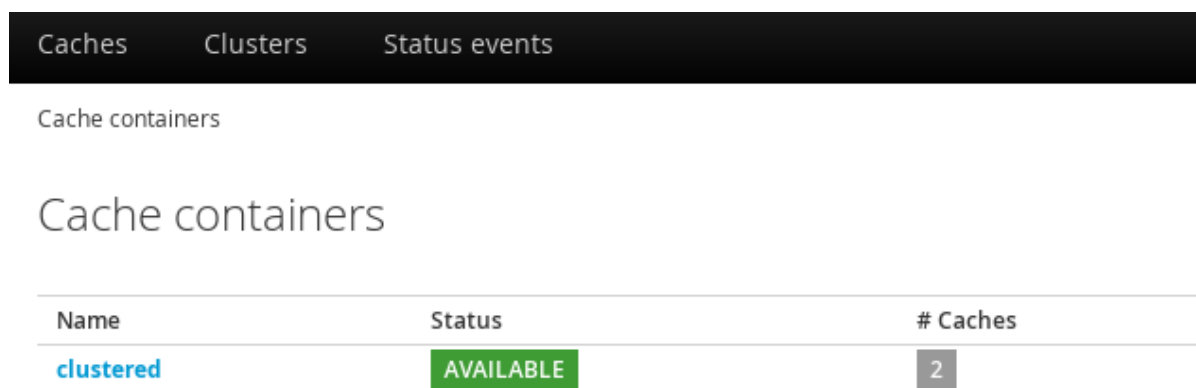
26.6.1. Cache Container Configuration

The JBoss Data Grid Administration Console allows users to view and set Cache Container level settings such as transport, thread pools, security, cache templates, deployment of remote Executables/Scripts. Each cache container is associated with a cluster.

The following procedure outlines the steps to access the Cache Container Configuration settings:

Accessing Cache Container Configuration Settings

1. In the Cache Container View, click on the name of the cache container.

Figure 26.47. Cache Container View

2. Click Configuration setting button at the top right hand side of the interface.

Figure 26.48. Configuration

The Cache Container Configuration interface is displayed.

Figure 26.49. Cache Container Configuration

Cache container configuration

[Schemas](#) Transport Thread pools Security Templates Tasks Deployments

26.6.2. Defining Protocol Buffer Schema

A Protocol Buffer Schema is defined in the Cache Container Configuration interface.

The following procedure outlines the steps to define a protobuf schema:

Defining a Protobuf Schema

1. Click **Add** at the right hand side of the Schema tab to launch the create schema window.
2. Enter the schema name and the schema in the respective fields and click **Create Schema**.

Figure 26.50. New Schema

Create new schema [X]

* Schema name:
Address

* Schema:

```

PRIVATE = 1;
PROFESSIONAL = 2;
}

message EmailAddress {
  required string email = 1;
  optional EmailType type = 2 [default = PROFESSIONAL];
}

repeated EmailAddress email = 5;
}

message Organization {
  required string name = 1;
  repeated Customer customer = 2;
}

```

Cancel Create schema

- The protocol buffer schema is added.

Figure 26.51. Protocol Buffer

Schemas Transport Thread pools Security Templates Tasks Deployments

Name	Type
Address.proto	ProtoBuf

26.6.3. Transport Setting

To access the Transport setting, click on the Transport tab in the Cache Container Configuration interface. Enter the Transport settings and click **Save**.

Figure 26.52. Transport Setting

Channel ⓘ (Undo) Restart needed

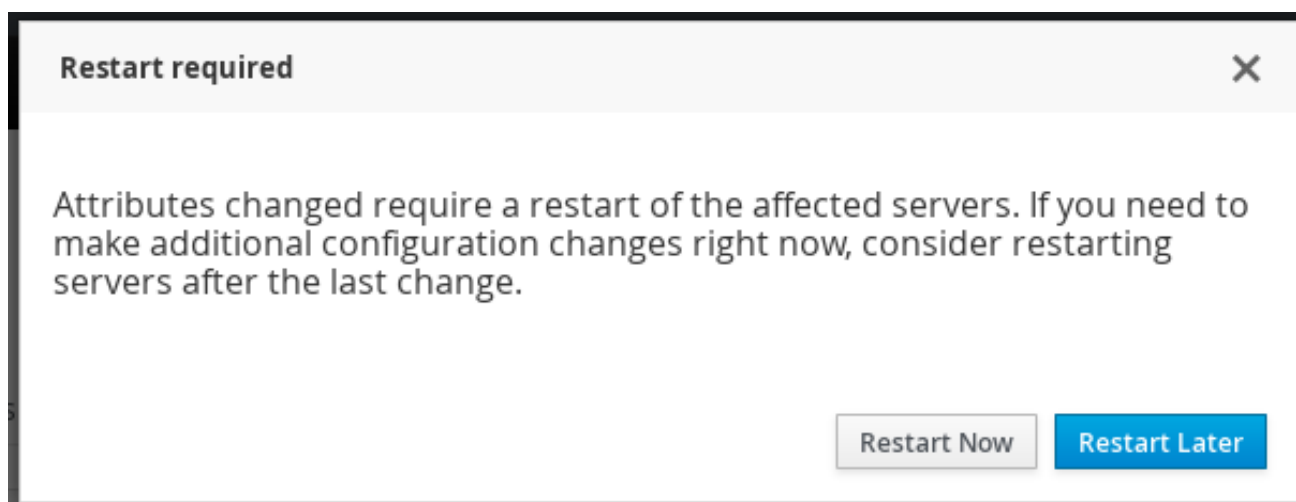
Lock Timeout ⓘ

Strict Peer To Peer ⓘ

Save Cancel

A dialog box will prompt to restart the server due to configuration changes. Restart to apply the changes.

Figure 26.53. Restart Confirmation



26.6.4. Defining Thread Pools

To define thread pools for different cache related operations, click on the Thread Pools tab in the Cache Container Configuration interface.

The JBoss Data Grid Administration Console allows administrators to set Thread Pool values for the following cache level operations:

Async Operations

Figure 26.54. Async Operations

Min Threads	<input type="text" value="25"/>	
Max Threads	<input type="text" value="25"/>	
Queue Length	<input type="text" value="1000"/>	
Keepalive Time	<input type="text" value="60000"/>	

The currently set value for each parameter is set by the console. Hover the cursor over the information icon to view the parameter description in form of a tooltip. To change a thread pool value, enter the new value in the parameter field and click **Save**. A server restart is needed after every change of values.

Expiration

For Expiration settings, the user can set values for the following parameters:





Figure 26.55. Expiration Values

Max Threads	<input type="text" value="1"/>	
Keepalive Time	<input type="text" value="60000"/>	

Listener

For Listener settings, the user can set values for the following parameters:





Figure 26.56. Listener Values

Min Threads	<input type="text" value="1"/>	
Max Threads	<input type="text" value="1"/>	
Queue Length	<input type="text" value="100000"/>	
Keepalive Time	<input type="text" value="60000"/>	

Persistence

For Persistence settings, the user can set values for the following parameters:





Figure 26.57. Persistence Values

Min Threads	<input type="text" value="1"/>	
Max Threads	<input type="text" value="4"/>	
Queue Length	<input type="text" value="0"/>	
Keepalive Time	<input type="text" value="60000"/>	

Remote Commands

For Remote Commands settings, the user can set values for the following parameters:

Figure 26.58. Remote Commands

Min Threads	<input type="text" value="25"/>	
Max Threads	<input type="text" value="25"/>	
Queue Length	<input type="text" value="100000"/>	
Keepalive Time	<input type="text" value="60000"/>	

Replication Queue

For Replication Queue settings, the user can set values for the following parameters:





Figure 26.59. Replication Queue Values

Max Threads	<input type="text" value="1"/>	
Keepalive Time	<input type="text" value="60000"/>	

State Transfer

For Listener settings, the user can set values for the following parameters:





Figure 26.60. State Transfer Values

State transfer		
Min Threads	<input type="text" value="1"/>	
Max Threads	<input type="text" value="60"/>	
Queue Length	<input type="text" value="0"/>	
Keepalive Time	<input type="text" value="60000"/>	

Transport

For Transport settings, the user can set values for the following parameters:

Figure 26.61. Transport Values

Transport		
Min Threads	<input type="text" value="25"/>	
Max Threads	<input type="text" value="25"/>	
Queue Length	<input type="text" value="100000"/>	
Keepalive Time	<input type="text" value="60000"/>	

26.6.5. Adding New Security Role

The following procedure outlines the steps to add a new security role:

Adding a Security Role

1. Click on the **Security** tab. If authorization is not defined for a cache container, click **Yes** to define.

Figure 26.62. Define Authorization

Schemas Transport Thread pools **Security** Templates Tasks Deployments

Authorization has not been defined for this cache container. Do you want to define it now?

2. Select the Role Mapper from the drop-down menu. Click **Add** to launch the permissions window.

Figure 26.63. Role Mapper Selection

Thread pools **Security** Templates Tasks Deployments

Role Mapper:

Roles:

Role	Actions
	<input type="button" value="Add"/>

3. In the **Permissions** window, enter the name of the new role and assign the permissions by checking the required check-boxes. Click **Save changes** to save the role.

Figure 26.64. Role Permissions

Permissions

* Role name: ⓘ

- READ
- WRITE
- BULK_READ
- BULK_WRITE
- EXEC
- LISTEN
- ADMIN

- The new security role is added.

Figure 26.65. New Security Role

Thread pools **Security** Templates Tasks Deployments

Role Mapper:

Roles

Role	Actions
Manager	<input type="button" value="Edit"/> <input type="button" value="Remove"/>

26.6.6. Creating Cache Configuration Template

The Templates tab in the Cache Container Configuration interface lists all the configured and available cache templates.

Figure 26.66. Cache Templates View

Schemas Transport Thread pools Security Templates Tasks Deployments

+ Create new template				
Name	Type	Mode	Traits	Actions
transactional	distributed-cache	SYNC	Transactional	Edit Remove
persistent-file-store	distributed-cache	SYNC		Edit Remove
indexed	distributed-cache	SYNC	Indexed	Edit Remove
memory-bounded	distributed-cache	SYNC	Bounded	Edit Remove
persistent-file-store-passivation	distributed-cache	SYNC	Bounded	Edit Remove
persistent-file-store-write-behind	distributed-cache	SYNC		Edit Remove
persistent-leveldb-store	distributed-cache	SYNC		Edit Remove
persistent-jdbc-string-keyed	distributed-cache	SYNC		Edit Remove

The following procedure outlines the steps to create a new Cache configuration template :

Creating New Cache Configuration Template

1. Click **Create new Template** on the right hand side of the templates list.
2. Enter the cache configuration template name and select the base configuration from the drop-down and click **Next**.

Figure 26.67. Cache Configuration Template

Add new cache configuration template ✕

Step 1: Please, enter the required cache configuration template properties:

* Template name: ⓘ

* Base configuration: ⓘ

3. Set the cache template attributes for the various cache operations such as Locking, Expiration, Indexing and others.

Figure 26.68. Cache Configuration Template

Main	
Type	distributed-cache ⓘ
Template	Config#1 ⓘ

Clustering	
Mode	SYNC ⓘ
Owners	2 ⓘ
Segments	80 ⓘ
L1 Lifespan	0 ⓘ
Capacity Factor	1 ⓘ
Remote Timeout	15000 ⓘ
Queue Size	0 ⓘ
Queue Flush Interval	10 ⓘ

Monitoring	
Statistics	<input type="checkbox"/> ⓘ
Jndi Name	<input type="text"/> ⓘ

- After entering the values, click **Create** to create the Cache Template.

26.7. CLUSTER ADMINISTRATION

26.7.1. Cluster Nodes View

Clusters centric view allows to view the nodes created for each server group and the list of deployed servers can be viewed. In Clusters view, you can add new nodes to the cluster group and view performance metrics of the particular nodes.



NOTE

The Cluster view will not appear when the server is running in standalone non-clustered mode. When running in standalone clustered mode the Cluster view will be displayed, but no operations on cluster nodes may be performed.

To access the Clusters view, navigate to the Clusters tab from the Dashboard and click on the name of the cluster.

Figure 26.69. Nodes View

STARTED Cluster: cluster Refresh Actions

Nodes

Quick search:

Add Node +	server-one master Coordinator 127.0.0.1	server-two master Coordinator 127.0.0.1	server-thre... master 127.0.0.1	MyNode1 master 127.0.0.1
	MyNode2 master 127.0.0.1	MyNode3 master 127.0.0.1	MyNode4 master 127.0.0.1	MyNode5 master 127.0.0.1

26.7.2. Cluster Nodes Mismatch

The total number of server nodes on the JBoss Data Grid cluster should ideally match the number of nodes shown in the JBoss Data Grid Administration Console. If in case, due to some reason, the expected nodes in the console do not match with the exact number of nodes on the JBoss Data Grid physical cluster, the console issues a mismatch warning by displaying the number of nodes detected and the number of expected nodes. Knowing the expected number of server nodes helps in handling Network Partitions.

If nodes mismatch occurs, it can be viewed in the clusters view, above the list of nodes as a warning. To access the Clusters view, navigate to the Clusters tab from the Dashboard and click on the name of the cluster.

In the following screen, the Console alerts the user in the form of a warning. The expected number of server nodes are 5 but only 3 are detected by the console.

Figure 26.70. Cluster Nodes Mismatch

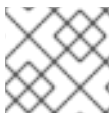
Showing 45 nodes out of 60 total.

3 nodes found, but 5 were expected.

Add Node +	Node #1 Coordinator 192.168.192.1	Node #2 192.168.192.2	Node #3 192.168.192.3
---------------	---	--------------------------	--------------------------

26.7.3. Cluster Rebalancing

The Red Hat JBoss Data Grid Administration Console allows the user to enable and disable cluster rebalancing at the cache container and cache levels.

**NOTE**

Cluster rebalancing is enabled by default.

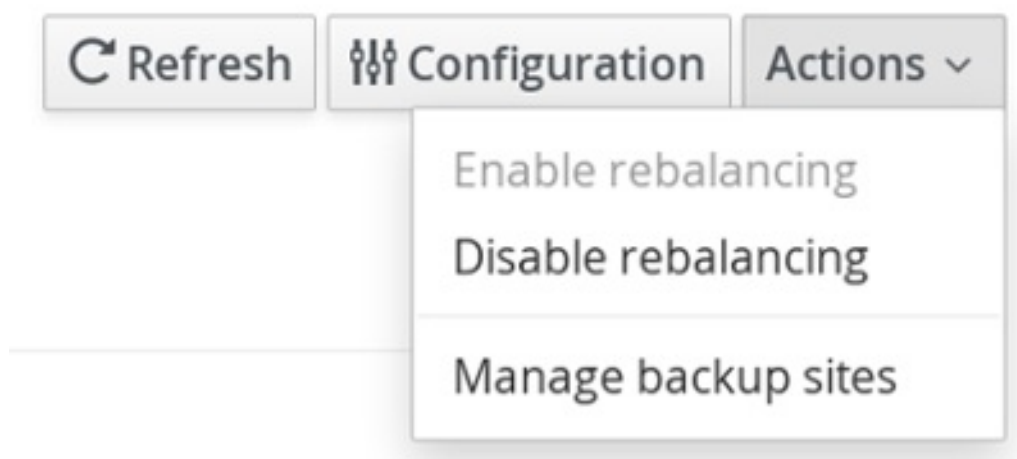
The following procedure outlines the steps to enable and disable cluster rebalancing at a cache container level :

Enable and Disable Rebalancing

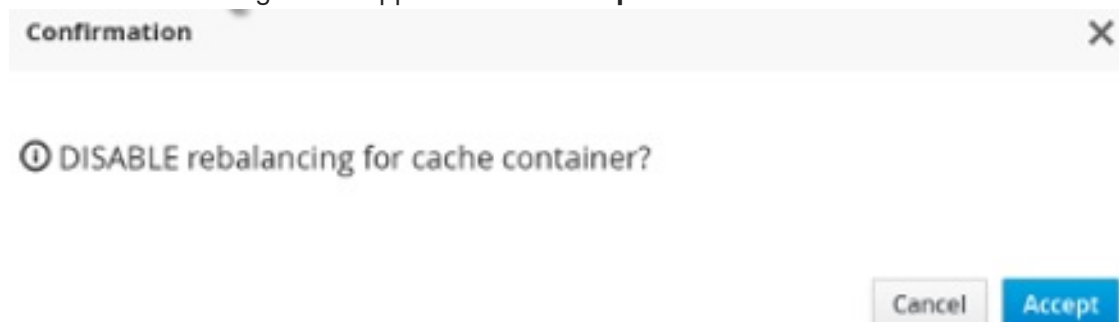
1. From the cache container view, click on the name of the cache container.
2. In the caches view, at the right hand side, click on **Actions**.



3. A callout menu is opened. Click **Disable Rebalancing**.



4. A confirmation dialogue box appears. Click **Accept**.



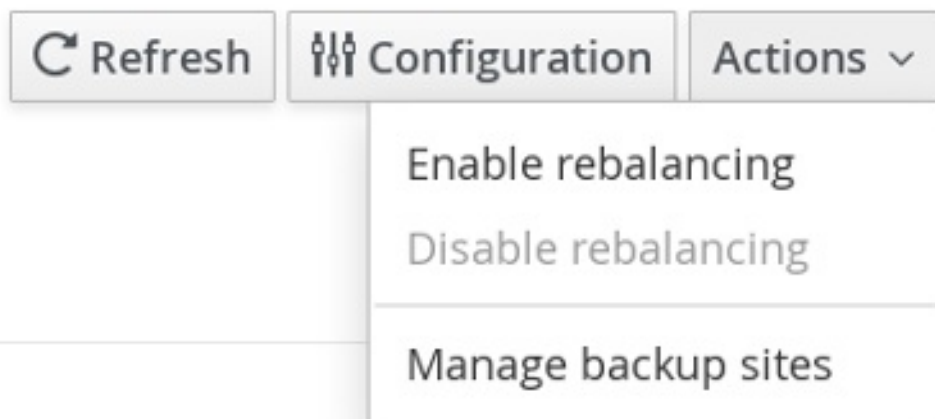
5. Cluster rebalancing is successfully disabled.

Cache containers » clustered - (cluster)

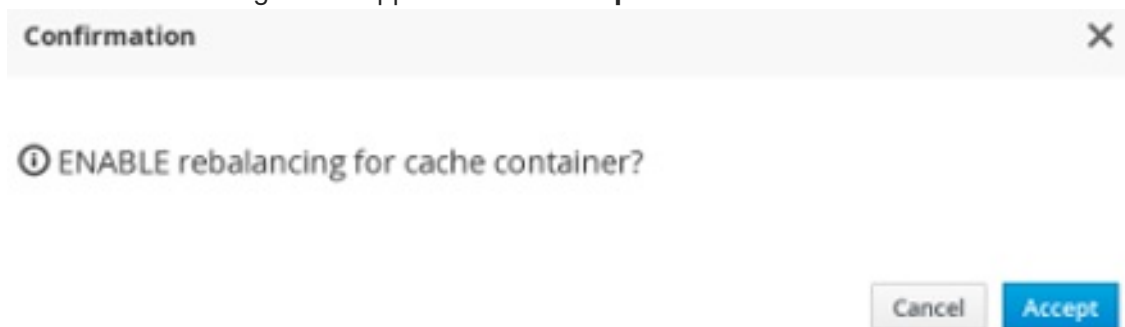
Available clustered - **⚠ Rebalancing is disabled**

✓ Success! The operation has been successfully executed.

6. To enable rebalancing, click **Actions Enable Rebalancing**.



7. A confirmation dialogue box appears. Click **Accept**.



Rebalancing is successfully enabled.

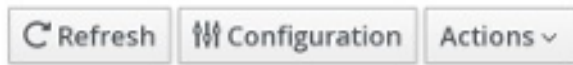
Available clustered

✓ Success! The operation has been successfully executed.

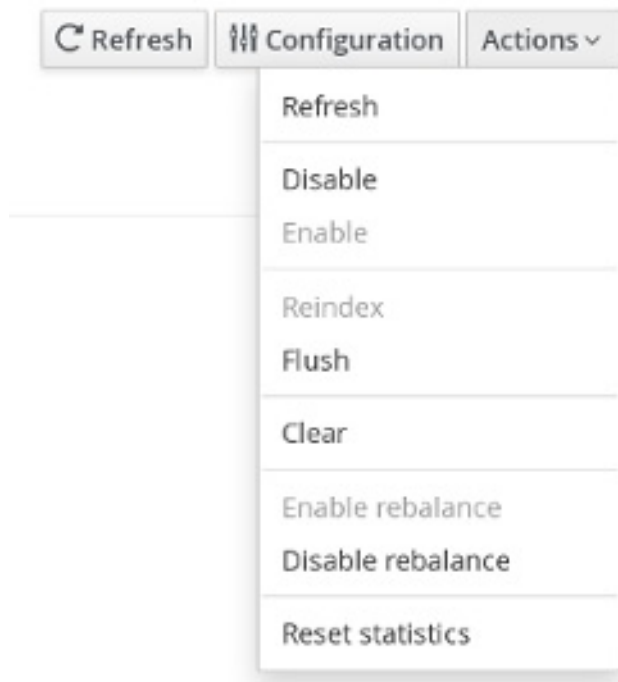
The following procedure outlines the steps to enable and disable cluster rebalancing at a cache level :

Enable and Disable Rebalancing

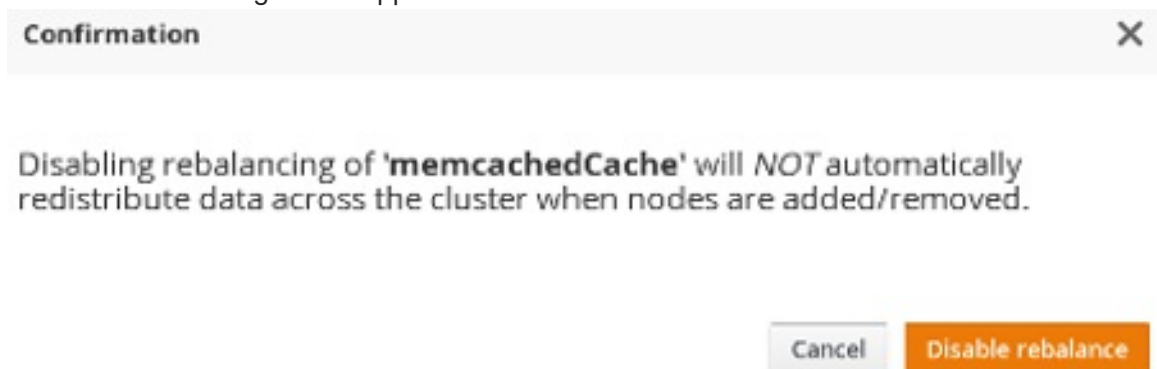
1. From the cache container view, click on the name of the cache container.
2. In the caches view, click on a specific cache.
3. The cache statistics page is displayed. At the right hand side, click **Actions**.



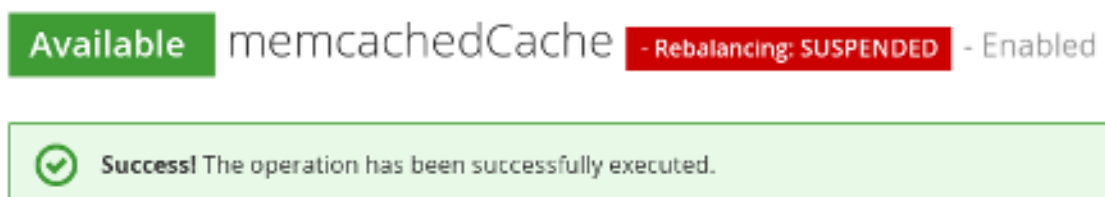
4. From the callout menu, click **Disable Rebalance**.



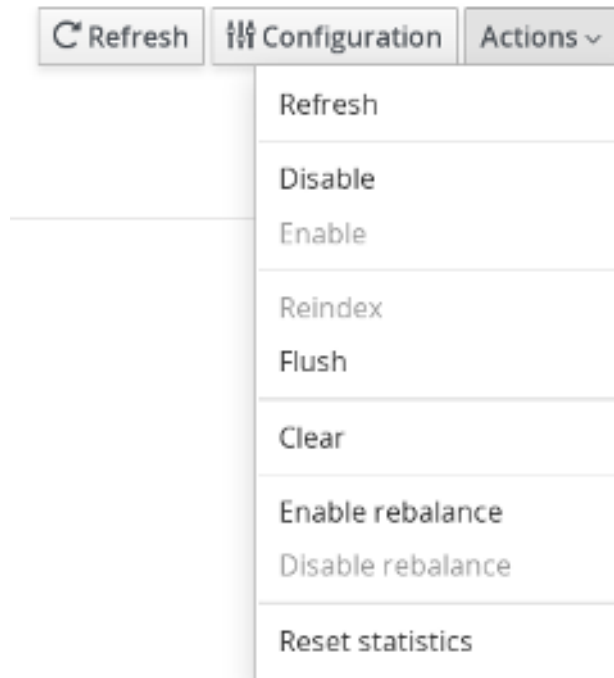
5. A confirmation dialogue box appears. Click **Disable Rebalance**.



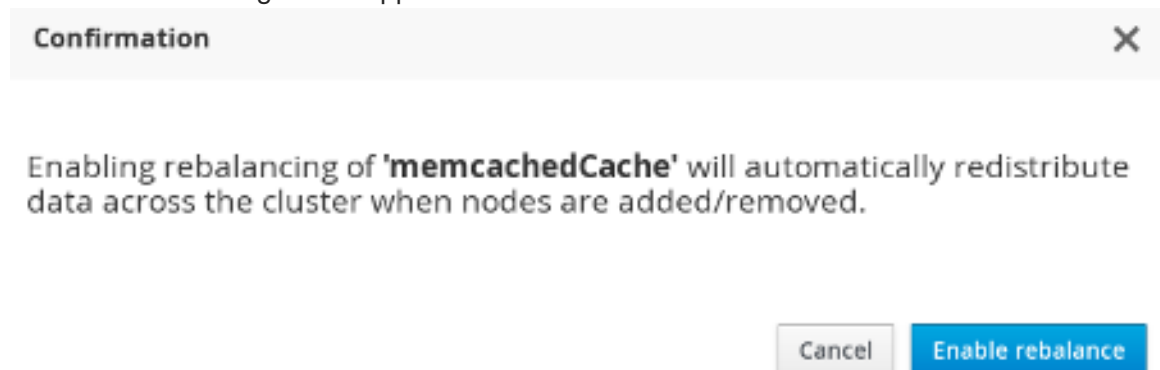
6. The rebalancing for the cache is successfully disabled.



7. To enable cache level rebalancing, click **Enable rebalance** from the **Actions** menu.




8. A confirmation dialogue box appears. Click **Enable rebalance**.



The rebalancing for the cache is successfully enabled.

Available memcachedCache - Rebalancing completed - Enabled

 **Success!** The operation has been successfully executed.

26.7.4. Cluster Partition Handling

The JBoss Data Grid Administration Console alerts the user with a visual warning when the cluster changes state to **DEGRADED**. The assumed causes for a **DEGRADED** cluster are occurrence of a network partition, unreachable node(s) or unexpected extra nodes.

The visual warning is displayed in the **Clusters** view. To access the Clusters view, navigate to the **Clusters** tab from the Dashboard and click on the name of the cluster. In the following screen, the visual warning **DEGRADED** is displayed next to the cluster name **JDG Cluster #1**.

Figure 26.71. Network Partition Warning

Clusters » JDG Cluster #1

Cluster: 'JDG Cluster #1' **DEGRADED** Rebalancing: Auto

This visual warning for a **DEGRADED** cluster is shown at Cluster, Cache Container, and Cache levels of the console.

26.7.5. Cluster Events

The JBoss Data Grid Console displays the cluster wide events such as cluster-split and cluster-merge events in a consolidated section.



NOTE

Cluster Events are not available when the server is running in standalone non-clustered mode.

Along with the cluster events, the console displays the timestamp of the associated event. Cluster events can be viewed in the Cache containers page, the Clusters view page and also in the Status Events tab of the Dashboard.

To view cluster events on the cache containers page, navigate to the default cache containers view which is the default landing interface after logging into the console. The Cluster events are displayed at the right hand side in a consolidated section under the title **Latest Grid Events**

⚠ Cluster 'Cluster #1': Split detected.

Thu Sep 3 16:10:02 CEST 2015

ⓘ Cluster 'Cluster #1': Rebalancing started.

Thu Sep 3 16:10:02 CEST 2015

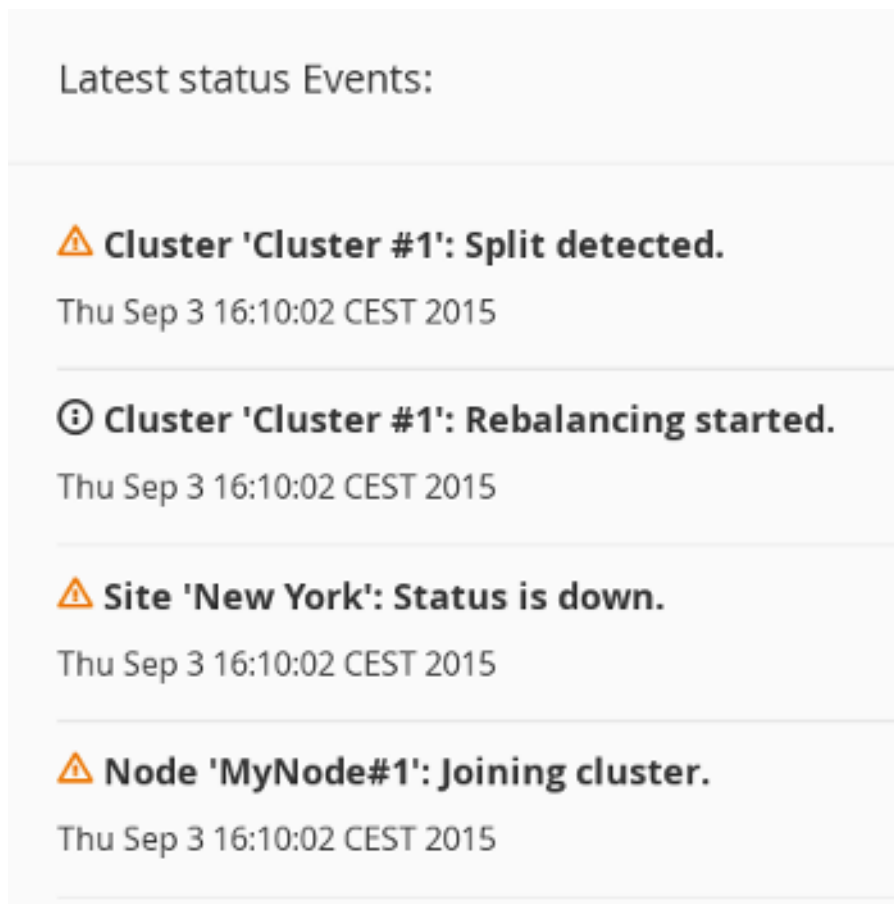
⚠ Site 'New York': Status is down.

Thu Sep 3 16:10:02 CEST 2015

⚠ Node 'MyNode#1': Joining cluster.

Thu Sep 3 16:10:02 CEST 2015

To view the cluster events on the Clusters view page, navigate to the Clusters view by clicking on the Clusters tab. The Cluster events are displayed at the right hand side in a consolidated section under the title **Latest status Events**



Latest status Events:

- ⚠ Cluster 'Cluster #1': Split detected.**
Thu Sep 3 16:10:02 CEST 2015
- ⓘ Cluster 'Cluster #1': Rebalancing started.**
Thu Sep 3 16:10:02 CEST 2015
- ⚠ Site 'New York': Status is down.**
Thu Sep 3 16:10:02 CEST 2015
- ⚠ Node 'MyNode#1': Joining cluster.**
Thu Sep 3 16:10:02 CEST 2015

26.7.6. Adding Node

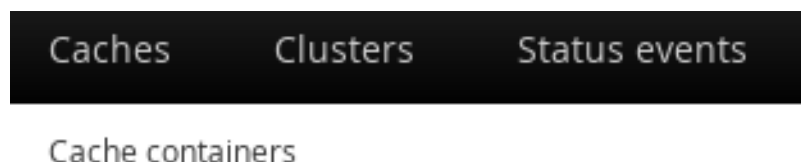
The JBoss Data Grid Administration Console allows administrators to configure new nodes.

The following procedure outlines the steps to add a new Node:

Adding a New Node

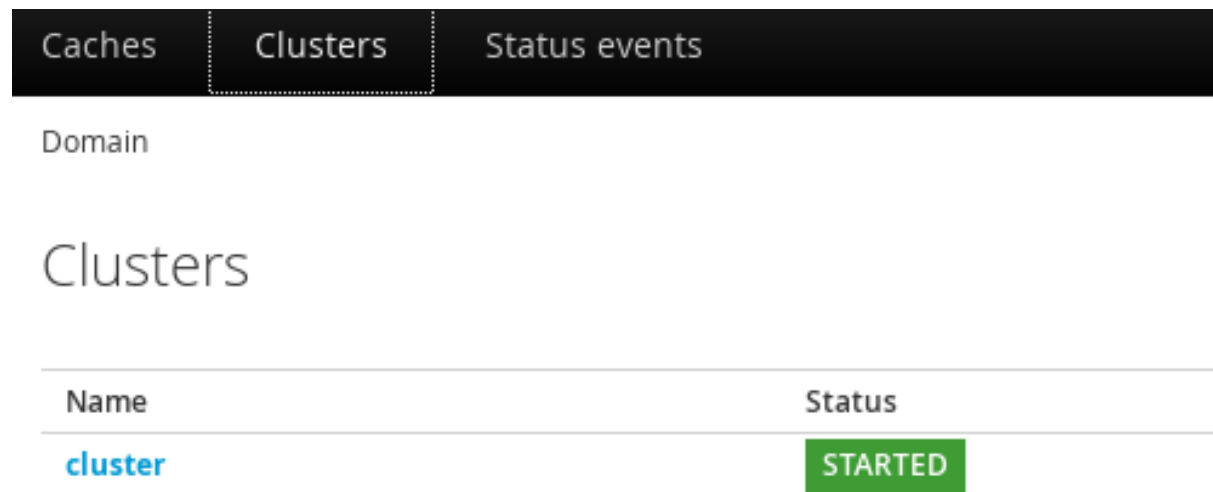
1. In the Dashboard view, click **Cluster** tab.

Figure 26.72. Clusters Tab



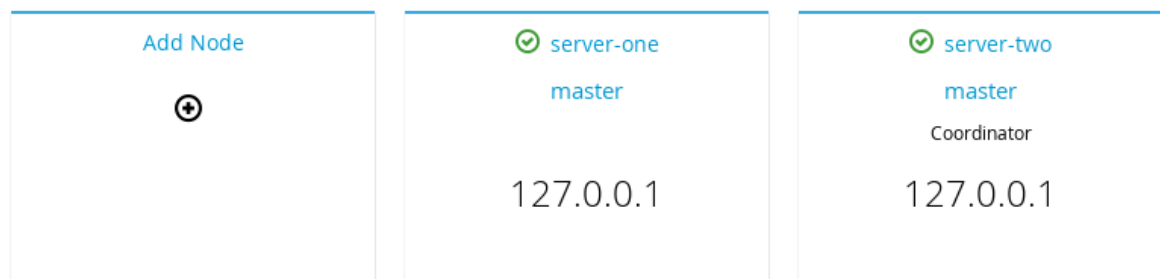
2. Click on the name of the cluster where the new node has to be added.

Figure 26.73. Cluster Selection



3. Click **Add Node**.

Figure 26.74. Add Node Created



4. The node configuration window is opened. Enter the node properties in the respective fields and click **Create**

Figure 26.75. Node Properties

Add new server node ✕

* Name: ⓘ

* Host: ⓘ

Port offset: ⓘ

JVM options:

Stack:

Heap:

Agent lib:

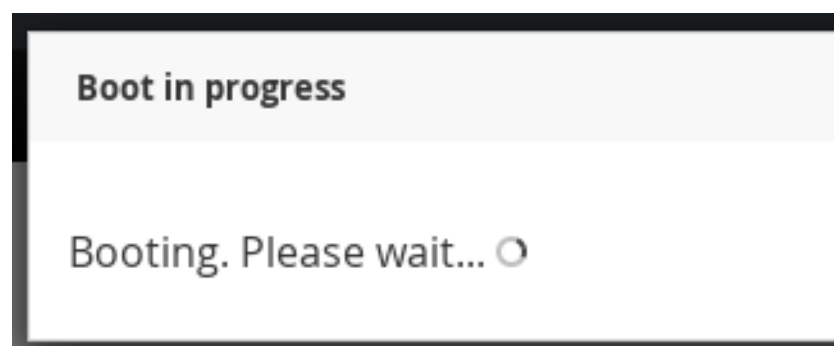
Agent path:

JVM options:

System properties:

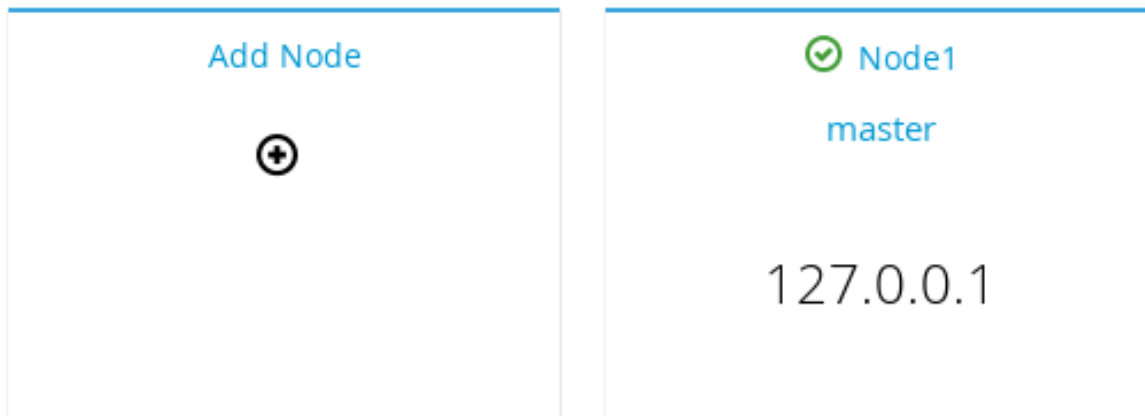
5. The system boots up.

Figure 26.76. System Boot



6. The new node is successfully created.

Figure 26.77. New Node

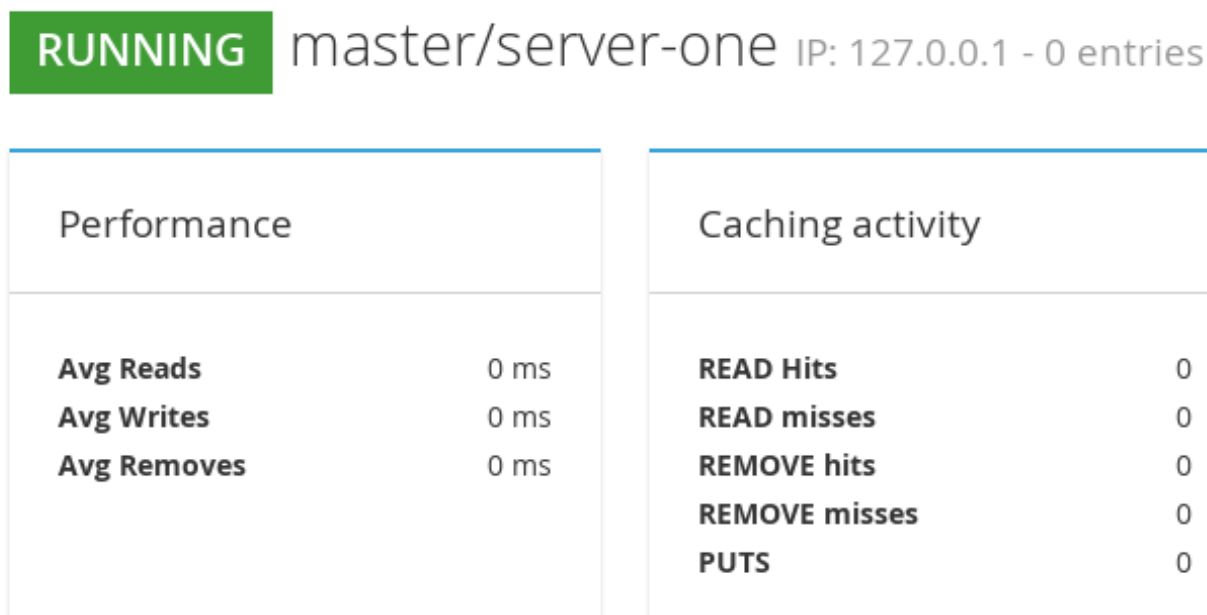


26.7.7. Node Statistics and Properties View

JBoss Data Grid Administration Console allows users to view the average time for reads, average times for writes, total number of entries, total number of reads, total number of failed reads, total number of writes and other data.

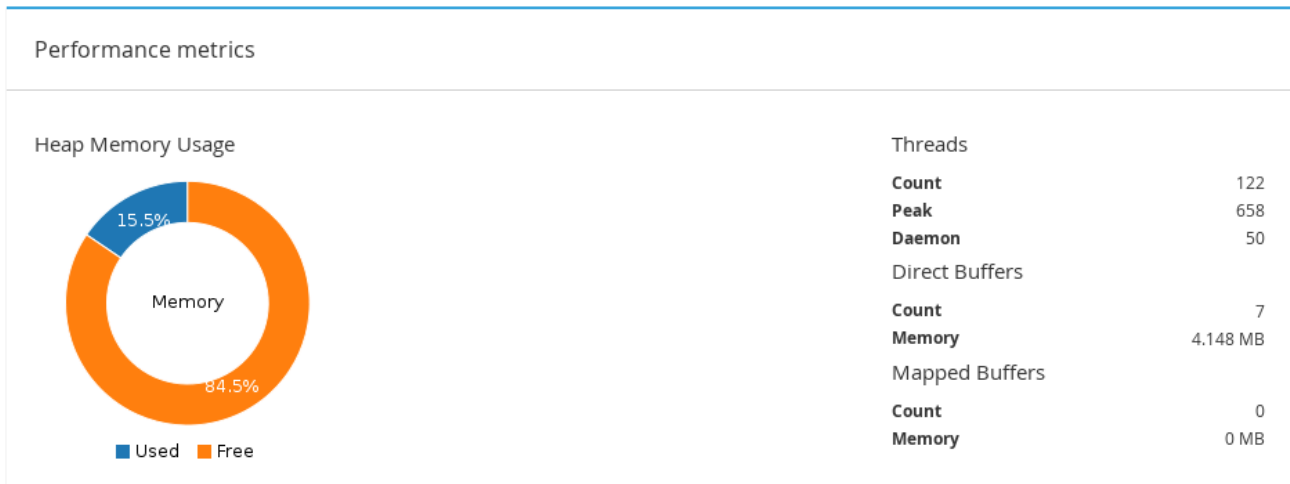
To view the Node statistics, click on the name of the Node in the Clusters tab on the JBoss Data Grid Administration Console.

Figure 26.78. Nodes Statistics



26.7.8. Node Performance Metrics View

To view the Node performance metrics, click on the name of the node in the Clusters tab of the JBoss Data Grid Administration Console

Figure 26.79. Node Performance Metrics

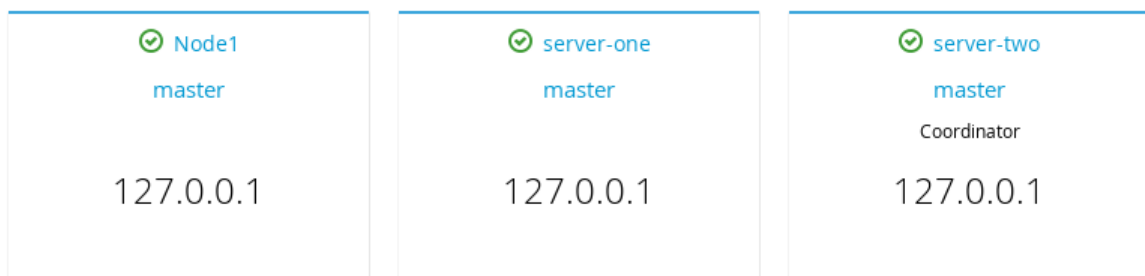
26.7.9. Disabling a Node

The JBoss Data Grid Administration Console allows administrators to disable nodes.

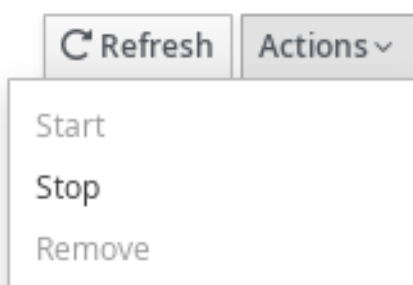
To disable a node of a cluster, follow these steps:

Adding a New Node

1. Click on the name of the cluster in the Cluster View of the JBoss Data Grid Administration Console.
2. In the Nodes view, click on the node to be disabled.

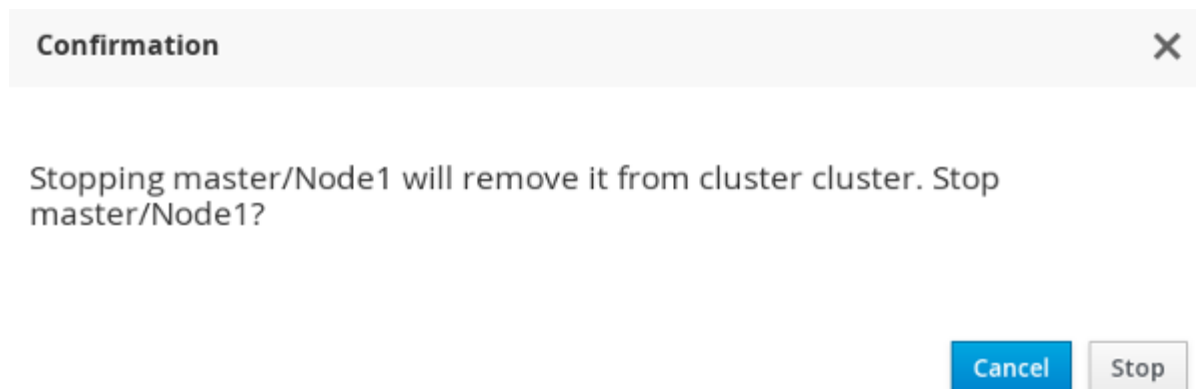
Figure 26.80. Nodes View

3. The Node statistics view is opened. Click on the Actions tab located at the right hand side of the page and then click **Stop**.

Figure 26.81. Nodes Stop

4. A confirmation box appears. Click **Stop** to shut down the node.

Figure 26.82. Confirmation Box



26.7.10. Cluster Shutdown and Restart

26.7.10.1. Cluster Shutdown

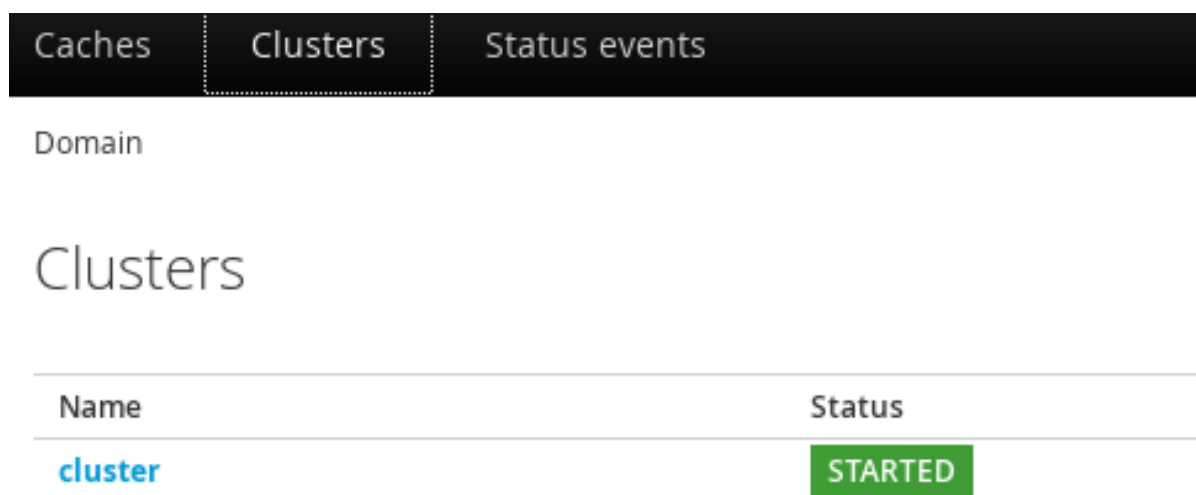
JBoss Data Grid Administration Console allows convenient and controlled shutdown of JBoss Data Grid clusters for maintenance purposes. For caches with a configured cache store, the data will be persisted without any data loss. For caches without a configured cache store, data will be lost after cluster shutdown.

To shut down or stop a cluster, follow these steps:

Shutting Down Cluster

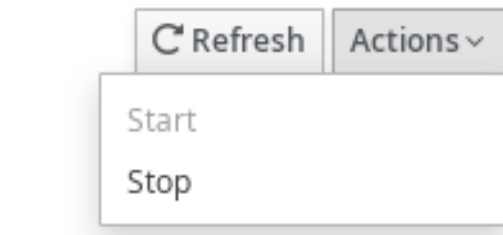
1. Navigate to the Clusters view in the JBoss Data Grid Administration console and click on the name of the cluster.

Figure 26.83. Clusters View



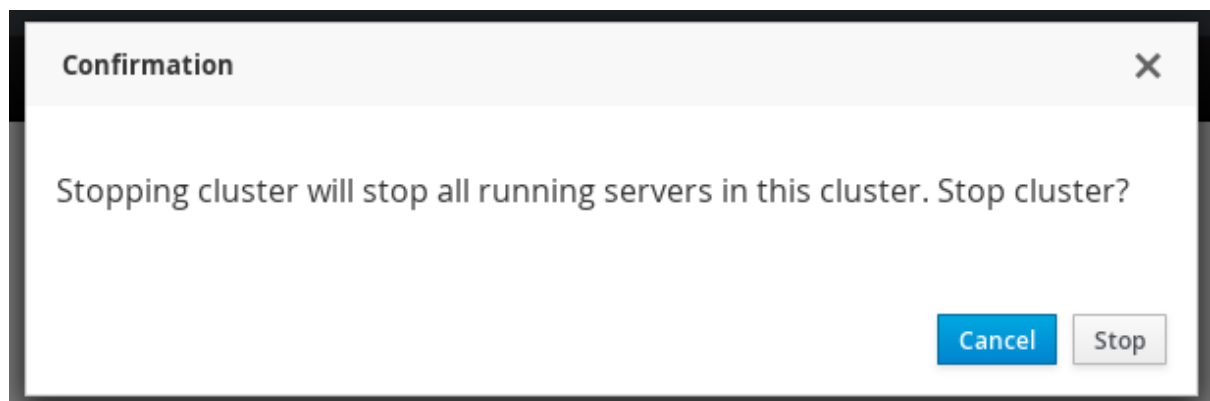
2. On the Nodes view page, locate the Actions tab to the top right hand side of the interface. Click on Actions tab and then click **Stop**.

Figure 26.84. Cluster Stop



3. A confirmation box will appear. To confirm, click Stop.

Figure 26.85. Confirmation Box



26.7.10.2. Cluster Start

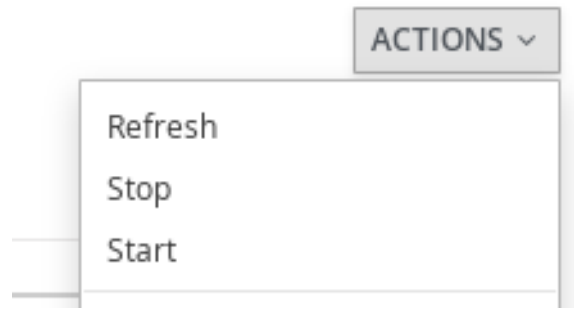
JBoss Data Grid Administration Console allows restarting a stopped cluster. The cache data is preloaded without any data loss for caches with configured cache-store. Caches without a configured cache store, will initially contain no data.

Preloading will only happen if preload is enabled on the cache store. If the local cache state on one of the nodes is corrupt, the cache will not start and manual intervention will be required.

To a cluster, follow these steps:

Starting Cluster

1. Navigate to the Clusters view in the JBoss Data Grid Administration console and click on the name of the cluster.
2. On the Nodes view page, locate the Actions tab to the top right hand side of the interface. Click on Actions tab and then click **Start**.

Figure 26.86. Cluster Start

3. A confirmation box will appear. Click Start to start the cluster.

PART XIII. SECURING DATA IN RED HAT JBOSS DATA GRID

CHAPTER 27. INTRODUCTION

27.1. SECURING DATA IN RED HAT JBOSS DATA GRID

In Red Hat JBoss Data Grid, data security can be implemented in the following ways:

Role-based Access Control

JBoss Data Grid features role-based access control for operations on designated secured caches. Roles can be assigned to users who access your application, with roles mapped to permissions for cache and cache-manager operations. Only authenticated users are able to perform the operations that are authorized for their role.

In Library mode, data is secured via role-based access control for CacheManagers and Caches, with authentication delegated to the container or application. In Remote Client-Server mode, JBoss Data Grid is secured by passing identity tokens from the Hot Rod client to the server, and role-based access control of Caches and CacheManagers.

Node Authentication and Authorization

Node-level security requires new nodes or merging partitions to authenticate before joining a cluster. Only authenticated nodes that are authorized to join the cluster are permitted to do so. This provides data protection by preventing unauthorized servers from storing your data.

Encrypted Communications Within the Cluster

JBoss Data Grid increases data security by supporting encrypted communications between the nodes in a cluster by using a user-specified cryptography algorithm, as supported by Java Cryptography Architecture (JCA).

JBoss Data Grid also provides audit logging for operations, and the ability to encrypt communication between the Hot Rod Client and Server using Transport Layer Security (TLS/SSL).

CHAPTER 28. RED HAT JBOSS DATA GRID SECURITY: AUTHORIZATION AND AUTHENTICATION

28.1. RED HAT JBOSS DATA GRID SECURITY: AUTHORIZATION AND AUTHENTICATION

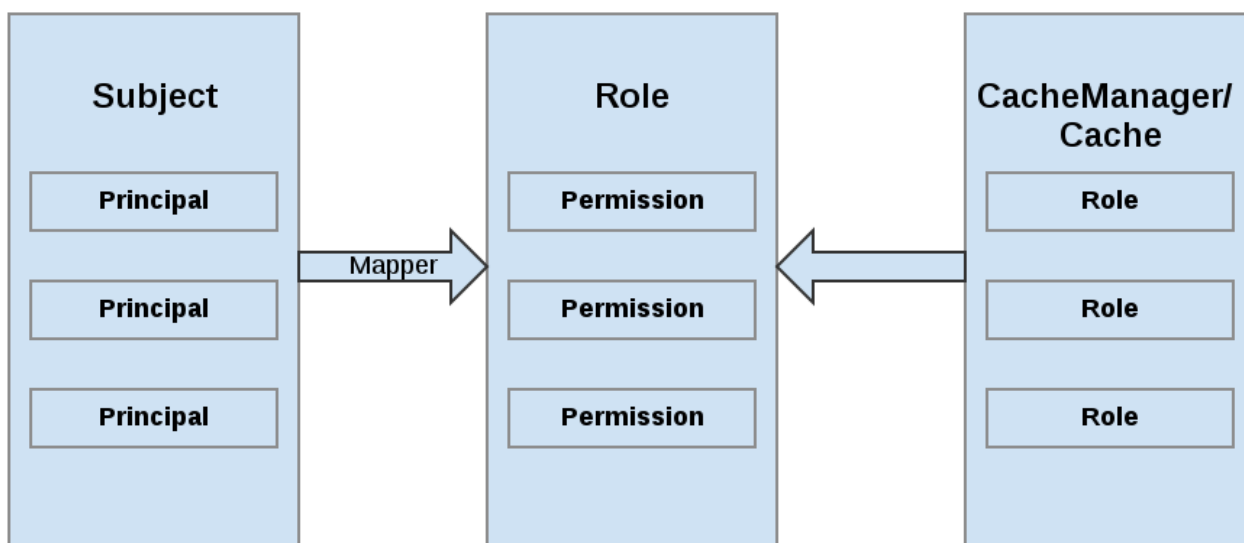
Red Hat JBoss Data Grid is able to perform authorization on CacheManagers and Caches. JBoss Data Grid authorization is built on standard security features available in a JDK, such as JAAS and the SecurityManager.

If an application attempts to interact with a secured CacheManager and Cache, it must provide an identity which JBoss Data Grid's security layer can validate against a set of required roles and permissions. Once validated, the client is issued a token for subsequent operations. Where access is denied, an exception indicating a security violation is thrown.

When a cache has been configured for with authorization, retrieving it returns an instance of **SecureCache**. **SecureCache** is a simple wrapper around a cache, which checks whether the "current user" has the permissions required to perform an operation. The "current user" is a Subject associated with the **AccessControlContext**.

JBoss Data Grid maps Principals names to roles, which in turn, represent one or more permissions. The following diagram represents these relationships:

Figure 28.1. Roles and Permissions Mapping



28.2. PERMISSIONS

Access to a CacheManager or a Cache is controlled using a set of required permissions. Permissions control the type of action that is performed on the CacheManager or Cache, rather than the type of data being manipulated. Some of these permissions can apply to specifically name entities, such as a named cache. Different types of permissions are available depending on the entity.

Table 28.1. CacheManager Permissions

Permission	Function	Description
CONFIGURATION	defineConfiguration	Whether a new cache configuration can be defined.
LISTEN	addListener	Whether listeners can be registered against a cache manager.
LIFECYCLE	stop, start	Whether the cache manager can be stopped or started respectively.
ALL		A convenience permission which includes all of the above.

Table 28.2. Cache Permissions

Permission	Function	Description
READ	get, contains	Whether entries can be retrieved from the cache.
WRITE	put, putIfAbsent, replace, remove, evict	Whether data can be written/replaced/removed/evicted from the cache.
EXEC	distexec, mapreduce	Whether code execution can be run against the cache.
LISTEN	addListener	Whether listeners can be registered against a cache.
BULK_READ	keySet, values, entrySet, query	Whether bulk retrieve operations can be executed.
BULK_WRITE	clear, putAll	Whether bulk write operations can be executed.
LIFECYCLE	start, stop	Whether a cache can be started / stopped.

Permission	Function	Description
ADMIN	getVersion, addInterceptor*, removeInterceptor, getInterceptorChain, getEvictionManager, getComponentRegistry, getDistributionManager, getAuthorizationManager, evict, getRpcManager, getCacheConfiguration, getCacheManager, getInvocationContextContainer, setAvailability, getDataContainer, getStats, getXAResource	Whether access to the underlying components/internal structures is allowed.
ALL		A convenience permission which includes all of the above.
ALL_READ		Combines READ and BULK_READ.
ALL_WRITE		Combines WRITE and BULK_WRITE.

**NOTE**

Some permissions may need to be combined with others in order to be useful. For example, EXEC with READ or with WRITE.

28.3. ROLE MAPPING

In order to convert the Principals in a Subject into a set of roles used for authorization, a **PrincipalRoleMapper** must be specified in the global configuration. Red Hat JBoss Data Grid ships with three mappers, and also allows you to provide a custom mapper.

Table 28.3. Mappers

Mapper Name	Java	XML	Description
IdentityRoleMapper	org.infinispan.security.impl.IdentityRoleMapper	<identity-role-mapper />	Uses the Principal name as the role name.

Mapper Name	Java	XML	Description
CommonNameRoleMapper	org.infinispan.security.impl.CommonRoleMapper	<common-name-role-mapper />	If the Principal name is a Distinguished Name (DN), this mapper extracts the Common Name (CN) and uses it as a role name. For example the DN cn=managers, ou=people, dc=example, dc=com will be mapped to the role managers .
ClusterRoleMapper	org.infinispan.security.impl.ClusterRoleMapper	<cluster-role-mapper />	Uses the ClusterRegistry to store principal to role mappings. This allows the use of the CLI's GRANT and DENY commands to add/remove roles to a Principal.
Custom Role Mapper		<custom-role-mapper class="a.b.c" />	Supply the fully-qualified class name of an implementation of org.infinispan.security.impl.PrincipalRoleMapper

28.4. CONFIGURING AUTHENTICATION AND ROLE MAPPING USING LOGIN MODULES

When using the authentication **login-module** for querying roles from LDAP, you must implement your own mapping of Principals to Roles, as custom classes are in use. An example implementation of this conversion is found in the JBoss Data Grid *Developer Guide*, while a declarative configuration example is below:

Example of LDAP Login Module Configuration

```
<security-domain name="ispn-secure" cache-type="default">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapLoginModule"
flag="required">
      <module-option name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory"/>
      <module-option name="java.naming.provider.url"
value="ldap://localhost:389"/>
      <module-option name="java.naming.security.authentication"
value="simple"/>
    </login-module>
  </authentication>
</security-domain>
```

```

        <module-option name="principalDNPrefix" value="uid="/>
        <module-option name="principalDNSuffix"
value=", ou=People, dc=infinispan, dc=org"/>
        <module-option name="rolesCtxDN"
value="ou=Roles, dc=infinispan, dc=org"/>
        <module-option name="uidAttributeID" value="member"/>
        <module-option name="matchOnUserDN" value="true"/>
        <module-option name="roleAttributeID" value="cn"/>
        <module-option name="roleAttributeIsDN" value="false"/>
        <module-option name="searchScope" value="ONELEVEL_SCOPE"/>
    </login-module>
</authentication>
</security-domain>

```

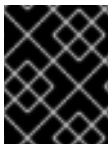
Example of Login Module Configuration

```

<security-domain name="krb-admin" cache-type="default">
  <authentication>
    <login-module code="Kerberos" flag="required">
      <module-option name="useKeyTab" value="true"/>
      <module-option name="principal"
value="admin@INFINISPAN.ORG"/>
      <module-option name="keyTab"
value="${basedir}/keytab/admin.keytab"/>
    </login-module>
  </authentication>
</security-domain>

```

When using GSSAPI authentication, this would typically involve using LDAP for role mapping, with the JBoss Data Grid server authenticating itself to the LDAP server via GSSAPI. For an example on configuring this authentication to an Active Directory server refer to [Active Directory Authentication Using Kerberos \(GSSAPI\)](#).



IMPORTANT

For information on configuring an LDAP server, or specifying users and roles in an LDAP server, refer to the Red Hat Directory Server *Administration Guide*.

28.5. CONFIGURING RED HAT JOSS DATA GRID FOR AUTHORIZATION

Authorization is configured at two levels: the cache container (CacheManager), and at the single cache.

CacheManager

The following is an example configuration for authorization at the CacheManager level:

CacheManager Authorization (Declarative Configuration)

```

<cache-container name="local" default-cache="default">
  <security>
    <authorization>
      <identity-role-mapper />
      <role name="admin" permissions="ALL"/>
    </authorization>
  </security>
</cache-container>

```

```

        <role name="reader" permissions="READ"/>
        <role name="writer" permissions="WRITE"/>
        <role name="supervisor" permissions="ALL_READ ALL_WRITE"/>
    </authorization>
</security>
</cache-container>

```

Each cache container determines:

- whether to use authorization.
- a class which will map principals to a set of roles.
- a set of named roles and the permissions they represent.

You can choose to use only a subset of the roles defined at the container level.

Roles

Roles may be applied on a cache-per-cache basis, using the roles defined at the cache-container level, as follows:

Defining Roles

```

<local-cache name="secured">
  <security>
    <authorization roles="admin reader writer supervisor"/>
  </security>
</local-cache>

```



IMPORTANT

Any cache that is intended to require authentication must have a listing of roles defined; otherwise authentication is not enforced as the no-anonymous policy is defined by the cache's authorization.



IMPORTANT

The REST protocol is not supported for use with authorization, and any attempts to access a cache with authorization enabled will result in a **SecurityException**.

28.6. AUTHORIZATION USING A SECURITYMANAGER

In Red Hat JBoss Data Grid's Remote Client-Server mode, authorization is able to work without a **SecurityManager** for basic cache operations. In Library mode, a **SecurityManager** may also be used to perform some of the more complex tasks, such as distexec and query among others.

In order to enforce access restrictions, enable the **SecurityManager** in your JVM using one of the following methods:

Command Line

```
java -Djava.security.manager ...
```

Programmatically

```
System.setSecurityManager(new SecurityManager());
```

Using the JDK's default implementation is not required; however, an appropriate policy file must be supplied. The policy file defines a set of permissions, which the **SecurityManager** examines when an application performs an action. If the action is allowed by the policy file, then the **SecurityManager** will permit the action to take place; however, if the action is not allowed by the policy then the **SecurityManager** denies that action.

Example policy files are below:

Library Mode Security Policy File Example

```
// Grant permissions to all of the Infinispan libraries. Modify the URLs
// of the codebases below to actually point to the physical location of the
// infinispan-embedded uberjar in your environment

grant codeBase "file://path/to/infinispan-embedded-8.4.0.Final-redhat-
2.jar" {
    permission java.lang.RuntimePermission "accessDeclaredMembers";
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

    // Modify this depending on the naming and location of your
    // configuration files
    permission java.io.FilePermission ".${/}jgroups.xml", "read";
    permission java.util.PropertyPermission "*" "read";
    permission java.net.SocketPermission "*";

    permission java.util.PropertyPermission "*" "read";

    // Modify this depending on the naming and location of your
    // configuration files
    permission java.io.FilePermission ".${/}infinispan.xml", "read";
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";

    // ForkJoin backport
    permission java.lang.RuntimePermission
"accessClassInPackage.sun.misc";

    // Infinispan shutdown hooks
    permission java.lang.RuntimePermission "shutdownHooks";
    permission java.util.PropertyPermission "user.dir" "read";

    // ConcurrentHashMap backports
    permission java.util.PropertyPermission
"java.util.concurrent.ForkJoinPool.common.parallelism" "read";
    permission java.util.PropertyPermission
"java.util.concurrent.ForkJoinPool.common.exceptionHandler" "read";
    permission java.util.PropertyPermission
"java.util.concurrent.ForkJoinPool.common.threadFactory" "read";

    // Infinispan security
    permission javax.security.auth.AuthPermission "doAs";
```

```

    permission javax.security.auth.AuthPermission "getSubject";
    permission org.infinispan.security.CachePermission "ALL";
}

```

Remote Client-Server Security Policy File Example

```

// Grant permissions to all of the Infinispan libraries. Modify the URLs
of the codebases below to actually point to the physical location of the
libraries in your environment

grant codeBase
"file://$JDG_HOME/modules/system/layers/base/org/jboss/marshalling/main/jb
oss-marshalling-osgi-1.4.10.Final-redhat-3.jar" {
    permission java.lang.RuntimePermission "accessDeclaredMembers";
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
}

grant codeBase
"file://$JDG_HOME/modules/system/layers/base/org/jgroups/main/jgroups-
4.0.0.CR2-redhat-1.jar" {
    // Modify this depending on the naming and location of your
configuration files
    permission java.io.FilePermission ".${/}jgroups.xml", "read";
    permission java.util.PropertyPermission "*" "read";
    permission java.net.SocketPermission "*";
}

grant codeBase
"file://$JDG_HOME/modules/system/layers/base/org/infinispan/commons/main/i
nfinispan-commons.jar" {
    permission java.util.PropertyPermission "*" "read";
}

grant codeBase
"file://$JDG_HOME/modules/system/layers/base/org/infinispan/main/infinispa
n-core.jar" {
    // Modify this depending on the naming and location of your
configuration files
    permission java.io.FilePermission ".${/}infinispan.xml", "read";
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
    permission java.lang.RuntimePermission
"accessClassInPackage.sun.misc"; // ForkJoin backport
    permission java.lang.RuntimePermission "shutdownHooks"; // Infinispan
shutdown hooks
    permission java.util.PropertyPermission "user.dir" "read";

    // ConcurrentHashMap backport
    permission java.util.PropertyPermission
"java.util.concurrent.ForkJoinPool.common.parallelism" "read";
    permission java.util.PropertyPermission
"java.util.concurrent.ForkJoinPool.common.exceptionHandler" "read";
    permission java.util.PropertyPermission
"java.util.concurrent.ForkJoinPool.common.threadFactory" "read";

    // Infinispan security
    permission javax.security.auth.AuthPermission "doAs";

```

```

    permission javax.security.auth.AuthPermission "getSubject";
    permission org.infinispan.security.CachePermission "ALL";
}

```

28.7. SECURITYMANAGER IN JAVA

28.7.1. About the Java Security Manager

Java Security Manager

The Java Security Manager is a class that manages the external boundary of the Java Virtual Machine (JVM) sandbox, controlling how code executing within the JVM can interact with resources outside the JVM. When the Java Security Manager is activated, the Java API checks with the security manager for approval before executing a wide range of potentially unsafe operations.

The Java Security Manager uses a security policy to determine whether a given action will be permitted or denied.

28.7.2. About Java Security Manager Policies

Security Policy

A set of defined permissions for different classes of code. The Java Security Manager compares actions requested by applications against the security policy. If an action is allowed by the policy, the Security Manager will permit that action to take place. If the action is not allowed by the policy, the Security Manager will deny that action. The security policy can define permissions based on the location of code, on the code's signature, or based on the subject's principals.

The Java Security Manager and the security policy used are configured using the Java Virtual Machine options `java.security.manager` and `java.security.policy`.

Basic Information

A security policy's entry consists of the following configuration elements, which are connected to the **policytool**:

CodeBase

The URL location (excluding the host and domain information) where the code originates from. This parameter is optional.

SignedBy

The alias used in the keystore to reference the signer whose private key was used to sign the code. This can be a single value or a comma-separated list of values. This parameter is optional. If omitted, presence or lack of a signature has no impact on the Java Security Manager.

Principals

A list of **principal_type/principal_name** pairs, which must be present within the executing thread's principal set. The Principals entry is optional. If it is omitted, it signifies that the principals of the executing thread will have no impact on the Java Security Manager.

Permissions

A permission is the access which is granted to the code. Many permissions are provided as part of the Java Enterprise Edition 6 (Java EE 6) specification. This document only covers additional permissions which are provided by JBoss EAP 6.



IMPORTANT

Refer to your container documentation on how to configure the security policy, as it may differ depending on the implementation.

28.7.3. Write a Java Security Manager Policy

Introduction

An application called **policytool** is included with most JDK and JRE distributions, for the purpose of creating and editing Java Security Manager security policies. Detailed information about **policytool** is linked from <http://docs.oracle.com/javase/6/docs/technotes/tools/>.

Setup a new Java Security Manager Policy

1. Start **policytool**
Start the **policytool** tool in one of the following ways.
 - a. Red Hat Enterprise Linux
From your GUI or a command prompt, run **/usr/bin/policytool**.
 - b. Microsoft Windows Server
Run **policytool.exe** from your Start menu or from the *bin* of your Java installation. The location can vary.
2. Create a policy.
To create a policy, select **Add Policy Entry**. Add the parameters you need, then click **Done**.
3. Edit an existing policy.
Select the policy from the list of existing policies, and select the **Edit Policy Entry** button. Edit the parameters as needed.
4. Delete an existing policy.
Select the policy from the list of existing policies, and select the **Remove Policy Entry** button.

28.7.4. Run Red Hat JBoss Data Grid Server Within the Java Security Manager

To specify a Java Security Manager policy, you need to edit the Java options passed to the server instance during the bootstrap process. For this reason, you cannot pass the parameters as options to the *standalone.sh* script. The following procedure guides you through the steps of configuring your instance to run within a Java Security Manager policy.

Prerequisites

Before you following this procedure, you need to write a security policy, using the **policytool** command which is included with your Java Development Kit (JDK). This procedure assumes that your policy is located at *JDG_HOME/bin/server.policy*. As an alternative, write the security policy using any text editor and manually save it as *JDG_HOME/bin/server.policy** The JBoss Data Grid server must be completely stopped before you edit any configuration files.

Perform the following procedure for each physical host or instance in your environment.

Configure the Security Manager for JBoss Data Grid Server

1. Open the configuration file.
Open the configuration file for editing. This location of this file is listed below by OS. Note that this is not the executable file used to start the server, but a configuration file that contains runtime parameters. **For Linux:** `JDG_HOME/bin/standalone.conf` For Windows: `JDG_HOME\bin\standalone.conf.bat`
2. Add the Java options to the file.
To ensure the Java options are used, add them to the code block that begins with:

```
if [ "x$JAVA_OPTS" = "x" ]; then
```

You can modify the `-Djava.security.policy` value to specify the exact location of your security policy. It should go onto one line only, with no line break. Using `==` when setting the `-Djava.security.policy` property specifies that the security manager will use *only* the specified policy file. Using `=` specifies that the security manager will use the specified policy *combined with* the policy set in the `policy.url` section of `JAVA_HOME/lib/security/java.security`.



IMPORTANT

JBoss Enterprise Application Platform releases from 6.2.2 onwards require that the system property `jboss.modules.policy-permissions` is set to `true`.

standalone.conf

```
JAVA_OPTS="$JAVA_OPTS -Djava.security.manager -
Djava.security.policy==$PWD/server.policy -
Djboss.home.dir=$JBOSS_HOME -Djboss.modules.policy-permissions=true"
```

standalone.conf.bat

```
set "JAVA_OPTS=%JAVA_OPTS% -Djava.security.manager -
Djava.security.policy==\path\to\server.policy -
Djboss.home.dir=%JBOSS_HOME% -Djboss.modules.policy-
permissions=true"
```

3. Start the server.
Start the server as normal.

28.8. DATA SECURITY FOR REMOTE CLIENT SERVER MODE

28.8.1. About Security Realms

A *security realm* is a series of mappings between users and passwords, and users and roles. Security realms are a mechanism for adding authentication and authorization to your EJB and Web applications. Red Hat JBoss Data Grid Server provides two security realms by default:

- **ManagementRealm** stores authentication information for the Management API, which provides the functionality for the Management CLI and web-based Management Console. It provides an authentication system for managing JBoss Data Grid Server itself. You could also use the

ManagementRealm if your application needed to authenticate with the same business rules you use for the Management API.

- **ApplicationRealm** stores user, password, and role information for Web Applications and EJBs.

Each realm is stored in two files on the filesystem:

- *REALM-users.properties* stores usernames and hashed passwords.
- *REALM-roles.properties* stores user-to-role mappings.
- *mgmt-groups.properties* stores user-to-role mapping file for **ManagementRealm**.

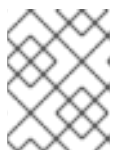
The properties files are stored in the *standalone/configuration/* directories. The files are written simultaneously by the *add-user.sh* or *add-user.bat* command. When you run the command, the first decision you make is which realm to add your new user to.

28.8.2. Add a New Security Realm

1. Run the Management CLI
Start the *cli.sh* or *cli.bat* command and connect to the server.
2. Create the new security realm itself
Run the following command to create a new security realm named **MyDomainRealm** on a domain controller or a standalone server.

```
/host=master/core-service=management/security-  
realm=MyDomainRealm:add()
```

3. Create the reference to the properties file which will store information about the new realm's users
Run the below command to define the location of the new security realm's properties file; this file contains information regarding the users of this security realm. The following command references a file named *myfile.properties* in the **jboss.server.config.dir**.



NOTE

The newly-created properties file is not managed by the included *add-user.sh* and *add-user.bat* scripts. It must be managed externally.

```
/host=master/core-service=management/security-  
realm=MyDomainRealm/authentication=properties:add(path="myfile.prope  
rties",relative-to="jboss.server.config.dir")
```

4. Reload the server
Reload the server so the changes will take effect.

```
:reload
```

Result

The new security realm is created. When you add users and roles to this new realm, the information will be stored in a separate file from the default security realms. You can manage this new file using your own applications or procedures.

28.8.3. Add a User to a Security Realm

1. Run the `add-user.sh` or `add-user.bat` command
Open a terminal and change directories to the `JDG_HOME/bin/` directory. If you run Red Hat Enterprise Linux or another UNIX-like operating system, run `add-user.sh`. If you run Microsoft Windows Server, run `add-user.bat`.
2. Choose whether to add a Management User or Application User
For this procedure, type **b** to add an Application User.
3. Choose the realm this user will be added to
By default, the only available realms are the **ManagementRealm** and **ApplicationRealm**; however, if a custom realm has been added, then its name may be entered instead.
4. Type the username, password, and roles, when prompted
Type the desired username, password, and optional roles when prompted. Verify your choice by typing **yes**, or type **no** to cancel the changes. The changes are written to each of the properties files for the security realm.

28.8.4. Configuring Security Realms Declaratively

In Remote Client-Server mode, a Hot Rod endpoint must specify a security realm.

The security realm declares an **authentication** and an **authorization** section.

Configuring Security Realms Declaratively

```
<security-realms>
  <security-realm name="ManagementRealm">
    <authentication>
      <local default-user="$local" skip-group-
loading="true"/>
      <properties path="mgmt-users.properties" relative-
to="jboss.server.config.dir"/>
    </authentication>
    <authorization map-groups-to-roles="false">
      <properties path="mgmt-groups.properties" relative-
to="jboss.server.config.dir"/>
    </authorization>
  </security-realm>
  <security-realm name="ApplicationRealm">
    <authentication>
      <local default-user="$local" allowed-users="*" skip-
group-loading="true"/>
      <properties path="application-users.properties"
relative-to="jboss.server.config.dir"/>
    </authentication>
    <authorization>
      <properties path="application-roles.properties"
relative-to="jboss.server.config.dir"/>
    </authorization>
  </security-realm>
</security-realms>
```

```

        </authorization>
    </security-realm>
</security-realms>

```

The **server-identities** parameter can also be used to specify certificates.

28.8.5. Loading Roles from LDAP for Authorization (Remote Client-Server Mode)

An LDAP directory contains entries for user accounts and groups, cross referenced by attributes. Depending on the LDAP server configuration, a user entity may map the groups the user belongs to through **memberOf** attributes; a group entity may map which users belong to it through **uniqueMember** attributes; or both mappings may be maintained by the LDAP server.

Users generally authenticate against the server using a simple user name. When searching for group membership information, depending on the directory server in use, searches could be performed using this simple name or using the distinguished name of the user's entry in the directory.

The authentication step of a user connecting to the server always happens first. Once the user is successfully authenticated the server loads the user's groups. The authentication step and the authorization step each require a connection to the LDAP server. The realm optimizes this process by reusing the authentication connection for the group loading step. As will be shown within the configuration steps below it is possible to define rules within the authorization section to convert a user's simple user name to their distinguished name. The result of a "user name to distinguished name mapping" search during authentication is cached and reused during the authorization query when the **force** attribute is set to "false". When **force** is true, the search is performed again during authorization (while loading groups). This is typically done when different servers perform authentication and authorization.

```

<authorization>
  <ldap connection="...">
    <!-- OPTIONAL -->
    <username-to-dn force="true">
      <!-- Only one of the following. -->
      <username-is-dn />
      <username-filter base-dn="..." recursive="..." user-dn-
attribute="..." attribute="..." />
      <advanced-filter base-dn="..." recursive="..." user-dn-
attribute="..." filter="..." />
    </username-to-dn>

    <group-search group-name="..." iterative="..." group-dn-
attribute="..." group-name-attribute="..." >
      <!-- One of the following -->
      <group-to-principal base-dn="..." recursive="..." search-
by="...">
        <membership-filter principal-attribute="..." />
      </group-to-principal>
      <principal-to-group group-attribute="..." />
    </group-search>
  </ldap>
</authorization>

```



IMPORTANT

These examples specify some attributes with their default values. This is done for demonstration. Attributes that specify their default values are removed from the configuration when it is persisted by the server. The exception is the **force** attribute. It is required, even when set to the default value of **false**.

username-to-dn

The **username-to-dn** element specifies how to map the user name to the distinguished name of their entry in the LDAP directory. This element is only required when *both* of the following are true:

- The authentication and authorization steps are against different LDAP servers.
- The group search uses the distinguished name.

1:1 username-to-dn

This specifies that the user name entered by the remote user is the user's distinguished name.

```
<username-to-dn force="false">
  <username-is-dn />
</username-to-dn>
```

+ This defines a 1:1 mapping and there is no additional configuration.

username-filter

The next option is very similar to the simple option described above for the authentication step. A specified attribute is searched for a match against the supplied user name.

```
<username-to-dn force="true">
  <username-filter base-dn="dc=people,dc=harold,dc=example,dc=com"
    recursive="false" attribute="sn" user-dn-attribute="dn" />
</username-to-dn>
```

+ The attributes that can be set here are:

- **base-dn**: The distinguished name of the context to begin the search.
- **recursive**: Whether the search will extend to sub contexts. Defaults to **false**.
- **attribute**: The attribute of the users entry to try and match against the supplied user name. Defaults to **uid**.
- **user-dn-attribute**: The attribute to read to obtain the users distinguished name. Defaults to **dn**.

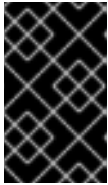
advanced-filter

The final option is to specify an advanced filter, as in the authentication section this is an opportunity to use a custom filter to locate the users distinguished name.

```
<username-to-dn force="true">
  <advanced-filter base-dn="dc=people,dc=harold,dc=example,dc=com"
    recursive="false" filter="sAMAccountName={0}" user-dn-attribute="dn" />
</username-to-dn>
```

+ For the attributes that match those in the *username-filter* example, the meaning and default values are the same. There is one new attribute:

- **filter**: Custom filter used to search for a user's entry where the user name will be substituted in the `{0}` place holder.



IMPORTANT

The XML must remain valid after the filter is defined so if any special characters are used such as `&` ensure the proper form is used. For example `&` for the `&` character.

The Group Search

There are two different styles that can be used when searching for group membership information. The first style is where the user's entry contains an attribute that references the groups the user is a member of. The second style is where the group contains an attribute referencing the users entry.

When there is a choice of which style to use Red Hat recommends that the configuration for a user's entry referencing the group is used. This is because with this method group information can be loaded by reading attributes of known distinguished names without having to perform any searches. The other approach requires extensive searches to identify the groups that reference the user.

Before describing the configuration here are some LDIF examples to illustrate this.

Principal to Group - LDIF example.

This example illustrates where we have a user **TestUserOne** who is a member of **GroupOne**, **GroupOne** is in turn a member of **GroupFive**. The group membership is shown by the use of a **memberOf** attribute which is set to the distinguished name of the group of which the user (or group) is a member.

It is not shown here but a user could potentially have multiple **memberOf** attributes set, one for each group of which the user is directly a member.

```
dn: uid=TestUserOne,ou=users,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
objectClass: organizationalPerson
cn: Test User One
sn: Test User One
uid: TestUserOne
distinguishedName: uid=TestUserOne,ou=users,dc=principal-to-
group,dc=example,dc=org
memberOf: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
memberOf: uid=Slashy/Group,ou=groups,dc=principal-to-
group,dc=example,dc=org
userPassword: :
e1NTSEF9WFpURzhLVjc4WVZBQUJNbEI3Ym96UVAvA0RTNlFNWUpLOTdTMUE9PQ==

dn: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
```

```

objectClass: top
objectClass: groupMember
objectClass: group
objectClass: uidObject
uid: GroupOne
distinguishedName: uid=GroupOne,ou=groups,dc=principal-to-
group,dc=example,dc=org
memberOf: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org

dn: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: group
objectClass: uidObject
uid: GroupFive
distinguishedName: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org

```

Group to Principal - LDIF Example

This example shows the same user **TestUserOne** who is a member of **GroupOne** which is in turn a member of **GroupFive** - however in this case it is an attribute **uniqueMember** from the group to the user being used for the cross reference.

Again the attribute used for the group membership cross reference can be repeated, if you look at *GroupFive* there is also a reference to another user *TestUserFive* which is not shown here.

```

dn: uid=TestUserOne,ou=users,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
objectClass: organizationalPerson
cn: Test User One
sn: Test User One
uid: TestUserOne
userPassword::
e1NTSEF9SjR00TRDR1ltaHc1VVZQ0EJvbXhUYjl1dkFVd1lQTmRLSEdzaWc9PQ==

dn: uid=GroupOne,ou=groups,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group One
uid: GroupOne
uniqueMember: uid=TestUserOne,ou=users,dc=group-to-
principal,dc=example,dc=org

dn: uid=GroupFive,ou=subgroups,ou=groups,dc=group-to-
principal,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject

```



```

cn: Group Five
uid: GroupFive
uniqueMember: uid=TestUserFive,ou=users,dc=group-to-
principal,dc=example,dc=org
uniqueMember: uid=GroupOne,ou=groups,dc=group-to-
principal,dc=example,dc=org

```

General Group Searching

Before looking at the examples for the two approaches shown above we first need to define the attributes common to both of these.

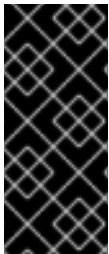
```

<group-search group-name="..." iterative="..." group-dn-attribute="..."
group-name-attribute="..." >
    ...
</group-search>

```

- **group-name**: This attribute is used to specify the form that should be used for the group name returned as the list of groups of which the user is a member. This can either be the simple form of the group name or the group's distinguished name. If the distinguished name is required this attribute can be set to **DISTINGUISHED_NAME**. Defaults to **SIMPLE**.
- **iterative**: This attribute is used to indicate if, after identifying the groups a user is a member of, we should also iteratively search based on the groups to identify which groups the groups are a member of. If iterative searching is enabled we keep going until either we reach a group that is not a member if any other groups or a cycle is detected. Defaults to **false**.

Cyclic group membership is not a problem. A record of each search is kept to prevent groups that have already been searched from being searched again.



IMPORTANT

For iterative searching to work the group entries need to look the same as user entries. The same approach used to identify the groups a user is a member of is then used to identify the groups of which the group is a member. This would not be possible if for group to group membership the name of the attribute used for the cross reference changes or if the direction of the reference changes.

- **group-dn-attribute**: On an entry for a group which attribute is its distinguished name. Defaults to **dn**.
- **group-name-attribute**: On an entry for a group which attribute is its simple name. Defaults to **uid**.

Principal to Group Example Configuration

Based on the example LDIF from above here is an example configuration iteratively loading a user's groups where the attribute used to cross reference is the **memberOf** attribute on the user.

```

<authorization>
  <ldap connection="LocalLdap">
    <username-to-dn>
      <username-filter base-dn="ou=users,dc=principal-to-
group,dc=example,dc=org" recursive="false" attribute="uid" user-dn-
attribute="dn" />
    </username-to-dn>
  </ldap connection>
</authorization>

```

```

        <group-search group-name="SIMPLE" iterative="true" group-dn-
attribute="dn" group-name-attribute="uid">
            <principal-to-group group-attribute="memberOf" />
        </group-search>
    </ldap>
</authorization>

```

The most important aspect of this configuration is that the **principal-to-group** element has been added with a single attribute.

- **group-attribute**: The name of the attribute on the user entry that matches the distinguished name of the group the user is a member of. Defaults to **memberOf**.

Group to Principal Example Configuration

This example shows an iterative search for the group to principal LDIF example shown above.

```

<authorization>
    <ldap connection="LocalLdap">
        <username-to-dn>
            <username-filter base-dn="ou=users,dc=group-to-
principal,dc=example,dc=org" recursive="false" attribute="uid" user-dn-
attribute="dn" />
        </username-to-dn>
        <group-search group-name="SIMPLE" iterative="true" group-dn-
attribute="dn" group-name-attribute="uid">
            <group-to-principal base-dn="ou=groups,dc=group-to-
principal,dc=example,dc=org" recursive="true" search-
by="DISTINGUISHED_NAME">
                <membership-filter principal-attribute="uniqueMember"
/>
            </group-to-principal>
        </group-search>
    </ldap>
</authorization>

```

Here an element **group-to-principal** is added. This element is used to define how searches for groups that reference the user entry will be performed. The following attributes are set:

- **base-dn**: The distinguished name of the context to use to begin the search.
- **recursive**: Whether sub-contexts also be searched. Defaults to **false**.
- **search-by**: The form of the role name used in searches. Valid values are **SIMPLE** and **DISTINGUISHED_NAME**. Defaults to **DISTINGUISHED_NAME**.

Within the *group-to-principal* element there is a *membership-filter* element to define the cross reference.

- **principal-attribute**: The name of the attribute on the group entry that references the user entry. Defaults to **member**.

28.9. SECURING INTERFACES

28.9.1. Hot Rod Interface Security

28.9.1.1. Publish Hot Rod Endpoints as a Public Interface

Red Hat JBoss Data Grid's Hot Rod server operates as a management interface as a default. To extend its operations to a public interface, alter the value of the **interface** parameter in the **socket-binding** element from **management** to **public** as follows:

```
<socket-binding name="hotrod" interface="public" port="11222" />
```

28.9.1.2. Encryption of communication between Hot Rod Server and Hot Rod client

Hot Rod can be encrypted using TLS/SSL, and has the option to require certificate-based client authentication.

Use the following procedure to secure the Hot Rod connector using SSL.

Secure Hot Rod Using SSL/TLS

1. Generate a Keystore

Create a Java Keystore using the `keytool` application distributed with the JDK and add your certificate to it. The certificate can be either self signed, or obtained from a trusted CA depending on your security policy.

2. Place the Keystore in the Configuration Directory

Put the keystore in the `~/JDG_HOME/standalone/configuration` directory with the `standalone-hotrod-ssl.xml` file from the `~/JDG_HOME/docs/examples/configs` directory.

3. Declare an SSL Server Identity

Declare an SSL server identity within a security realm in the management section of the configuration file. The SSL server identity must specify the path to a keystore and its secret key.

```
<server-identities>
  <ssl protocol="...">
    <keystore path="..." relative-to="..." keystore-
password="{VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::ENCRYPTED_VALUE}" />
  </ssl>
  <secret value="..." />
</server-identities>
```

See [Configure Hot Rod Authentication \(X.509\)](#) for details about these parameters.

4. Add the Security Element

Add the security element to the Hot Rod connector as follows:

```
<hotrod-connector socket-binding="hotrod" cache-container="local">
  <encryption ssl="true" security-realm="ApplicationRealm"
require-ssl-client-auth="false" />
</hotrod-connector>
```

- a. Server Authentication of Certificate

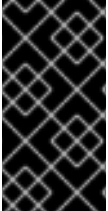
If you require the server to perform authentication of the client certificate, create a truststore that contains the valid client certificates and set the **require-ssl-client-auth** attribute to **true**.

5. Start the Server

Start the server using the following:

```
bin/standalone.sh -c standalone-hotrod-ssl.xml
```

This will start a server with a Hot Rod endpoint on port 11222. This endpoint will only accept SSL connections.



IMPORTANT

To prevent plain text passwords from appearing in configurations or source codes, plain text passwords should be changed to Vault passwords. For more information about how to set up Vault passwords, see the [Password Vault](#) section of the JBoss Enterprise Application Platform security documentation. .

28.9.1.3. Securing Hot Rod to LDAP Server using SSL

When connecting to an LDAP server with SSL enabled it may be necessary to specify a trust store or key store containing the appropriate certificates.

[Encryption of communication between Hot Rod Server and Hot Rod client](#) describes how to set up SSL for Hot Rod client-server communication. This can be used, for example, for secure Hot Rod client authentication with **PLAIN** username/password. When the username/password is checked against credentials in LDAP, a secure connection from the Hot Rod server to the LDAP server is also required. To enable connection from the Hot Rod server to LDAP via SSL, a security realm must be defined as follows:

Hot Rod Client Authentication to LDAP Server

```
<management>
  <security-realms>
    <security-realm name="LdapSSLRealm">
      <authentication>
        <truststore path="ldap.truststore" relative-
to="jboss.server.config.dir" keystore-
password=${VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::ENCRYPTED_VALUE} />
      </authentication>
    </security-realm>
  </security-realms>
  <outbound-connections>
    <ldap name="LocalLdap" url="ldaps://localhost:10389" search-
dn="uid=wildfly,dc=simple,dc=wildfly,dc=org" search-credential="secret"
security-realm="LdapSSLRealm" />
  </outbound-connections>
</management>
```



IMPORTANT

To prevent plain text passwords from appearing in configurations or source codes, plain text passwords should be changed to Vault passwords. For more information about how to set up Vault passwords, see the *Red Hat Enterprise Application Platform Security Guide* .

28.9.1.4. User Authentication over Hot Rod Using SASL

28.9.1.4.1. User Authentication over Hot Rod Using SASL

User authentication over Hot Rod can be implemented using the following Simple Authentication and Security Layer (SASL) mechanisms:

- **PLAIN** is the least secure mechanism because credentials are transported in plain text format. However, it is also the simplest mechanism to implement. This mechanism can be used in conjunction with encryption (**SSL**) for additional security.
- **DIGEST-MD5** is a mechanism that hashes the credentials before transporting them. As a result, it is more secure than the **PLAIN** mechanism.
- **GSSAPI** is a mechanism that uses Kerberos tickets. As a result, it requires a correctly configured Kerberos Domain Controller (for example, Microsoft Active Directory).
- **EXTERNAL** is a mechanism that obtains the required credentials from the underlying transport (for example, from a **X.509** client certificate) and therefore requires client certificate encryption to work correctly.

28.9.1.4.2. Configure Hot Rod Authentication (GSSAPI/Kerberos)

Use the following steps to set up Hot Rod Authentication using the SASL GSSAPI/Kerberos mechanism:

Configure SASL GSSAPI/Kerberos Authentication - Server-side Configuration

1. Define a Kerberos security login module using the security domain subsystem:

```
<system-properties>
  <property name="java.security.krb5.conf"
value="/tmp/infinispan/krb5.conf"/>
  <property name="java.security.krb5.debug" value="true"/>
  <property name="jboss.security.disable.secdomain.option"
value="true"/>
</system-properties>

<security-domain name="infinispan-server" cache-type="default">
  <authentication>
    <login-module code="Kerberos" flag="required">
      <module-option name="debug" value="true"/>
      <module-option name="storeKey" value="true"/>
      <module-option name="refreshKrb5Config" value="true"/>
      <module-option name="useKeyTab" value="true"/>
      <module-option name="doNotPrompt" value="true"/>
      <module-option name="keyTab"
value="/tmp/infinispan/infinispan.keytab"/>
      <module-option name="principal"
value="HOTROD/localhost@INFINISPAN.ORG"/>
    </login-module>
  </authentication>
</security-domain>
```

2. Ensure that the cache-container has authorization roles defined, and these roles are applied in the cache's authorization block as seen in [Configuring Red Hat JBoss Data Grid for Authorization](#).

3. Configure a Hot Rod connector as follows:

```

<hotrod-connector socket-binding="hotrod"
  cache-container="default">
  <authentication security-realm="ApplicationRealm">
    <sasl server-name="node0"
      mechanisms="{mechanism_name}"
      qop="{qop_name}"
      strength="{value}">
    <policy>
      <no-anonymous value="true" />
    </policy>
    <property
name="com.sun.security.sasl.digest.utf8">true</property>
  </sasl>
  </authentication>
</hotrod-connector>

```

- The **server-name** attribute specifies the name that the server declares to incoming clients. The client configuration must also contain the same server name value.
 - The **server-context-name** attribute specifies the name of the login context used to retrieve a server subject for certain SASL mechanisms (for example, GSSAPI).
 - The **mechanisms** attribute specifies the authentication mechanism in use. See [User Authentication over Hot Rod Using SASL](#) for a list of supported mechanisms.
 - The **qop** attribute specifies the SASL quality of protection value for the configuration. Supported values for this attribute are **auth** (authentication), **auth-int** (authentication and integrity, meaning that messages are verified against checksums to detect tampering), and **auth-conf** (authentication, integrity, and confidentiality, meaning that messages are also encrypted). Multiple values can be specified, for example, **auth-int auth-conf**. The ordering implies preference, so the first value which matches both the client and server's preference is chosen.
 - The **strength** attribute specifies the SASL cipher strength. Valid values are **low**, **medium**, and **high**.
 - The **no-anonymous** element within the **policy** element specifies whether mechanisms that accept anonymous login are permitted. Set this value to **false** to permit and **true** to deny.
4. Perform the Client-Side configuration on each client. As the Hot Rod client is configured programmatically information on this configuration is found in the JBoss Data Grid *Developer Guide*.

28.9.1.4.3. Configure Hot Rod Authentication (MD5)

Use the following steps to set up Hot Rod Authentication using the SASL using the MD5 mechanism:

Configure Hot Rod Authentication (MD5)

1. Set up the Hot Rod Connector configuration by adding the **sasl** element to the **authentication** element (for details on the **authentication** element, see [Configuring Security Realms Declaratively](#)) as follows:

```

<hotrod-connector socket-binding="hotrod"
                  cache-container="default">
  <authentication security-realm="ApplicationRealm">
    <sasl server-name="myhotrodserver"
          mechanisms="DIGEST-MD5"
          qop="auth" />
  </authentication>
</hotrod-connector>

```

- The **server-name** attribute specifies the name that the server declares to incoming clients. The client configuration must also contain the same server name value.
 - The **mechanisms** attribute specifies the authentication mechanism in use. See [User Authentication over Hot Rod Using SASL](#) for a list of supported mechanisms.
 - The **qop** attribute specifies the SASL quality of production value for the configuration. Supported values for this attribute are **auth**, **auth-int**, and **auth-conf**.
2. Configure each client to be connected to the Hot Rod connector. As this step is performed programmatically instructions are found in JBoss Data Grid's *Developer Guide* .

28.9.1.4.4. Configure Hot Rod Using LDAP/Active Directory

Use the following to configure authentication over Hot Rod using LDAP or Microsoft Active Directory:

```

<security-realms>
  <security-realm name="ApplicationRealm">
    <authentication>
      <ldap connection="ldap_connection"
            recursive="true"
            base-dn="cn=users,dc=infinispan,dc=org">
        <username-filter attribute="cn" />
      </ldap>
    </authentication>
  </security-realm>
</security-realms>
<outbound-connections>
  <ldap name="ldap_connection"
        url="ldap://my_ldap_server"
        search-dn="CN=test,CN=Users,DC=infinispan,DC=org"
        search-credential="Test_password"/>
</outbound-connections>

```

The following are some details about the elements and parameters used in this configuration:

- The **security-realm** element's **name** parameter specifies the security realm to reference to use when establishing the connection.
- The **authentication** element contains the authentication details.
- The **ldap** element specifies how LDAP searches are used to authenticate a user. First, a connection to LDAP is established and a search is conducted using the supplied user name to identify the distinguished name of the user. A subsequent connection to the server is established using the password supplied by the user. If the second connection succeeds, the authentication is a success.

- The **connection** parameter specifies the name of the connection to use to connect to LDAP.
- The (optional) **recursive** parameter specifies whether the filter is executed recursively. The default value for this parameter is **false**.
- The **base-dn** parameter specifies the distinguished name of the context to use to begin the search from.
- The (optional) **user-dn** parameter specifies which attribute to read for the user's distinguished name after the user is located. The default value for this parameter is **dn**.
- The **outbound-connections** element specifies the name of the connection used to connect to the LDAP directory.
- The **ldap** element specifies the properties of the outgoing LDAP connection.
 - The **name** parameter specifies the unique name used to reference this connection.
 - The **url** parameter specifies the URL used to establish the LDAP connection.
 - The **search-dn** parameter specifies the distinguished name of the user to authenticate and to perform the searches.
 - The **search-credential** parameter specifies the password required to connect to LDAP as the **search-dn**.
 - The (optional) **initial-context-factory** parameter allows the overriding of the initial context factory. the default value of this parameter is **com.sun.jndi.ldap.LdapCtxFactory**.

28.9.1.4.5. Configure Hot Rod Authentication (X.509)

The **X.509** certificate can be installed at the node, and be made available to other nodes for authentication purposes for inbound and outbound SSL connections. This is enabled using the `<server-identities/>` element of a security realm definition, which defines how a server appears to external applications. This element can be used to configure a password to be used when establishing a remote connection, as well as the loading of an **X.509** key.

The following example shows how to install an **X.509** certificate on the node.

```
<security-realm name="ApplicationRealm">
  <server-identities>
    <ssl protocol="...">
      <keystore path="..." relative-to="..." keystore-password="..."
alias="..." key-password="..." />
    </ssl>
  </server-identities>

  [... authentication/authorization ...]
</security-realms>
```

In the provided example, the SSL element contains the `<keystore/>` element, which is used to define how to load the key from the file-based keystore. The following parameters are available for this element.

Table 28.4. <server-identities/> Options

Parameter	Mandatory/Optional	Description
path	Mandatory	This is the path to the keystore, this can be an absolute path or relative to the next attribute.
relative-to	Optional	The name of a service representing a path the keystore is relative to.
keystore-password	Mandatory	The password required to open the keystore.
alias	Optional	The alias of the entry to use from the keystore - for a keystore with multiple entries in practice the first usable entry is used but this should not be relied on and the alias should be set to guarantee which entry is used.
key-password	Optional	The password to load the key entry, if omitted the keystore-password will be used instead.

**NOTE**

If the following error occurs, specify a **key-password** as well as an **alias** to ensure only one key is loaded.

```
UnrecoverableKeyException: Cannot recover key
```

28.9.2. REST Interface Security

28.9.2.1. Publish REST Endpoints as a Public Interface

Red Hat JBoss Data Grid's REST server operates as a management interface by default. To extend its operations to a public interface, alter the value of the **interface** parameter in the **socket-binding** element from **management** to **public** as follows:

```
<socket-binding name="http"
  interface="public"
  port="8080"/>
```

28.9.2.2. Enable Security for the REST Endpoint

Use the following procedure to enable security for the REST endpoint in Red Hat JBoss Data Grid.

**NOTE**

The REST endpoint supports any of the JBoss Enterprise Application Platform security subsystem providers.

Enable Security for the REST Endpoint

To enable security for JBoss Data Grid when using the REST interface, make the following changes to *standalone.xml*:

1. Ensure that the rest endpoint defines a valid configuration for the **authentication** attribute. An example configuration is below:

```
<subsystem xmlns="urn:infinispan:server:endpoint:8.1">
  <rest-connector socket-binding="rest" cache-
container="security">
    <authentication security-realm="ApplicationRealm" auth-
method="BASIC"/>
  </rest-connector>
</subsystem>
```

2. Check Security Domain Declaration

Ensure that the security subsystem contains the corresponding security-domain declaration. For details about setting up security-domain declarations, see the JBoss Enterprise Application Platform 7 documentation.

3. Add an Application User

Run the relevant script and enter the configuration settings to add an application user.

- a. Run the *adduser.sh* script (located in *\$JDG_HOME/bin*).
 - i. On a Windows system, run the *adduser.bat* file (located in *\$JDG_HOME/bin*) instead.
- b. When prompted about the type of user to add, select **Application User (application-users.properties)** by entering **b**.
- c. Accept the default value for realm (**ApplicationRealm**) by pressing the return key.
- d. Specify a username and password.
- e. When prompted for a group, enter **REST**.
- f. Ensure the username and application realm information is correct when prompted and enter "yes" to continue.

4. Verify the Created Application User

Ensure that the created application user is correctly configured.

- a. Check the configuration listed in the *application-users.properties* file (located in *\$JDG_HOME/standalone/configuration/*). The following is an example of what the correct configuration looks like in this file:

```
user1=2dc3eacfed8cf95a4a31159167b936fc
```

- b. Check the configuration listed in the *application-roles.properties* file (located in *\$JDG_HOME/standalone/configuration/*). The following is an example of what the correct configuration looks like in this file:

```
user1=REST
```

5. Test the Server

Start the server and enter the following link in a browser window to access the REST endpoint:

```
http://localhost:8080/rest/namedCache
```



NOTE

If testing using a GET request, a **405** response code is expected and indicates that the server was successfully authenticated.

28.9.3. Memcached Interface Security

28.9.3.1. Publish Memcached Endpoints as a Public Interface

Red Hat JBoss Data Grid's memcached server operates as a management interface by default. It is possible to extend the memcached operations to a public interface, but there is no additional security available for this interface. If security is a concern then it is recommended to keep this interface on an isolated, internal network, or to use either the REST or Hot Rod interfaces.

To configure the memcached interface as a public interface, alter the value of the **interface** parameter in the **socket-binding** element from **management** to **public** as follows:

```
<socket-binding name="memcached"
  interface="public"
  port="11211" />
```

28.10. ACTIVE DIRECTORY AUTHENTICATION (NON-KERBEROS)

See [Example of LDAP Login Module Configuration](#) for a non-Kerberos Active Directory Authentication configuration example.

28.11. ACTIVE DIRECTORY AUTHENTICATION USING KERBEROS (GSSAPI)

When using Red Hat JBoss Data Grid with Microsoft Active Directory, data security can be enabled via Kerberos authentication. To configure Kerberos authentication for Microsoft Active Directory, use the following procedure.

Configure Kerberos Authentication for Active Directory (Library Mode)

1. Configure JBoss EAP server to authenticate itself to Kerberos. This can be done by configuring a dedicated security domain, for example:

```
<security-domain name="ldap-service" cache-type="default">
  <authentication>
```

```

        <login-module code="Kerberos" flag="required">
            <module-option name="storeKey" value="true"/>
            <module-option name="useKeyTab" value="true"/>
            <module-option name="refreshKrb5Config" value="true"/>
            <module-option name="principal"
value="ldap/localhost@INFINISPAN.ORG"/>
            <module-option name="keyTab"
value="${basedir}/keytab/ldap.keytab"/>
            <module-option name="doNotPrompt" value="true"/>
        </login-module>
    </authentication>
</security-domain>

```

- The security domain for authentication must be configured correctly for JBoss EAP, an application must have a valid Kerberos ticket. To initiate the Kerberos ticket, you must reference another security domain using

```

<module-option name="usernamePasswordDomain" value="krb-admin"/>

```

This points to the standard Kerberos login module described in Step 3.

```

<security-domain name="ispn-admin" cache-type="default">
    <authentication>
        <login-module code="SPNEGO" flag="requisite">
            <module-option name="password-stacking"
value="useFirstPass"/>
            <module-option name="serverSecurityDomain" value="ldap-
service"/>
            <module-option name="usernamePasswordDomain"
value="krb-admin"/>
        </login-module>
        <login-module code="AdvancedAdLdap" flag="required">
            <module-option name="password-stacking"
value="useFirstPass"/>
            <module-option name="bindAuthentication"
value="GSSAPI"/>
            <module-option name="jaasSecurityDomain" value="ldap-
service"/>
            <module-option name="java.naming.provider.url"
value="ldap://localhost:389"/>
            <module-option name="baseCtxDN"
value="ou=People,dc=infinispan,dc=org"/>
            <module-option name="baseFilter" value="
(krb5PrincipalName={0})"/>
            <module-option name="rolesCtxDN"
value="ou=Roles,dc=infinispan,dc=org"/>
            <module-option name="roleFilter" value="(member={1})"/>
            <module-option name="roleAttributeID" value="cn"/>
        </login-module>
    </authentication>
</security-domain>

```

- The security domain authentication configuration described in the previous step points to the following standard Kerberos login module:

```

<security-domain name="krb-admin" cache-type="default">
  <authentication>
    <login-module code="Kerberos" flag="required">
      <module-option name="useKeyTab" value="true"/>
      <module-option name="principal"
value="admin@INFINISPAN.ORG"/>
      <module-option name="keyTab"
value="${basedir}/keytab/admin.keytab"/>
    </login-module>
  </authentication>
</security-domain>

```

28.12. THE SECURITY AUDIT LOGGER

28.12.1. The Security Audit Logger

Red Hat JBoss Data Grid includes a logger to audit security logs for the cache, specifically whether a cache or a cache manager operation was allowed or denied for various operations.

The default audit logger is `org.infinispan.security.impl.DefaultAuditLogger`. This logger outputs audit logs using the available logging framework (for example, JBoss Logging) and provides results at the **TRACE** level and the **AUDIT** category.

To send the **AUDIT** category to either a log file, a JMS queue, or a database, use the appropriate log appender.

28.12.2. Configure the Security Audit Logger (Library Mode)

Use the following to configure the audit logger in Red Hat JBoss Data Grid:

```

<infinispan>
  ...
  <global-security>
    <authorization audit-logger =
"org.infinispan.security.impl.DefaultAuditLogger">
      ...
    </authorization>
  </global-security>
  ...
</infinispan>

```

28.12.3. Configure the Security Audit Logger (Remote Client-Server Mode)

Use the following code to configure the audit logger in Red Hat JBoss Data Grid Remote Client-Server Mode.

To use a different audit logger, specify it in the **authorization** element. The **authorization** element must be within the **cache-container** element in the Infinispan subsystem (in the *standalone.xml* configuration file).

```

<cache-container name="local" default-cache="default">
  <security>
    <authorization audit-

```

```

logger="org.infinispan.security.impl.DefaultAuditLogger">
  <identity-role-mapper/>
  <role name="admin" permissions="ALL"/>
  <role name="reader" permissions="READ"/>
  <role name="writer" permissions="WRITE"/>
  <role name="supervisor" permissions="ALL_READ ALL_WRITE"/>
</authorization>
</security>
<local-cache name="default">
  <locking isolation="NONE" acquire-timeout="30000" concurrency-
level="1000" striping="false"/>
  <transaction mode="NONE"/>
  <security>
    <authorization roles="admin reader writer supervisor"/>
  </security>
</local-cache>
[...]
```



NOTE

The default audit logger for server mode is **org.jboss.as.clustering.infinispan.subsystem.ServerAuditLogger** which sends the log messages to the server audit log. See the *Management Interface Audit Logging* chapter in the JBoss Enterprise Application Platform *Administration and Configuration Guide* for more information.

28.12.4. Custom Audit Loggers

Users can implement custom audit loggers in Red Hat JBoss Data Grid Library and Remote Client-Server Mode. The custom logger must implement the **org.infinispan.security.AuditLogger** interface. If no custom logger is provided, the default logger (**DefaultAuditLogger**) is used.

CHAPTER 29. SECURITY FOR CLUSTER TRAFFIC

29.1. NODE AUTHENTICATION AND AUTHORIZATION (REMOTE CLIENT-SERVER MODE)

29.1.1. Node Authentication and Authorization (Remote Client-Server Mode)

Security can be enabled at node level via SASL protocol, which enables node authentication against a security realm. This requires nodes to authenticate each other when joining or merging with a cluster. For detailed information about security realms, see [About Security Realms](#).

The following example depicts the `<sasl />` element, which leverages the SASL protocol. Both **DIGEST-MD5** or **GSSAPI** mechanisms are currently supported.

Configure SASL Authentication

```
<management>
  <security-realms>
    <!-- Additional configuration information here -->
    <security-realm name="ClusterRealm">
      <authentication>
        <properties path="cluster-users.properties" relative-
to="jboss.server.config.dir"/>
      </authentication>
      <authorization>
        <properties path="cluster-roles.properties" relative-
to="jboss.server.config.dir"/>
      </authorization>
    </security-realm>
  </security-realms>
  <!-- Additional configuration information here -->
</management>

<stack name="udp">
  <!-- Additional configuration information here -->
  <sasl mech="DIGEST-MD5" security-realm="ClusterRealm" cluster-
role="cluster">
    <property name="client_name">node1</property>
    <property name="client_password">password</property>
  </sasl>
  <!-- Additional configuration information here -->
</stack>
```

In the provided example, the nodes use the **DIGEST-MD5** mechanism to authenticate against the **ClusterRealm**. In order to join, nodes must have the **cluster** role.

The **cluster-role** attribute determines the role all nodes must belong to in the security realm in order to **JOIN** or **MERGE** with the cluster. Unless it has been specified, the **cluster-role** attribute is the name of the clustered **<cache-container>** by default. Each node identifies itself using the **client-name** property. If none is specified, the hostname on which the server is running will be used.

This name can also be overridden by specifying the `jboss.node.name` system property that can be overridden on the command line. For example:

```
$ standalone.sh -Djboss.node.name=node001
```



NOTE

JGroups AUTH protocol is not integrated with security realms, and its use is not advocated for Red Hat JBoss Data Grid.

29.1.2. Configure Node Authentication for Cluster Security (DIGEST-MD5)

The following example demonstrates how to use **DIGEST-MD5** with a properties-based security realm, with a dedicated realm for cluster node.

Using the DIGEST-MD5 Mechanism

```
<management>
  <security-realms>
    <security-realm name="ClusterRealm">
      <authentication>
        <properties path="cluster-users.properties"
relative-to="jboss.server.config.dir"/>
      </authentication>
      <authorization>
        <properties path="cluster-roles.properties"
relative-to="jboss.server.config.dir"/>
      </authorization>
    </security-realm>
  </security-realms>
</management>
<subsystem xmlns="urn:infinispan:server:jgroups:8.0" default-
stack="${jboss.default.jgroups.stack:udp}">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp"/>
    <protocol type="PING"/>
    <protocol type="MERGE2"/>
    <protocol type="FD SOCK" socket-binding="jgroups-udp-fd"/>
    <protocol type="FD_ALL"/>
    <protocol type="pbcast.NAKACK"/>
    <protocol type="UNICAST2"/>
    <protocol type="pbcast.STABLE"/>
    <protocol type="pbcast.GMS"/>
    <protocol type="UFC"/>
    <protocol type="MFC"/>
    <protocol type="FRAG3"/>
    <protocol type="RSVP"/>
    <sasl security-realm="ClusterRealm" mech="DIGEST-MD5">
      <property name="client_password">...</property>
    </sasl>
  </stack>
</subsystem>
<subsystem xmlns="urn:infinispan:server:core:8.4" default-cache-
container="clustered">
  <cache-container name="clustered" default-cache="default">
```



```

        <transport executor="infinispan-transport" lock-timeout="60000"
stack="udp"/>
        <!-- various clustered cache definitions here -->
    </cache-container>
</subsystem>

```

In the provided example, supposing the hostnames of the various nodes are **node001**, **node002**, **node003**, the **cluster-users.properties** will contain:

- **node001**=/**<node001passwordhash>**/
- **node002**=/**<node002passwordhash>**/
- **node003**=/**<node003passwordhash>**/

The **cluster-roles.properties** will contain:

- node001=clustered
- node002=clustered
- node003=clustered

To generate these values, the following **add-users.sh** script can be used:

```

$ add-user.sh -up cluster-users.properties -gp cluster-roles.properties -r
ClusterRealm -u node001 -g clustered -p <password>

```

The **MD5** password hash of the node must also be placed in the "**client_password**" property of the **<sasl/>** element.

```

<property name="client_password">...</property>

```



NOTE

To increase security, it is recommended that this password be stored using a Vault. For more information about vault expressions, see the *Red Hat Enterprise Application Platform Security Guide*

Once node security has been set up as discussed here, the cluster coordinator will validate each **JOINing** and **MERGEing** node's credentials against the realm before letting the node become part of the cluster view.

29.1.3. Configure Node Authentication for Cluster Security (GSSAPI/Kerberos)

When using the **GSSAPI** mechanism, the **client_name** is used as the name of a Kerberos-enabled login module defined within the security domain subsystem. For a full procedure on how to do this, see [Configure Hot Rod Authentication \(GSSAPI/Kerberos\)](#).

Using the Kerberos Login Module

```

<security-domain name="krb-node0" cache-type="default">
  <authentication>

```

```

        <login-module code="Kerberos" flag="required">
            <module-option name="storeKey" value="true"/>
            <module-option name="useKeyTab" value="true"/>
            <module-option name="refreshKrb5Config" value="true"/>
            <module-option name="principal"
value="jgroups/node0/clustered@INFINISPAN.ORG"/>
            <module-option name="keyTab"
value="{jboss.server.config.dir}/keytabs/jgroups_node0_clustered.keytab"/
>
            <module-option name="doNotPrompt" value="true"/>
        </login-module>
    </authentication>
</security-domain>

```

The following property must be set in the `<sasl/>` element to reference it:

```

<sasl <!-- Additional configuration information here --> >
    <property name="login_module_name">
        <!-- Additional configuration information here -->
    </property>
</sasl>

```

As a result, the **authentication** section of the security realm is ignored, as the nodes will be validated against the Kerberos Domain Controller. The **authorization** configuration is still required, as the node principal must belong to the required cluster-role.

In all cases, it is recommended that a shared authorization database, such as LDAP, be used to validate node membership in order to simplify administration.

By default, the principal of the joining node must be in the following format:

```
jgroups/$NODE_NAME/$CACHE_CONTAINER_NAME@REALM
```

29.2. CONFIGURE NODE SECURITY IN LIBRARY MODE

29.2.1. Configure Node Security in Library Mode

In Library mode, node authentication is configured directly in the JGroups configuration. JGroups can be configured so that nodes must authenticate each other when joining or merging with a cluster. The authentication uses SASL and is enabled by adding the **SASL** protocol to your JGroups XML configuration.

SASL relies on JAAS notions, such as **CallbackHandlers**, to obtain certain information necessary for the authentication handshake. Users must supply their own **CallbackHandlers** on both client and server sides.



IMPORTANT

The **JAAS** API is only available when configuring user authentication and authorization, and is not available for node security.

**NOTE**

In the provided example, **CallbackHandler** classes are examples only, and not contained in the Red Hat JBoss Data Grid release. Users must provide the appropriate **CallbackHandler** classes for their specific LDAP implementation.

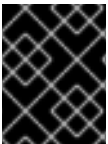
Setting Up SASL Authentication in JGroups

```
<SASL mech="DIGEST-MD5"
  client_name="node_user"
  client_password="node_password"

  server_callback_handler_class="org.example.infinispan.security.JGroupsSasl
  ServerCallbackHandler"

  client_callback_handler_class="org.example.infinispan.security.JGroupsSasl
  ClientCallbackHandler"
  sasl_props="com.sun.security.sasl.digest.realm=test_realm" />
```

The above example uses the **DIGEST-MD5** mechanism. Each node must declare the user and password it will use when joining the cluster.

**IMPORTANT**

The SASL protocol must be placed before the GMS protocol in order for authentication to take effect.

29.2.2. Simple Authorizing Callback Handler

For instances where a more complex Kerberos or LDAP approach is not needed the *SimpleAuthorizingCallbackHandler* class may be used. To enable this set both the **server_callback_handler** and the **client_callback_handler** to *org.jgroups.auth.sasl.SimpleAuthorizingCallbackHandler*, as seen in the below example:

```
<SASL mech="DIGEST-MD5"
  client_name="node_user"
  client_password="node_password"

  server_callback_handler_class="org.jgroups.auth.sasl.SimpleAuthorizingCall
  backHandler"

  client_callback_handler_class="org.jgroups.auth.sasl.SimpleAuthorizingCall
  backHandler"
  sasl_props="com.sun.security.sasl.digest.realm=test_realm" />
```

The **SimpleAuthorizingCallbackHandler** may be configured either programmatically, by passing the constructor an instance of *java.util.Properties*, or via standard Java system properties, set on the command line using the **-DpropertyName=propertyValue** notation. The following properties are available:

- **sasl.credentials.properties** - the path to a property file which contains principal/credential mappings represented as `principal=password`.

- **sasl.local.principal** - the name of the principal that is used to identify the local node. It must exist in the `sasl.credentials.properties` file.
- **sasl.roles.properties** - (optional) the path to a property file which contains principal/roles mappings represented as `principal=role1,role2,role3`.
- **sasl.role** - (optional) if present, authorizes joining nodes only if their principal is.
- **sasl.realm** - (optional) the name of the realm to use for the SASL mechanisms that require it

29.2.3. Configure Node Authentication for Library Mode (DIGEST-MD5)

The behavior of a node differs depending on whether it is the coordinator node or any other node. The coordinator acts as the SASL server, with the joining or merging nodes behaving as SASL clients. When using the DIGEST-MD5 mechanism in Library mode, the server and client callback must be specified so that the server and client are aware of how to obtain the credentials. Therefore, two **CallbackHandlers** are required:

- The **server_callback_handler_class** is used by the coordinator.
- The **client_callback_handler_class** is used by other nodes.

The following example demonstrates these **CallbackHandlers**.

Callback Handlers

```
<SASL mech="DIGEST-MD5"
  client_name="node_name"
  client_password="node_password"

  client_callback_handler_class="${CLIENT_CALLBACK_HANDLER_IN_CLASSPATH}"

  server_callback_handler_class="${SERVER_CALLBACK_HANDLER_IN_CLASSPATH}"
  sasl_props="com.sun.security.sasl.digest.realm=test_realm"
/>
```

JGroups is designed so that all nodes are able to act as coordinator or client depending on cluster behavior, so if the current coordinator node goes down, the next node in the succession chain will become the coordinator. Given this behavior, both server and client callback handlers must be identified within SASL for Red Hat JBoss Data Grid implementations.

29.2.4. Configure Node Authentication for Library Mode (GSSAPI)

When performing node authentication in Library mode using the GSSAPI mechanism, the **login_module_name** parameter must be specified instead of **callback**.

This login module is used to obtain a valid Kerberos ticket, which is used to authenticate a client to the server. The **server_name** must also be specified, as the client principal is constructed as `jgroups/$server_name@REALM`.

Specifying the login module and server on the coordinator node

```
<SASL mech="GSSAPI"
  server_name="node0/clustered"
  login_module_name="krb-node0"
```

```
server_callback_handler_class="org.infinispan.test.integration.security.ut
ils.SaslPropCallbackHandler" />
```

On the coordinator node, the `server_callback_handler_class` must be specified for node authorization. This will determine if the authenticated joining node has permission to join the cluster.



NOTE

The server principal is always constructed as `jgroups/server_name`, therefore the server principal in Kerberos must also be `jgroups/server_name`. For example, if the server name in Kerberos is `jgroups/node1/mycache`, then the server name must be `node1/mycache`.

29.3. JGROUPS ENCRYPTION

29.3.1. JGroups Encryption

JGroups includes the `SYM_ENCRYPT` and `ASYM_ENCRYPT` protocols to provide encryption for cluster traffic.



IMPORTANT

The `ENCRYPT` protocol has been deprecated and should not be used in production environments. It is recommended to use either `SYM_ENCRYPT` or `ASYM_ENCRYPT`

By default, both of these protocols only encrypt the message body; they do not encrypt message headers. To encrypt the entire message, including all headers, as well as destination and source addresses, the property `encrypt_entire_message` must be `true`. When defining these protocols they should be placed directly under `NAKACK2`.

Both protocols may be used to encrypt and decrypt communication in JGroups, and are used in the following ways:

- **SYM_ENCRYPT**: Configured with a secret key in a keystore using the **JCEKS** store type.
- **ASYM_ENCRYPT**: Configured with algorithms and key sizes. In this scenario the secret key is not retrieved from the keystore, but instead generated by the coordinator and distributed to new members. Once a member joins the cluster they send a request for the secret key to the coordinator; the coordinator responds with the secret key back to the new member encrypted with the member's public key.

Each message is identified as encrypted with a specific encryption header identifying the encrypt header and an MD5 digest identifying the version of the key being used to encrypt and decrypt messages.

29.3.2. Configuring JGroups Encryption Protocols

JGroups encryption protocols are placed in the JGroups configuration file, and there are three methods of including this file depending on how JBoss Data Grid is in use:

- Standard Java properties can also be used in the configuration, and it is possible to pass the path to JGroups configuration via the `-D` option during start up.

- The default, pre-configured JGroups files are packaged in *infinispan-embedded.jar*, alternatively, you can create your own configuration file. See [Configure JGroups \(Library Mode\)](#) for instructions on how to set up JBoss Data Grid to use custom JGroups configurations in library mode.
- In Remote Client-Server mode, the JGroups configuration is part of the main server configuration file.

When defining both the **SYM_ENCRYPT** and **ASYM_ENCRYPT** protocols, place them directly under **NAKACK2** in the configuration file.

29.3.3. SYM_ENCRYPT: Using a Key Store

SYM_ENCRYPT uses store type JCEKS. To generate a keystore compatible with JCEKS, use the following command line options to keytool:

```
$ keytool -genseckey -alias myKey -keypass changeit -storepass changeit -keyalg Blowfish -keysize 56 -keystore defaultStore.keystore -storetype JCEKS
```

SYM_ENCRYPT can then be configured by adding the following information to the JGroups file used by the application.

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



NOTE

The `defaultStore.keystore` must be found in the classpath.

29.3.4. ASYM_ENCRYPT: Configured with Algorithms and Key Sizes

In this encryption mode, the coordinator selects the secretKey and distributes it to all peers. There is no keystore, and keys are distributed using a public/private key exchange. Instead, encryption occurs as follows:

1. The secret key is generated and distributed by the coordinator.
2. When a view change occurs, a peer requests the secret key by sending a key request with its own public key.
3. The coordinator encrypts the secret key with the public key, and sends it back to the peer.
4. The peer then decrypts and installs the key as its own secret key.
5. Any further communications are encrypted and decrypted using the secret key.

ASYM_ENCRYPT Example

```
...
<VERIFY_SUSPECT/>
```

```

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

<pbcast.NAKACK2/>
<UNICAST3/>
<pbcast.STABLE/>
<FRAG3/>
<AUTH auth_class="org.jgroups.auth.MD5Token"
    auth_value="chris"
    token_hash="MD5"/>
<pbcast.GMS join_timeout="2000" />

```

In the provided example, **ASYM_ENCRYPT** has been placed immediately below **NAKACK2**, and **encrypt_entire_message** has been enabled, indicating that the message headers will be encrypted along with the message body. This means that the **NAKACK2** and **UNICAST3** protocols are also encrypted. In addition, **AUTH** has been included as part of the configuration, so that only authenticated nodes may request the secret key from the coordinator.

View changes that identify a new controller result in a new secret key being generated and distributed to all peers. This is a substantial overhead in an application with high peer churn. A new secret key may optionally be generated when a cluster member leaves by setting **change_key_on_leave** to true.

When encrypting an entire message, the message must be marshalled into a byte buffer before being encrypted, resulting in decreased performance.

29.3.5. JGroups Encryption Configuration Parameters

The following table provides configuration parameters for the **ENCRYPT** JGroups protocol, which both **SYM_ENCRYPT** and **ASYM_ENCRYPT** extend:

Table 29.1. ENCRYPT Configuration Parameters

Name	Description
asym_algorithm	Cipher engine transformation for asymmetric algorithm. Default is RSA.
asym_keylength	Initial public/private key length. Default is 512.
asym_provider	Cryptographic Service Provider. Default is Bouncy Castle Provider.
encrypt_entire_message	By default only the message body is encrypted. Enabling encrypt_entire_message ensures that all headers, destination and source addresses, and the message body is encrypted.
sym_algorithm	Cipher engine transformation for symmetric algorithm. Default is AES.

Name	Description
sym_keylength	Initial key length for matching symmetric algorithm. Default is 128.
sym_provider	Cryptographic Service Provider. Default is Bouncy Castle Provider.

The following table provides a list of the **SYM_ENCRYPT** protocol parameters

Table 29.2. SYM_ENCRYPT Configuration Parameters

Name	Description
alias	Alias used for recovering the key. Change the default.
key_password	Password for recovering the key. Change the default.
keystore_name	File on classpath that contains keystore repository.
store_password	Password used to check the integrity/unlock the keystore. Change the default.

The following table provides a list of the **ASYM_ENCRYPT** protocol parameters

Table 29.3. ASYM_ENCRYPT Configuration Parameters

Name	Description
change_key_on_leave	When a member leaves the view, change the secret key, preventing old members from eavesdropping.

PART XIV. COMMAND LINE TOOLS

CHAPTER 30. INTRODUCTION

30.1. COMMAND LINE TOOLS

Red Hat JBoss Data Grid includes two command line tools for interacting with the caches in the data grid:

- The JBoss Data Grid Library CLI. For more information, see [Red Hat JBoss Data Grid Library Mode CLI](#).
- The JBoss Data Grid Server CLI. For more information, see [Red Hat JBoss Data Grid Server CLI](#).

CHAPTER 31. RED HAT JBOSS DATA GRID CLIS

31.1. JBOSS DATA GRID CLIS

Red Hat JBoss Data Grid includes two Command Line Interfaces: a Library Mode CLI (see [Red Hat JBoss Data Grid Library Mode CLI](#) for details) and a Server Mode CLI (see [Red Hat JBoss Data Grid Server CLI](#) for details).

31.2. RED HAT JBOSS DATA GRID LIBRARY MODE CLI

31.2.1. Red Hat JBoss Data Grid Library Mode CLI

Red Hat JBoss Data Grid includes the Red Hat JBoss Data Grid Library Mode Command Line Interface (CLI) that is used to inspect and modify data within caches and internal components (such as transactions, cross-datacenter replication sites, and rolling upgrades). The JBoss Data Grid Library Mode CLI can also be used for more advanced operations such as transactions.

31.2.2. Start the Library Mode CLI (Server)

Start the Red Hat JBoss Data Grid CLI's server-side module with the *standalone* and *domain* files. For Linux, use the *standalone.sh* or *domain.sh* script and for Windows, use the *standalone.bat* or *domain.bat* file.

31.2.3. Start the Library Mode CLI (Client)

Start the Red Hat JBoss Data Grid CLI client using the *cli* files in the *bin* directory. For Linux, run *bin/cli.sh* and for Windows, run *bin\cli.bat*.

When starting up the CLI client, specific command line switches can be used to customize the start up.

31.2.4. CLI Client Switches for the Command Line

The listed command line switches are appended to the command line when starting the Red Hat JBoss Data Grid CLI command:

Table 31.1. CLI Client Command Line Switches

Short Option	Long Option	Description
-c	--connect=\${URL}	Connects to a running Red Hat JBoss Data Grid instance. For example, for JMX over RMI use jmx://[username[:password]]@host:port[/container[/cache]] and for JMX over JBoss Remoting use remoting://[username[:password]]@host:port[/container[/cache]]

Short Option	Long Option	Description
-f	--file=\${FILE}	Read the input from the specified file rather than using interactive mode. If the value is set to - then the stdin is used as the input.
-h	--help	Displays the help information.
-v	--version	Displays the CLI version information.

31.2.5. Connect to the Application

Use the following command to connect to the application using the CLI:

```
[disconnected//]> connect 127.0.0.1:9990
[standalone@127.0.0.1:9990/>
```



NOTE

The port value **9990** depends on the value the JVM is started with. This port may be changed by starting the JVM with the - **Dcom.sun.management.jmxremote.port=\$PORT_NUMBER** command line parameter. When the remoting protocol (**remoting://localhost:9990**) is used, the Red Hat JBoss Data Grid server administration port is used (the default is port **9990**).

The command line prompt displays the active connection information, along with the current directory.

Use the **container** command to select a cache manager, and then select a cache with the **cache** command. A cache must be selected before performing cache operations. The CLI supports tab completion, therefore using the **cache** and pressing the tab button displays a list of active caches. The following example assumes a cache manager, **MyCacheManager**, defined in the configuration.

```
[standalone@127.0.0.1:9990/> container MyCacheManager
[standalone@127.0.0.1:9990/MyCacheManager/> cache
default namedCache
[standalone@127.0.0.1:9990/MyCacheManager/]> cache default
[standalone@127.0.0.1:9990/MyCacheManager/default]>
```

Additionally, pressing tab displays a list of valid commands for the CLI.

31.3. RED HAT JBOSS DATA GRID SERVER CLI

31.3.1. Red Hat Data Grid Server Mode CLI

Red Hat JBoss Data Grid includes a new Remote Client-Server mode CLI. This CLI can only be used for specific use cases, such as manipulating the server subsystem for the following:

- configuration
- management
- obtaining metrics

31.3.2. Start the Server Mode CLI

Use the following commands to run the JBoss Data Grid Server CLI from the command line:

For Linux:

```
$ JDG_HOME/bin/cli.sh
```

For Windows:

```
C:\>JDG_HOME\bin\cli.bat
```

31.4. CLI COMMANDS

31.4.1. CLI Commands

Unless specified otherwise, all listed commands for the JBoss Data Grid CLIs can be used with both the Library Mode and Server Mode CLIs. Specifically, the **deny** (see [The deny Command](#)), **grant** (see [The grant Command](#)), and **roles** (see [The roles command](#)) commands are only available on the Server Mode CLI.

31.4.2. The abort Command

The **abort** command aborts a running batch initiated using the **start** command. Batching must be enabled for the specified cache. The following is a usage example:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> start
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> abort
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> get a
null
```

31.4.3. The begin Command

The **begin** command starts a transaction. This command requires transactions enabled for the cache it targets. An example of this command's usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> begin
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put b b
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> commit
```

31.4.4. The cache Command

The **cache** command specifies the default cache used for all subsequent operations. If invoked without any parameters, it shows the currently selected cache. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> cache default
[standalone@127.0.0.1:9990/MyCacheManager/default]> cache
default
[standalone@127.0.0.1:9990/MyCacheManager/default]>
```

31.4.5. The clearcache Command

The **clearcache** command clears all content from the cache. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> clearcache
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> get a
null
```

31.4.6. The commit Command

The **commit** command commits changes to an ongoing transaction. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> begin
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put b b
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> commit
```

31.4.7. The container Command

The **container** command selects the default cache container (cache manager). When invoked without any parameters, it lists all available containers. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> container
MyCacheManager OtherCacheManager
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> container
OtherCacheManager
[standalone@127.0.0.1:9990/OtherCacheManager/]>
```

31.4.8. The create Command

The **create** command creates a new cache based on the configuration of an existing cache definition. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> create newCache
like namedCache
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> cache newCache
[standalone@127.0.0.1:9990/MyCacheManager/newCache]>
```

31.4.9. The deny Command

When authorization is enabled and the role mapper has been configured to be the ClusterRoleMapper, principal to role mappings are stored within the cluster registry (a replicated cache available to all nodes). The **deny** command can be used to deny roles previously assigned to a principal:

```
[standalone@127.0.0.1:9990]> deny supervisor to user1
```



NOTE

The **deny** command is only available to the JBoss Data Grid Server Mode CLI.

31.4.10. The disconnect Command

The **disconnect** command disconnects the currently active connection, which allows the CLI to connect to another instance. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> disconnect
[disconnected//]
```

31.4.11. The encoding Command

The **encoding** command sets a default codec to use when reading and writing entries to and from a cache. If invoked with no arguments, the currently selected codec is displayed. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> encoding
none
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> encoding --list
memcached
hotrod
none
rest
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> encoding hotrod
```

31.4.12. The end Command

The **end** command ends a running batch initiated using the **start** command. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> start
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> end
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> get a
a
```

31.4.13. The evict Command

The **evict** command evicts an entry associated with a specific key from the cache. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> evict a
```

31.4.14. The get Command

The **get** command shows the value associated with a specified key. For primitive types and Strings, the **get** command prints the default representation. For other objects, a **JSON** representation of the object is printed. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> get a
a
```

31.4.15. The grant Command

When authorization is enabled and the role mapper has been configured to be the **ClusterRoleMapper**, the principal to role mappings are stored within the cluster registry (a replicated cache available to all nodes). The **grant** command can be used to grant new roles to a principal as follows:

```
[standalone@127.0.0.1:9990]> grant supervisor to user1
```



NOTE

The **grant** command is only available to the JBoss Data Grid Server Mode CLI.

31.4.16. The info Command

The **info** command displays the configuration of a selected cache or container. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> info
GlobalConfiguration{asyncListenerExecutor=ExecutorFactoryConfiguration{factory=org.infinispan.executors.DefaultExecutorFactory@98add58},
asyncTransportExecutor=ExecutorFactoryConfiguration{factory=org.infinispan.executors.DefaultExecutorFactory@7bc9c14c},
evictionScheduledExecutor=ScheduledExecutorFactoryConfiguration{factory=org.infinispan.executors.DefaultScheduledExecutorFactory@7ab1a411},
replicationQueueScheduledExecutor=ScheduledExecutorFactoryConfiguration{factory=org.infinispan.executors.DefaultScheduledExecutorFactory@248a9705},
globalJmxStatistics=GlobalJmxStatisticsConfiguration{allowDuplicateDomains=true, enabled=true, jmxDomain='jboss.infinispan',
mBeanServerLookup=org.jboss.as.clustering.infinispan.MBeanServerProvider@6c0dc01, cacheManagerName='local', properties={}},
transport=TransportConfiguration{clusterName='ISPN', machineId='null', rackId='null', siteId='null', strictPeerToPeer=false,
distributedSyncTimeout=240000, transport=null, nodeName='null', properties={}},
serialization=SerializationConfiguration{advancedExternalizers={1100=org.infinispan.server.core.CacheValue$Externalizer@5fab91d,
1101=org.infinispan.server.memcached.MemcachedValue$Externalizer@720bffd,
1104=org.infinispan.server.hotrod.ServerAddress$Externalizer@771c7eb2},
marshaller=org.infinispan.marshall.VersionAwareMarshaller@6fc21535,
version=52,
```



```
classResolver=org.jboss.marshalling.ModularClassResolver@2efe83e5},
shutdown=ShutdownConfiguration{hookBehavior=DONT_REGISTER}, modules={},
site=SiteConfiguration{localSite='null'}}}
```

31.4.17. The locate Command

The **locate** command displays the physical location of a specified entry in a distributed cluster. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> locate a
[host/node1,host/node2]
```

31.4.18. The put Command

The **put** command inserts an entry into the cache. If a mapping exists for a key, the **put** command overwrites the old value. The CLI allows control over the type of data used to store the key and value. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put b 100
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put c 41391
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put d true
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put e {
"package.MyClass": {"i": 5, "x": null, "b": true } }
```

Optionally, the **put** can specify a life span and maximum idle time value as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a expires 10s
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a expires 10m
maxidle 1m
```

31.4.19. The replace Command

The **replace** command replaces an existing entry in the cache with a specified new value. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> replace a b
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> get a
b
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> replace a b c
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> get a
c
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> replace a b d
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> get a
c
```

31.4.20. The roles command

When authorization is enabled and the role mapper has been configured to be the **ClusterRoleMapper**, the principal to role mappings are stored within the cluster registry (a replicated cache available to all nodes). The **roles** command can be used to list the roles associated to a specific

user, or to all users if one is not given:

```
[standalone@127.0.0.1:9990]> roles user1
[supervisor, reader]
```



NOTE

The **roles** command is only available to the JBoss Data Grid Server Mode CLI.

31.4.21. The rollback Command

The **rollback** command rolls back any changes made by an ongoing transaction. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> begin
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put b b
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> rollback
```

31.4.22. The site Command

The **site** command performs administration tasks related to cross-datacenter replication. This command also retrieves information about the status of a site and toggles the status of a site. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> site --status NYC
online
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> site --offline NYC
ok
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> site --status NYC
offline
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> site --online NYC
```

31.4.23. The start Command

The **start** command initiates a batch of operations. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> start
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put a a
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> put b b
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> end
```

31.4.24. The stats Command

The **stats** command displays statistics for the cache. An example of its usage is as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> stats
Statistics: {
  averageWriteTime: 143
  evictions: 10
  misses: 5
```

```

hitRatio: 1.0
readWriteRatio: 10.0
removeMisses: 0
timeSinceReset: 2123
statisticsEnabled: true
stores: 100
elapsedTime: 93
averageReadTime: 14
removeHits: 0
numberOfEntries: 100
hits: 1000
}
LockManager: {
  concurrencyLevel: 1000
  numberOfLocksAvailable: 0
  numberOfLocksHeld: 0
}

```

31.4.25. The upgrade Command

The **upgrade** command implements the rolling upgrade procedure. For details about rolling upgrades, refer to [Rolling Upgrades](#).

An example of the **upgrade** command's use is as follows:

```

[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> upgrade --
synchronize=hotrod --all
[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> upgrade --
disconnectsource=hotrod --all

```

31.4.26. The version Command

The **version** command displays version information for the CLI client and server. An example of its usage is as follows:

```

[standalone@127.0.0.1:9990/MyCacheManager/namedCache]> version
Client Version 5.2.1.Final
Server Version 5.2.1.Final

```

PART XV. OTHER RED HAT JBOSS DATA GRID FUNCTIONS

CHAPTER 32. SET UP THE L1 CACHE

32.1. ABOUT THE L1 CACHE

The Level 1 (or **L1**) cache stores remote cache entries after they are initially accessed, preventing unnecessary remote fetch operations for each subsequent use of the same entries. The L1 cache is only available when Red Hat JBoss Data Grid's cache mode is set to distribution. In other cache modes any configuration related to the L1 cache is ignored.

When caches are configured with distributed mode, the entries are evenly distributed between all clustered caches. Each entry is copied to a desired number of owners, which can be less than the total number of caches. As a result, the system's scalability is improved but also means that some entries are not available on all nodes and must be fetched from their owner node. In this situation, configure the Cache component to use the L1 Cache to temporarily store entries that it does not own to prevent repeated fetching for subsequent uses.

Each time a key is updated an invalidation message is generated. This message is multicast to each node that contains data that corresponds to current L1 cache entries. The invalidation message ensures that each of these nodes marks the relevant entry as invalidated. Also, when the location of an entry changes in the cluster, the corresponding L1 cache entry is invalidated to prevent outdated cache entries.

32.2. L1 CACHE CONFIGURATION

32.2.1. L1 Cache Configuration (Library Mode)

The following sample configuration shows the L1 cache default values in Red Hat JBoss Data Grid's Library Mode.

L1 Cache Configuration in Library Mode

```
<distributed-cache name="distributed_cache"
  l1-lifespan="0"
  l1-cleanup-interval="60000"/>
```

The following attributes control the L1 cache behavior:

- The **l1-lifespan** attribute indicates the maximum lifespan in milliseconds of entries placed in the L1 cache, and is not allowed in non-distributed caches. By default L1 this value is 0, indicating that L1 caching is disabled, and is only enabled if a positive value is defined.
- The **l1-cleanup-interval** parameter controls how often a cleanup task to prune L1 tracking data is run, in milliseconds, and by default is defined to 10 minutes.

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

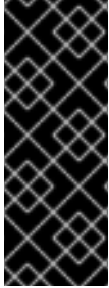
The following sample configuration shows the L1 cache default value of **0**, indicating it is disabled, in Red Hat JBoss Data Grid's Remote Client-Server mode:

L1 Cache Configuration for Remote Client-Server Mode

```
<distributed-cache l1-lifespan="0">  
  <!-- Additional configuration information here -->  
</distributed-cache>
```

The **l1-lifespan** element is added to a **distributed-cache** element to enable L1 caching and to set the life span of the L1 cache entries for the cache. This element is only valid for distributed caches.

If **l1-lifespan** is set to **0** or a negative number (**-1**), L1 caching is disabled. L1 caching is enabled when the **l1-lifespan** value is greater than **0**.



IMPORTANT

When the cache is accessed remotely via the Hot Rod protocol, the client accesses the owner node directly. Therefore, using L1 Cache via the Hot Rod protocol is not recommended; instead, refer to the Near Caching section in the *JBoss Data Grid Developer Guide*. Other remote clients (Memcached, REST) cannot target the owner, therefore, using L1 Cache may increase the performance (at the cost of higher memory consumption).



NOTE

In Remote Client-Server mode, the L1 cache was enabled by default when distributed cache was used, even if the **l1-lifespan** attribute is not set. The default lifespan value was 10 minutes. Since JBoss Data Grid 6.3, the default lifespan is 0 which disables the L1 cache. Set a non-zero value for the **l1-lifespan** parameter to enable the L1 cache.

CHAPTER 33. SET UP TRANSACTIONS

33.1. ABOUT TRANSACTIONS

33.1.1. About Transactions

A transaction consists of a collection of interdependent or related operations or tasks. All operations within a single transaction must succeed for the overall success of the transaction. If any operations within a transaction fail, the transaction as a whole fails and rolls back any changes. Transactions are particularly useful when dealing with a series of changes as part of a larger operation.

In Red Hat JBoss Data Grid, transactions are only available in Library mode.

33.1.2. About the Transaction Manager

In Red Hat JBoss Data Grid, the Transaction Manager coordinates transactions across a single or multiple resources. The responsibilities of a Transaction Manager include:

- initiating and concluding transactions
- managing information about each transaction
- coordinating transactions as they operate over multiple resources
- recovering from a failed transaction by rolling back changes

33.1.3. XA Resources and Synchronizations

XA Resources are fully fledged transaction participants. In the prepare phase (see [About Two Phase Commit \(2PC\)](#) for details), the XA Resource returns a vote with either the value **OK** or **ABORT**. If the Transaction Manager receives **OK** votes from all XA Resources, the transaction is committed, otherwise it is rolled back.

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

Unless recovery is required, it is not necessary to register as a full XA resource. An advantage of synchronizations is that they allow the Transaction Manager to optimize 2PC (Two Phase Commit) with a 1PC (One Phase Commit) where only one other resource is enlisted with that transaction (last resource commit optimization). This makes registering a synchronization more efficient.

However, if the operation fails in the prepare phase within Red Hat JBoss Data Grid, the transaction is not rolled back and if there are more participants in the transaction, they can ignore this failure and commit. Additionally, errors encountered in the commit phase are not propagated to the application code that commits the transaction.

By default JBoss Data Grid registers to the transaction as a synchronization.

33.1.4. Optimistic and Pessimistic Transactions

Pessimistic transactions acquire the locks when the first write operation on the key executes. After the key is locked, no other transaction can modify the key until this transaction is committed or rolled back. It is up to the application code to acquire the locks in correct order to prevent deadlocks.

With optimistic transactions locks are acquired at transaction prepare time and are held until the transaction commits (or rolls back). Also, Red Hat JBoss Data Grid sorts keys for all entries modified within a transaction automatically, preventing any deadlocks occurring due to the incorrect order of keys being locked. This results in:

- less messages being sent during the transaction execution
- locks held for shorter periods
- improved throughput



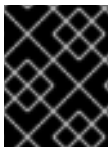
NOTE

Read operations never acquire any locks. Acquiring the lock for a read operation on demand is possible only with pessimistic transactions, using the **FORCE_WRITE_LOCK** flag with the operation.

33.1.5. Write Skew Checks

A common use case for entries is that they are read and subsequently written in a transaction. However, a third transaction can modify the entry between these two operations. In order to detect such a situation and roll back a transaction Red Hat JBoss Data Grid offers entry versioning and write skew checks. If the modified version is not the same as when it was last read during the transaction, the write skew checks throws an exception and the transaction is rolled back.

Enabling write skew check requires the **REPEATABLE_READ** isolation level. Also, in clustered mode (distributed or replicated modes), set up entry versioning. For local mode, entry versioning is not required.



IMPORTANT

With optimistic transactions, write skew checks are required for (atomic) conditional operations.

33.1.6. Transactions Spanning Multiple Cache Instances

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

33.2. CONFIGURE TRANSACTIONS

33.2.1. Configure Transactions (Library Mode)

In Red Hat JBoss Data Grid, transactions in Library mode can be configured with synchronization and transaction recovery. Transactions in their entirety (which includes synchronization and transaction recovery) are not available in Remote Client-Server mode.

In order to execute a cache operation, the cache requires a reference to the environment's Transaction Manager. Configure the cache with the class name that belongs to an implementation of the **TransactionManagerLookup** interface. When initialized, the cache creates an instance of the specified class and invokes its **getTransactionManager()** method to locate and return a reference to the Transaction Manager.

In Library mode, transactions are configured as follows:

Configure Transactions in Library Mode (XML Configuration)

```
<local-cache name="default" <!-- Additional configuration information here -->
-->
  <transaction mode="BATCH"
    stop-timeout="60000"
    auto-commit="true"
    protocol="DEFAULT"
    recovery-cache="recoveryCache">
  <locking <!-- Additional configuration information here --> >
  <versioning versioningScheme="SIMPLE"/>
  <!-- Additional configuration information here -->
</local-cache>
```

1. Enable **transactions** by defining a **mode**. By default the mode is **NONE**, therefore disabling transactions. Valid transaction modes are **BATCH**, **NON_XA**, **NON_DURABLE_XA**, **FULL_XA**.
2. Define a **stop-timeout**, so that if there are any ongoing transactions when a cache is stopped the instance will wait for ongoing transactions to finish. Defaults to **30000** milliseconds.
3. Enable **auto-commit**, so that single operation transactions do not need to be manually initiated. Defaults to **true**.
4. Define the commit **protocol** in use. Valid commit protocols are **DEFAULT** and **TOTAL_ORDER**.
5. Define the name of the **recovery-cache**, where recovery related information is kept. Defaults to **_recoveryInfoCacheName_**.
6. Enable **versioning** of entries by defining the **versioningScheme** attribute as **SIMPLE**. Defaults to **NONE**, indicating that versioning is disabled.

33.2.2. Configure Transactions (Remote Client-Server Mode)

Red Hat JBoss Data Grid does not offer transactions in Remote Client-Server mode. The default configuration is non-transactional, which is set as follows:

Default Transaction Configuration in Remote Client-Server Mode

```
<cache>
  <!-- Additional configuration elements here -->
  <transaction mode="NONE" />
  <!-- Additional configuration elements here -->
</cache>
```



IMPORTANT

JBoss Data Grid in Remote Client-Server mode does not support transactions directly. However, if you use conditional operators such as **replaceWithVersion**, **putIfAbsent**, or **removeWithVersion** you should enable transactions.

To enable transactions for the use of conditional operators use the following configuration:

Transaction Configuration for Conditional Operators in Remote Client-Server Mode

```
<cache>
  <!-- Additional configuration elements here -->
  <transaction mode="NON_XA" />
  <!-- Additional configuration elements here -->
</cache>
```

33.3. TRANSACTION RECOVERY

33.3.1. Transaction Recovery

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

JBoss Data Grid uses JMX for operations that explicitly force transactions to commit or roll back. These methods receive byte arrays that describe the XID instead of the number associated with the relevant transactions.

The System Administrator can use such JMX operations to facilitate automatic job completion for transactions that require manual intervention. This process uses the Transaction Manager's transaction recovery process and has access to the Transaction Manager's XID objects.

33.3.2. Transaction Recovery Process

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

The Transaction Recovery Process

1. The Transaction Manager creates a list of transactions that require intervention.
2. The system administrator, connected to JBoss Data Grid using JMX, is presented with the list of transactions (including transaction IDs) using email or logs. The status of each transaction is either **COMMITTED** or **PREPARED**. If some transactions are in both **COMMITTED** and **PREPARED** states, it indicates that the transaction was committed on some nodes while in the preparation state on others.
3. The System Administrator visually maps the XID received from the Transaction Manager to a JBoss Data Grid internal ID. This step is necessary because the XID (a byte array) cannot be conveniently passed to the JMX tool and then reassembled by JBoss Data Grid without this mapping.
4. The system administrator forces the commit or rollback process for a transaction based on the mapped internal ID.

33.3.3. Transaction Recovery Example

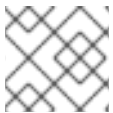
The following example describes how transactions are used in a situation where money is transferred from an account stored in a database to an account stored in Red Hat JBoss Data Grid.

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

1. The `TransactionManager.commit()` method is invoked to run the two phase commit protocol between the source (the database) and the destination (JBoss Data Grid) resources.
2. The `TransactionManager` tells the database and JBoss Data Grid to initiate the prepare phase (the first phase of a Two Phase Commit).

During the commit phase, the database applies the changes but JBoss Data Grid fails before receiving the Transaction Manager's commit request. As a result, the system is in an inconsistent state due to an incomplete transaction. Specifically, the amount to be transferred has been subtracted from the database but is not yet visible in JBoss Data Grid because the prepared changes could not be applied.

Transaction recovery is used here to reconcile the inconsistency between the database and JBoss Data Grid entries.



NOTE

To use JMX to manage transaction recoveries, JMX support must be explicitly enabled.

33.4. DEADLOCK DETECTION

33.4.1. Deadlock Detection

A deadlock occurs when multiple processes or tasks wait for the other to release a mutually required resource. Deadlocks can significantly reduce the throughput of a system, particularly when multiple transactions operate against one key set.

Red Hat JBoss Data Grid provides deadlock detection to identify such deadlocks. Deadlock detection is enabled by default.

33.4.2. Enable Deadlock Detection

Deadlock detection in Red Hat JBoss Data Grid is enabled by default, and may be configured by adjusting the `deadlock-detection-spin` attribute of the `cache` configuration element, as seen below:

```
<local-cache [...] deadlock-detection-spin="1000"/>
```

The `deadlock-detection-spin` attribute defines how often lock acquisition is attempted within the maximum time allowed to acquire a particular lock (in milliseconds). This value defaults to **100** milliseconds, and negative values disable deadlock detection.

Deadlock detection can only be applied to individual caches. Deadlocks that are applied on more than one cache cannot be detected by JBoss Data Grid.

CHAPTER 34. CONFIGURE JGROUPS

34.1. ABOUT JGROUPS

JGroups is the underlying group communication library used to connect Red Hat JBoss Data Grid instances. For a full list of JGroups protocols supported in JBoss Data Grid, see [Supported JGroups Protocols](#).

34.2. CONFIGURE RED HAT JOSS DATA GRID INTERFACE BINDING (REMOTE CLIENT-SERVER MODE)

34.2.1. Interfaces

Red Hat JBoss Data Grid allows users to specify an interface type rather than a specific (unknown) IP address.

- **link-local**: Uses a **169.x.x.x** or **254.x.x.x** address. This suits the traffic within one box.

```
<interfaces>
  <interface name="link-local">
    <link-local-address/>
  </interface>
  <!-- Additional configuration elements here -->
</interfaces>
```

- **site-local**: Uses a private IP address, for example **192.168.[replaceable].x.[replaceable].x**. This prevents extra bandwidth charged from GoGrid, and similar providers.

```
<interfaces>
  <interface name="site-local">
    <site-local-address/>
  </interface>
  <!-- Additional configuration elements here -->
</interfaces>
```

- **global**: Picks a public IP address. This should be avoided for replication traffic.

```
<interfaces>
  <interface name="global">
    <any-address/>
  </interface>
  <!-- Additional configuration elements here -->
</interfaces>
```

- **non-loopback**: Uses the first address found on an active interface that is not a **127.x.x.x** address.

```
<interfaces>
  <interface name="non-loopback">
    <not>
      <loopback />
  </interface>
  <!-- Additional configuration elements here -->
</interfaces>
```

```

    </not>
    </interface>
</interfaces>

```

34.2.2. Binding Sockets

34.2.2.1. Binding Sockets

Socket bindings provide a method of associating a name with networking details, such as an interface, a port, a multicast-address, or other details. Sockets may be bound to the interface either individually or using a socket binding group.

34.2.2.2. Binding a Single Socket Example

The following is an example depicting the use of JGroups interface socket binding to bind an individual socket using the **socket-binding** element.

Socket Binding

```

<socket-binding name="jgroups-udp" <!-- Additional configuration elements
here --> interface="site-local"/>

```

34.2.2.3. Binding a Group of Sockets Example

The following is an example depicting the use of Groups interface socket bindings to bind a group, using the **socket-binding-group** element:

Bind a Group

```

<socket-binding-group name="ha-sockets" default-interface="global">
  <!-- Additional configuration elements here -->
  <socket-binding name="jgroups-tcp" port="7600"/>
  <socket-binding name="jgroups-tcp-fd" port="57600"/>
  <!-- Additional configuration elements here -->
</socket-binding-group>

```

The two sample socket bindings in the example are bound to the same **default-interface** (**global**), therefore the interface attribute does not need to be specified.

34.2.3. Configure JGroups Socket Binding

Each JGroups stack, configured in the JGroups subsystem, uses a specific socket binding. Set up the socket binding as follows:

JGroups UDP Socket Binding Configuration

The following example utilizes UDP automatically from the cluster. In this example the **jgroups-udp** socket binding is defined for the transport:

```

<subsystem xmlns="urn:jboss:domain:jgroups:3.0" default-stack="udp">
  <stack name="udp">
    <transport type="UDP" socket-binding="jgroups-udp">
      <!-- Additional configuration elements here -->
    </transport>
  </stack>
</subsystem>

```

```

    </transport>
    <protocol type="PING" />
    <protocol type="MERGE3" />
    <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd" />
    <protocol type="FD_ALL" />
    <protocol type="VERIFY_SUSPECT" />
    <protocol type="pbcast.NAKACK2" />
    <protocol type="UNICAST3" />
    <protocol type="pbcast.STABLE" />
    <protocol type="pbcast.GMS" />
    <protocol type="UFC" />
    <protocol type="MFC" />
    <protocol type="FRAG3" />
  </stack>
</subsystem>

```

JGroups TCP Socket Binding Configuration

The following example uses TCP to establish direct communication between two clusters nodes. In the example below node1 is located at 192.168.1.2:7600, and node2 is located at 192.168.1.3:7600. The port in use will be defined by the **jgroups-tcp** property in the **socket-binding** section.

```

<subsystem xmlns="urn:infinispan:server:jgroups:8.0" default-stack="tcp">
  <stack name="tcp">
    <transport type="TCP" socket-binding="jgroups-tcp" />
    <protocol type="TCPPING">
      <property
name="initial_hosts">192.168.1.2[7600],192.168.1.3[7600]</property>
      <property name="num_initial_members">2</property>
      <property name="port_range">0</property>
      <property name="timeout">2000</property>
    </protocol>
    <protocol type="MERGE3" />
    <protocol type="FD_SOCK" socket-binding="jgroups-tcp-fd" />
    <protocol type="FD_ALL" />
    <protocol type="VERIFY_SUSPECT" />
    <protocol type="pbcast.NAKACK2">
      <property name="use_mcast_xmit">>false</property>
    </protocol>
    <protocol type="UNICAST3" />
    <protocol type="pbcast.STABLE" />
    <protocol type="pbcast.GMS" />
    <protocol type="MFC" />
    <protocol type="FRAG3" />
  </stack>
</subsystem>

```

The decision of UDP vs TCP must be made in each environment. By default JGroups uses UDP, as it allows for dynamic detection of clustered members and scales better in larger clusters due to a smaller network footprint. In addition, when using UDP only one packet per cluster is required, as multicast packets are received by all subscribers to the multicast address; however, in environments where multicast traffic is prohibited, or if UDP traffic can not reach the remote cluster nodes, such as when cluster members are on separate VLANs, TCP traffic can be used to create a cluster.



IMPORTANT

When using UDP as the JGroups transport, the socket binding has to specify the regular (unicast) port, multicast address, and multicast port.

34.3. CONFIGURE JGROUPS (LIBRARY MODE)

34.3.1. Configure JGroups for Clustered Modes

Red Hat JBoss Data Grid must have an appropriate JGroups configuration in order to operate in clustered mode.

JGroups XML Configuration

```
<infinispan xmlns="urn:infinispan:config:8.4">
  <jgroups>
    <stack-file name="jgroupsStack"
path="/path/to/jgroups/xml/jgroups.xml"/>
  </jgroups>
  <cache-container name="default" default-cache="default">
    <transport stack="jgroupsStack" lock-timeout="600000">
cluster="default" />
  </cache-container>
</infinispan>
```

JBoss Data Grid will first search for *jgroups.xml* in the classpath; if no instances are found in the classpath it will then search for an absolute path name.

34.3.2. JGroups Transport Protocols

34.3.2.1. JGroups Transport Protocols

A transport protocol is the protocol at the bottom of a protocol stack. Transport Protocols are responsible for sending and receiving messages from the network.

Red Hat JBoss Data Grid ships with both UDP and TCP transport protocols.

34.3.2.2. The UDP Transport Protocol

UDP is a transport protocol that uses:

- IP multicasting to send messages to all members of a cluster.
- UDP datagrams for unicast messages, which are sent to a single member.

When the UDP transport is started, it opens a unicast socket and a multicast socket. The unicast socket is used to send and receive unicast messages, the multicast socket sends and receives multicast sockets. The physical address of the channel will be the same as the address and port number of the unicast socket.

UDP is the default transport protocol used in JBoss Data Grid as it allows for dynamic discovery of additional cluster nodes, and scales well with cluster sizes.

34.3.2.3. The TCP Transport Protocol

TCP/IP is a replacement transport for UDP in situations where IP multicast cannot be used, such as operations over a WAN where routers may discard IP multicast packets.

TCP is a transport protocol used to send unicast and multicast messages.

- When sending multicast messages, TCP sends multiple unicast messages.

As IP multicasting cannot be used to discover initial members, another mechanism must be used to find initial membership.

34.3.2.4. Using the TCPping Protocol

Some networks only allow TCP to be used. The pre-configured *default-configs/default-jgroups-tcp.xml* includes the *MPING* protocol, which uses *UDP* multicast for discovery. When *UDP* multicast is not available, the *MPING* protocol, has to be replaced by a different mechanism. The recommended alternative is the *TCPping* protocol. The *TCPping* configuration contains a static list of IP addresses which are contacted for node discovery.

Configure the JGroups Subsystem to Use *TCPping*

```
<TCP bind_port="7800" />
<TCPping
  initial_hosts="{jgroups.tcping.initial_hosts:HostA[7800],HostB[7801]}"
  port_range="1" />
```

34.3.3. Pre-Configured JGroups Files

34.3.3.1. Pre-Configured JGroups Files

Red Hat JBoss Data Grid ships with a number of pre-configured JGroups files packaged in *infinispan-embedded.jar*, and are available on the classpath by default. In order to use one of these files, specify one of these file names instead of using *jgroups.xml*.

The JGroups configuration files shipped with JBoss Data Grid are intended to be used as a starting point for a working project. JGroups will usually require fine-tuning for optimal network performance.

The available configurations are:

- *default-configs/default-jgroups-udp.xml*
- *default-configs/default-jgroups-tcp.xml*
- *default-configs/default-jgroups-ec2.xml*
- *default-configs/default-jgroups-google.xml*

34.3.3.2. *default-jgroups-udp.xml*

The *default-configs/default-jgroups-udp.xml* file is a pre-configured JGroups configuration in Red Hat JBoss Data Grid. The *default-jgroups-udp.xml* configuration

- uses **UDP** as a transport and **UDP** multicast for discovery, allowing for dynamic discovery of nodes.

- is suitable for larger clusters, as it requires less resources than **TCP**.
- is recommended for clusters that send the same information to all nodes, such as when using Invalidation or Replication modes.

The behavior of some of these settings can be altered by adding certain system properties to the JVM at startup. The settings that can be configured are shown in the following table.

Table 34.1. default-jgroups-udp.xml System Properties

System Property	Description	Default	Required?
jgroups.udp.mcast_addr	IP address to use for multicast (both for communications and discovery). Must be a valid Class D IP address, suitable for IP multicast	228.6.7.8	No
jgroups.udp.mcast_port	Port to use for multicast socket	46655	No
jgroups.ip_ttl	Specifies the time-to-live (TTL) for IP multicast packets. The value here refers to the number of network hops a packet is allowed to make before it is dropped	2	No
jgroups.thread_pool.min_threads	Minimum thread pool size for the thread pool	0	No
jgroups.thread_pool.max_threads	Maximum thread pool size for the thread pool	200	No
jgroups.join_timeout	The maximum number of milliseconds to wait for a new node JOIN request to succeed	5000	No

34.3.3.3. default-jgroups-tcp.xml

The `default-configs/default-jgroups-tcp.xml` file is a pre-configured JGroups configuration in Red Hat JBoss Data Grid. The `default-jgroups-tcp.xml` configuration

- uses **TCP** as a transport and **UDP** multicast for discovery.
- is typically used where multicast **UDP** is not an option.
- is recommended for point-to-point (unicast) communication, such as when using Distribution mode, as it has a more refined method of flow control.

As with other pre-configured JGroups files, the behavior of some of these settings can be altered by adding certain system properties to the JVM at startup. The settings that can be configured are shown in the following table.

Table 34.2. default-jgroups-tcp.xml System Properties

System Property	Description	Default	Required?
jgroups.tcp.address	IP address to use for the TCP transport.	127.0.0.1	Not required, but should be changed for clustering servers located on different systems
jgroups.tcp.port	Port to use for TCP socket	7800	No
jgroups.thread_pool.min_threads	Minimum thread pool size for the thread pool	0	No
jgroups.thread_pool.max_threads	Maximum thread pool size for the thread pool	200	No
jgroups.mping.mcast_address	IP address to use for multicast (for discovery). Must be a valid Class D IP address, suitable for IP multicast.	228.2.4.6	No
jgroups.mping.mcast_port	Port to use for multicast socket	43366	No
jgroups.udp.ip_ttl	Specifies the time-to-live (TTL) for IP multicast packets. The value here refers to the number of network hops a packet is allowed to make before it is dropped	2	No
jgroups.join_timeout	The maximum number of milliseconds to wait for a new node JOIN request to succeed	5000	No

34.3.3.4. default-jgroups-ec2.xml

The `default-configs/default-jgroups-ec2.xml` file is a pre-configured JGroups configuration in Red Hat JBoss Data Grid. The `default-jgroups-ec2.xml` configuration

- uses TCP as a transport and S3_PING for discovery.

- is suitable on **EC2** nodes, as UDP multicast is not available.

As with other pre-configured JGroups files, the behavior of some of these settings can be altered by adding certain system properties to the JVM at startup. The settings that can be configured are shown in the following table.

Table 34.3. default-jgroups-ec2.xml System Properties

System Property	Description	Default	Required?
jgroups.tcp.address	IP address to use for the TCP transport.	127.0.0.1	Not required, but should be changed for clustering servers located on different EC2 nodes
jgroups.tcp.port	Port to use for TCP socket	7800	No
jgroups.thread_pool.min_threads	Minimum thread pool size for the thread pool	0	No
jgroups.thread_pool.max_threads	Maximum thread pool size for the thread pool	200	No
jgroups.s3.access_key	The Amazon S3 access key used to access an S3 bucket		Yes
jgroups.s3.secret_access_key	The Amazon S3 secret key used to access an S3 bucket		Yes
jgroups.s3.bucket	Name of the Amazon S3 bucket to use. Must be unique and must already exist		Yes
jgroups.s3.pre_signed_delete_url	The pre-signed URL to be used for the DELETE operation.		Yes
jgroups.s3.pre_signed_put_url	The pre-signed URL to be used for the PUT operation.		Yes
jgroups.s3.prefix	If set, S3_PING searches for a bucket with a name that starts with the prefix value.		No

System Property	Description	Default	Required?
jgroups.join_timeout	The maximum number of milliseconds to wait for a new node JOIN request to succeed	5000	No

34.3.3.5. default-jgroups-google.xml

The *default-configs/default-jgroups-google.xml* file is a pre-configured JGroups configuration in Red Hat JBoss Data Grid. The *default-jgroups-google.xml* configuration

- uses TCP as a transport and GOOGLE_PING for discovery.
- is suitable on **Google Compute Engine** nodes, as UDP multicast is not available.

As with other pre-configured JGroups files, the behavior of some of these settings can be altered by adding certain system properties to the JVM at startup. The settings that can be configured are shown in the following table.

Table 34.4. default-jgroups-google.xml System Properties

System Property	Description	Default	Required?
jgroups.tcp.address	IP address to use for the TCP transport.	127.0.0.1	Not required, but should be changed for clustering systems located on different nodes
jgroups.tcp.port	Port to use for TCP socket	7800	No
jgroups.thread_pool.min_threads	Minimum thread pool size for the thread pool	0	No
jgroups.thread_pool.max_threads	Maximum thread pool size for the thread pool	200	No
jgroups.google.access_key	The Google Compute Engine User's access key used to access the bucket		Yes
jgroups.google.secret_access_key	The Google Compute Engine User's secret access key used to access the bucket		Yes

System Property	Description	Default	Required?
<code>jgroups.google.bucket</code>	Name of the Google Compute Engine bucket to use. Must be unique and already exist		Yes
<code>jgroups.join_timeout</code>	The maximum number of milliseconds to wait for a new node JOIN request to succeed	5000	No

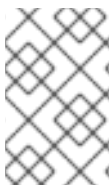
34.4. TEST MULTICAST USING JGROUPS

34.4.1. Test Multicast Using JGroups

Learn how to ensure that the system has correctly configured multicasting within the cluster.

34.4.2. Testing With Different Red Hat JBoss Data Grid Versions

The following table details which Red Hat JBoss Data Grid versions are compatible with this multicast test:



NOTE

`${infinispan.version}` corresponds to the version of Infinispan included in the specific release of JBoss Data Grid. This will appear in a x.y.z format, with the major version, minor version, and revision being included.

Table 34.5. Testing with Different JBoss Data Grid Versions

Version	Test Case	Details
JBoss Data Grid 7.1.0	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> For library mode, they are inside the <i>infinispan-embedded-<code>\${infinispan.version}</code>.Final-redhat-#</i> JAR file For Remote Client-Server mode, they are in the JGroups JAR file in the <code>\${JDG_HOME}/modules/system/layers/base/org/jgroups/main/</code> directory."

Version	Test Case	Details
JBoss Data Grid 7.0.0	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • For library mode, they are inside the <i>infinispan-embedded-<code>{infinispan.version}.Final-redhat-#</code></i> JAR file • For Remote Client-Server mode, they are in the JGroups JAR file in the <code>{JDG_HOME}/modules/system/layers/base/org/jgroups/main/</code> directory."
JBoss Data Grid 6.6.0	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • For library mode, they are inside the <i>infinispan-embedded-<code>{infinispan.version}.Final-redhat-#</code></i> JAR file • For Remote Client-Server mode, they are in the JGroups JAR file in the <code>{JDG_HOME}/modules/system/layers/base/org/jgroups/main/</code> directory."
JBoss Data Grid 6.5.1	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • For library mode, they are inside the <i>infinispan-embedded-<code>{infinispan.version}.Final-redhat-#</code></i> JAR file • For Remote Client-Server mode, they are in the JGroups JAR file in the <code>{JDG_HOME}/modules/system/layers/base/org/jgroups/main/</code> directory."

Version	Test Case	Details
JBoss Data Grid 6.5.0	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • For library mode, they are inside the <i>infinispan-embedded-<code>{infinispan.version}.Final-redhat-#</code></i> JAR file • For Remote Client-Server mode, they are in the JGroups JAR file in the <code>{JDG_HOME}/modules/system/layers/base/org/jgroups/main/</code> directory."
JBoss Data Grid 6.4.0	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • For library mode, they are inside the <i>infinispan-embedded-<code>{infinispan.version}.Final-redhat-#</code></i> JAR file • For Remote Client-Server mode, they are in the JGroups JAR file in the <code>{JDG_HOME}/modules/system/layers/base/org/jgroups/main/</code> directory."
JBoss Data Grid 6.3.0	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • In Library mode, they are in the JGroups JAR file in the lib directory. • In Remote Client-Server mode, they are in the JGroups JAR file in the <code>{JDG_HOME}/modules/system/layers/base/org/jgroups/main</code> .

Version	Test Case	Details
JBoss Data Grid 6.2.1	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • In Library mode, they are in the JGroups JAR file in the lib directory. • In Remote Client-Server mode, they are in the JGroups JAR file in the <code>\${JDG_HOME}/modules/system/layers/base/org/jgroups/main</code>
JBoss Data Grid 6.2.0	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • In Library mode, they are in the JGroups JAR file in the lib directory. • In Remote Client-Server mode, they are in the JGroups JAR file in the <code>\${JDG_HOME}/modules/system/layers/base/org/jgroups/main</code> .
JBoss Data Grid 6.1.0	Available	<p>The location of the test classes depends on the distribution:</p> <ul style="list-style-type: none"> • In Library mode, they are in the JGroups JAR file in the <i>lib</i> directory. • In Remote Client-Server mode, they are in the JGroups JAR file in the <code>\${JDG_HOME}/modules/org/jgroups/main/</code> directory.
JBoss Data Grid 6.0.1	Not Available	This version of JBoss Data Grid is based on JBoss Enterprise Application Platform 6.0, which does not include the test classes used for this test.
JBoss Data Grid 6.0.0	Not Available	This version of JBoss Data Grid is based on JBoss Enterprise Application Server 6.0, which does not include the test classes used for this test.

34.4.3. Testing Multicast Using JGroups

The following procedure details the steps to test multicast using JGroups if you are using Red Hat JBoss Data Grid:

Prerequisites

Ensure that the following prerequisites are met before starting the testing procedure.

1. Set the **bind_addr** value to the appropriate IP address for the instance.
2. For added accuracy, set **mcast_addr** and **port** values that are the same as the cluster communication values.
3. Start two command line terminal windows. Navigate to the location of the JGroups JAR file for one of the two nodes in the first terminal and the same location for the second node in the second terminal.

Test Multicast Using JGroups

1. Run the Multicast Server on Node One

Run the following command on the command line terminal for the first node (replace *jgroups.jar* with the *infinispan-embedded.jar* for Library mode):

```
java -cp jgroups.jar org.jgroups.tests.McastReceiverTest -mcast_addr
230.1.2.3 -port 5555 -bind_addr $YOUR_BIND_ADDRESS
```

2. Run the Multicast Server on Node Two

Run the following command on the command line terminal for the second node (replace *jgroups.jar* with the *infinispan-embedded.jar* for Library mode):

```
java -cp jgroups.jar org.jgroups.tests.McastSenderTest -mcast_addr
230.1.2.3 -port 5555 -bind_addr $YOUR_BIND_ADDRESS
```

3. Transmit Information Packets

Enter information on instance for node two (the node sending packets) and press enter to send the information.

4. View Receives Information Packets

View the information received on the node one instance. The information entered in the previous step should appear here.

5. Confirm Information Transfer

Repeat steps 3 and 4 to confirm all transmitted information is received without dropped packets.

6. Repeat Test for Other Instances

Repeat steps 1 to 4 for each combination of sender and receiver. Repeating the test identifies other instances that are incorrectly configured.

Result

All information packets transmitted from the sender node must appear on the receiver node. If the sent information does not appear as expected, multicast is incorrectly configured in the operating system or the network.

CHAPTER 35. USE RED HAT JBOSS DATA GRID WITH AMAZON WEB SERVICES

35.1. THE S3_PING JGROUPS DISCOVERY PROTOCOL

S3_PING is a discovery protocol that is ideal for use with Amazon's Elastic Compute Cloud (EC2) because EC2 does not allow multicast and therefore *MPING* is not allowed.

Each EC2 instance adds a small file to an S3 data container, known as a bucket. Each instance then reads the files in the bucket to discover the other members of the cluster.

35.2. S3_PING CONFIGURATION OPTIONS

35.2.1. S3_PING Configuration Options

Red Hat JBoss Data Grid works with Amazon Web Services in two ways:

- In Library mode, use JGroups' *default-configs/default-jgroups-ec2.xml* file (see [default-jgroups-ec2.xml](#) for details) or use the *S3_PING* protocol.
- In Remote Client-Server mode, use JGroups' *S3_PING* protocol.

In Library and Remote Client-Server mode, there are three ways to configure the *S3_PING* protocol for clustering to work in Amazon AWS:

- Use Private S3 Buckets. These buckets use Amazon AWS credentials.
- Use Pre-Signed URLs. These pre-assigned URLs are assigned to buckets with private write and public read rights.
- Use Public S3 Buckets. These buckets do not have any credentials.

35.2.2. Using Private S3 Buckets

This configuration requires access to a private bucket that can only be accessed with the appropriate AWS credentials. To confirm that the appropriate permissions are available, confirm that the user has the following permissions for the bucket:

- List
- Upload/Delete
- View Permissions
- Edit Permissions

Ensure that the *S3_PING* configuration includes the following properties:

- the **location** where the bucket is found.
- the **access_key** and **secret_access_key** properties for the AWS user.

**NOTE**

If a **403** error displays when using this configuration, verify that the properties have the correct values. If the problem persists, confirm that the system time in the EC2 node is correct. Amazon S3 rejects requests with a time stamp that is more than **15** minutes old compared to their server's times for security purposes.

Start the Red Hat JBoss Data Grid Server with a Private Bucket

Run the following command from the top level of the server directory to start the Red Hat JBoss Data Grid server using a private S3 bucket:

```
bin/standalone.sh
-c cloud.xml
-Djboss.node.name={node_name}
-Djboss.socket.binding.port-offset={port_offset}
-Djboss.default.jgroups.stack=s3-private
-Djgroups.s3.bucket={s3_bucket_name}
-Djgroups.s3.access_key={access_key}
-Djgroups.s3.secret_access_key={secret_access_key}
```

1. Replace **{node_name}** with the server's desired node name.
2. Replace **{port_offset}** with the port offset. To use the default ports specify this as **0**.
3. Replace **{s3_bucket_name}** with the appropriate bucket name.
4. Replace **{access_key}** with the user's access key.
5. Replace **{secret_access_key}** with the user's secret access key.

35.2.3. Using Pre-Signed URLs

35.2.3.1. Using Pre-Signed URLs

For this configuration, create a publically readable bucket in S3 by setting the List permissions to Everyone to allow public read access. Each node in the cluster may share a pre-signed URL that points to a single file, allowing a single file to be shared across every node in the cluster. This URL points to a unique file and can include a folder path within the bucket.

**NOTE**

Longer paths will cause errors in *S3_PING*. For example, a path such as *my_bucket/DemoCluster/jgroups.list* works while a longer path such as *my_bucket/Demo/Cluster/jgroups.list* will not.

35.2.3.2. Generating Pre-Signed URLs

JGroup's *S3_PING* class includes a utility method to generate pre-signed URLs. The last argument for this method is the time when the URL expires expressed in the number of seconds since the Unix epoch (January 1, 1970).

The syntax to generate a pre-signed URL is as follows:

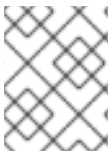
```
String url = S3_PING.generatePreSignedUrl("{access_key}", "
{secret_access_key}", "{operation}", "{bucket_name}", "{path}",
{seconds});
```

1. Replace **{operation}** with either **PUT** or **DELETE**.
2. Replace **{access_key}** with the user's access key.
3. Replace **{secret_access_key}** with the user's secret access key.
4. Replace **{bucket_name}** with the name of the bucket.
5. Replace **{path}** with the desired path to the file within the bucket.
6. Replace **{seconds}** with the number of seconds since the Unix epoch (January 1, 1970) that the path remains valid.

Generate a Pre-Signed URL

```
String putUrl = S3_PING.generatePreSignedUrl("access_key",
"secret_access_key", "put", "my_bucket", "DemoCluster/jgroups.list",
1234567890);
```

Ensure that the *S3_PING* configuration includes the **pre_signed_put_url** and **pre_signed_delete_url** properties generated by the call to **S3_PING.generatePreSignedUrl()**. This configuration is more secure than one using private S3 buckets, because the AWS credentials are not stored on each node in the cluster



NOTE

If a pre-signed URL is entered into an XML file, then the **&** characters in the URL must be replaced with its XML entity (**&**).

35.2.3.3. Set Pre-Signed URLs Using the Command Line

To set the pre-signed URLs using the command line, use the following guidelines:

- Enclose the URL in double quotation marks (" ").
- In the URL, each occurrence of the ampersand (&) character must be escaped with a backslash (\)

Start a JBoss Data Grid Server with a Pre-Signed URL

```
bin/standalone.sh
-c cloud.xml
-Djboss.node.name={node_name}
-Djboss.socket.binding.port-offset={port_offset}
-Djboss.default.jgroups.stack=s3-presigned
-
Djgroups.s3.pre_signed_delete_url="http://{s3_bucket_name}.s3.amazonaws.com/
jgroups.list?AWSAccessKeyId={access_key}&Expires=
{expiration_time}&Signature={signature}"
-
```

```
Djgroups.s3.pre_signed_put_url="http://{s3_bucket_name}.s3.amazonaws.com/j
groups.list?AWSAccessKeyId={access_key}&Expires=
{expiration_time}&Signature={signature}"
```

1. Replace **{node_name}** with the server's desired node name.
2. Replace **{port_offset}** with the port offset. To use the default ports specify this as **0**.
3. Replace **{s3_bucket_name}** with the appropriate bucket name.
4. Replace **{access_key}** with the user's access key.
5. Replace **{expiration_time}** with the values for the URL that are passed into the **S3_PING.generatePreSignedUrl()** method.
6. Replace **{signature}** with the values generated by the **S3_PING.generatePreSignedUrl()** method.

35.2.4. Using Public S3 Buckets

This configuration involves an S3 bucket that has public read and write permissions, which means that Everyone has permissions to List , Upload/Delete , View Permissions , and Edit Permissions for the bucket.

The **location** property must be specified with the bucket name for this configuration. This configuration method is the least secure because any user who knows the name of the bucket can upload and store data in the bucket and the bucket creator's account is charged for this data.

To start the Red Hat JBoss Data Grid server, use the following command:

```
bin/standalone.sh
-c cloud.xml
-Djboss.node.name={node_name}
-Djboss.socket.binding.port-offset={port_offset}
-Djboss.default.jgroups.stack=s3-public
-Djgroups.s3.bucket={s3_bucket_name}
```

1. Replace **{node_name}** with the server's desired node name.
2. Replace **{port_offset}** with the port offset. To use the default ports specify this as **0**.
3. Replace **{s3_bucket_name}** with the appropriate bucket name.

35.3. UTILIZING AN ELASTIC IP ADDRESS

While each node in the cluster is able to discover other nodes in the cluster using the S3_PING protocol, all network traffic is over the internal private network. It is recommended to configure an Elastic IP, or static IP, for a single node, so that a consistent address is available for configuring the cluster, such as through the Administration Console, across restarts. If no Elastic IP is configured each instance will contain a randomized IP address on its public network whenever it is started.

Full instructions for configuring an Elastic IP address may be found in Amazon's [Getting Started Guide](#).

CHAPTER 36. USE RED HAT JBOSS DATA GRID WITH GOOGLE COMPUTE ENGINE

36.1. THE GOOGLE_PING PROTOCOL

GOOGLE_PING is a discovery protocol used by JGroups during cluster formation. It is ideal to use with Google Compute Engine (GCE) and uses Google Cloud Storage to store information about individual cluster members.

36.2. GOOGLE_PING CONFIGURATION

36.2.1. GOOGLE_PING Configuration

Red Hat JBoss Data Grid works with Google Compute Engine in the following way:

- In Library mode, use the JGroups' configuration file *default-configs/default-jgroups-google.xml* or use the *GOOGLE_PING* protocol in an existing configuration file.
- In Remote Client-Server mode, define the properties on the command line when you start the server to use the JGroups Google stack (see example in [Starting the Server in Google Compute Engine](#)).

To configure the *GOOGLE_PING* protocol to work in Google Compute Engine in Library and Remote Client-Server mode:

- Use JGroups bucket. These buckets use Google Compute Engine credentials.
- Use the access key.
- Use the secret access key.



NOTE

Only the *TCP* protocol is supported in Google Compute Engine since multicasts are not allowed.

36.2.2. Starting the Server in Google Compute Engine

This configuration requires access to a bucket that can only be accessed with the appropriate Google Compute Engine credentials.

Ensure that the *GOOGLE_PING* configuration includes the following properties:

- the **access_key** and the **secret_access_key** properties for the Google Compute Engine user.

Start the Red Hat JBoss Data Grid Server with a Bucket

Run the following command from the top level of the server directory to start the Red Hat JBoss Data Grid server using a bucket:

```
bin/standalone.sh
-c cloud.xml
```

```
-Djboss.node.name={node_name}
-Djboss.socket.binding.port-offset={port_offset}
-Djboss.default.jgroups.stack=google
-Djgroups.google.bucket={google_bucket_name}
-Djgroups.google.access_key={access_key}
-Djgroups.google.secret_access_key={secret_access_key}
```

1. Replace **{node_name}** with the server's desired node name.
2. Replace **{port_offset}** with the port offset. To use the default ports specify this as **0**.
3. Replace **{google_bucket_name}** with the appropriate bucket name.
4. Replace **{access_key}** with the user's access key.
5. Replace **{secret_access_key}** with the user's secret access key.

36.3. UTILIZING A STATIC IP ADDRESS

While each node in the cluster is able to discover other nodes in the cluster using the `GOOGLE_PING` protocol, all network traffic is over the internal private network. It is recommended to configure an external static IP address for a single node, so that a consistent address is available for configuring the cluster, such as through the Administration Console, across restarts. If no static address is configured each instance will contain a randomized IP address on its public network whenever it is started.

Full instructions for configuring an external static IP address may be found in Google's [Configuring an Instance's IP Address](#) documentation.

CHAPTER 37. INTEGRATION WITH THE SPRING FRAMEWORK

37.1. INTEGRATION WITH THE SPRING FRAMEWORK

JBoss Data Grid allows users to define a Spring Cache provider, providing applications a method of easily adding caching support, and allowing users familiar with Spring's programming model a way to have caching fulfilled by JBoss Data Grid.

The following steps and examples demonstrate methods that Administrators can use to configure JBoss Data Grid nodes for Spring support. Additional information, including how to include Spring annotations inside applications, can be found in the JBoss Data Grid *Developer Guide*.

37.2. ENABLING SPRING CACHE SUPPORT DECLARATIVELY (LIBRARY MODE)

Spring's cache support can be enabled through the xml file by performing the following steps:

1. Add `<cache:annotation-driven/>` to the xml file. This line enables the standard spring annotations to be used by the application.
2. Define a cache manager using the `<infinispan:embedded-cache-manager ... />`.

The following example demonstrates these changes:

Sample Declarative Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:infinispan="http://www.infinispan.org/schemas/spring"
       xmlns:cache="http://www.springframework.org/schema/cache"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache.xsd
http://www.infinispan.org/schemas/spring
http://www.infinispan.org/schemas/infinispan-spring.xsd">
[... ]
<cache:annotation-driven/>

<infinispan:embedded-cache-manager
       configuration="classpath:/path/to/cache-config.xml"/>
[... ]
```

37.3. ENABLING SPRING CACHE SUPPORT DECLARATIVELY (REMOTE CLIENT-SERVER MODE)

Spring's cache support can be enabled declaratively by performing the following steps:

1. Add `<cache:annotation-driven/>` to the xml file. This line enables the standard spring annotations to be used by the application.
2. Define the HotRod client properties using the `<infinispan:remote-cache-manager ... />`.

The following example demonstrates these changes:

Sample Declarative Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:infinispan="http://www.infinispan.org/schemas/spring"
       xmlns:cache="http://www.springframework.org/schema/cache"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache.xsd
http://www.infinispan.org/schemas/spring
http://www.infinispan.org/schemas/infinispan-spring.xsd">
[... ]
<cache:annotation-driven/>

<infinispan:remote-cache-manager
       configuration="classpath:/path/to/hotrod-client.properties"/>
[... ]
```

CHAPTER 38. HIGH AVAILABILITY USING SERVER HINTING

38.1. HIGH AVAILABILITY USING SERVER HINTING

In Red Hat JBoss Data Grid, Server Hinting ensures that backed up copies of data are not stored on the same physical server, rack, or data center as the original. Server Hinting does not apply to total replication because total replication mandates complete replicas on every server, rack, and data center.

Data distribution across nodes is controlled by the Consistent Hashing mechanism. JBoss Data Grid offers a pluggable policy to specify the consistent hashing algorithm. For details on configuring this policy refer to the **ConsistentHashFactories** section in the JBoss Data Grid *Developer Guide*.

Setting a **machineId**, **rackId**, or **siteId** in the transport configuration will trigger the use of **TopologyAwareConsistentHashFactory**, which is the equivalent of the **DefaultConsistentHashFactory** with Server Hinting enabled.

Server Hinting is particularly important when ensuring the high availability of your JBoss Data Grid implementation.

38.2. ESTABLISHING SERVER HINTING WITH JGROUPS

When setting up a clustered environment in Red Hat JBoss Data Grid, Server Hinting is configured when establishing JGroups configuration.

JBoss Data Grid ships with several JGroups files pre-configured for clustered mode. These files can be used as a starting point when configuring Server Hinting in JBoss Data Grid.

See Also: [Pre-Configured JGroups Files](#)

38.3. CONFIGURE SERVER HINTING (REMOTE CLIENT-SERVER MODE)

In Red Hat JBoss Data Grid's Remote Client-Server mode, Server Hinting is configured in the JGroups subsystem on the **transport** element for the default stack, as follows:

Configure Server Hinting in Remote Client-Server Mode

```
<subsystem xmlns="urn:jboss:domain:jgroups:3.0"
  default-stack="${jboss.default.jgroups.stack:udp}">
  <stack name="udp">
    <transport type="UDP"
      socket-binding="jgroups-udp"
      site="${jboss.jgroups.transport.site:s1}"
      rack="${jboss.jgroups.transport.rack:r1}"
      machine="${jboss.jgroups.transport.machine:m1}">
      <!-- Additional configuration elements here -->
    </transport>
  </stack>
</subsystem>
```

1. Find the JGroups subsystem configuration
2. Enable Server Hinting via the **transport** Element

- a. Set the site ID using the **site** parameter.
- b. Set the rack ID using the **rack** parameter.
- c. Set the machine ID using the **machine** parameter.

38.4. CONFIGURE SERVER HINTING (LIBRARY MODE)

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

Configure Server Hinting for Library Mode

The following configuration attributes are used to configure Server Hinting in JBoss Data Grid.

```
<transport cluster = "MyCluster"
           machine = "LinuxServer01"
           rack = "Rack01"
           site = "US-WestCoast" />
```

1. The **cluster** attribute specifies the name assigned to the cluster.
2. The **machine** attribute specifies the JVM instance that contains the original data. This is particularly useful for nodes with multiple JVMs and physical hosts with multiple virtual hosts.
3. The **rack** attribute specifies the rack that contains the original data, so that other racks are used for backups.
4. The **site** attribute differentiates between nodes in different data centers replicating to each other.

The listed parameters are optional in a JBoss Data Grid configuration.

If **machine**, **rack**, or **site** are included in the configuration, **TopologyAwareConsistentHashFactory** is selected automatically, enabling Server Hinting. However, if Server Hinting is not configured, JBoss Data Grid's distribution algorithms are allowed to store replications in the same physical machine/rack/data center as the original data.

CHAPTER 39. SET UP CROSS-DATACENTER REPLICATION

39.1. CROSS-DATACENTER REPLICATION

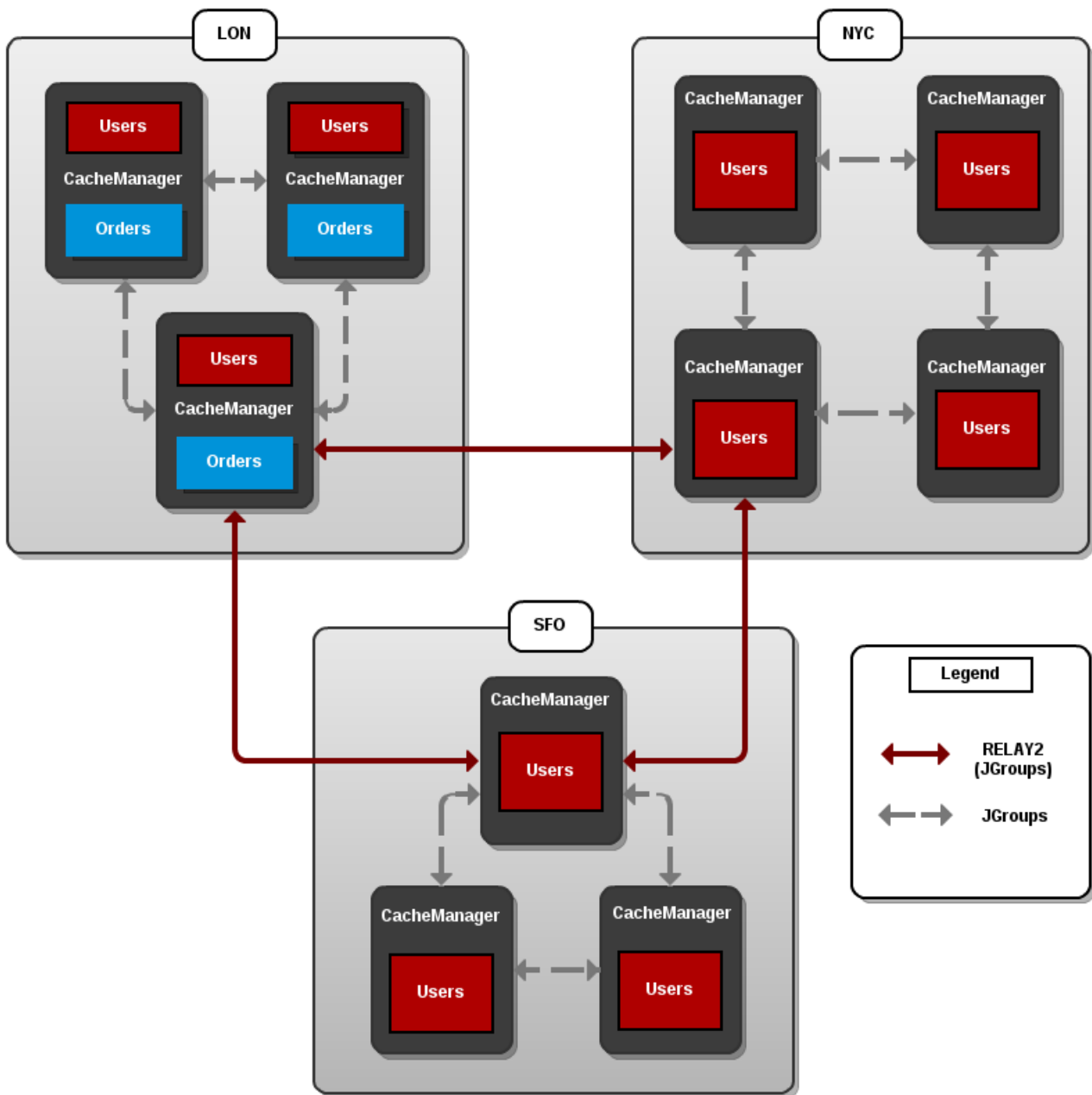
In Red Hat JBoss Data Grid, Cross-Datacenter Replication allows the administrator to create data backups in multiple clusters. These clusters can be at the same physical location or different ones. JBoss Data Grid's Cross-Site Replication implementation is based on JGroups' *RELAY2* protocol.

Cross-Datacenter Replication ensures data redundancy across clusters. In addition to creating backups for data restoration, these datasets may also be used in an active-active mode. When configured in this manner systems in separate environments are able to handle sessions should one cluster fail. Ideally, each of these clusters should be in a different physical location than the others.

39.2. CROSS-DATACENTER REPLICATION OPERATIONS

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

Figure 39.1. Cross-Datcenter Replication Example



Three sites are configured in this example: **LON**, **NYC** and **SFO**. Each site hosts a running JBoss Data Grid cluster made up of three to four physical nodes.

The **Users** cache is active in all three sites - **LON**, **NYC** and **SFO**. Changes to the **Users** cache at the any one of these sites will be replicated to the other two as long as the cache defines the other two sites as its backups through configuration. The **Orders** cache, however, is only available locally at the **LON** site because it is not replicated to the other sites.

The **Users** cache can use different replication mechanisms each site. For example, it can back up data synchronously to **SFO** and asynchronously to **NYC** and **LON**.

The **Users** cache can also have a different configuration from one site to another. For example, it can be configured as a distributed cache with **owners** set to **2** in the **LON** site, as a replicated cache in the **NYC** site and as a distributed cache with **owners** set to **1** in the **SFO** site.

JGroups is used for communication within each site as well as inter-site communication. Specifically, a JGroups protocol called *RELAY2* facilitates communication between sites. For more information, see [About RELAY2](#).

39.3. CONFIGURE CROSS-DATACENTER REPLICATION

39.3.1. Configure Cross-Datacenter Replication (Remote Client-Server Mode)

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

Set Up Cross-Datacenter Replication

1. Set Up RELAY

Add the following configuration to the *standalone.xml* file to set up *RELAY* :

```
<subsystem xmlns="urn:infinispan:server:jgroups:8.0">
  <channels default="cluster">
    <channel name="cluster"/>
    <channel name="xsite" stack="tcp"/>
  </channels>
  <stacks default="udp">
    <stack name="udp">
      <transport type="UDP" socket-binding="jgroups-udp"/>
      <...other protocols...>
      <relay site="LON">
        <remote-site name="NYC" channel="xsite"/>
        <remote-site name="SFO" channel="xsite"/>
      </relay>
    </stack>
  </stacks>
</subsystem>
```

The *RELAY* protocol creates an additional stack (running parallel to the existing *UDP* stack) to communicate with the remote site. In the above example the **xsite** channel references the current **tcp** stack. If a *TCP* based stack is used for the local cluster, two *TCP* based stack configurations are required: one for local communication and one to connect to the remote site. For an illustration, see [Cross-Datacenter Replication Operations](#).

2. Set Up Sites

Use the following configuration in the *standalone.xml* file to set up sites for each distributed cache in the cluster:

```
<distributed-cache name="namedCache">
  <!-- Additional configuration elements here -->
  <backups>
    <backup site="{FIRSTSITE}" strategy="{SYNC/ASYNC}" />
    <backup site="{SECONDSITE}" strategy="{SYNC/ASYNC}" />
  </backups>
</distributed-cache>
```

3. Configure the Transport

In our first step we indicated the remote datacenters would be connecting over *TCP*, via the **xsite** channel. Now the *TCP* stack is configured to point to the remote sites:

```

<stack name="tcp">
  <transport type="TCP" socket-binding="jgroups-tcp"/>
  <protocol type="TCPPING">
    <property
name="initial_hosts">lon.hostname[7600],nyc.hostname[7600],sfo.hostn
ame[7600]</property>
    <property name="ergonomics">>false</property>
  </protocol>
  <!-- Additional configuration elements here -->
</stack>

```

4. Configure the Backup Sites

Repeat steps 1-3 for each site in this configuration, adjusting the site names in the RELAY configuration as appropriate.

A cross-datacenter example configuration file may be found at `$JDG_SERVER/docs/examples/configs/clustered-xsite.xml`.

39.3.2. Configure Cross-Datacenter Replication (Library Mode)

39.3.2.1. Configure Cross-Datacenter Replication Declaratively

When configuring Cross-Datacenter Replication, the `relay.RELAY2` protocol creates an additional stack (running parallel to the existing `TCP` stack) to communicate with the remote site. If a `TCP`-based stack is used for the local cluster, two `TCP` based stack configurations are required: one for local communication and one to connect to the remote site.

In JBoss Data Grid's Library mode, cross-datacenter replication is set up as follows:

Setting Up Cross-Datacenter Replication

1. Configure the Local Site

```

<infinispan
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:infinispan:config:8.4
http://www.infinispan.org/schemas/infinispan-config-8.4.xsd"
  xmlns="urn:infinispan:config:8.4">

  <jgroups>
    <stack-file name="udp" path="jgroups-with-relay.xml"/>
  </jgroups>

  <cache-container default-cache="default">
    <transport cluster="infinispan-cluster" lock-timeout="50000"
      stack="udp" node-name="node1"
      machine="machine1" rack="rack1" site="LON"/>
    <local-cache name="default">
      <backups>
        <backup site="NYC" strategy="SYNC" failure-
policy="IGNORE" timeout="12003"/>
        <backup site="SFO" strategy="ASYNC"/>
      </backups>
    </local-cache>

```

```

<!-- Additional configuration information here -->
</infinispan>

```

- a. Add the **site** attribute to the **transport** element to define the local site (in this example, the local site is named **LON**).
- b. Cross-site replication requires a non-default JGroups configuration. Define the **jgroups** element and define a custom **stack-file**, passing in the name of the file to be referenced and the location to this custom configuration. In this example, the JGroups configuration file is named *jgroups-with-relay.xml*.
- c. Configure the cache in site **LON** to back up to the sites **NYC** and **SFO**.
- d. Configure the back up caches:
 - i. Configure the cache in site **NYC** to receive back up data from **LON**:

```

<local-cache name="backupNYC">
  <backups/>
  <backup-for remote-cache="default" remote-site="LON"/>
</local-cache>

```

- ii. Configure the cache in site **SFO** to receive back up data from **LON**:

```

<local-cache name="backupSFO">
  <backups/>
  <backup-for remote-cache="default" remote-site="LON"/>
</local-cache>

```

2. Add the Contents of the Configuration File

As a default, Red Hat JBoss Data Grid includes JGroups configuration files such as *default-configs/default-jgroups-tcp.xml* and *default-configs/default-jgroups-udp.xml* in the *infinispan-embedded-{VERSION}.jar* package.

Copy the JGroups configuration to a new file (in this example, it is named *jgroups-with-relay.xml*) and add the provided configuration information to this file. Note that the *relay.RELAY2* protocol configuration must be the last protocol in the configuration stack.

```

<config>
  ...
  <relay.RELAY2 site="LON"
    config="relay.xml"
    relay_multicasts="false" />
</config>

```

3. Configure the relay.xml File

Set up the *relay.RELAY2* configuration in the *relay.xml* file. This file describes the global cluster configuration.

```

<RelayConfiguration>
  <sites>
    <site name="LON"
      id="0">
    <bridges>

```



```

        <bridge config="jgroups-global.xml"
            name="global"/>
    </bridges>
</site>
<site name="NYC"
    id="1">
    <bridges>
        <bridge config="jgroups-global.xml"
            name="global"/>
    </bridges>
</site>
<site name="SFO"
    id="2">
    <bridges>
        <bridge config="jgroups-global.xml"
            name="global"/>
    </bridges>
</site>
</sites>
</RelayConfiguration>

```

4. Configure the Global Cluster

The file *jgroups-global.xml* referenced in *relay.xml* contains another JGroups configuration which is used for the global cluster: communication between sites.

The global cluster configuration is usually *TCP*-based and uses the *TCPPING* protocol (instead of *PING* or *MPING*) to discover members. Copy the contents of *default-configs/default-jgroups-tcp.xml* into *jgroups-global.xml* and add the following configuration in order to configure *TCPPING*:

```

<config>
    <TCP bind_port="7800" ... />
    <TCPPING
initial_hosts="lon.hostname[7800],nyc.hostname[7800],sfo.hostname[78
00]"
        ergonomics="false" />
    <!-- Rest of the protocols -->
</config>

```

Replace the hostnames (or IP addresses) in **TCPPING.initial_hosts** with those used for your site masters. The ports (**7800** in this example) must match the **TCP.bind_port**.

For more information about the *TCPPING* protocol, see [Using the TCPing Protocol](#).

39.4. TAKING A SITE OFFLINE

39.4.1. Taking a Site Offline

In Red Hat JBoss Data Grid's Cross-datacenter replication configuration, if backing up to one site fails a certain number of times during a time interval, that site can be marked as offline automatically. This feature removes the need for manual intervention by an administrator to mark the site as offline.

It is possible to configure JBoss Data Grid to take down a site automatically when specified conditions are met, or for an administrator to manually take down a site:

- Configure automatically taking a site offline:
 - Declaratively in Remote Client-Server mode.
 - Declaratively in Library mode.
 - Using the programmatic method.
- Manually taking a site offline:
 - Using JBoss Operations Network (JON).
 - Using the JBoss Data Grid Command Line Interface (CLI).

39.4.2. Taking a Site Offline

To take a site offline in either mode of Red Hat JBoss Data Grid's add the **take-offline** element to the **backup** element. This will configure when a site is automatically taken offline.

Taking a Site Offline in Remote Client-Server Mode

```
<backup>
  <take-offline after-failures="${NUMBER}"
    min-wait="${PERIOD}" />
</backup>
```

The **take-offline** element use the following parameters to configure when to take a site offline:

- The **after-failures** parameter specifies the number of times attempts to contact a site can fail before the site is taken offline.
- The **min-wait** parameter specifies the number (in milliseconds) to wait to mark an unresponsive site as offline. The site is offline when the **min-wait** period elapses after the first attempt, and the number of failed attempts specified in the **after-failures** parameter occur.

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

A site can be taken offline in Red Hat JBoss Data Grid using the JBoss Operations Network operations. For a list of the metrics, see [JBoss Operations Network Plugin Operations](#).

39.4.4. Taking a Site Offline via the CLI

Use Red Hat JBoss Data Grid's Command Line Interface (CLI) to manually take a site from a cross-datacenter replication configuration down if it is unresponsive using the **site** command.

The **site** command can be used to check the status of a site as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache] site --status
${SITENAME}
```

The result of this command would either be **online** or **offline** according to the current status of the named site.

The command can be used to bring a site online or offline by name as follows:

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache] site --offline
${SITENAME}
```

```
[standalone@127.0.0.1:9990/MyCacheManager/namedCache] site --online
${SITENAME}
```

If the command is successful, the output **ok** displays after the command. As an alternate, the site can also be brought online using JMX (see [Bring a Site Back Online](#) for details).

For more information about the JBoss Data Grid CLI and its commands, see the *Developer Guide's* chapter on the JBoss Data Grid Command Line Interface (CLI)

39.4.5. Bring a Site Back Online

After a site is taken offline, the site can be brought back online either using the JMX console to invoke the **bringSiteOnline([replaceable]siteName)** operation on the **XSiteAdmin** MBean (See [XSiteAdmin](#) for details) or using the CLI (see [Taking a Site Offline via the CLI](#) for details).

39.5. STATE TRANSFER BETWEEN SITES

39.5.1. State Transfer Between Sites

When an offline master site is back online, it is necessary to synchronize its state with the latest data from the backup site. State transfer allows state to be transferred from one site to another, meaning the master site is synchronized and made consistent with the backup site. Similarly, when a backup site becomes available, state transfer can be utilized to make it consistent with the master site.

Consider a scenario of two sites - Master site A and Backup site B. Clients can originally access only Master site A whereas Backup Site B acts as an invisible backup. Cross Site State Transfer can be pushed bidirectionally. When the new backup site B goes online, in order to synchronize its state with the master site A, a State Transfer can be initiated to push the state from the Master site A to the Backup site B.

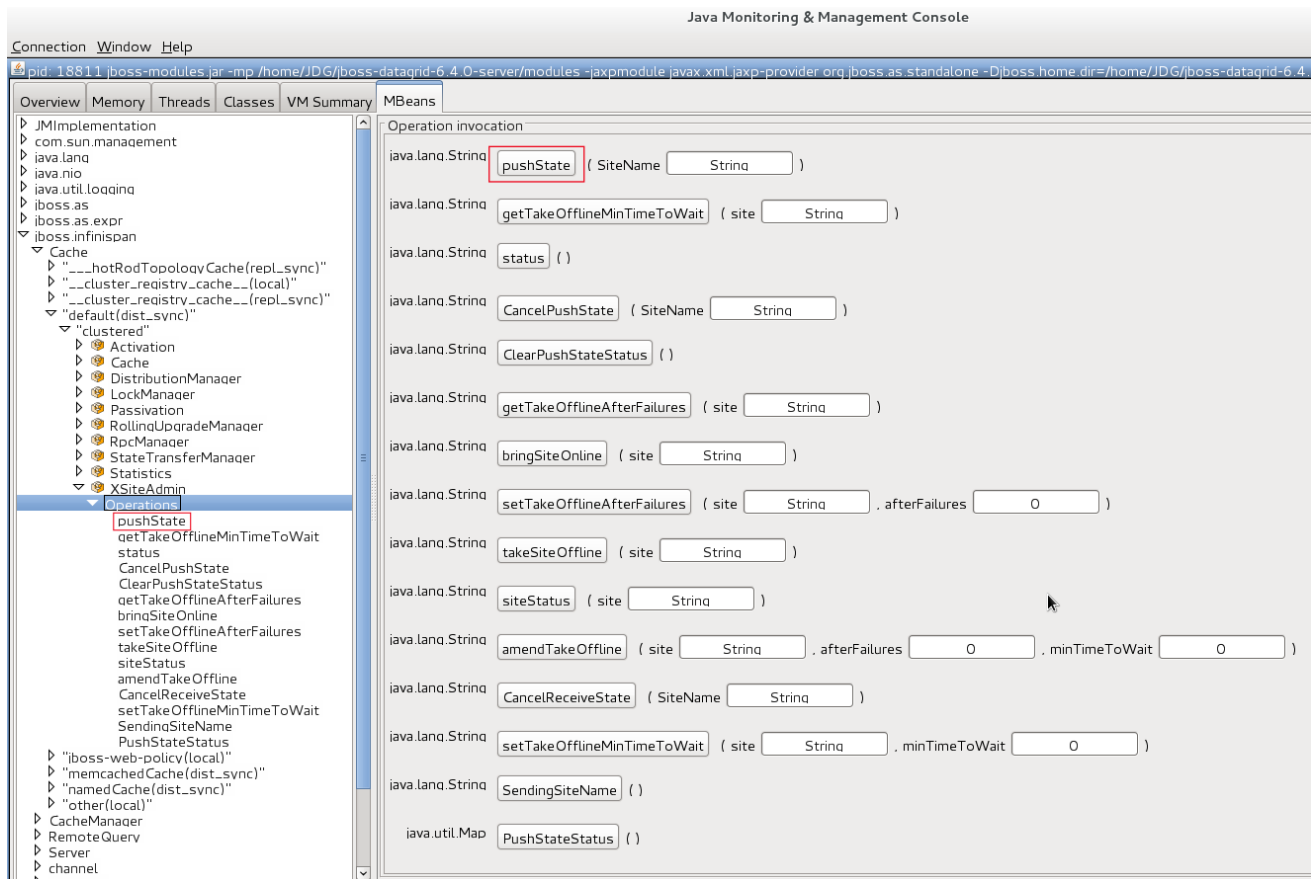
Similarly, when the Master site A is brought back online, in order to synchronize it with the Backup site B, a State Transfer can be initiated to push the state from Backup site B to Master Site A.

The use cases applies for both Active-Passive and Active-Active State Transfer. The difference is that during Active-Active State Transfer we assume that cache operations can be performed in the site, which consumes state.

A system administrator or an authorized entity initiates the state transfer manually using JMX. The system administrator invokes the **pushState(SiteName String)** operation available in the **XSiteAdminOperations** MBean.

The following interface shows the **pushState(SiteName String)** operation in JConsole:

Figure 39.2. PushState Operation



State transfer is also invoked using the Command Line Interface (CLI) by the `site push sitename` command. For example, when the master site is brought back online, the system administrator invokes the state transfer operation in the backup site, specifying the master site name that is to receive the state.

The master site can be offline at the time of the push operation. On successful state transfer, the state data common to both the sites is overwritten on the master site. For example, if key A exists on the master site but not on the backup site, key A will not be deleted from the master site. Whereas, if key B exists on the backup as well as the master site, key B is overwritten on the master site.



NOTE

Updates on keys performed after initiating state transfer are not overwritten by incoming state transfer.

Cross-site state transfer can be transactional and supports 1PC and 2PC transaction options. 1PC and 2PC options define whether data modified inside a transaction is backed up to a remote site in one or two phases. 2PC includes a prepare phase in which backup sites acknowledges that transaction has been successfully prepared. Both options are supported.

39.5.2. Active-Passive State Transfer

The active-passive state transfer is used when cross-site replication is used to back up the master site. The master site processes all the requests but if it goes offline, the backup site starts to handle them. When the master site is back online, it receives the state from the backup site and starts to handle the client requests. In Active-Passive state transfer mode, transactional writes happen concurrently with state transfer on the site which sends the state.

In active-passive state transfer mode, the client read-write requests occurs only on the backup site. The master site acts as an invisible backup until the client requests are switched to it when the state transfer is completed. The active-passive state transfer mode is fully supported in cross-datacenter replication.

When an Active-Passive State Transfer is interrupted by a network failure, the System Administrator invokes the JMX operation manually to resume the state transfer. To transfer the state, for example from Master site A to Backup site B, invoke the JMX operation on Master site A. Similarly, to transfer state from Backup site B to Master site A, invoke the JMX operation on the Backup site B.

The JMX operation is invoked on the site from which the state is transferred to the other site that is online to synchronize the states.

For example, there is a running backup site and the system administrator wants to bring back the master site online. To use active-passive state transfer, the system administrator will perform the following steps.

- Boot the Red Hat JBoss Data Grid cluster in the master site.
- Command the backup site to push state to the master site.
- Wait until the state transfer is complete.
- Make the clients aware that the master site is available to process the requests.

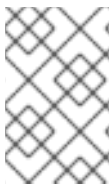
39.5.3. Active-Active State Transfer

In active-active state transfer mode, the client requests occur concurrently in both the sites while the state transfer is in progress. The current implementation supports handling requests in the new site while the state transfer is in progress, which may break the data consistency.



WARNING

Active-active state transfer mode is not fully supported, as it may lead to data inconsistencies.



NOTE

In active-active state transfer mode, both the sites, the master and the backup sites share the same role. There is no clear distinction between the master and backup sites in the active-active state transfer mode

For example, there is a running site and the system administrator wants to bring a new site online. To use active-active state transfer, the system administrator must perform the following steps.

- Boot the Red Hat JBoss Data Grid cluster in the new site.
- Command the running site to push state to the new site.
- Make the clients aware that the new site is available to process the requests.

39.5.4. State Transfer Configuration

State transfer between sites is not enabled or disabled but it allows to tune some parameters. The only configuration is done by the system administrator while configuring the load balancer to switch the request to the master site during or after the state transfer. The implementation handles a case in which a key is updated by a client before it receives the state, ignoring when it is delivered.

The following are default parameter values:

```
<backups>
  <backup site="NYC"
    strategy="SYNC"
    failure-policy="FAIL">
    <state-transfer chunk-size="512"
      timeout="1200000"
      max-retries="30"
      wait-time="2000" />
  </backup>
</backups>
```

39.6. CONFIGURE MULTIPLE SITE MASTERS

39.6.1. Configure Multiple Site Masters

A standard Red Hat JBoss Data Grid cross-datacenter replication configuration includes one master node for each site. The master node is a gateway for other nodes to communicate with the master nodes at other sites.

This standard configuration works for a simple cross-datacenter replication configuration. However, with a larger volume of traffic between the sites, passing traffic through a single master node can create a bottleneck, which slows communication across nodes.

In JBoss Data Grid, configure multiple master nodes for each site to optimize traffic across multiple sites.

39.6.2. Multiple Site Master Operations

When multiple site masters are enabled and configured, the master nodes in each site joins the local cluster (i.e. the local site) as well as the global cluster (which includes nodes that are members of multiple sites).

Each node that acts as a site master and maintains a routing table that consists of a list of target sites and site masters. When a message arrives, a random master node for the destination site is selected. The message is then forwarded to the random master node, where it is sent to the destination node (unless the randomly selected node was the destination).

39.6.3. Configure Multiple Site Masters (Remote Client-Server Mode)

Prerequisites

Configure Cross-Datacenter Replication for Red Hat JBoss Data Grid's Remote Client-Server Mode.

Set Multiple Site Masters in Remote Client-Server Mode

```
<relay site="LON">
  <remote-site name="NYC" stack="tcp" cluster="global"/>
  <remote-site name="SFO" stack="tcp" cluster="global"/>
```

```

<property name="relay_multicasts">false</property>
<property name="max_site_masters">16</property>
<property name="can_become_site_master">true</property>
</relay>

```

1. Locate the Target Configuration

Locate the target site's configuration in the *clustered-xsite.xml* example configuration file. The sample configuration looks like example provided above.

2. Configure Maximum Sites

Use the **max_site_masters** property to determine the maximum number of master nodes within the site. Set this value to the number of nodes in the site to make every node a master.

3. Configure Site Master

Use the **can_become_site_master** property to allow the node to become the site master. This flag is set to **true** as a default. Setting this flag to **false** prevents the node from becoming a site master. This is required in situations where the node does not have a network interface connected to the external network.

39.6.4. Configure Multiple Site Masters (Library Mode)

To configure multiple site masters in Red Hat JBoss Data Grid's Library Mode:

Configure Multiple Site Masters (Library Mode)

1. Configure Cross-Datcenter Replication

Configure Cross-Datcenter Replication in JBoss Data Grid. Use the instructions in [Configure Cross-Datcenter Replication Declaratively](#) for an XML configuration. For instructions on a programmatic configuration refer to the JBoss Data Grid *Developer Guide*.

2. Add the Contents of the Configuration File

Add the **can_become_site_master** and **max_site_masters** parameters to the configuration as follows:

```

<config>
  <!-- Additional configuration information here -->
  <relay.RELAY2 site="LON"
    config="relay.xml"
    relay_multicasts="false"
    can_become_site_master="true"
    max_site_masters="16"/>
</config>

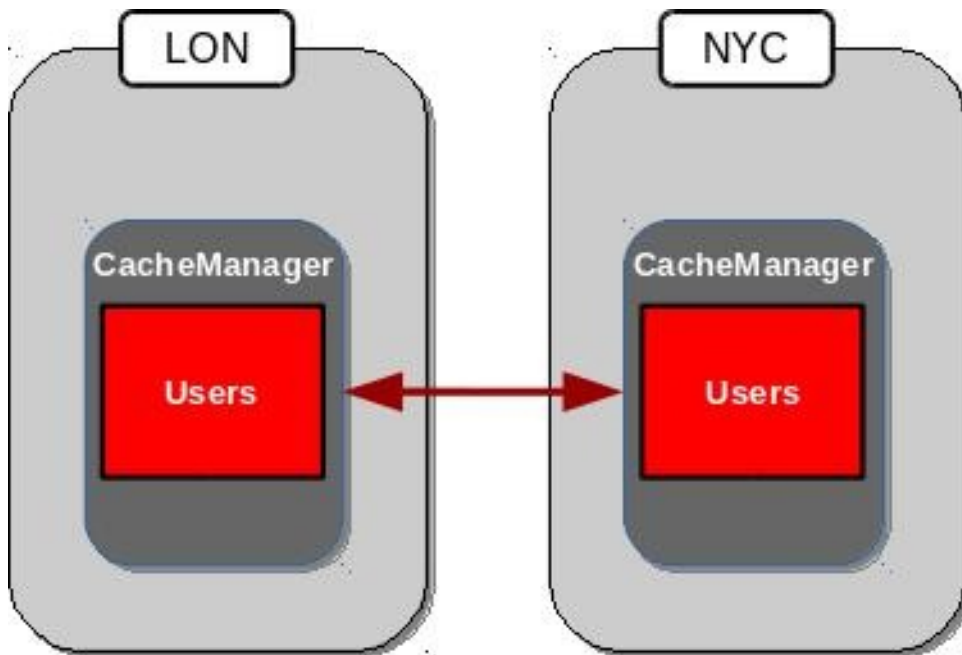
```

Set the **max_site_masters** value to the number of nodes in the cluster to make all nodes masters.

39.7. CROSS-DATACENTER REPLICATION CONCERNS

When using Cross-Datcenter Replication in active-active mode, where each site is using the other as an active backup, there may be issues if the same key is written to both locations simultaneously.

Consider the following image:



In this scenario both LON and NYC are backing up the **Users** cache to the other location. If the same entry were edited in both locations simultaneously, each site would update their local copy and then replicate this copy to the other site. This replication would overwrite the local copy's value with the newly received, replicated, copy, resulting in LON and NYC swapping the expected value. The following example demonstrates this:

1. Both LON and NYC have an entry in the **Users** cache with a key of **1** and a value of **Smith**.
2. LON updates the value to **Johnson**.
3. NYC updates the value to **Williams**.
4. The cache is replicated to its backups, resulting in LON containing a value of **Williams**, and NYC containing a value of **Johnson**.

CHAPTER 40. ROLLING UPGRADES

40.1. ROLLING UPGRADES

In Red Hat JBoss Data Grid, rolling upgrades permit a cluster to be upgraded from one version to a new version without experiencing any downtime. This allows nodes to be upgraded without the need to restart the application or risk losing data.

In JBoss Data Grid, rolling upgrades can only be performed in Remote Client-Server mode.



IMPORTANT

When performing a rolling upgrade it is recommended to not update any cache entries in the source cluster, as this may lead to data inconsistency.

40.2. ROLLING UPGRADES USING HOT ROD

The following process is used to perform rolling upgrades on Red Hat JBoss Data Grid running in Remote Client-Server mode, using Hot Rod. This procedure is designed to upgrade the data grid itself, and does not upgrade the client application.



IMPORTANT

Ensure that the correct version of the Hot Rod protocol is used with your JBoss Data Grid version. This version must be specified on the client programmatically, and instructions on defining this are found inside the JBoss Data Grid *Developer Guide*. A list of Hot Rod protocol versions from each release are found below:

- For JBoss Data Grid 7.1, use Hot Rod protocol version 2.5
- For JBoss Data Grid 7.0, use Hot Rod protocol version 2.5
- For JBoss Data Grid 6.6, use Hot Rod protocol version 2.3
- For JBoss Data Grid 6.5, use Hot Rod protocol version 2.0
- For JBoss Data Grid 6.4, use Hot Rod protocol version 2.0
- For JBoss Data Grid 6.3, use Hot Rod protocol version 2.0
- For JBoss Data Grid 6.2, use Hot Rod protocol version 1.3
- For JBoss Data Grid 6.1, use Hot Rod protocol version 1.2

Prerequisite

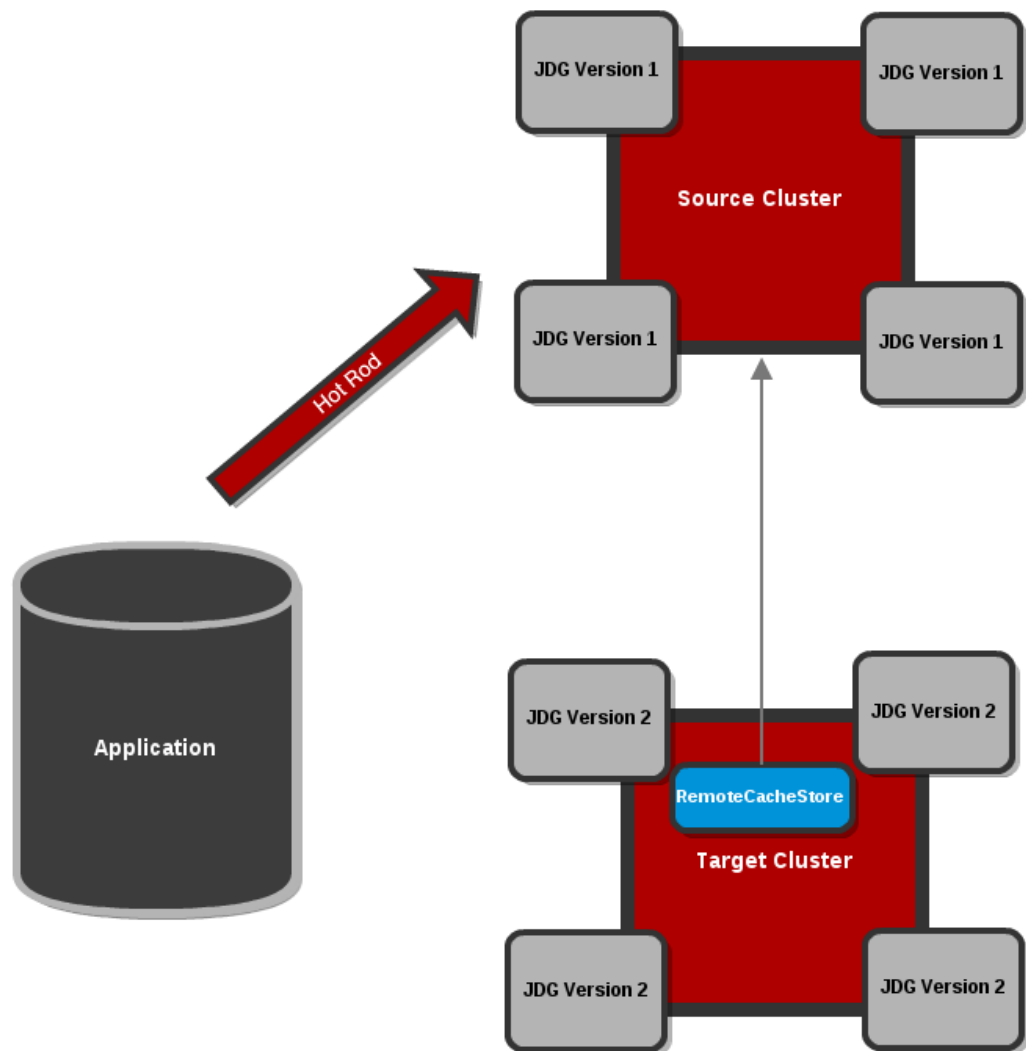
This procedure assumes that a cluster is already configured and running, and that it is using an older version of JBoss Data Grid. This cluster is referred to below as the Source Cluster and the Target Cluster refers to the new cluster to which data will be migrated.

1. Configure the Target Cluster

Use either different network settings or a different JGroups cluster name to set the Target Cluster (consisting of nodes with new JBoss Data Grid) apart from the Source Cluster. For each cache, configure a **RemoteCacheStore** with the following settings:

- a. Ensure that **remote-server** points to the Source Cluster.
- b. Ensure that the cache name matches the name of the cache on the Source Cluster.
- c. Ensure that **hotrod-wrapping** is enabled (set to **true**).
- d. Ensure that **purge** is disabled (set to **false**).
- e. Ensure that **passivation** is disabled (set to **false**).

Figure 40.1. Configure the Target Cluster with a RemoteCacheStore



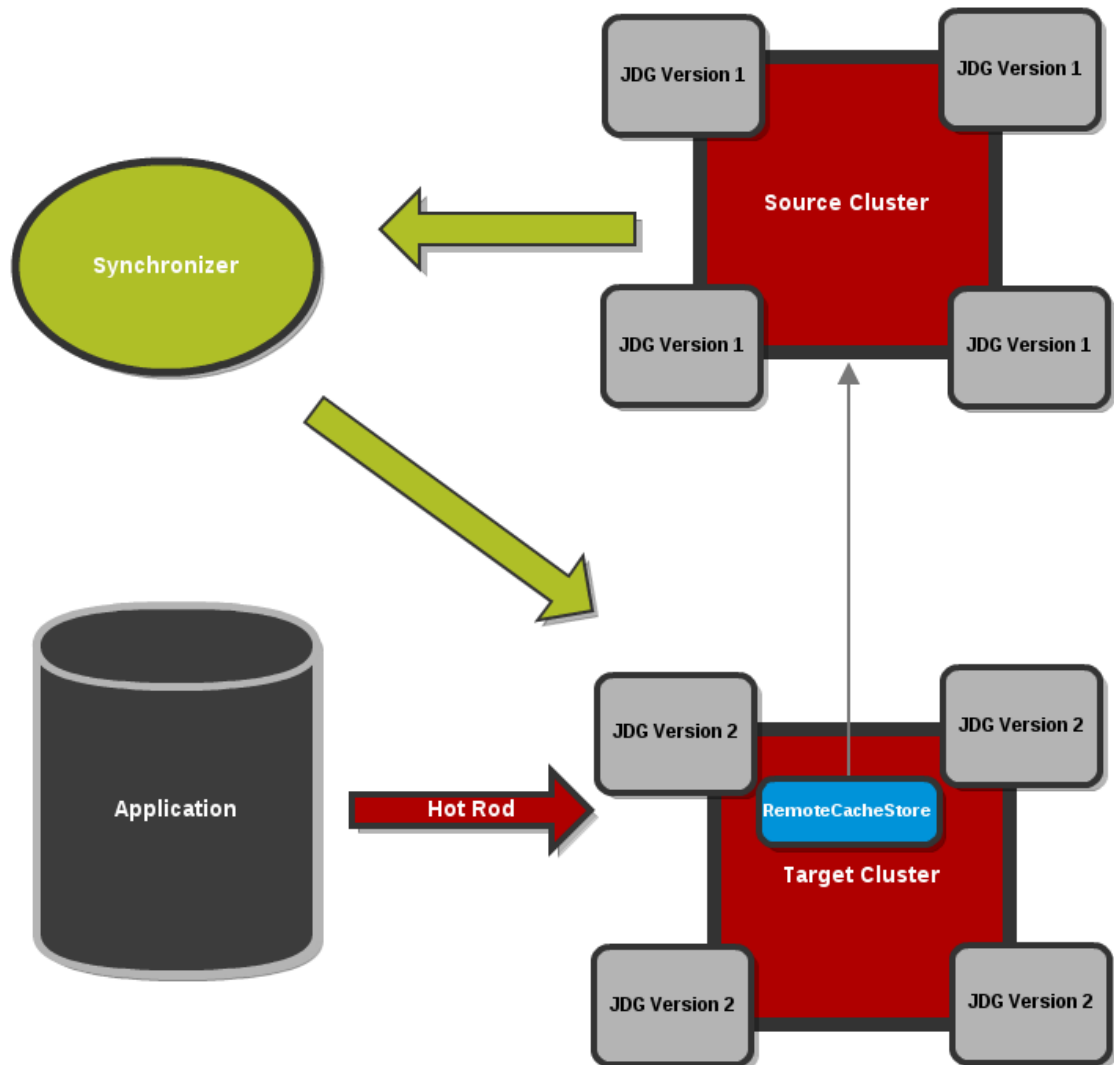
NOTE

See the `$JDG_HOME/docs/examples/configs/[path]standalone-hotrod-rolling-upgrade.xml` file for a full example of the Target Cluster configuration for performing Rolling Upgrades.

2. Start the Target Cluster

Start the Target Cluster's nodes. Configure each client to point to the Target Cluster instead of the Source Cluster. Eventually, the Target Cluster handles all requests instead of the Source Cluster. The Target Cluster then lazily loads data from the Source Cluster on demand using the **RemoteCacheStore**.

Figure 40.2. Clients point to the Target Cluster with the Source Cluster as RemoteCacheStore for the Target Cluster.



3. Dump the Source Cluster keyset

When all connections are using the Target Cluster, the keyset on the Source Cluster must be dumped. This can be done using either JMX or the CLI:

a. JMX

Invoke the **recordKnownGlobalKeyset** operation on the **RollingUpgradeManager** MBean on the Source Cluster for every cache that must be migrated.

b. CLI

Invoke the **upgrade --dumpkeys** command on the Source Cluster for every cache that must be migrated, or use the **--all** switch to dump all caches in the cluster.

4. Fetch remaining data from the Source Cluster

The Target Cluster fetches all remaining data from the Source Cluster. Again, this can be done using either JMX or CLI:

a. JMX

Invoke the **synchronizeData** operation and specify the **hotrod** parameter on the **RollingUpgradeManager** MBean on the Target Cluster for every cache that must be migrated.

- b. CLI
Invoke the **upgrade --synchronize=hotrod** command on the Target Cluster for every cache that must be migrated, or use the **--all** switch to synchronize all caches in the cluster.
5. Disabling the RemoteCacheStore
Once the Target Cluster has obtained all data from the Source Cluster, the **RemoteCacheStore** on the Target Cluster must be disabled. This can be done as follows:
 - a. JMX
Invoke the **disconnectSource** operation specifying the **hotrod** parameter on the **RollingUpgradeManager** MBean on the Target Cluster.
 - b. CLI
Invoke the **upgrade --disconnectsource=hotrod** command on the Target Cluster.
6. Decommission the Source Cluster
As a final step, decommission the Source Cluster.

40.3. ROLLING UPGRADES USING REST

The following procedure outlines using Red Hat JBoss Data Grid installations as a remote grid using the REST protocol. This procedure applies to rolling upgrades for the grid, not the client application.

Perform Rolling Upgrades Using REST

In the instructions, the Source Cluster refers to the old cluster that is currently in use and the Target Cluster refers to the destination cluster for our data.

1. Configure the Target Cluster
Use either different network settings or a different JGroups cluster name to set the Target Cluster (consisting of nodes with new JBoss Data Grid) apart from the Source Cluster. For each cache, configure a **RestCacheStore** with the following settings:
 - a. Ensure that the host and port values point to the Source Cluster.
 - b. Ensure that the path value points to the Source Cluster's REST endpoint.
2. Start the Target Cluster
Start the Target Cluster's nodes. Configure each client to point to the Target Cluster instead of the Source Cluster. Eventually, the Target Cluster handles all requests instead of the Source Cluster. The Target Cluster then lazily loads data from the Source Cluster on demand using the **RestCacheStore**.
3. Do not dump the Key Set during REST Rolling Upgrades
The REST Rolling Upgrades use case is designed to fetch all the data from the Source Cluster without using the **recordKnownGlobalKeyset** operation.

**WARNING**

Do not invoke the **recordKnownGlobalKeyset** operation for REST Rolling Upgrades. If you invoke this operation, it will cause data corruption and REST Rolling Upgrades will not complete successfully.

4. Fetch the Remaining Data

The Target Cluster must fetch all the remaining data from the Source Cluster. This is done either using JMX or the CLI as follows:

a. Using JMX

Invoke the **synchronizeData** operation with the **rest** parameter specified on the **RollingUpgradeManager** MBean on the Target Cluster for all caches to be migrated.

b. Using the CLI

Run the **upgrade --synchronize=rest** on the Target Cluster for all caches to be migrated. Optionally, use the **--all** switch to synchronize all caches in the cluster.

5. Disable the RestCacheStore

Disable the **RestCacheStore** on the Target Cluster using either JMX or the CLI as follows:

a. Using JMX

Invoke the **disconnectSource** operation with the **rest** parameter specified on the **RollingUpgradeManager** MBean on the Target Cluster.

b. Using the CLI

Run the **upgrade --disconnectsource=rest** command on the Target Cluster. Optionally, use the **--all** switch to disconnect all caches in the cluster.

Result

Migration to the Target Cluster is complete. The Source Cluster can now be decommissioned.

40.4. ROLLINGUPGRADEMANAGER OPERATIONS

The **RollingUpgradeManager** Mbean handles the operations that allow data to be migrated from one version of Red Hat JBoss Data Grid to another when performing rolling upgrades. The **RollingUpgradeManager** operations are:

- **recordKnownGlobalKeyset** retrieves the entire keyset from the cluster running on the old version of JBoss Data Grid.
- **synchronizeData** performs the migration of data from the Source Cluster to the Target Cluster, which is running the new version of JBoss Data Grid.
- **disconnectSource** disables the Source Cluster, the older version of JBoss Data Grid, once data migration to the Target Cluster is complete.

40.5. REMOTECACHESTORE PARAMETERS FOR ROLLING UPGRADES

40.5.1. rawValues and RemoteCacheStore

By default, the RemoteCacheStore store's values are wrapped in InternalCacheEntry. Enabling the **rawValues** parameter causes the raw values to be stored instead for interoperability with direct access by RemoteCacheManagers.

rawValues must be enabled in order to interact with a Hot Rod cache via both RemoteCacheStore and RemoteCacheManager.

40.5.2. hotRodWrapping

The **hotRodWrapping** parameter is a shortcut that enables rawValues and sets an appropriate marshaller and entry wrapper for performing Rolling Upgrades.

CHAPTER 41. EXTERNALIZE SESSIONS

41.1. EXTERNALIZE SESSIONS

Red Hat JBoss Data Grid can be used as an external cache for containers, such as JBoss Enterprise Application Platform (EAP). This allows JBoss Data Grid to store HTTP Sessions, among other data, independent of the application layer, which provides the following benefits:

Application Elasticity

By making the application stateless additional nodes may be added to the EAP cluster without expensive data rebalancing operations. The EAP cluster may also be replaced without downtime by keeping the state in the JBoss Data Grid layer, as upgraded nodes may be brought online and retrieve the sessions.

Failover Across Data Centers

Should a data center become unavailable the session data persists, as it is stored safely within the JBoss Data Grid cluster. This allows a load balancer to redirect incoming requests to a second cluster to retrieve the session information.

Reduced Memory Footprint

There is reduced memory pressure, resulting in shorter garbage collection time and frequency of collections, as the HTTP Sessions have been moved out of the application layer and into the backing caches.

41.2. EXTERNALIZE HTTP SESSION FROM JBOSS EAP TO JBOSS DATA GRID

The below procedure applies for both standalone and domain mode of EAP; however, in domain mode each server group requires a unique remote cache configured. While multiple server groups can utilize the same Red Hat JBoss Data Grid cluster the respective remote caches will be unique to the EAP server group.



NOTE

The following procedures have been tested and confirmed to function on JBoss EAP 7.0 and JBoss Data Grid 7.0; when externalizing HTTP sessions with JBoss Data Grid 7.x only use these, or later, versions of each product.



NOTE

For each distributable application, an entirely new cache must be created. It can be created in an existing cache container, for example, web.

Externalize HTTP Sessions

1. Ensure the remote cache containers are defined in EAP's **infinispan** subsystem; in the example below the **cache** attribute in the **remote-store** element defines the cache name on the remote JBoss Data Grid server:

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
  [...]
  <cache-container name="web" default-cache="dist"
```

```

module="org.jboss.as.clustering.web.infinispan" statistics-
enabled="true">
  <transport lock-timeout="60000"/>
  <invalidation-cache name="jdg">
    <locking isolation="REPEATABLE_READ"/>
    <transaction mode="BATCH"/>
    <remote-store remote-servers="remote-jdg-server1 remote-jdg-
server2"
                cache="default" socket-timeout="60000"
                preload="true" passivation="false"
purge="false" shared="true"/>
  </replicated-cache>
</cache-container>
</subsystem>

```

2. Define the location of the remote Red Hat JBoss Data Grid server by adding the networking information to the **socket-binding-group**:

```

<socket-binding-group ...>
  <outbound-socket-binding name="remote-jdg-server1">
    <remote-destination host="JDGHostName1" port="11222"/>
  </outbound-socket-binding>
  <outbound-socket-binding name="remote-jdg-server2">
    <remote-destination host="JDGHostName2" port="11222"/>
  </outbound-socket-binding>
</socket-binding-group>

```

3. Repeat the above steps for each cache-container and each Red Hat JBoss Data Grid server. Each server defined must have a separate **<outbound-socket-binding>** element defined.
4. Add passivation and cache information into the application's **jboss-web.xml**. In the following example **web** is the name of the cache container, and **jdg** is the name of the default cache located in this container. An example file is shown below:

```

<?xml version="1.0" encoding="UTF-8"?>
<jboss-web xmlns="http://www.jboss.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.jboss.com/xml/ns/javaee
http://www.jboss.org/j2ee/schema/jboss-web_10_0.xsd"
           version="10.0">

  <replication-config>
    <replication-granularity>SESSION</replication-granularity>
    <cache-name>web.jdg</cache-name>
  </replication-config>

</jboss-web>

```



NOTE

The passivation timeouts above are provided assuming that a typical session is abandoned within 15 minutes and uses the default HTTP session timeout in JBoss EAP of 30 minutes. These values may need to be adjusted based on each application's workload.

41.3. EXTERNALIZE HTTP SESSIONS FROM JBOSS WEB SERVER (JWS) TO JBOSS DATA GRID

41.3.1. Externalize HTTP Session from JBoss Web Server (JWS) to JBoss Data Grid

A session manager has been provided as part of the JBoss Data Grid distribution, allowing JWS users to externalize sessions to JBoss Data Grid by integrating with an extension of Tomcat's **Manager**. This allows the Tomcat layer to remain stateless while providing session management persistence.

41.3.2. Prerequisites

This manager requires the following versions, or later, be installed:

- JBoss Web Server 3.0
- JBoss Data Grid 7.1

41.3.3. Installation

To install this manager, for both Tomcat 7 and Tomcat 8, complete the following steps:

1. Download **jboss-datagrid-7.1.0-jws-library** from JBoss Data Grid's product page for the version of Tomcat in use.
2. Extract the downloaded archive.
3. Copy the **lib/** directory from the extracted archive into **\$CATALINA_HOME**.
4. Define the implementation of the Session Manager in **context.xml**, as seen below:

```
<Manager
  className="org.wildfly.clustering.tomcat.hotrod.HotRodManager"
  server_list="www.server1.com:7600;www.server2.com:7600"
  <!-- Additional Configuration Elements -->
/>
```

41.3.4. Session Management Details

When using the **HotRodManager** all sessions are placed into the **default** cache located on the Remote JBoss Data Grid server. Cache names are not configurable.

Sessions stored in JBoss Web Server are all mutable by default. If an object changes during the course of the request then it will be replicated after the request ends. To define immutable objects, use one of the following annotations:

- The Wildfly specific annotation - **org.wildfly.clustering.web.annotation.Immutable**.
- Any generic immutable annotation.
- Any known immutable type from the JDK implementation.

Objects may have custom marshalling by defining an **Externalizer**. By default the Wildfly

Externalizer is recognized; however, any implementation of this **Externalizer** may be used. Additionally, non-serializable objects may be stored without issue as long as they have an **Externalizer** defined.

41.3.5. Configure the JBoss Web Server Session Manager

The **HotRodManager** is configured by defining properties on the **Manager** element inside of **context.xml**. These are pulled from two separate lists:

- **org.apache.catalina.Manager** - As the session manager implements this class many of the **Common Attributes** are configurable.
- **ConfigurationParameters** - This session manager also uses the HotRod Configuration Properties.

The following table displays all of the configurable elements

Table 41.1. Common Attributes from Tomcat's Manager

Attribute	Description
name	The name of this cluster manager. The name is used to identify a session manager on a node. The name might get modified by the Cluster element to make it unique to the container.
sessionIdLength	The length of session ids created by this Manager, measured in bytes, excluding subsequent conversion to a hexadecimal string and excluding any JVM route information used for load balancing. This attribute is deprecated. Set the length on a nested SessionIdGenerator element instead.
secureRandomClass	Name of the Java class that extends java.security.SecureRandom to use to generate session IDs. If not specified, the default value is java.security.SecureRandom .
secureRandomProvider	Name of the provider to use to create the java.security.SecureRandom instances that generate session IDs. If an invalid algorithm and/or provider is specified, the Manager will use the platform default provider and the default algorithm. If not specified, the platform default provider will be used.

Attribute	Description
secureRandomAlgorithm	Name of the algorithm to use to create the <code>java.security.SecureRandom</code> instances that generate session IDs. If an invalid algorithm and/or provider is specified, the Manager will use the platform default provider and the default algorithm. If not specified, the default algorithm of SHA1PRNG will be used. If the default algorithm is not supported, the platform default will be used. To specify that the platform default should be used, do not set the <code>secureRandomProvider</code> attribute and set this attribute to the empty string.
recordAllActions	Flag whether send all actions for session across Tomcat cluster nodes. If set to false, if already done something to the same attribute, make sure don't send multiple actions across Tomcat cluster nodes. In that case, sends only the actions that have been added at last. Default is false.

There is also a property specific to the JWS `HotRodManager`, shown below:

Attribute	Description
persistenceStrategy	Determines whether or not all attributes that compose a session should be serialized together (COARSE) or individually (FINE). When using a COARSE strategy relationships between objects will be preserved. FINE uses less memory, as only the attributes that have changed are serialized. Defaults to COARSE .

In addition to the attributes inherited from Tomcat, the `HotRodManager` may use any of the properties typically available to a `RemoteCacheManager`. These are outlined in [HotRod Properties](#).

When using HotRod properties only the property name itself is required. For instance, to configure TCP KEEPALIVE and TCP NODELAY on the manager the following xml snippet would be used:

```
<Manager className="org.wildfly.clustering.tomcat.hotrod.HotRodManager"
    tcp_no_delay="true"
    tcp_keep_alive="true"/>
```

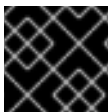
CHAPTER 42. DATA INTEROPERABILITY

42.1. PROTOCOL INTEROPERABILITY

Protocol interoperability enables clients that are written in different programming languages to read and write cache entries from any Red Hat JBoss Data Grid endpoint, such as REST, Memcached, or Hot Rod.

Each endpoint stores data in a suitable format so that data transformation is not required to retrieve entries from the cache. However, when accessing data from multiple protocols, Red Hat JBoss Data Grid must convert data between the formats for each endpoint.

To access data in a cache from multiple protocols, you must enable compatibility mode on that cache.



IMPORTANT

Compatibility mode supports only strings and primitives. Objects are not supported.

42.1.1. Enabling Compatibility Mode

To enable compatibility mode, add **enabled=true** to the **compatibility** element as follows:

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



NOTE

JBoss Data Grid uses the **GenericJBossMarshaller** marshaller by default in compatibility mode. However, when using memcached, you must explicitly set this marshaller in the configuration. For more information, see [Protocol Interoperability](#) in the *Developer Guide*.

CHAPTER 43. HANDLING NETWORK PARTITIONS (SPLIT BRAIN)

43.1. NETWORK PARTITION RECOVERY

Network Partitions occur when a cluster breaks into two or more partitions. As a result, the nodes in each partition are unable to locate or communicate with nodes in the other partitions. This results in an unintentionally partitioned network.

In the event of a network partition in a distributed system like Red Hat JBoss Data Grid, the CAP (Brewer's) theorem comes into play. The CAP theorem states that in the event of a Network Partition [P], a distributed system can provide either Consistency [C] or Availability [A] for the data, but not both.

By default, Partition Handling is disabled in JBoss Data Grid. During a network partition, the partitions continue to remain Available [A], at the cost of Consistency [C].

However, when Partition Handling is enabled, JBoss Data Grid prioritizes consistency [C] of data over Availability [A].

Red Hat JBoss Data Grid offers the primary partitions strategy to repair a split network. When the network partition occurs and the cache is split into two or more partitions, at most one partition becomes the primary partition (and stays available) and the others are designated as secondary partitions (and enter Degraded Mode). When the partitions merge back into a single cache, the primary partition is then used as a reference for all secondary partitions. All members of the secondary partitions must remove their current state information and replace it with fresh state information from a member of the primary partition. If there was no primary partition during the split, the state on every node is assumed to be correct.

In JBoss Data Grid, a cache consists of data stored on a number of nodes. To prevent data loss if a node fails, JBoss Data Grid replicates a data item over multiple nodes. In distribution mode, this redundancy is configured using the **owners** configuration attribute, which specifies the number of replicas for each cache entry in the cache. As a result, as long as the number of nodes that have failed are less than the value of **owners**, JBoss Data Grid retains a copy of the lost data and can recover.



NOTE

In JBoss Data Grid's replication mode, however, **owners** is always equal to the number of nodes in the cache, because each node contains a copy of every data item in the cache in this mode.

In certain cases, a number of nodes greater than the value of **owners** can disappear from the cache. Two common reasons for this are:

- **Split-Brain:** Usually, as the result of a router crash, the cache is divided into two or more partitions. Each of the partitions operates independently of the other and each may contain different versions of the same data.
- **Successive Crashed Nodes:** A number of nodes greater than the value of **owners** crashes in succession for any reason. JBoss Data Grid is unable to properly balance the state between crashes, and the result is partial data loss.

43.2. DETECTING AND RECOVERING FROM A SPLIT-BRAIN PROBLEM

When a Split-Brain occurs in the data grid, each network partition installs its own JGroups view with nodes from other partitions removed. The partitions remain unaware of each other, therefore there is no way to determine how many partitions the network has split into. Red Hat JBoss Data Grid assumes that the cache has unexpectedly split if one or more nodes disappear from the JGroups cache without sending an explicit leaving message, while in reality the cause can be physical (crashed switches, cable failure, etc.) to virtual (stop-the-world garbage collection).

This state is dangerous because each of the newly split partitions operates independently and can store conflicting updates for the same data entries.

When Partition Handling mode is enabled (see [Configure Partition Handling](#) for instructions) and JBoss Data Grid suspects that one or more nodes are no longer accessible, each partition does not start a rebalance immediately, but first it checks whether it should enter degraded mode instead. To enter Degraded Mode, one of the following conditions must be true:

- At least one segment has lost all its owners, which means that a number of nodes equal to or greater than the value of **owners** have left the JGroups view.
- The partition does not contain a majority of nodes (greater than half) of the nodes from the latest stable topology. The stable topology is updated each time a rebalance operation successfully concludes and the coordinator determines that additional rebalancing is not required.

If neither of the conditions are met, the partition continues normal operations and JBoss Data Grid attempts to rebalance its nodes. Based on these conditions, at most one partition can remain in Available mode. Other partitions will enter Degraded Mode.

When a partition enters into Degraded Mode, it only allows read/write access to those entries for which all owners (copies) of the entry exist on nodes within the same partition. Read and write requests for an entry for which one or more of its owners (copies) exist on nodes that have disappeared from the partition are rejected with an **AvailabilityException**.



NOTE

A possible limitation is that if two partitions start as isolated partitions and do not merge, they can read and write inconsistent data. JBoss Data Grid does not identify such partitions as split partitions.



WARNING

Data consistency can be at risk from the time (t1) when the cache physically split to the time (t2) when JBoss Data Grid detects the connectivity change and changes the state of the partitions:

- Transactional writes that were in progress at t1 when the split physically occurred may be rolled back on some of the owners. This can result in inconsistency between the copies (after the partitions rejoin) of an entry that is affected by such a write. However, transactional writes that started after t1 will fail as expected.
- If the write is non-transactional, then during this time window, a value written only in a minor partition (due to physical split and because the partition has not yet been Degraded) can be lost when partitions rejoin, if this minor partition receives state from a primary (Available) partition upon rejoin. If the partition does not receive state upon rejoin (i.e. all partitions are degraded), then the value is not lost, but an inconsistency can remain.
- There is also a possibility of a stale read in a minor partition during this transition period, as an entry is still Available until the minor partition enters Degraded state.

When partitions merge after a network partition has occurred:

- If one of the partitions was Available during the network partition, then the joining partition(s) are wiped out and state transfer occurs from the Available (primary) partition to the joining nodes.
- If all joining partitions were Degraded during the Split Brain, then no state transfer occurs during the merge. The combined cache is then Available only if the merging partitions contain a simple majority of the members in the latest stable topology (one with the highest topology ID) and has at least an owner for each segment (i.e. keys are not lost).

**WARNING**

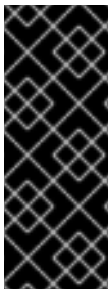
Between the time (t1) when partitions begin merging to the time (t2) when the merge is complete, nodes reconnect through a series of merge events. During this time window, it is possible that a node can be reported as having temporarily left the cluster. For a Transactional cache, if during this window between t1 and t2, such a node is executing a transaction that spans other nodes, then this transaction may not execute on the remote node, but still succeed on the originating node. The result is a potential stale value for affected entries on a node that did not commit this transaction.

After t2, once the merge has completed on all nodes, this situation will not occur for subsequent transactions. However, an inconsistency introduced on entries that were affected by a transaction in progress during the time window between t1 and t2 is not resolved until these entries are subsequently updated or deleted. Until then, a read on such impacted entries can potentially return the stale value.

43.3. SPLIT BRAIN TIMING: DETECTING A SPLIT

When using the **FD_ALL** protocol a given node becomes suspected after the following amount of milliseconds have passed:

```
FD_ALL.timeout + FD_ALL.interval + VERIFY_SUSPECT.timeout +
GMS.view_ack_collection_timeout
```

**IMPORTANT**

The amount of time taken in the formulas above is how long it takes JBoss Data Grid to install a cluster view without the leavers; however, as JBoss Data Grid runs inside a JVM excessive Garbage Collection (GC) times can increase this time beyond the failure detection outlined above. JBoss Data Grid has no control over these GC times, and excessive GC on the coordinator can delay this detection by an amount equal to the GC time.

43.4. SPLIT BRAIN TIMING: RECOVERING FROM A SPLIT

After a split occurs JBoss Data Grid will merge the partitions back, and the maximum time to detect a merge after the network partition is healed is:

```
3.1 * MERGE3.max_interval
```

In some cases multiple merges will occur after a split so that the cluster may contain all available partitions. In this case, where multiple merges occur, time should be allowed for all of these to complete, and as there may be as many as three merges occurring sequentially the total delay should be no more than the following:

```
10 * MERGE3.max_interval
```




IMPORTANT

The amount of time taken in the formulas above is how long it takes JBoss Data Grid to install a cluster view without the leavers; however, as JBoss Data Grid runs inside a JVM excessive Garbage Collection (GC) times can increase this time beyond the failure detection outlined above. JBoss Data Grid has no control over these GC times, and excessive GC on the coordinator can delay this detection by an amount equal to the GC time.

In addition, when merging cluster views JBoss Data Grid tries to confirm all members are present; however, there is no upper bound on waiting for these responses, and merging the cluster views may be delayed due to networking issues.

43.5. DETECTING AND RECOVERING FROM SUCCESSIVE CRASHED NODES

Red Hat JBoss Data Grid cannot distinguish whether a node left the cluster because of a process or machine crash, or because of a network failure.

If a single node exits the cluster, and if the value of **owners** is greater than 1, the cluster remains available and JBoss Data Grid attempts to create new replicas of the lost data. However, if additional nodes crash during this rebalancing process, it is possible that for some entries, all copies of its data have left the node and therefore cannot be recovered.

The recommended way to protect the data grid against successive crashed nodes is to enable partition handling (see [Configure Partition Handling](#) for instructions) and to set an appropriately high value for **owners** to ensure that even if a large number of nodes leave the cluster in rapid succession, JBoss Data Grid is able to rebalance the nodes to recover the lost data.

Alternatively, if you can tolerate some data loss, you can force JBoss Data Grid into AVAILABLE mode from DEGRADED mode using the **Cache JMX MBean**. See [Cache JMX MBean](#).

Likewise, the **AdvancedCache** interface lets you read and change the cache availability. See [The AdvancedCache Interface](#) in the Developer Guide.

43.6. NETWORK PARTITION RECOVERY EXAMPLES

43.6.1. Network Partition Recovery Examples

The following examples illustrate how network partitions occur in Red Hat JBoss Data Grid clusters and how they are dealt with and eventually merged. The following examples scenarios are described in detail:

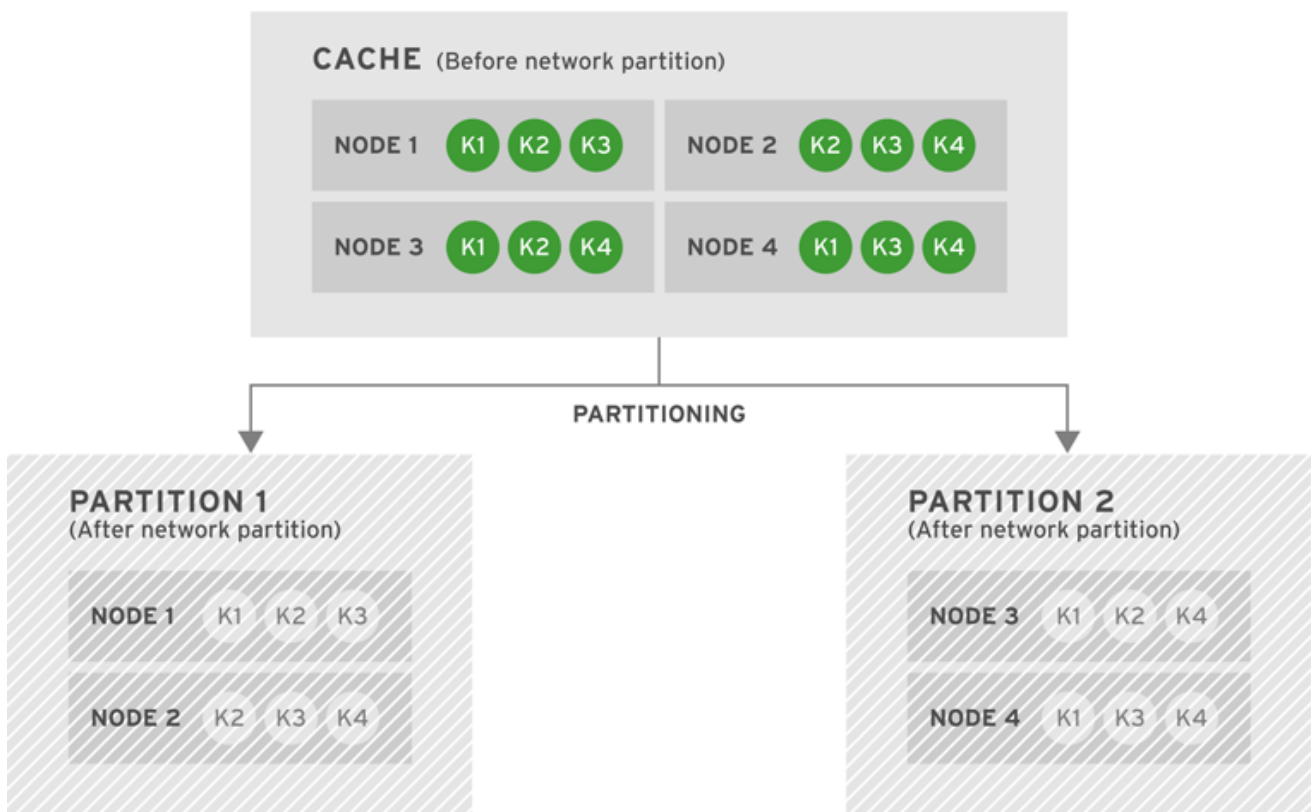
1. A distributed four node cluster with **owners** set to 3 at [Distributed 4-Node Cache Example With 3 Owners](#)
2. A distributed four node cluster with **owners** set to 2 at [Distributed 4-Node Cache Example With 2 Owners](#)
3. A distributed five node cluster with **owners** set to 3 at [Distributed 5-Node Cache Example With 3 Owners](#)
4. A replicated four node cluster with **owners** set to 4 at [Replicated 4-Node Cache Example With 4 Owners](#)

5. A replicated five node cluster with **owners** set to 5 at [Replicated 5-Node Cache Example With 5 Owners](#)
6. A replicated eight node cluster with **owners** set to 8 at [Replicated 8-Node Cache Example With 8 Owners](#)

43.6.2. Distributed 4-Node Cache Example With 3 Owners

The first example scenario includes a four-node distributed cache that contains four data entries (**k1**, **k2**, **k3**, and **k4**). For this cache, **owners** equals 3, which means that each data entry must have three copies on various nodes in the cache.

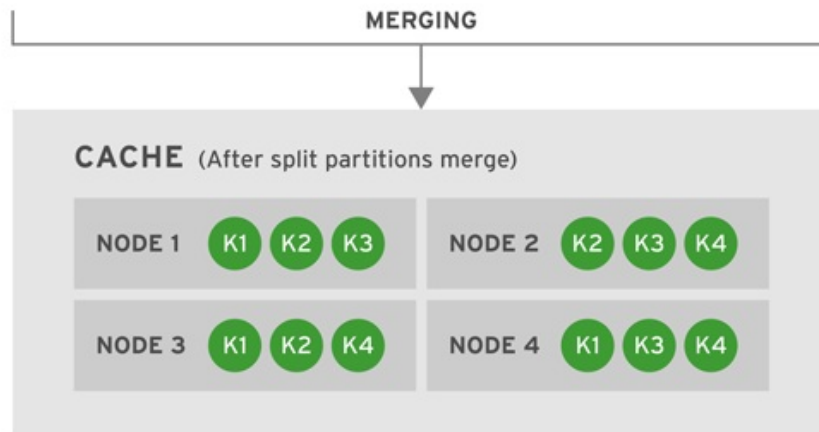
Figure 43.1. Cache Before and After a Network Partition



JBoss_11_335370_0415

As seen in the diagram, after the network partition occurs, Node 1 and Node 2 form Partition 1 while Node 3 and Node 4 form a Partition 2. After the split, the two partitions enter into Degraded Mode (represented by grayed-out nodes in the diagram) because neither has at least 3 (the value of **owners**) nodes left from the last stable view. As a result, none of the four entries (**k1**, **k2**, **k3**, and **k4**) are available for reads or writes. No new entries can be written in either degraded partition, as neither partition can store 3 copies of an entry.

Figure 43.2. Cache After Partitions Are Merged



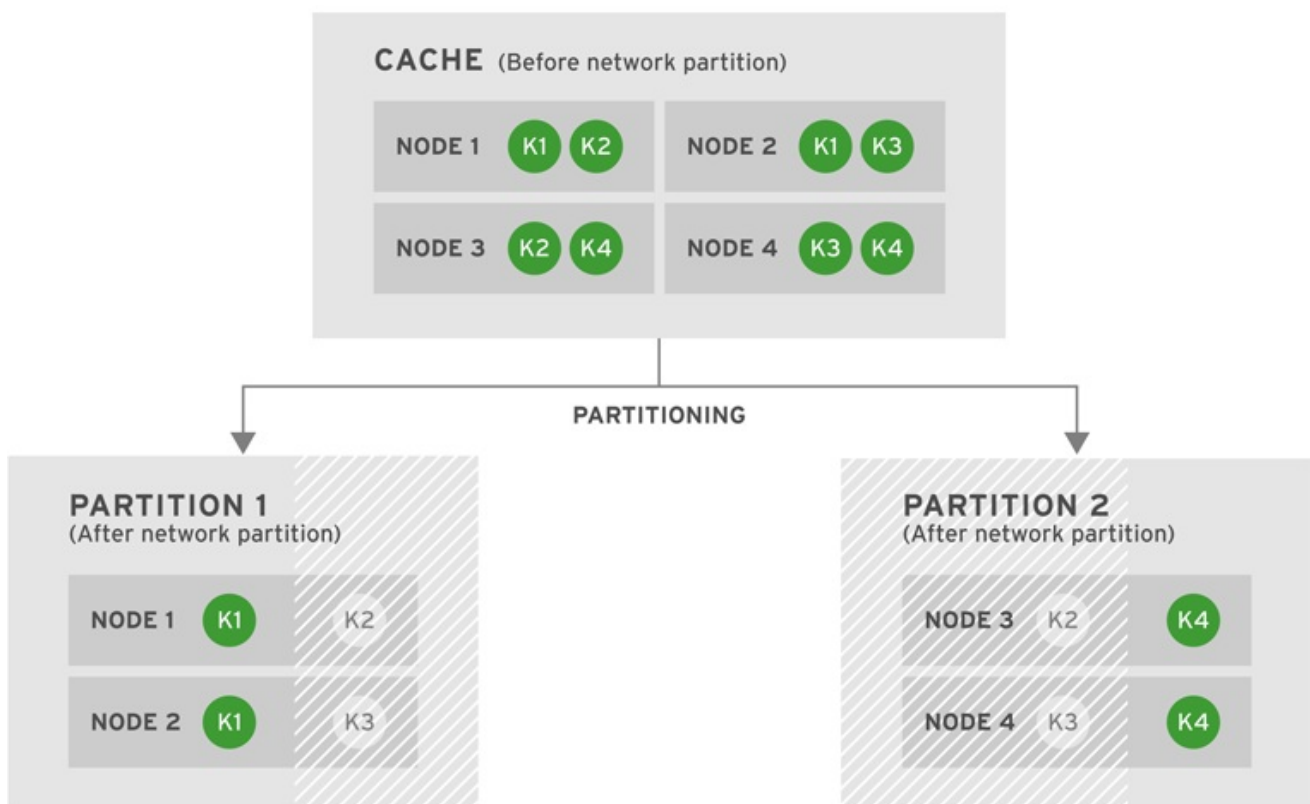
JBoss_12_335370_0415

JBoss Data Grid subsequently merges the two split partitions. No state transfer is required and the new merged cache is subsequently in Available Mode with four nodes and four entries (**k1**, **k2**, **k3**, and **k4**).

43.6.3. Distributed 4-Node Cache Example With 2 Owners

The second example scenario includes a distributed cache with four nodes. In this scenario, **owners** equals 2, so the four data entries (**k1**, **k2**, **k3** and **k4**) have two copies each in the cache.

Figure 43.3. Cache Before and After a Network Partition

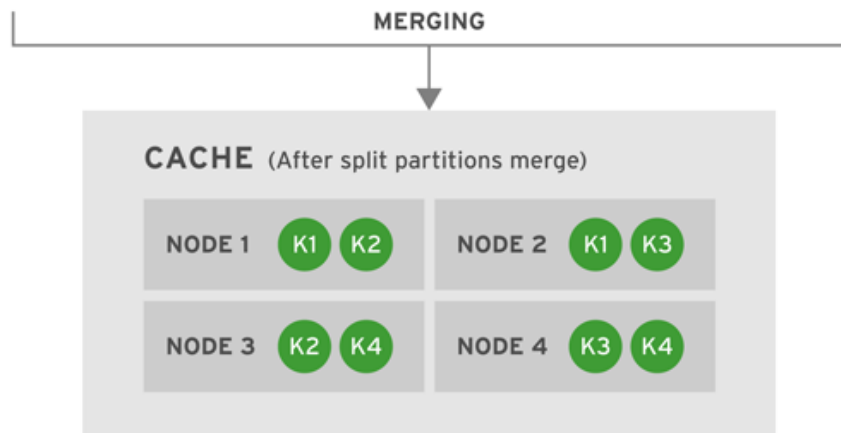


JBoss_13_335370_0615

After the network partition occurs, Partitions 1 and 2 enter Degraded mode (depicted in the diagram as grayed-out nodes). Within each partition, an entry will only be available for read or write operations if both its copies are in the same partition. In Partition 1, the data entry **k1** is available for reads and writes

because **owners** equals 2 and both copies of the entry remain in Partition 1. In Partition 2, **k4** is available for reads and writes for the same reason. The entries **k2** and **k3** become unavailable in both partitions, as neither partition contains all copies of these entries. A new entry **k5** can be written to a partition only if that partition were to own both copies of **k5**.

Figure 43.4. Cache After Partitions Are Merged



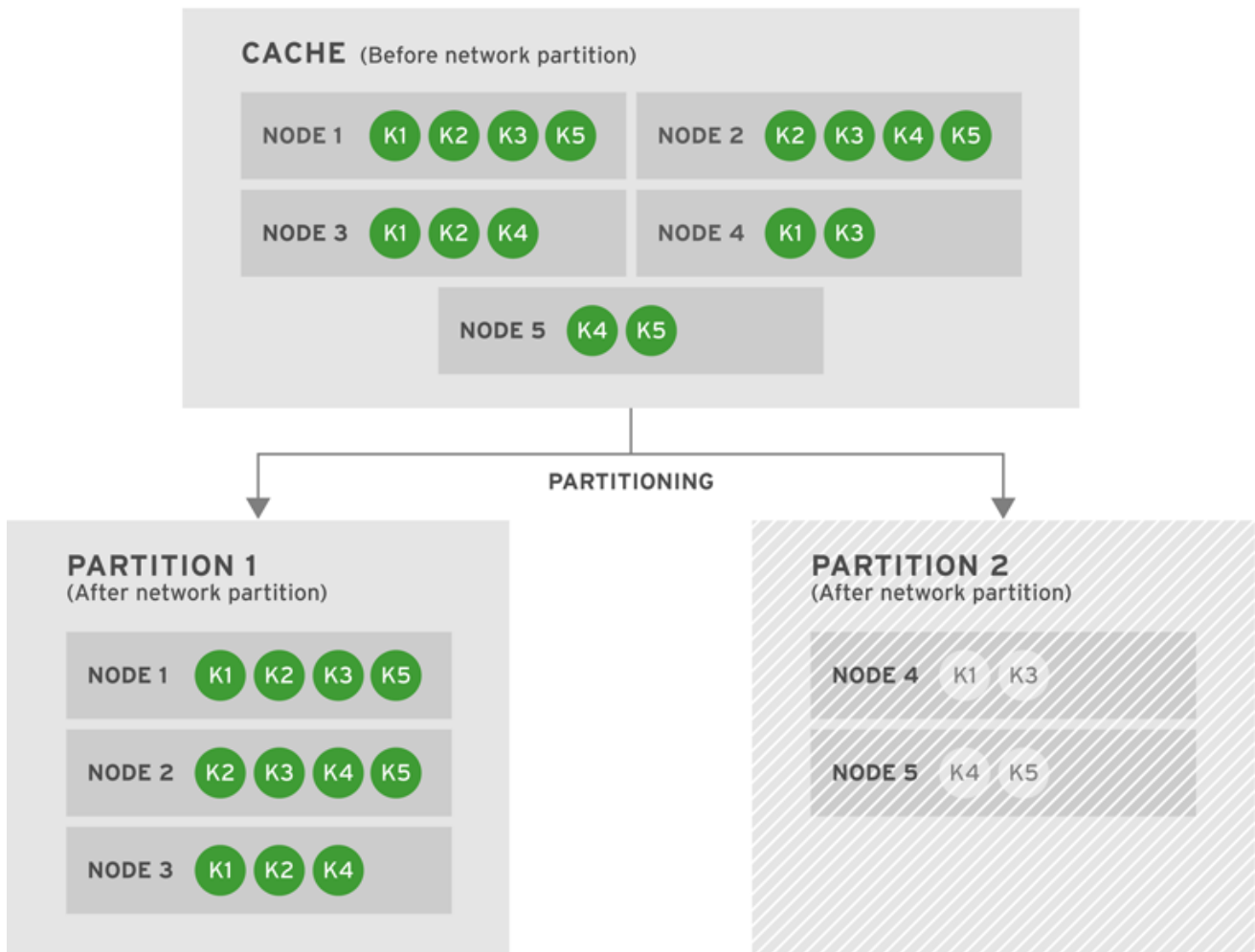
JBOSS_14_335370_0415

JBoss Data Grid subsequently merges the two split partitions into a single cache. No state transfer is required and the cache returns to Available Mode with four nodes and four data entries (**k1**, **k2**, **k3** and **k4**).

43.6.4. Distributed 5-Node Cache Example With 3 Owners

The third example scenario includes a distributed cache with five nodes and with **owners** equal to 3.

Figure 43.5. Cache Before and After a Network Partition

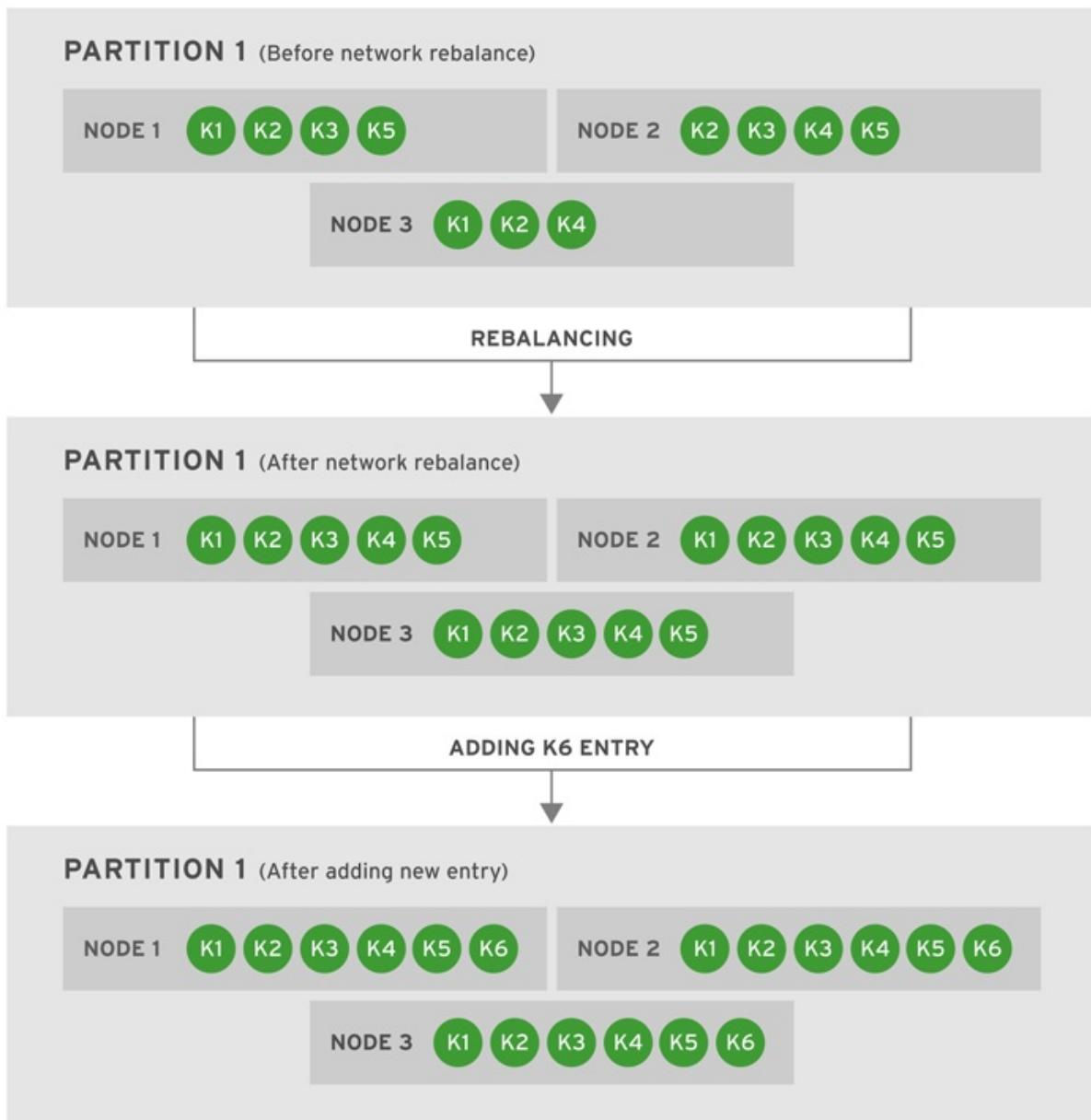


JBOSS_15_335370_0415

After the network partition occurs, the cache splits into two partitions. Partition 1 includes Node 1, Node 2, and Node 3 and Partition 2 includes Node 4 and Node 5. Partition 2 is Degraded because it does not include the majority of nodes from the total number of nodes in the cache. Partition 1 remains Available because it has the majority of nodes and lost less than **owners** nodes.

No new entries can be added to Partition 2 because this partition is Degraded and it cannot own all copies of the data.

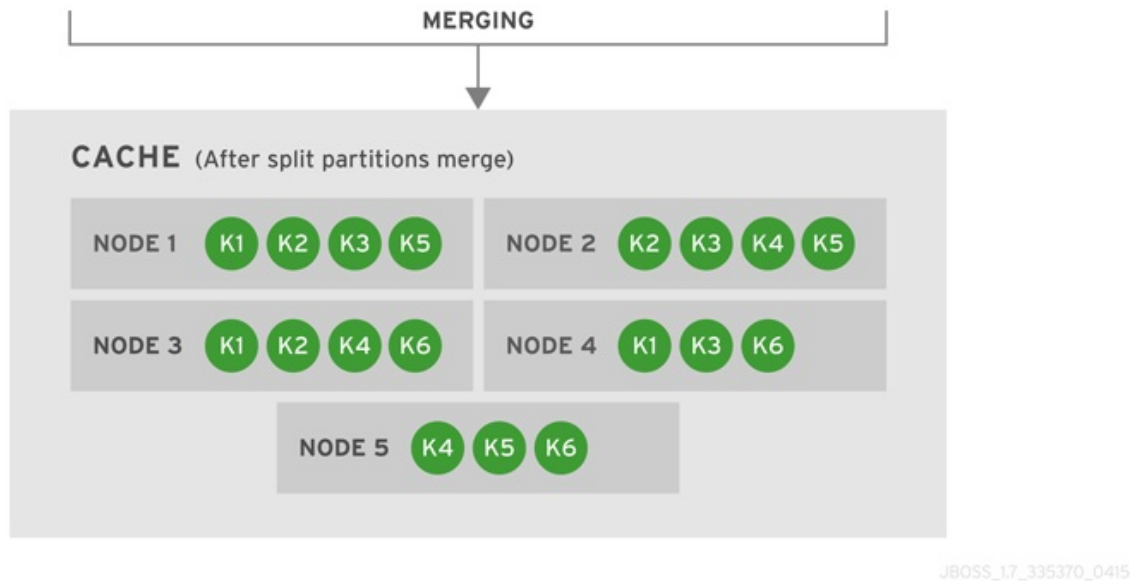
Figure 43.6. Partition 1 Rebalances and Another Entry is Added



JBoss_16_335370_0415

After the partition split, Partition 1 retains the majority of nodes and therefore can rebalance itself by creating copies to replace the missing entries. As displayed in the diagram above, rebalancing ensures that there are three copies of each entry (**owners** equals 3) in the cache. As a result, each of the three nodes contains a copy of every entry in the cache. Next, we add a new entry, **k6**, to the cache. Since the **owners** value is still 3, and there are three nodes in Partition 1, each node includes a copy of **k6**.

Figure 43.7. Cache After Partitions Are Merged

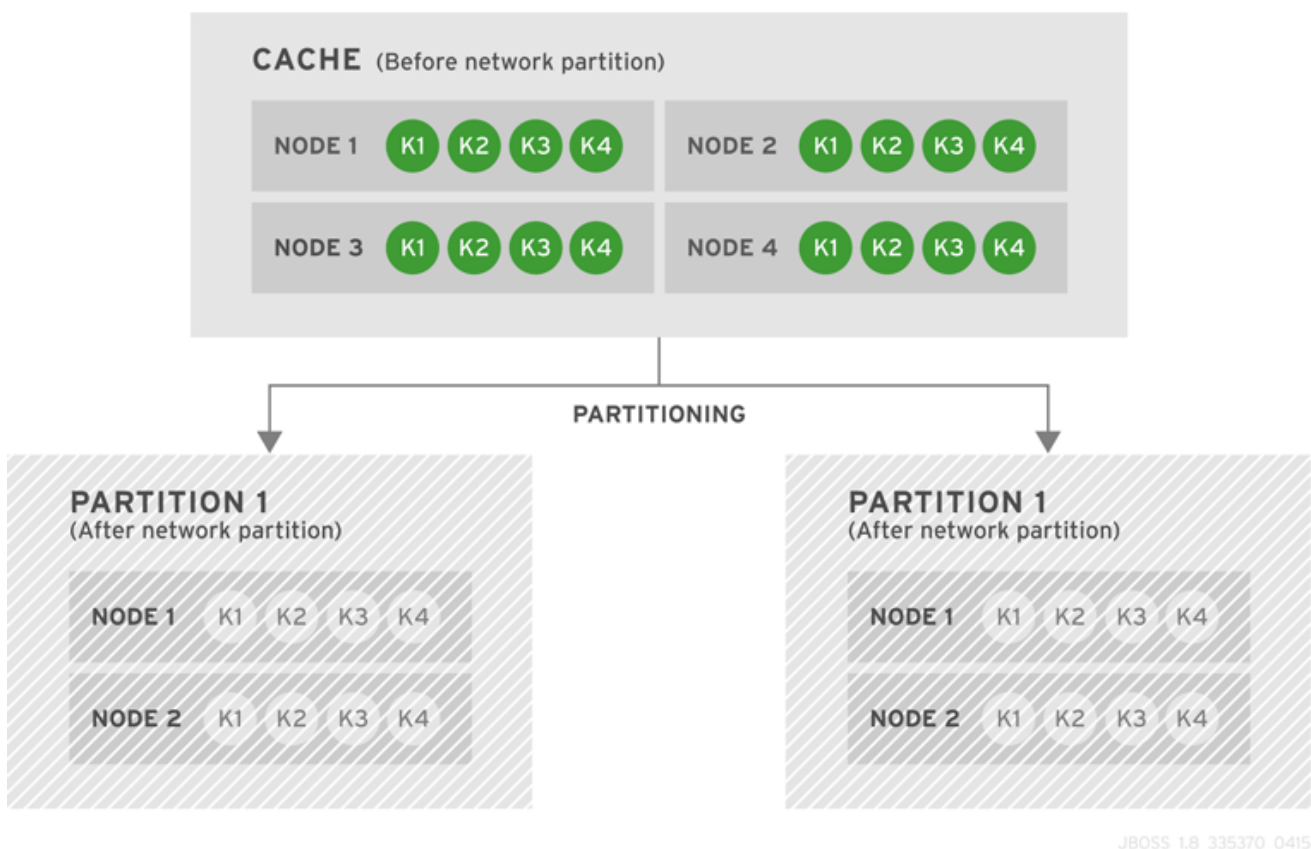


Eventually, Partition 1 and 2 are merged into a cache. Since only three copies are required for each data entry (**owners=3**), JBoss Data Grid rebalances the nodes so that the data entries are distributed between the four nodes in the cache. The new combined cache becomes fully available.

43.6.5. Replicated 4-Node Cache Example With 4 Owners

The fourth example scenario includes a replicated cache with four nodes and with **owners** equal to 4.

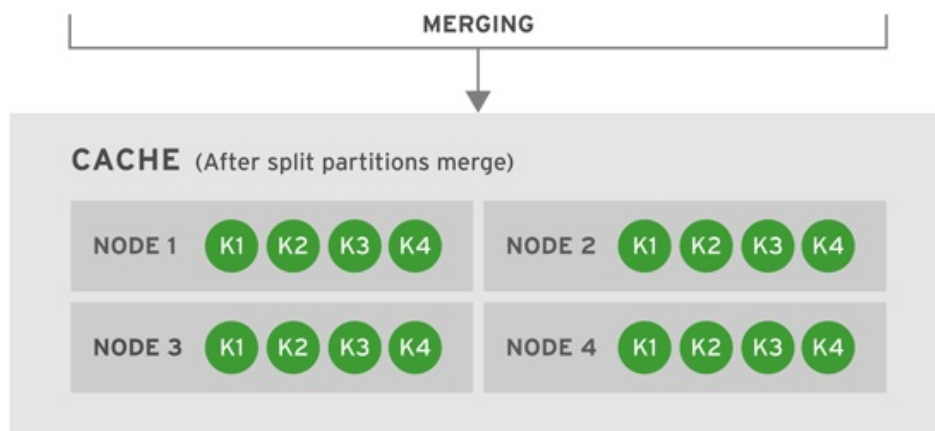
Figure 43.8. Cache Before and After a Network Partition



After a network partition occurs, Partition 1 contains Node 1 and Node 2 while Node 3 and Node 4 are in

Partition 2. Both partitions enter Degraded Mode because neither has a simple majority of nodes. All four keys (**k1**, **k2**, **k3**, and **k4**) are unavailable for reads and writes because neither of the two partitions owns all copies of any of the four keys.

Figure 43.9. Cache After Partitions Are Merged



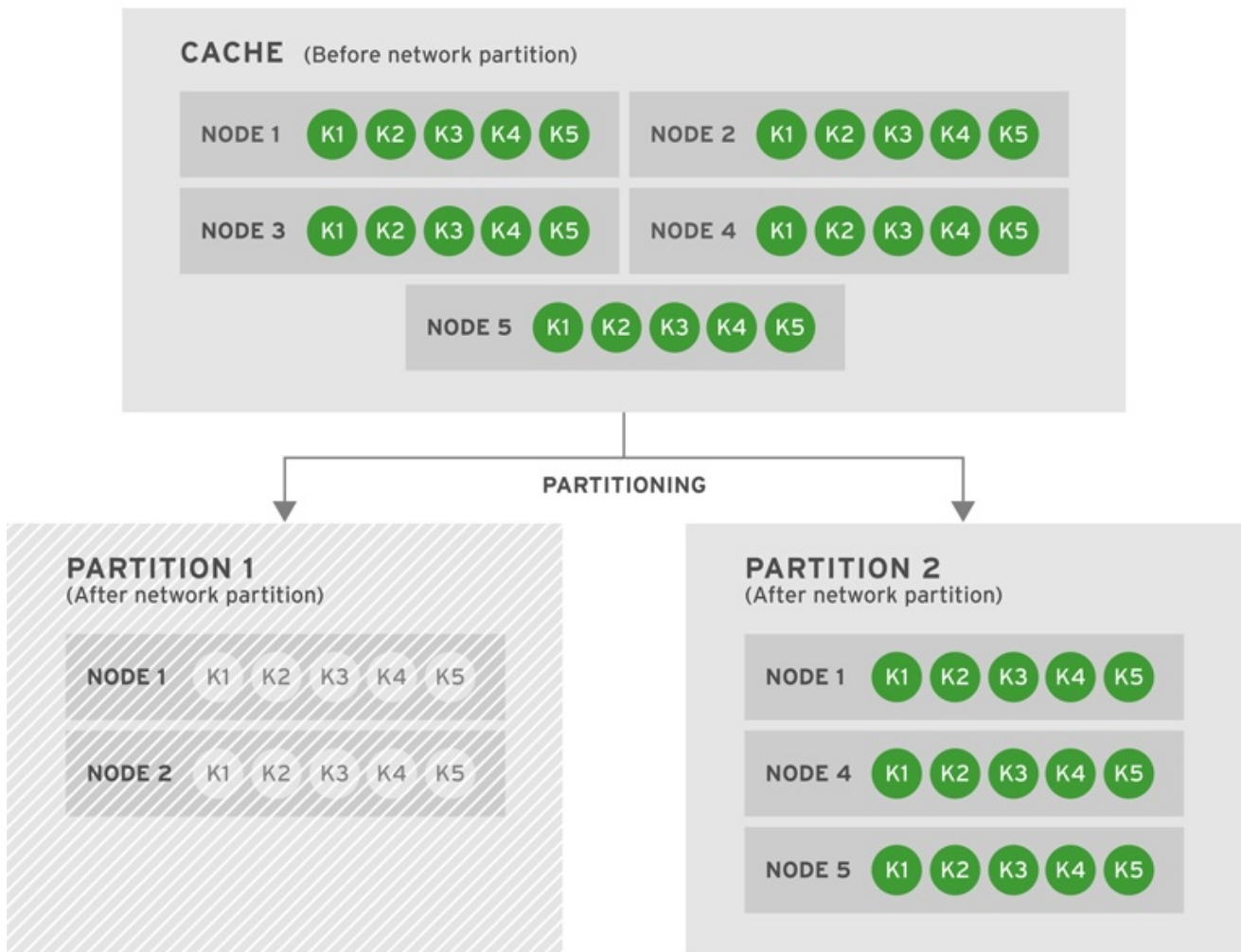
JBOSS_19_335370_0415

JBoss Data Grid subsequently merges the two split partitions into a single cache. No state transfer is required and the cache returns to its original state in Available Mode with four nodes and four data entries (**k1**, **k2**, **k3**, and **k4**).

43.6.6. Replicated 5-Node Cache Example With 5 Owners

The fifth example scenario includes a replicated cache with five nodes and with **owners** equal to 5.

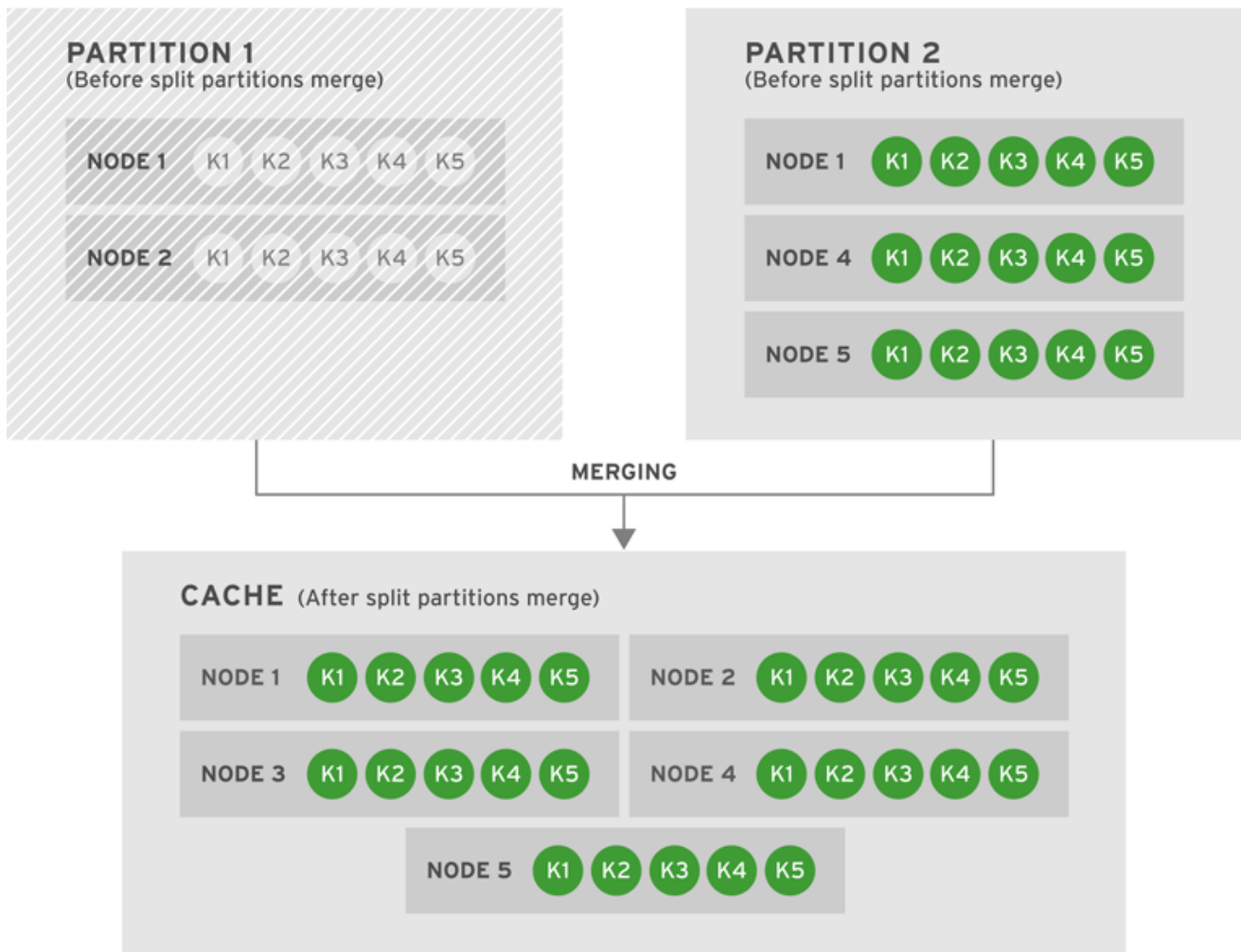
Figure 43.10. Cache Before and After a Network Partition



JBOSS_110_335370_0415

After a network partition occurs, the cache splits into two partitions. Partition 1 contains Node 1 and Node 2 and Partition 2 includes Node 3, Node 4, and Node 5. Partition 1 enters Degraded Mode (indicated by the grayed-out nodes) because it does not contain the majority of nodes. Partition 2, however, remains available.

Figure 43.11. Both Partitions Are Merged Into One Cache



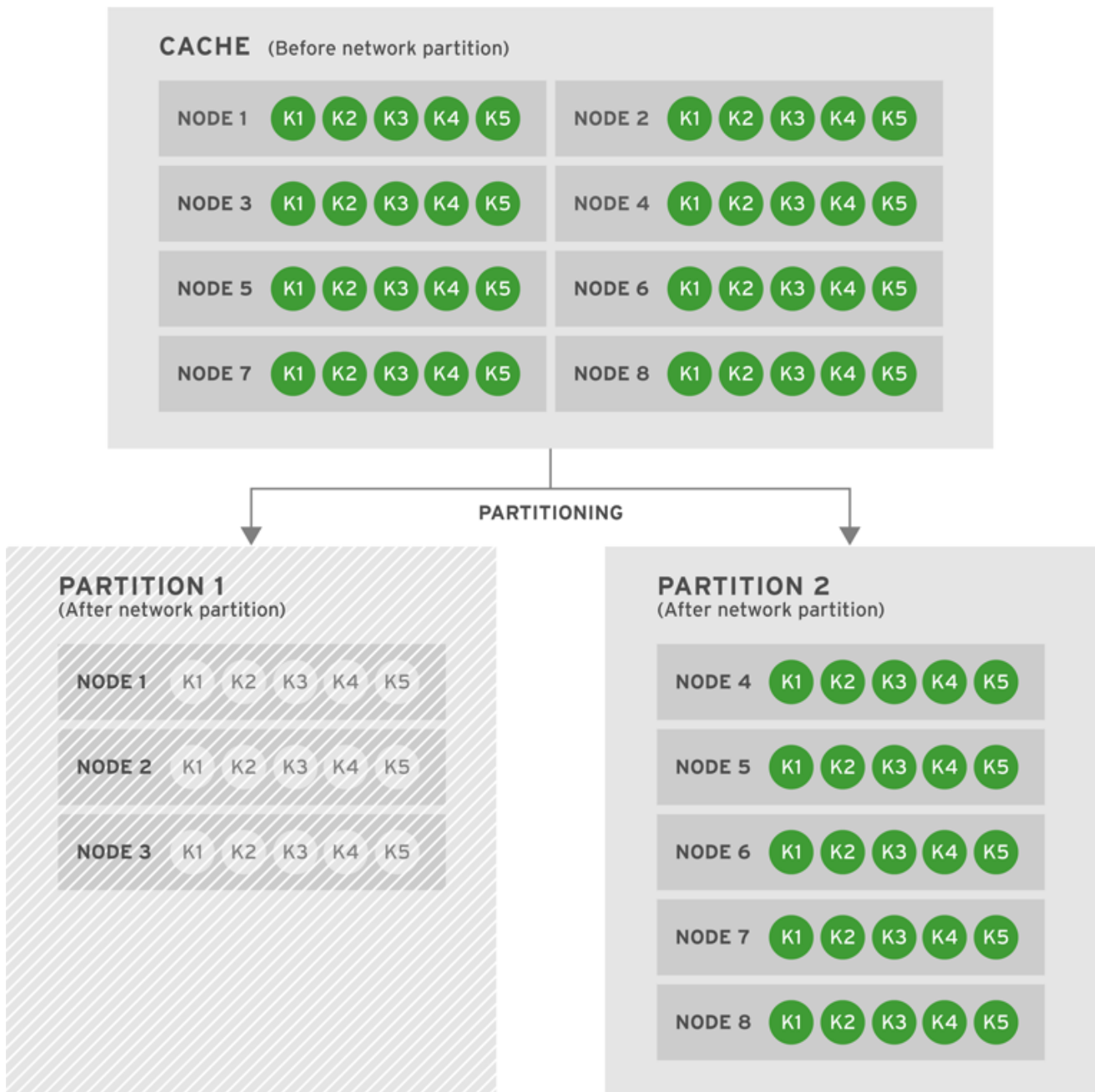
JBOSSE_1.11_335370_0415

When JBoss Data Grid merges partitions in this example, Partition 2, which was fully available, is considered the primary partition. State is transferred from Partition 1 and to Partition 2. The combined cache becomes fully available."

43.6.7. Replicated 8-Node Cache Example With 8 Owners

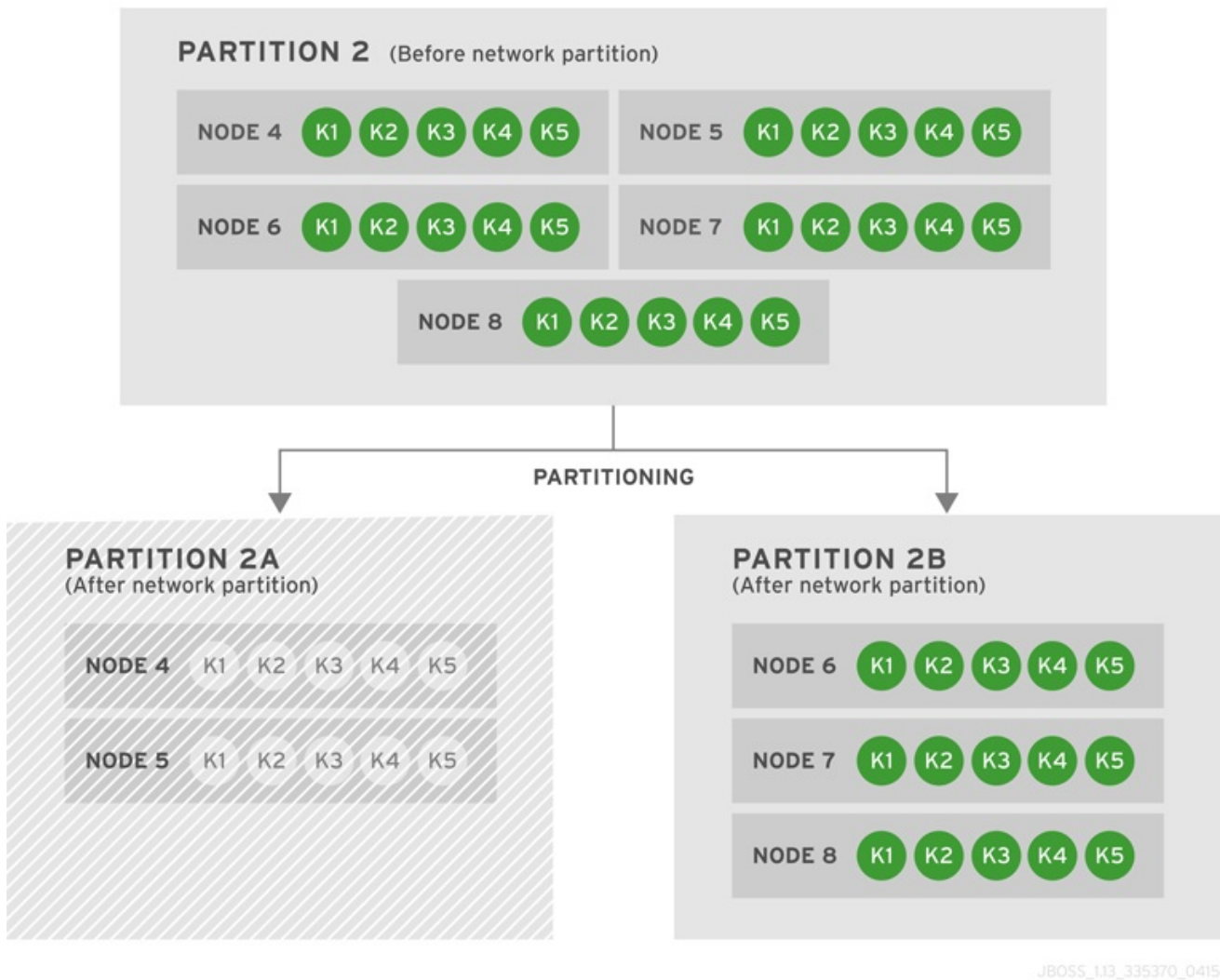
The sixth scenario is for a replicated cache with eight nodes and **owners** equal to 8.

Figure 43.12. Cache Before and After a Network Partition



JBOSS_112_335370_0415

A network partition splits the cluster into Partition 1 with 3 nodes and Partition 2 with 5 nodes. Partition 1 enters Degraded state, but Partition 2 remains Available.

Figure 43.13. Partition 2 Further Splits into Partitions 2A and 2B

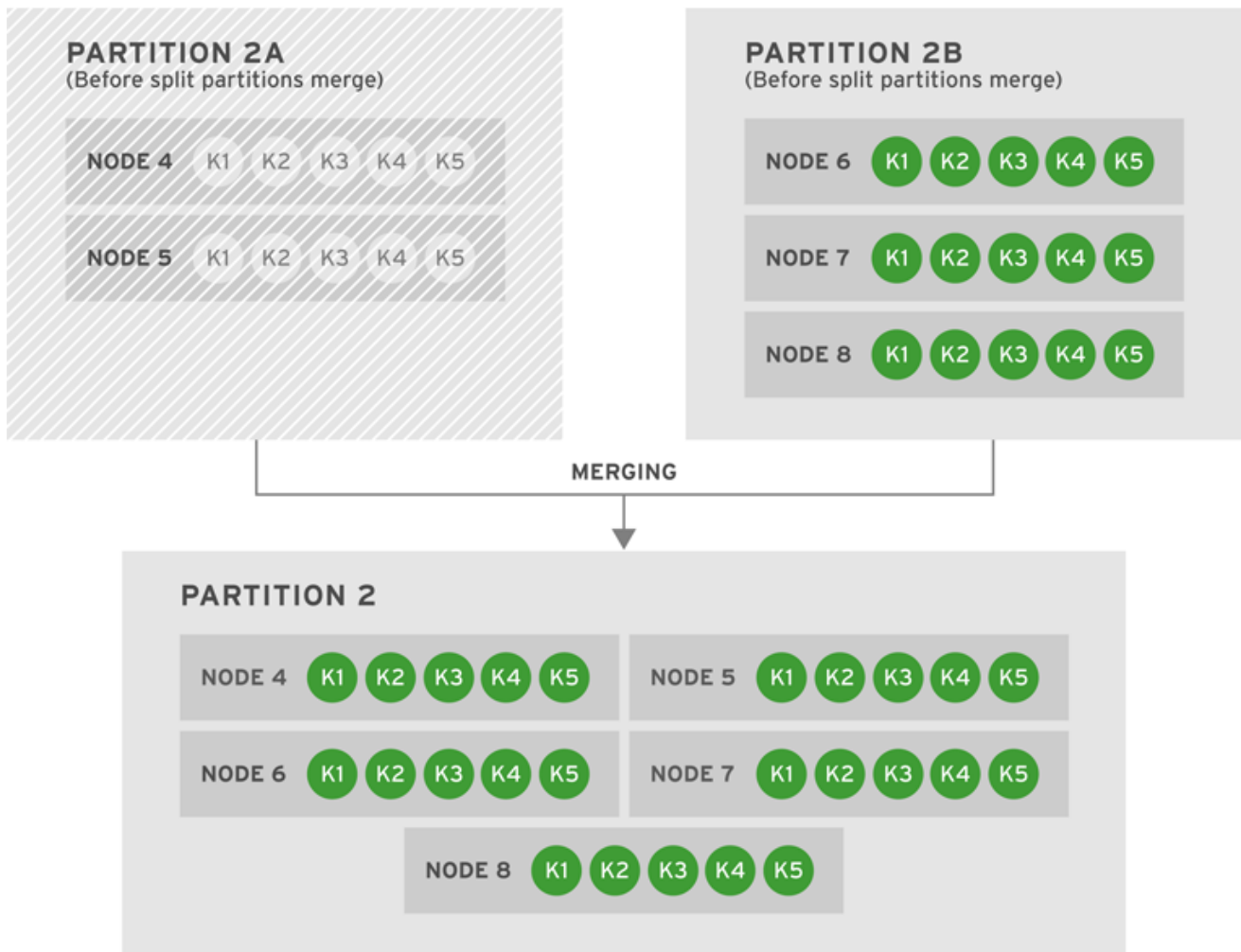
Now another network partition affects Partition 2, which subsequently splits further into Partition 2A and 2B. Partition 2A contains Node 4 and Node 5 while Partition 2B contains Node 6, Node 7, and Node 8. Partition 2A enters Degraded Mode because it does not contain the majority of nodes. However, Partition 2B remains Available.

Potential Resolution Scenarios

There are four potential resolutions for the caches from this scenario:

- Case 1: Partitions 2A and 2B Merge
- Case 2: Partition 1 and 2A Merge
- Case 3: Partition 1 and 2B Merge
- Case 4: Partition 1, Partition 2A, and Partition 2B Merge Together

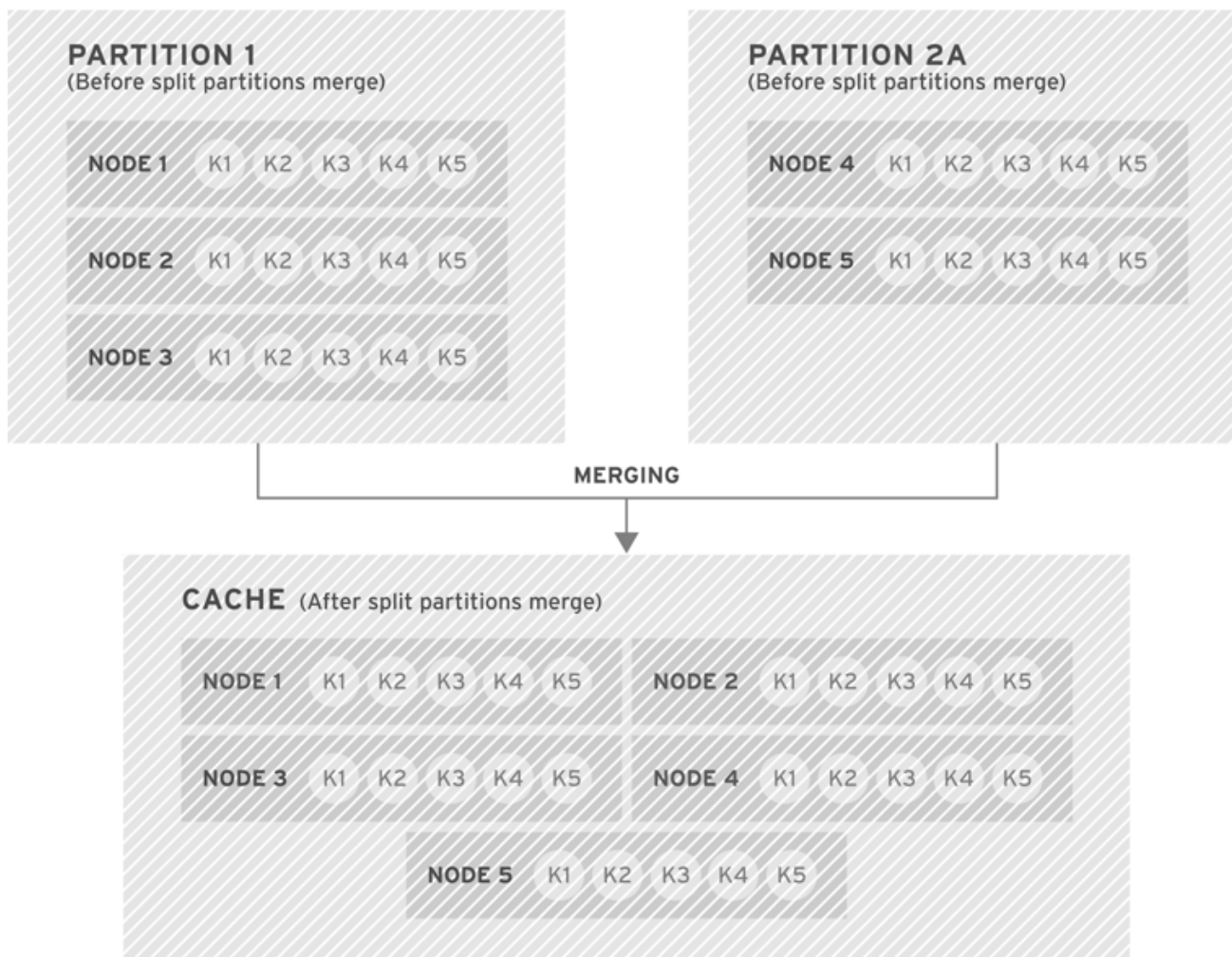
Figure 43.14. Case 1: Partitions 2A and 2B Merge



JBOSS_114_335370_0415

The first potential resolution to the partitioned network involves Partition 2B's state information being copied into Partition 2A. The result is Partition 2, which contains Node 5, Node 6, Node 7, and Node 8. The newly merged partition becomes Available.

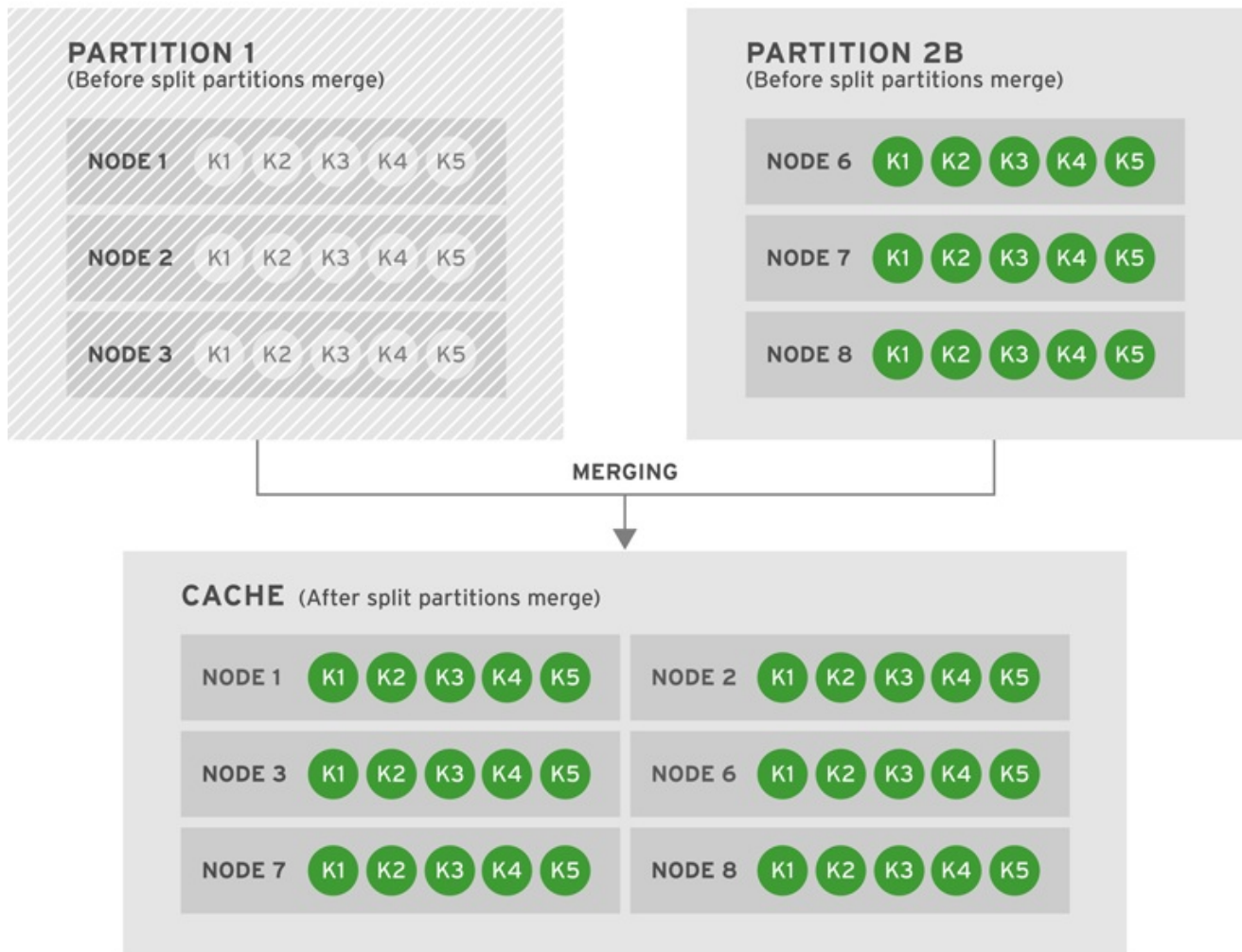
Figure 43.15. Case 2: Partition 1 and 2A Merge



JBOSS_115_335370_0415

The second potential resolution to the partitioned network involves Partition 1 and Partition 2A merging. The combined partition contains Node 1, Node 2, Node 3, Node 4, and Node 5. As neither partition has the latest stable topology, the resulting merged partition remains in Degraded mode.

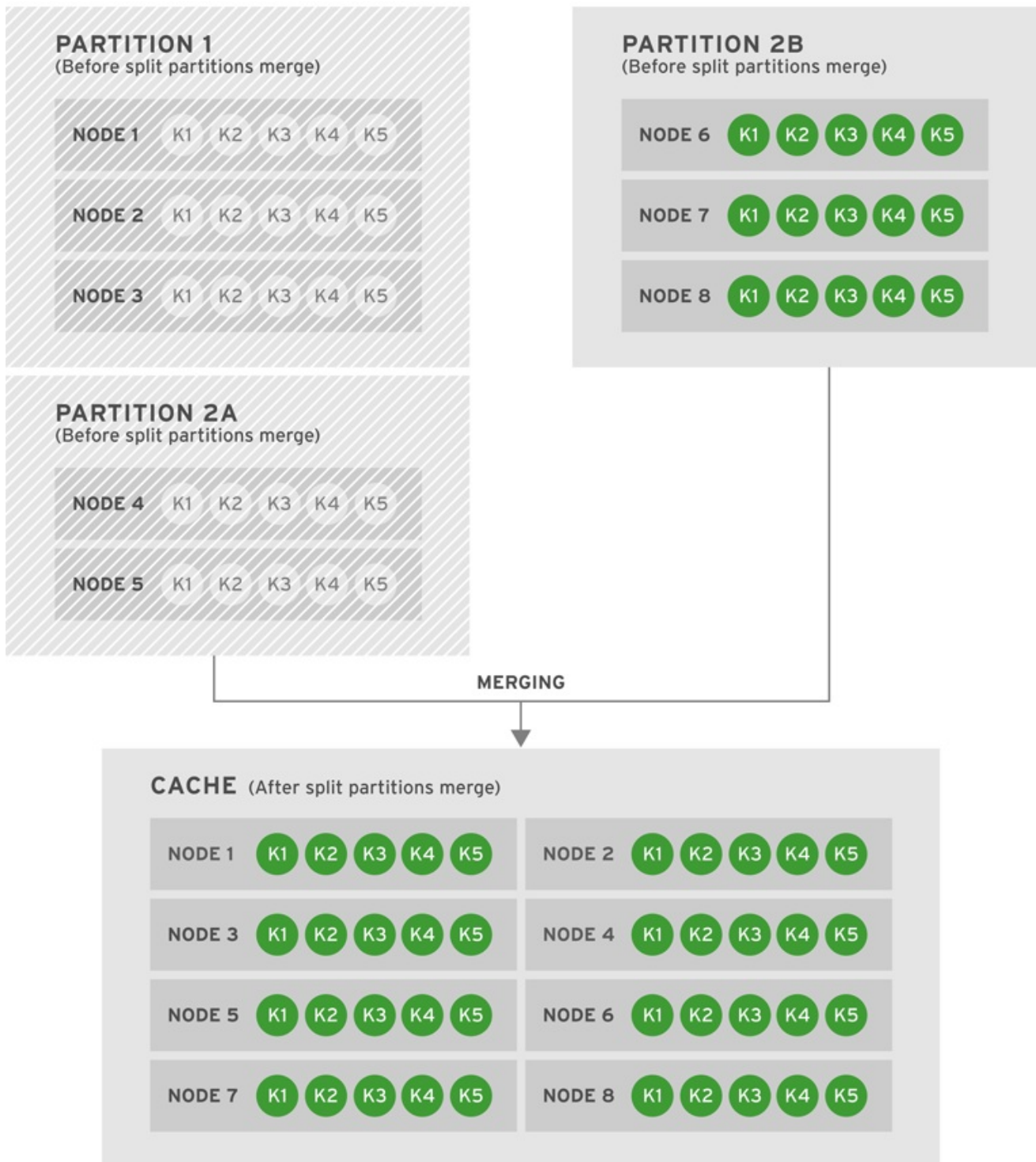
Figure 43.16. Case 3: Partition 1 and 2B Merge



JBOSS_116_335370_0415

The third potential resolution to the partitioned network involves Partition 1 and Partition 2B merging. Partition 1 receives state information from Partition 2B, and the combined partition becomes Available.

Figure 43.17. Case 4: Partition 1, Partition 2A, and Partition 2B Merge Together



JBOSS_117_335370_0415

The fourth and final potential resolution to the partitioned network involves Partition 1, Partition 2A, and Partition 2B merging to form Partition 1. The state is transferred from Partition 2B to both partitions 1 and 2A. The resulting cache contains eight nodes (Node 1, Node 2, Node 3, Node 4, Node 5, Node 6, Node 7, and Node 8) and is Available.

43.7. CONFIGURE PARTITION HANDLING

In Red Hat JBoss Data Grid, partition handling is disabled as a default.

Declarative Configuration (Library Mode)

Enable partition handling declaratively as follows:

```
<distributed-cache name="distributed_cache"
  l1-lifespan="20000">
  <partition-handling enabled="true"/>
</distributed-cache>
```

Declarative Configuration (Remote Client-server Mode)

Enable partition handling declaratively in remote client-server mode by using the following configuration:

```
<subsystem xmlns="urn:infinispan:server:core:8.4" default-cache-
container="clustered">
  <cache-container name="clustered" default-cache="default"
statistics="true">
    <distributed-cache name="default" >
      <partition-handling enabled="true" />
      <locking isolation="READ_COMMITTED" acquire-timeout="30000"
        concurrency-level="1000" striping="false"/>
    </distributed-cache>
  </cache-container>
</subsystem>
```

APPENDIX A. RECOMMENDED JGROUPS VALUES FOR JBOSS DATA GRID

A.1. SUPPORTED JGROUPS PROTOCOLS

The table contains a list of the JGroups protocols supported in JBoss Data Grid.

Table A.1. Supported JGroups Protocols

Protocol	Details
TCP	<p>TCP/IP is a replacement transport for UDP in situations where IP multicast cannot be used, such as operations over a WAN where routers may discard IP multicast packets.</p> <p>TCP is a transport protocol used to send unicast and multicast messages.</p> <ul style="list-style-type: none"> • When sending multicast messages, TCP sends multiple unicast messages. • When using TCP, each message to all cluster members is sent as multiple unicast messages, or one to each member. <p>As IP multicasting cannot be used to discover initial members, another mechanism must be used to find initial membership.</p> <p>Red Hat JBoss Data Grid's Hot Rod is a custom TCP client/server protocol.</p>
UDP	<p>UDP is a transport protocol that uses:</p> <ul style="list-style-type: none"> • IP multicasting to send messages to all members of a cluster. • UDP datagrams for unicast messages, which are sent to a single member. <p>When the UDP transport is started, it opens a unicast socket and a multicast socket. The unicast socket is used to send and receive unicast messages, the multicast socket sends and receives multicast sockets. The physical address of the channel will be the same as the address and port number of the unicast socket.</p>

Protocol	Details
PING	<p>The PING protocol is used for the initial discovery of members. It is used to detect the coordinator, which is the oldest member, by multicasting PING requests to an IP multicast address.</p> <p>Each member responds to the ping with a packet containing the coordinator's and their own address. After a specified number of milliseconds (N) or replies (M), the joiner determines the coordinator from the responses and sends it a JOIN request (handled by GMS). If there is no response, the joiner is considered the first member of the group.</p> <p>PING differs from TCPPING because it used dynamic discovery, which means that a member does not need to know in advance where the other cluster members are. PING uses the transport's IP multicasting abilities to send a discovery request to the cluster. As a result, PING requires UDP as transport.</p>
TCPPING	<p>The TCPPING protocol uses a set of known members and pings them for discovery. This protocol has a static configuration.</p>
MPING	<p>The MPING (Multicast PING) protocol uses IP multicast to discover the initial membership. It can be used with all transports, but is usually used in combination with TCP.</p>
S3_PING	<p>S3_PING is a discovery protocol that is ideal for use with Amazon's Elastic Compute Cloud (EC2) because EC2 does not allow multicast and therefore MPING is not allowed.</p> <p>Each EC2 instance adds a small file to an S3 data container, known as a bucket. Each instance then reads the files in the bucket to discover the other members of the cluster.</p>
JDBC_PING	<p>JDBC_PING is a discovery protocol that utilizes a shared database to store information regarding nodes in the cluster.</p>
TCPGOSSIP	<p>TCPGOSSIP is a discovery protocol that uses one or more configured GossipRouter processes to store information about the nodes in the cluster.</p>

Protocol	Details
MERGE3	<p>The MERGE3 protocol is available in JGroups 3.1 onwards. Unlike MERGE2, in MERGE3, all members periodically send an INFO message with their address (UUID), logical name, physical address and View ID. Periodically, each coordinator reviews the INFO details to ensure that there are no inconsistencies.</p>
FD_ALL	<p>Used for failure detection, FD_ALL uses a simple heartbeat protocol. Each member maintains a table of all other members (except itself) and periodically multicasts a heartbeat. For example, when data or a heartbeat from P is received, the timestamp for P is set to the current time. Periodically, expired members are identified using the timestamp values.</p>
FD SOCK	<p>FD SOCK is a failure detection protocol based on a ring of TCP sockets created between cluster members. Each cluster member connects to its neighbor (the last member connects to the first member), which forms a ring. Member B is suspected when its neighbor A detects an abnormal closing of its TCP socket (usually due to node B crashing). However, if member B is leaving gracefully, it informs member A and does not become suspected when it does exit.</p>
FD_HOST	<p>FD_HOST is a failure detection protocol that detects the crashing or hanging of entire hosts and suspects all cluster members of that host through ICMP ping messages or custom commands. FD_HOST does not detect the crashing or hanging of single members on the local hosts, but only checks whether all the other hosts in the cluster are live and available. It is therefore used in conjunction with other failure detection protocols such as FD_ALL and FD SOCK. This protocol is typically used when multiple cluster members are running on the same physical box.</p> <p>The FD_HOST protocol is supported on Windows for JBoss Data Grid. The cmd parameter must be set to ping.exe and the ping count must be specified.</p>
VERIFY_SUSPECT	<p>The VERIFY_SUSPECT protocol verifies whether a suspected member is dead by pinging the member before excluding it. If the member responds, the suspect message is discarded.</p>

Protocol	Details
NAKACK2	<p>The NAKACK2 protocol is a successor to the NAKACK protocol and was introduced in JGroups 3.1.</p> <p>The NACKACK2 protocol is used for multicast messages and uses NAK. Each message is tagged with a sequence number. The receiver tracks the sequence numbers and delivers the messages in order. When a gap in the sequence numbers is detected, the receiver asks the sender to retransmit the missing message.</p>
UNICAST3	<p>The UNICAST3 protocol provides reliable delivery (no message sent by a sender is lost because they are sent in a numbered sequence) and uses the FIFO (First In First Out) properties for point to point messages between a sender and a receiver.</p> <p>UNICAST3 uses positive acks for retransmission. For example, sender A keeps sending message M until receiver B receives message M and returns an ack to indicate a successful delivery. Sender A keeps resending message M until it receives an ack from B, until B leaves the cluster, or A crashes.</p>
STABLE	<p>The STABLE protocol is a garbage collector for messages that have been viewed by all members in a cluster. Each member stores all messages because retransmission may be required. A message can only be removed from the retransmission buffers when all members have seen the message. The STABLE protocol periodically gossips its highest and lowest messages seen. The lowest value is used to compute the min (all lowest sequence numbers for all members) and messages with a sequence number below the min value can be discarded</p>
GMS	<p>The GMS protocol is the group membership protocol. This protocol handles joins/leaves/crashes (suspicions) and emits new views accordingly.</p>
MFC	<p>MFC is the Multicast version of the flow control protocol.</p>
UFC	<p>UFC is the Unicast version of the flow control protocol.</p>

Protocol	Details
FRAG3	<p>The FRAG3 protocol fragments large messages into smaller ones and then sends the smaller messages. At the receiver side, the smaller fragments are reassembled into larger, complete messages and delivered to the application. FRAG3 is used for both multicast and unicast messages.</p>
SYM_ENCRYPT	<p>JGroups includes the SYM_ENCRYPT protocol to provide encryption for cluster traffic. By default, encryption only encrypts the message body; it does not encrypt message headers. To encrypt the entire message, including all headers, as well as destination and source addresses, the property encrypt_entire_message must be true. When defining this protocol it should be placed directly under NAKACK2.</p> <p>The SYM_ENCRYPT layer is used to encrypt and decrypt communication in JGroups by defining a secret key in a keystore.</p> <p>Each message is identified as encrypted with a specific encryption header identifying the encrypt header and an MD5 digest identifying the version of the key being used to encrypt and decrypt messages.</p>
ASYM_ENCRYPT	<p>JGroups includes the ASYM_ENCRYPT protocol to provide encryption for cluster traffic. By default, encryption only encrypts the message body; it does not encrypt message headers. To encrypt the entire message, including all headers, as well as destination and source addresses, the property encrypt_entire_message must be true. When defining this protocol it should be placed directly under NAKACK2.</p> <p>The ASYM_ENCRYPT layer is used to encrypt and decrypt communication in JGroups by having a coordinator generate a secret key using defined algorithms and key sizes.</p> <p>Each message is identified as encrypted with a specific encryption header identifying the encrypt header and an MD5 digest identifying the version of the key being used to encrypt and decrypt messages.</p>
SASL	<p>The SASL (Simple Authentication and Security Layer) protocol is a framework that provides authentication and data security services in connection-oriented protocols using replaceable mechanisms. Additionally, SASL provides a structured interface between protocols and mechanisms.</p>

Protocol	Details
RELAY2	<p>The RELAY protocol bridges two remote clusters by creating a connection between one node in each site. This allows multicast messages sent out in one site to be relayed to the other and vice versa.</p> <p>JGroups includes the RELAY2 protocol, which is used for communication between sites in Red Hat JBoss Data Grid's Cross-Site Replication.</p> <p>The RELAY2 protocol works similarly to RELAY but with slight differences. Unlike RELAY, the RELAY2 protocol:</p> <ul style="list-style-type: none"> • connects more than two sites. • connects sites that operate autonomously and are unaware of each other. • offers both unicast and multicast routing between sites.

A.2. TCP DEFAULT AND RECOMMENDED VALUES

To learn more about JGroups and using TCP and UDP, see [Configure JGroups \(Library Mode\)](#).



NOTE

- Values in **JGroups Default Value** indicate values that are configured internally to JGroups, but may be overridden by a custom configuration file or by a JGroups configuration file shipped with JBoss Data Grid.
- Values in **JBoss Data Grid Configured Values** indicate values that are in use by default when using one of the configuration files for JGroups as shipped with JBoss Data Grid. It is recommended to use these values when custom configuration files for JGroups are in use with JBoss Data Grid. For more information on the configuration files included with JBoss Data Grid refer to [JBoss Data Grid JGroups Configuration Files](#).

Table A.2. Recommended and Default Values for TCP

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
bind_addr	Any non-loopback	Set address on specific interface
bind_port	Any free port	Set specific port
loopback	true	Same as default

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
port_range	50	Set based on desired range of ports
recv_buf_size	150,000	Same as default
send_buf_size	150,000	640,000
sock_conn_timeout	2,000	300
bundler_type	transfer-queue	no-bundler
max_bundle_size	64,000	64,000
enable_diagnostics	true	false
thread_pool.enabled	true	Same as default
thread_pool.min_threads	2	This should equal the number of nodes
thread_pool.max_threads	30	This should be higher than thread_pool.min_threads . For example, for a smaller grid (2-10 nodes), set this value to twice the number of nodes, but for a larger grid (20 or more nodes), the ratio should be lower. As an example, if a grid contains 20 nodes, set this value to 25 and if the grid contains 100 nodes, set the value to 110.
thread_pool.keep_alive_time	30,000	60,000



NOTE

Red Hat JBoss Data Grid 7.1 uses JGroups 4.0.0.CR2, in which the TCP ping timeout value no longer exists. Use the `pbcast.GMS join_timeout` value to indicate the timeout period instead.

Recommended Values for S3_PING

See [S3_PING Configuration Options](#) for details about configuring S3_PING for JBoss Data Grid.

Recommended Values for TCPGOSSIP

See [TCPCGOSSIP Configuration Options](#) for details about configuring TCPCGOSSIP for JBoss Data Grid.

Table A.3. Recommended Values for MPING

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
bind_addr	Any non-loopback	Set address on specific interface
break_on_coord_rsp	true	Same as default
mcast_addr	230.5.6.7	228.2.4.6
mcast_port	7555	43366
ip_ttl	8	2



NOTE

In JGroups 3.6.1, the MPING timeout value was removed and the `pbcast.GMS join_timeout` value indicates the timeout period instead.

Table A.4. Recommended Values for MERGE3

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
min_interval	1,000	10,000
max_interval	10,000	30,000

Table A.5. Recommended Values for FD_SOCK

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
client_bind_por	0 (randomly selects a port and uses it)	Same as default
get_cache_timeout	1000 milliseconds	Same as default
keep_alive	true	Same as default
num_tries	3	Same as default
start_port	0 (randomly selects a port and uses it)	Same as default

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
suspect_msg_interval	5000 milliseconds.	Same as default

Table A.6. Recommended Values for FD_ALL

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
timeout	40,000	60,000. The FD_ALL timeout value is set to two times the longest possible stop the world garbage collection pause in the CMS garbage collector. In a well-tuned JVM, the longest pause is proportional to heap size and should not exceed 1 second per GB of heap. For example, an 8GB heap should not have a pause longer than 8 seconds, so the FD_ALL timeout value must be set to 16 seconds. If longer garbage collection pauses are used, then this timeout value should be increased to avoid false failure detection on a node.
interval	8,000	15,000. The FD_ALL interval value must be at least four times smaller than the value set for FD_ALL's timeout value.
timeout_check_interval	2,000	5,000

Table A.7. Recommended Values for FD_HOST

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
check_timeout	3,000	5,000
cmd	InetAddress.isReachable() (ICMP ping)	-
interval	20,000	15,000. The interval value for FD_HOST must be four times smaller than FD_HOST's timeout value.

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
timeout	60,000	60,000.

Table A.8. Recommended Values for VERIFY_SUSPECT

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
timeout	2,000	5,000

Table A.9. Recommended Values for pbcast.NAKACK2

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
use_mcast_xmit	true	false
xmit_interval	1,000	100
xmit_table_num_rows	50	Same as default
xmit_table_msgs_per_row	10,000	1,024
xmit_table_max_compaction_time	10,000	30,000
max_msg_batch_size	100	Same as default
resend_last_seqno	false	true

Table A.10. Recommended Values for UNICAST3

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
xmit_interval	500	100
xmit_table_num_rows	100	50
xmit_table_msgs_per_row	1,0000	1,024
xmit_table_max_compaction_time	600,000	30,000
max_msg_batch_size	500	100

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
conn_close_timeout	60,000	No recommended value.
conn_expiry_timeout	120,000	0

Table A.11. Recommended Values for pbcaster.STABLE

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
stability_delay	6,000	500
desired_avg_gossip	20,000	5,000
max_bytes	2,000,000	1,000,000

Table A.12. Recommended Values for pbcaster.GMS

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
print_local_addr	true	false
join_timeout	5,000	Same as default
view_bundling	true	Same as default

Table A.13. Recommended Values for MFC

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
max_credits	500,000	2,000,000
min_threshold	0.40	Same as default

Table A.14. Recommended Values for FRAG3

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
frag_size	60,000	Same as default

Table A.15. Recommended Values for SYM_ENCRYPT

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
sym_algorithm	AES	-
sym_keylength	128	-
provider	Bouncy Castle Provider	-

Table A.16. Recommended Values for ASYM_ENCRYPT

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
asym_algorithm	RSA	-
asym_keylength	512	-
sym_provider	Bouncy Castle Provider	-
change_key_on_leave	false	-

Recommended Values for SASL

See the *Red Hat JBoss Data Grid Developer Guide's User Authentication over Hot Rod Using SASL* section for details.

Recommended Values for RELAY2

See [Set Up Cross-Datacenter Replication](#) for details.

A.3. UDP DEFAULT AND RECOMMENDED VALUES

To learn more about JGroups and using TCP and UDP, see [Configure JGroups \(Library Mode\)](#).



NOTE

- Values in **JGroups Default Value** indicate values that are configured internally to JGroups, but may be overridden by a custom configuration file or by a JGroups configuration file shipped with JBoss Data Grid.
- Values in **JBoss Data Grid Configured Values** indicate values that are in use by default when using one of the configuration files for JGroups as shipped with JBoss Data Grid. It is recommended to use these values when custom configuration files for JGroups are in use with JBoss Data Grid. For more information on the configuration files included with JBoss Data Grid refer to [JBoss Data Grid JGroups Configuration Files](#).

Table A.17. Recommended Values for UDP

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
bind_addr	Any non-loopback	Set address on specific interface
bind_port	Any free port	Set specific port
loopback	true	true
port_range	50	Set based on desired range of ports
mcast_addr	228.8.8.8	228.6.7.8
mcast_port	7600	46655
tos	8	Same as default
ucast_recv_buf_size	64,000	5,000,000
ucast_send_buf_size	100,000	2,000,000
mcast_recv_buf_size	500,000	5,000,000
mcast_send_buf_size	100,000	1,000,000
ip_ttl	8	2
thread_naming_pattern	cl	pl
bundler_type	transfer-queue	no-bundler
max_bundle_size	64,000	8700
enable_diagnostics	true	false
thread_pool.enabled	true	Same as default
thread_pool.min_threads	2	This should equal the number of nodes.

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
thread_pool.max_threads	30	This should be higher than thread_pool.min_threads. For example, for a smaller grid (2-10 nodes), set this value to twice the number of nodes, but for a larger grid (20 or more nodes), the ratio should be lower. As an example, if a grid contains 20 nodes, set this value to 25 and if the grid contains 100 nodes, set the value to 110.
thread_pool.keep_alive_time	30,000	60,000

**NOTE**

In JGroups 3.5, the PING timeout value is removed and the pbcast.GMS **join_timeout** value indicates the timeout period instead.

Table A.18. Recommended Values for MERGE3

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
min_interval	1,000	10,000
max_interval	10,000	30,000

Table A.19. Recommended Values for FD SOCK

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
client_bind_por	0 (randomly selects a port and uses it)	Same as default
get_cache_timeout	1000 milliseconds	Same as default
keep_alive	true	Same as default
num_tries	3	Same as default
start_port	0 (randomly selects a port and uses it)	Same as default
suspect_msg_interval	5000 milliseconds.	Same as default

Table A.20. Recommended Values for FD_ALL

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
timeout	40,000	60,000. The FD_ALL timeout value is set to two times the longest possible stop the world garbage collection pause in the CMS garbage collector. In a well-tuned JVM, the longest pause is proportional to heap size and should not exceed 1 second per GB of heap. For example, an 8GB heap should not have a pause longer than 8 seconds, so the FD_ALL timeout value must be set to 16 seconds. If longer garbage collection pauses are used, then this timeout value should be increased to avoid false failure detection on a node.
interval	8,000	15,000. The FD_ALL interval value must be at least four times smaller than the value set for FD_ALL's timeout value.
timeout_check_interval	2,000	5,000

Table A.21. Recommended Values for FD_HOST

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
check_timeout	3,000	5,000
cmd	InetAddress.isReachable() (ICMP ping)	-
interval	20,000	15,000. The interval value for FD_HOST must be four times smaller than FD_HOST's timeout value.
timeout	-	60,000.

Table A.22. Recommended Values for VERIFY_SUSPECT

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
timeout	2,000	5,000

Table A.23. Recommended Values for pbcast.NAKACK2

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
use_mcast_xmit	true	Same as default
xmit_interval	1,000	100
xmit_table_num_rows	50	Same as default
xmit_table_msgs_per_row	10,000	1,024
xmit_table_max_compaction_time	10,000	30,000
max_msg_batch_size	100	Same as default
resend_last_seqno	false	true

Table A.24. Recommended Values for UNICAST3

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
xmit_interval	500	100
xmit_table_num_rows	100	50
xmit_table_msgs_per_row	1,0000	1,024
xmit_table_max_compaction_time	600,000	30,000
max_msg_batch_size	500	100
conn_close_timeout	60,000	No recommended value
conn_expiry_timeout	120,000	0

Table A.25. Recommended Values for pbcast.STABLE

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
stability_delay	6,000	500
desired_avg_gossip	20,000	5,000
max_bytes	2,000,000	1,000,000

Table A.26. Recommended Values for pbcaster.GMS

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
print_local_addr	true	false
join_timeout	5,000	Same as default
view_bundling	true	Same as default

Table A.27. Recommended Values for UFC

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
max_credits	500,000	2,000,000
min_threshold	0.40	Same as default

Table A.28. Recommended Values for MFC

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
max_credits	500,000	2,000,000
min_threshold	0.40	Same as default

Table A.29. Recommended Values for FRAG3

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
frag_size	60,000	Same as default

Table A.30. Recommended Values for SYM_ENCRYPT

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
sym_algorithm	AES	-
sym_keylength	128	-
provider	Bouncy Castle Provider	-

Table A.31. Recommended Values for ASYM_ENCRYPT

Parameter	JGroups Default Value	JBoss Data Grid Configured Values
asym_algorithm	RSA	-
asym_keylength	512	-
provider	Bouncy Castle Provider	-
change_key_on_leave	false	-

Recommended Values for SASL

See the *Red Hat JBoss Data Grid Developer Guide's User Authentication over Hot Rod Using SASL* section for details.

Recommended Values for RELAY2

See [Set Up Cross-Datacenter Replication](#) for details.

A.4. THE TCPGOSSIP JGROUPS PROTOCOL

The **TCPGOSSIP** discovery protocol uses one or more configured GossipRouter processes to store information about the nodes in the cluster.



IMPORTANT

It is vital that the GossipRouter process consistently be available to all nodes in the cluster, as without this process it will not be possible to add additional nodes. For this reason it is strongly recommended to deploy this process in a highly available method; for example, an Availability Set with multiple virtual machines may be used.

Running the GossipRouter

The GossipRouter is included in the JGroups jar file, and must be running before any nodes are started. This process may be started by pointing to the **GossipRouter** class in the JGroups jar file included with JBoss Data Grid:

```
java -classpath jgroups-${jgroups.version}.jar
org.jgroups.stack.GossipRouter -bindaddress IP_ADDRESS -port PORT
```

In the event that multiple GossipRouters are available, and specified, a node will always register with all specified GossipRouters; however, it will only retrieve information from the first available GossipRouter. If a GossipRouter is unavailable it will be marked as failed and removed from the list, with a background thread started to periodically attempt reconnecting to the failed GossipRouter. Once the thread successfully reconnects the GossipRouter will be reinserted into the list.

Configuring JBoss Data Grid to use TCPGOSSIP (Library Mode)

In Library Mode the JGroups xml file should be used to configure **TCPGOSSIP**; however, there is no **TCPGOSSIP** configuration included by default. It is recommended to use one of the preexisting files specified in [Pre-Configured JGroups Files](#) and then adjust the configuration to include **TCPGOSSIP**. For instance, *default-configs/default-jgroups-ec2.xml* could be selected and the **S3_PING** protocol removed, and then the following block added in its place:

```
<TCPGOSSIP initial_hosts="IP_ADDRESS_0[PORT_0],IP_ADDRESS_1[PORT_1]" />
```

Configuring JBoss Data Grid to use TCPGOSSIP (Remote Client-Server Mode)

In Remote Client-Server Mode a **stack** may be defined for **TCPGOSSIP** in the **jgroups** subsystem of the server's configuration file. The following configuration snippet contains an example of this:

```
<subsystem xmlns="urn:infinispan:server:jgroups:8.0" default-
stack="${jboss.default.jgroups.stack:tcpgossip}">
[... ]
  <stack name="jdbc_ping">
    <transport type="TCP" socket-binding="jgroups-tcp"/>
    <protocol type="TCPGOSSIP">
      <property
name="initial_hosts">IP_ADDRESS_0[PORT_0],IP_ADDRESS_1[PORT_1]</property>
    </protocol>
    <protocol type="MERGE3"/>
    <protocol type="FD_SOCK" socket-binding="jgroups-tcp-fd"/>
    <protocol type="FD_ALL"/>
    <protocol type="VERIFY_SUSPECT"/>
    <protocol type="pbcast.NAKACK2">
      <property name="use_mcast_xmit">false</property>
    </protocol>
    <protocol type="UNICAST3"/>
    <protocol type="pbcast.STABLE"/>
    <protocol type="pbcast.GMS"/>
    <protocol type="MFC"/>
    <protocol type="FRAG3"/>
  </stack>
[... ]
</subsystem>
```

A.5. TCPGOSSIP CONFIGURATION OPTIONS

The following **TCPGOSSIP** specific properties may be configured:

- **initial_hosts** - Comma delimited list of hosts to be contacted for initial membership.

- **reconnect_interval** - Interval (in milliseconds) by which a disconnected node attempts to reconnect to the Gossip Router.
- **sock_conn_timeout** - Max time (in milliseconds) allowed for socket creation. Defaults to **1000**.

A.6. JBOSS DATA GRID JGROUPS CONFIGURATION FILES

The following configuration files are included with JBoss Data Grid and contain the recommended values for JGroups. All of the following files are included in *infinispan-embedded- $\{infinispan.version\}$.jar* found with the Library mode distribution.

- **default-configs/default-jgroups-ec2.xml**
- **default-configs/default-jgroups-google.xml**
- **default-configs/default-jgroups-tcp.xml**
- **default-configs/default-jgroups-udp.xml**

APPENDIX B. HOTROD.PROPERTIES

B.1. HOTROD.PROPERTIES

The following is a list of parameters that may be used to configure the behavior of a **RemoteCacheManager**. These elements are placed into the **hotrod-client.properties** file which is read when starting the application.

Table B.1. Hotrod-Client Configuration Properties

Name	Description
<code>infinispan.client.hotrod.request_balancing_strategy</code>	For replicated (vs. distributed) Hot Rod server clusters, the client balances requests to the servers according to this strategy. Defaults to <code>org.infinispan.client.hotrod.impl.transport.tcp.RoundRobinBalancingStrategy</code>
<code>infinispan.client.hotrod.server_list</code>	This is the initial list of Hot Rod servers to connect to, specified in the following format: <code>host1:port1;host2:port2...</code> . At least one <code>host:port</code> must be specified. Defaults to <code>127.0.0.1:11222</code> .
<code>infinispan.client.hotrod.force_return_values</code>	Whether or not to implicitly force return values for all calls. Defaults to <code>false</code> .
<code>infinispan.client.hotrod.tcp_no_delay</code>	Affects TCP NODELAY on the TCP stack. Defaults to <code>true</code> .
<code>infinispan.client.hotrod.tcp_keep_alive</code>	Affects TCP KEEPALIVE on the TCP stack. Defaults to <code>false</code> .
<code>infinispan.client.hotrod.ping_on_startup</code>	If true, a ping request is sent to a back end server in order to fetch cluster's topology. Defaults to <code>true</code> .
<code>infinispan.client.hotrod.transport_factory</code>	Controls which transport to use. Currently only the <code>TcpTransport</code> is supported. Defaults to <code>org.infinispan.client.hotrod.impl.transport.tcp.TcpTransportFactory</code> .
<code>infinispan.client.hotrod.marshaller</code>	Allows you to specify a custom <code>org.infinispan.marshall.Marshaller</code> implementation to serialize and deserialize user objects. For portable serialization payloads, you should configure the marshaller to be <code>org.infinispan.client.hotrod.marshall1.ProtoStreamMarshaller</code> . Defaults to <code>org.infinispan.marshall.jboss.GenericJBossMarshaller</code> .

Name	Description
infinispan.client.hotrod.async_executor_factory	Allows you to specify a custom asynchronous executor for async calls. Defaults to org.infinispan.client.hotrod.impl.async.DefaultAsyncExecutorFactory .
infinispan.client.hotrod.default_executor_factory.pool_size	If the default executor is used, this configures the number of threads to initialize the executor with. Defaults to 10 .
infinispan.client.hotrod.default_executor_factory.queue_size	If the default executor is used, this configures the queue size to initialize the executor with. Defaults to 100000 .
infinispan.client.hotrod.hash_function_impl.1	This specifies the version of the hash function and consistent hash algorithm in use, and is closely tied with the HotRod server version used. By default it uses the hash function specified by the server in the responses as indicated in org.infinispan.client.hotrod.impl.consistenthash.ConsistentHashFactory .
infinispan.client.hotrod.key_size_estimate	This hint allows sizing of byte buffers when serializing and deserializing keys, to minimize array resizing. Defaults to 64 .
infinispan.client.hotrod.value_size_estimate	This hint allows sizing of byte buffers when serializing and deserializing values, to minimize array resizing. Defaults to 512 .
infinispan.client.hotrod.socket_timeout	This property defines the maximum socket read timeout before giving up waiting for bytes from the server. Defaults to 60000 (60 seconds).
infinispan.client.hotrod.protocol_version	This property defines the protocol version that this client should use. Defaults to 2.0 .
infinispan.client.hotrod.connect_timeout	This property defines the maximum socket connect timeout before giving up connecting to the server. Defaults to 60000 (60 seconds).
infinispan.client.hotrod.max_retries	This property defines the maximum number of retries in case of a recoverable error. A valid value should be greater or equals to 0 (zero). Zero mean no retry. Defaults to 10 .
infinispan.client.hotrod.use_ssl	This property defines if SSL is enabled. Defaults to false .

Name	Description
infinispan.client.hotrod.key_store_file_name	Specifies the filename of a keystore to use to create the SSLContext . A key_store_password must also be defined.
infinispan.client.hotrod.key_store_password	Specifies the password needed to open the keystore. A key_store_file_name must also be defined.
infinispan.client.hotrod.trust_store_file_name	Specifies the filename of a truststore to use to create the SSLContext . A trust_store_password must also be defined.
infinispan.client.hotrod.trust_store_password	Specified the password needed to open the truststore. A trust_store_file_name must also be defined.

APPENDIX C. CONNECTING WITH JCONSOLE

C.1. CONNECT TO JDG VIA JCONSOLE

JConsole is a JMX GUI that allows a user to connect to a JVM, either local or remote, to monitor the JVM, its MBeans, and execute operations.

Add Management User to JBoss Data Grid

Before being able to connect to a remote JBoss Data Grid instance a user will need to be created; to add a user execute the following steps on the remote instance.

1. Navigate to the **bin** directory

```
cd $JDG_HOME/bin
```

2. Execute the **add-user.sh** script.

```
./add-user.sh
```

3. Accept the default option of **ManagementUser** by pressing return.
4. Accept the default option of **ManagementRealm** by pressing return.
5. Enter the desired username. In this example **jmxadmin** will be used.
6. Enter and confirm the password.
7. Accept the default option of no groups by pressing return.
8. Confirm that the desired user will be added to the **ManagementRealm** by entering **yes**.
9. Enter **no** as this user will not be used for connections between processes.
10. The following image shows an example execution run.

Figure C.1. Execution of add-user.sh

```
what type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a):
Enter the details of the new user to add.
Realm (ManagementRealm) :
Username : jmxadmin
Password requirements are listed below. To modify these restrictions edit the add-user.properties configuration file.
- The password must not be one of the following restricted values {root, admin, administrator}
- The password must contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
- The password must be different from the username
Password :
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[ ]:
About to add user 'jmxadmin' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'jmxadmin' to file '/opt/jboss/jboss-datagrid-6.5.0-server/standalone/configuration/mgmt-users.properties'
Added user 'jmxadmin' with groups to file '/opt/jboss/jboss-datagrid-6.5.0-server/standalone/configuration/mgmt-groups.properties'
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.
yes/no? no
```

Binding the Management Interface

By default JBoss Data Grid will start with the management interface binding to 127.0.0.1. In order to connect remotely this interface must be bound to an IP address that is visible by the network. Either of the following options will correct this:

- **Option 1: Runtime** - By adjusting the `jboss.bind.address.management` property on startup a new IP address can be specified. In the following example JBoss Data Grid is starting with this bound to 192.168.122.5:

```
./standalone.sh ... -Djboss.bind.address.management=192.168.122.5
```

- **Option 2: Configuration** - Adjust the `jboss.bind.address.management` in the configuration file. This is found in the `interfaces` subsystem. A snippet of the configuration file, with the IP adjusted to 192.168.122.5, is provided below:

```
<interfaces>
  <interface name="management">
    <inet-address
value="{jboss.bind.address.management:192.168.122.5}"/>
    </interface>
    [...]
  </interface>
```

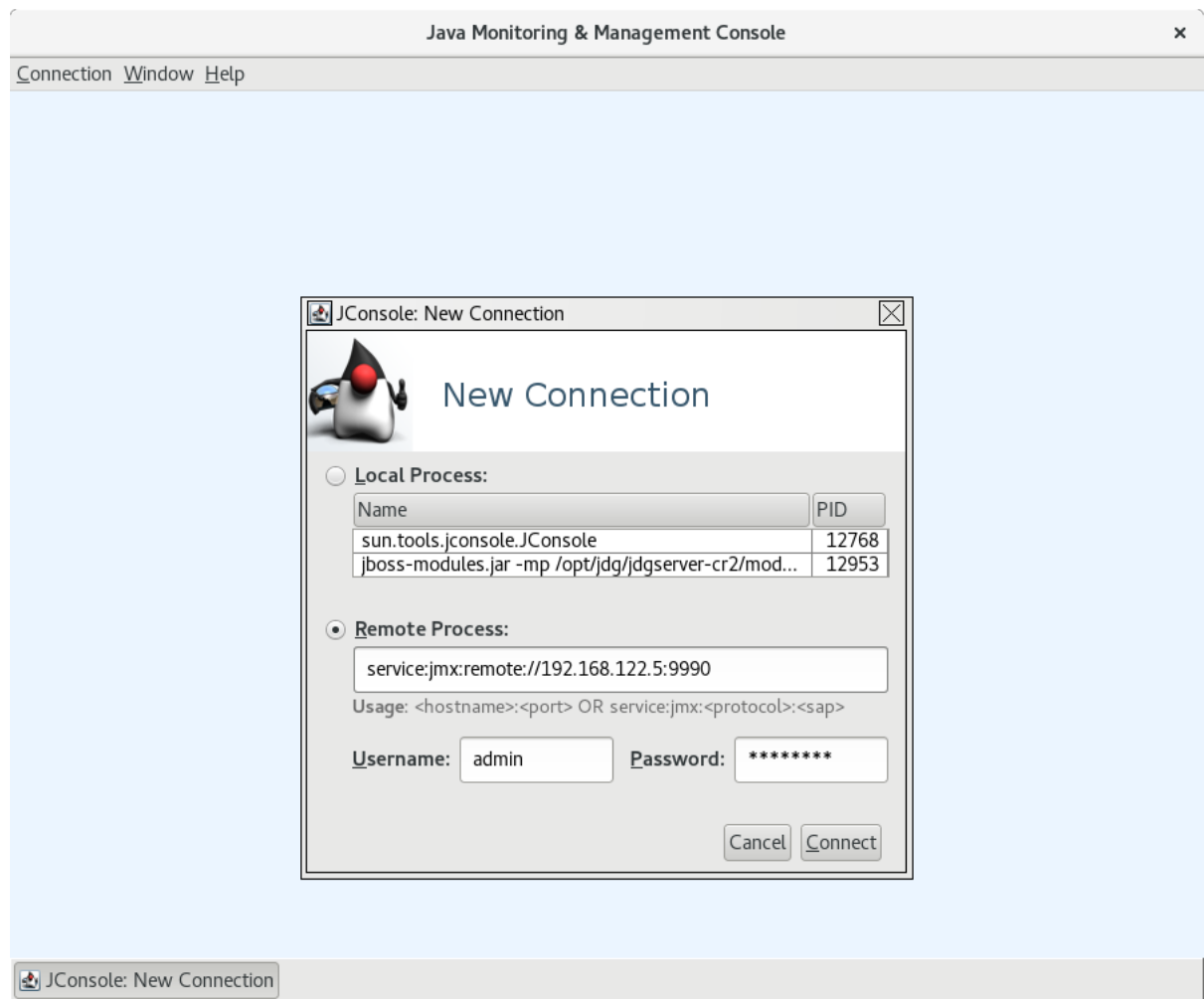
Running JConsole

A `jconsole.sh` script is provided in the `$JDG_HOME/bin` directory. Executing this script will launch JConsole.

Connecting to a remote JBoss Data Grid instance using JConsole.

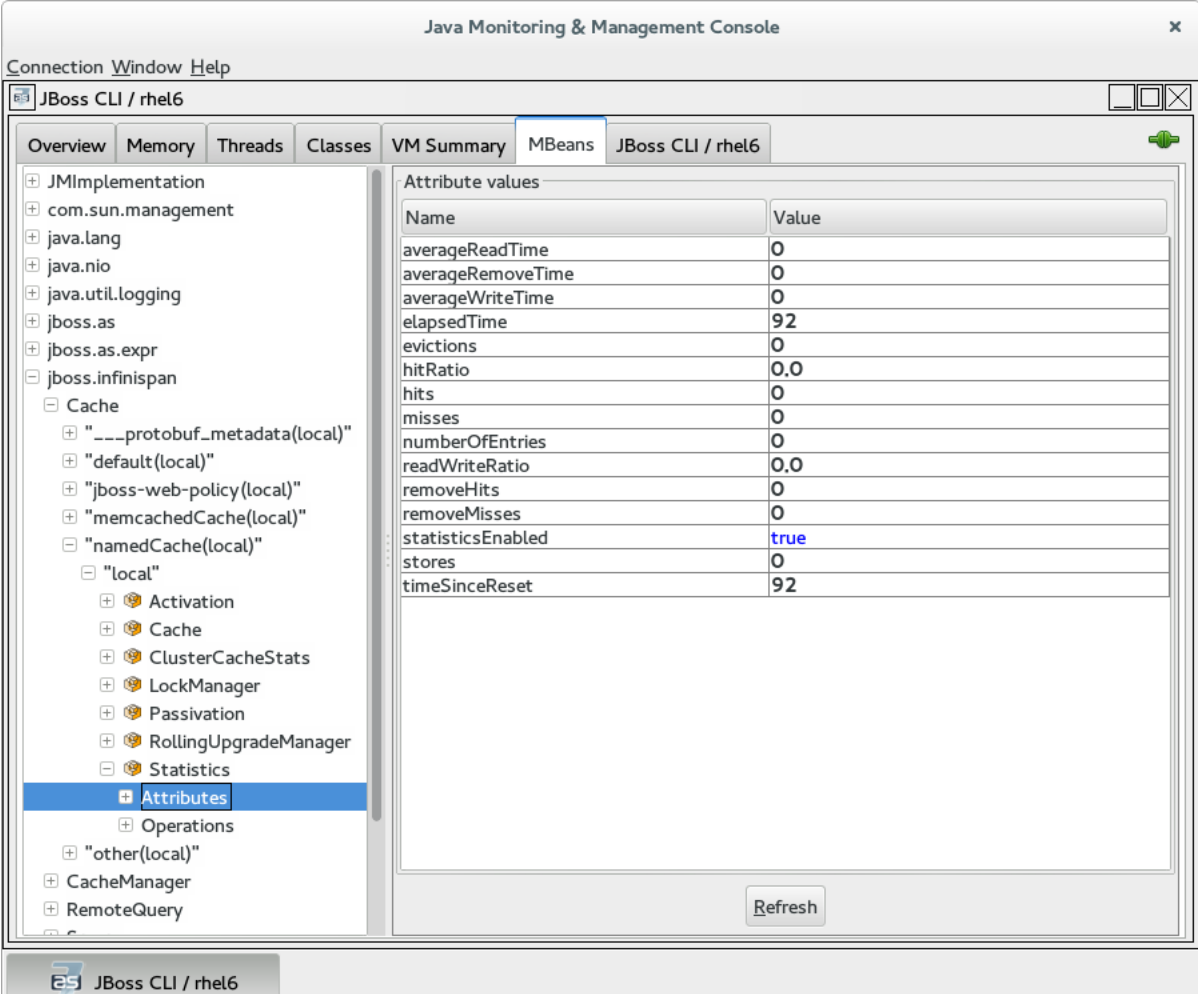
1. Execute the `$JDG_HOME/bin/jconsole.sh` script. This will result in the following window appearing:

Figure C.2. JConsole



2. Select **Remote Process**.
3. Enter **service:jmx:remote://\$IP:9990** in the text area.
4. Enter the username and password, created from the **add-user.sh** script.
5. Click **Connect** to initiate the connection.
6. Once connected ensure that the cache-related nodes may be viewed. The following screenshot shows such a node.

Figure C.3. JConsole: Showing a Cache



Java Monitoring & Management Console

Connection Window Help

JBoss CLI / rhel6

Overview Memory Threads Classes VM Summary MBeans JBoss CLI / rhel6

Tree View:

- JMImplementation
- com.sun.management
- java.lang
- java.nio
- java.util.logging
- jboss.as
- jboss.as.expr
- jboss.infinispan
 - Cache
 - "__protobuf_metadata(local)"
 - "default(local)"
 - "jboss-web-policy(local)"
 - "memcachedCache(local)"
 - "namedCache(local)"
 - "local"
 - Activation
 - Cache
 - ClusterCacheStats
 - LockManager
 - Passivation
 - RollingUpgradeManager
 - Statistics
 - Attributes**
 - Operations
 - "other(local)"
 - CacheManager
 - RemoteQuery

Attribute values:

Name	Value
averageReadTime	0
averageRemoveTime	0
averageWriteTime	0
elapsedTime	92
evictions	0
hitRatio	0.0
hits	0
misses	0
numberOfEntries	0
readWriteRatio	0.0
removeHits	0
removeMisses	0
statisticsEnabled	true
stores	0
timeSinceReset	92

Refresh

JBoss CLI / rhel6

APPENDIX D. JMX MBEANS IN RED HAT JBOSS DATA GRID

D.1. ACTIVATION

`org.infinispan.eviction.ActivationManagerImpl`

Activates entries that have been passivated to the CacheStore by loading the entries into memory.

Table D.1. Attributes

Name	Description	Type	Writable
activations	Number of activation events.	String	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table D.2. Operations

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

D.2. CACHE

`org.infinispan.CacheImpl`

The Cache component represents an individual cache instance.

Table D.3. Attributes

Name	Description	Type	Writable
cacheName	Returns the cache name.	String	No
cacheStatus	Returns the cache status.	String	No
configurationAsProperties	Returns the cache configuration in form of properties.	Properties	No
version	Returns the version of Infinispan	String	No

Name	Description	Type	Writable
cacheAvailability	Returns the cache availability	String	Yes

Table D.4. Operations

Name	Description	Signature
start	Starts the cache.	void start()
stop	Stops the cache.	void stop()
clear	Clears the cache.	void clear()

D.3. CACHECONTAINERSTATS

`org.infinispan.stats.impl.CacheContainerStatsImpl`

The `CacheContainerStats` component contains statistics such as timings, hit/miss ratio, and operation information.

Table D.5. Attributes

Name	Description	Type	Writable
averageReadTime	Cache container total average number of milliseconds for all read operations in this cache container.	long	No
averageRemoveTime	Cache container total average number of milliseconds for all remove operations in this cache container.	long	No
averageWriteTime	Cache container total average number of milliseconds for all write operations in this cache container.	long	No
evictions	Cache container total number of cache eviction operations.	long	No

Name	Description	Type	Writable
hitRatio	Cache container total percentage hit/(hit+miss) ratio for this cache.	double	No
hits	Cache container total number of cache attribute hits.	long	No
misses	Cache container total number of cache attribute misses.	long	No
numberOfEntries	Cache container total number of entries currently in all caches from this cache container.	int	No
readWriteRatio	Cache container read/writes ratio in all caches from this cache container.	double	No
removeHits	Cache container total number of removal hits.	double	No
removeMisses	Cache container total number of cache removals where keys were not found.	long	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes
stores	Cache container total number of cache attribute put operations.	long	No

D.4. CACHELOADER

`org.infinispan.interceptors.CacheLoaderInterceptor`

This component loads entries from a `CacheStore` into memory.

Table D.6. Attributes

Name	Description	Type	Writable
cacheLoaderLoads	Number of entries loaded from the cache store.	long	No
cacheLoaderMisses	Number of entries that did not exist in cache store.	long	No
stores	Returns a collection of cache loader types which are configured and enabled.	Collection	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table D.7. Operations

Name	Description	Signature
disableStore	Disable all cache loaders of a given type, where type is a fully qualified class name of the cache loader to disable.	void disableStore(String storeType)
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

D.5. CACHEMANAGER

`org.infinispan.manager.DefaultCacheManager`

The CacheManager component acts as a manager, factory, and container for caches in the system.

Table D.8. Attributes

Name	Description	Type	Writable
cacheManagerStatus	The status of the cache manager instance.	String	No
clusterMembers	Lists members in the cluster.	String	No
clusterName	Cluster name.	String	No

Name	Description	Type	Writable
clusterSize	Size of the cluster in the number of nodes.	int	No
createdCacheCount	The total number of created caches, including the default cache.	String	No
definedCacheCount	The total number of defined caches, excluding the default cache.	String	No
definedCacheNames	The defined cache names and their statuses. The default cache is not included in this representation.	String	No
name	The name of this cache manager.	String	No
nodeAddress	The network address associated with this instance.	String	No
physicalAddresses	The physical network addresses associated with this instance.	String	No
runningCacheCount	The total number of running caches, including the default cache.	String	No
version	Infinispan version.	String	No
globalConfigurationAsProperties	Global configuration properties	Properties	No

Table D.9. Operations

Name	Description	Signature
startCache	Starts the default cache associated with this cache manager.	void startCache()

Name	Description	Signature
startCache	Starts a named cache from this cache manager.	void startCache (String p0)

D.6. CACHESTORE

`org.infinispan.interceptors.CacheWriterInterceptor`

The CacheStore component stores entries to a CacheStore from memory.

Table D.10. Attributes

Name	Description	Type	Writable
writesToTheStores	Number of writes to the store.	long	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table D.11. Operations

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

D.7. CLUSTERCACHESTATS

`org.infinispan.stats.impl.ClusterCacheStatsImpl`

The ClusterCacheStats component contains statistics such as timings, hit/miss ratio, and operation information for the whole cluster.

Table D.12. Attributes

Name	Description	Type	Writable
activations	The total number of activations in the cluster.	long	No
averageReadTime	Cluster wide total average number of milliseconds for a read operation on the cache.	long	No

Name	Description	Type	Writable
averageRemoveTime	Cluster wide total average number of milliseconds for a remove operation in the cache.	long	No
averageWriteTime	Cluster wide average number of milliseconds for a write operation in the cache.	long	No
cacheLoaderLoads	The total number of cacheloader load operations in the cluster.	long	No
cacheLoaderMisses	The total number of cacheloader load misses in the cluster.	long	No
evictions	Cluster wide total number of cache eviction operations.	long	No
hitRatio	Cluster wide total percentage hit/(hit+miss) ratio for this cache.	double	No
hits	Cluster wide total number of cache hits.	long	No
invalidations	The total number of invalidations in the cluster.	long	No
misses	Cluster wide total number of cache attribute misses.	long	No
numberOfEntries	Cluster wide total number of entries currently in the cache.	int	No
numberOfLocksAvailable	Total number of exclusive locks available in the cluster.	int	No

Name	Description	Type	Writable
numberOfLocksHeld	The total number of locks held in the cluster.	int	No
passivations	The total number of passivations in the cluster.	long	No
readWriteRatio	Cluster wide read/writes ratio for the cache.	double	No
removeHits	Cluster wide total number of cache removal hits.	double	No
removeMisses	Cluster wide total number of cache removals where keys were not found.	long	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes
storeWrites	The total number of cachestore store operations in the cluster.	long	No
stores	Cluster wide total number of cache attribute put operations.	long	No
timeSinceStart	Number of seconds since the first cache node started.	long	No

Table D.13. Operations

Name	Description	Signature
setStaleStatsTreshold	Sets the threshold for cluster wide stats refresh (in milliseconds).	void setStaleStatsTreshold(long staleStatsThreshold)
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()

D.8. DEADLOCKDETECTINGLOCKMANAGER

org.infinispan.util.concurrent.locks.DeadlockDetectingLockManager

This component provides information about the number of deadlocks that were detected.

Table D.14. Attributes

Name	Description	Type	Writable
detectedLocalDeadlocks	Number of local transactions that were rolled back due to deadlocks.	long	No
detectedRemoteDeadlocks	Number of remote transactions that were rolled back due to deadlocks.	long	No
overlapWithNotDeadlockAwareLockOwners	Number of situations when we try to determine a deadlock and the other lock owner is NOT a transaction. In this scenario we cannot run the deadlock detection mechanism.	long	No
totalNumberOfDetectedDeadlocks	Total number of local detected deadlocks.	long	No
concurrencyLevel	The concurrency level that the MVCC Lock Manager has been configured with.	int	No
numberOfLocksAvailable	The number of exclusive locks that are available.	int	No
numberOfLocksHeld	The number of exclusive locks that are held.	int	No

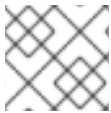
Table D.15. Operations

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

D.9. DISTRIBUTIONMANAGER

org.infinispan.distribution.DistributionManagerImpl

The DistributionManager component handles the distribution of content across a cluster.

**NOTE**

The DistrubutionManager component is only available in clustered mode.

Table D.16. Operations

Name	Description	Signature
isAffectedByRehash	Determines whether a given key is affected by an ongoing rehash.	boolean isAffectedByRehash(Object p0)
isLocatedLocally	Indicates whether a given key is local to this instance of the cache. Only works with String keys.	boolean isLocatedLocally(String p0)
locateKey	Locates an object in a cluster. Only works with String keys.	List locateKey(String p0)

D.10. INTERPRETER**org.infinispan.cli.interpreter.Interpreter**

The Interpreter component executes command line interface (**CLI** operations).

Table D.17. Attributes

Name	Description	Type	Writable
cacheNames	Retrieves a list of caches for the cache manager.	String[]	No

Table D.18. Operations

Name	Description	Signature
createSessionId	Creates a new interpreter session.	String createSessionId(String cacheName)
execute	Parses and executes IspnCliQL statements.	String execute(String p0, String p1)

D.11. INVALIDATION**org.infinispan.interceptors.InvalidatorInterceptor**

The Invalidation component invalidates entries on remote caches when entries are written locally.

Table D.19. Attributes

Name	Description	Type	Writable
invalidations	Number of invalidations.	long	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table D.20. Operations

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

D.12. LOCKMANAGER

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

The LockManager component handles MVCC locks for entries.

Table D.21. Attributes

Name	Description	Type	Writable
concurrencyLevel	The concurrency level that the MVCC Lock Manager has been configured with.	int	No
numberOfLocksAvailable	The number of exclusive locks that are available.	int	No
numberOfLocksHeld	The number of exclusive locks that are held.	int	No

D.13. LOCALTOPOLOGYMANAGER

`org.infinispan.topology.LocalTopologyManagerImpl`

The LocalTopologyManager component controls the cache membership and state transfer in Red Hat JBoss Data Grid.



NOTE

The LocalTopologyManager component is only available in clustered mode.

Table D.22. Attributes

Name	Description	Type	Writable
rebalancingEnabled	If false, newly started nodes will not join the existing cluster nor will the state be transferred to them. If any of the current cluster members are stopped when rebalancing is disabled, the nodes will leave the cluster but the state will not be rebalanced among the remaining nodes. This will result in fewer copies than specified by the owners attribute until rebalancing is enabled again.	boolean	Yes
clusterAvailability	If AVAILABLE the node is currently operating regularly. If DEGRADED then data can not be safely accessed due to either a network split, or successive nodes leaving.	String	No

D.14. MASSINDEXER

`org.infinispan.query.MassIndexer`

The MassIndexer component rebuilds the index using cached data.

Table D.23. Operations

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



NOTE

This operation is available only for caches with indexing enabled. For more information, see the Red Hat JBoss Data Grid *Developer Guide*

D.15. PASSIVATION

`org.infinispan.eviction.PassivationManager`

The Passivation component handles the passivation of entries to a CacheStore on eviction.

Table D.24. Attributes

Name	Description	Type	Writable
passivations	Number of passivation events.	String	No
statisticsEnabled	Enables or disables the gathering of statistics by this component	boolean	Yes

Table D.25. Operations

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

D.16. RECOVERYADMIN

`org.infinispan.transaction.xa.recovery.RecoveryAdminOperations`

The RecoveryAdmin component exposes tooling for handling transaction recovery.

Table D.26. Operations

Name	Description	Signature
forceCommit	Forces the commit of an in-doubt transaction.	String forceCommit(long p0)
forceCommit	Forces the commit of an in-doubt transaction	String forceCommit(int p0, byte[] p1, byte[] p2)
forceRollback	Forces the rollback of an in-doubt transaction.	String forceRollback(long p0)
forceRollback	Forces the rollback of an in-doubt transaction	String forceRollback(int p0, byte[] p1, byte[] p2)
forget	Removes recovery info for the given transaction.	String forget(long p0)
forget	Removes recovery info for the given transaction.	String forget(int p0, byte[] p1, byte[] p2)

Name	Description	Signature
showInDoubtTransactions	Shows all the prepared transactions for which the originating node crashed.	String showInDoubtTransactions()

D.17. ROLLINGUPGRADEMANAGER

`org.infinispan.upgrade.RollingUpgradeManager`

The `RollingUpgradeManager` component handles the control hooks in order to migrate data from one version of Red Hat JBoss Data Grid to another.

Table D.27. Operations

Name	Description	Signature
disconnectSource	Disconnects the target cluster from the source cluster according to the specified migrator.	void disconnectSource(String p0)
recordKnownGlobalKeyset	Dumps the global known keyset to a well-known key for retrieval by the upgrade process.	void recordKnownGlobalKeyset()
synchronizeData	Synchronizes data from the old cluster to this using the specified migrator.	long synchronizeData(String p0)

D.18. RPCMANAGER

`org.infinispan.remoting.rpc.RpcManagerImpl`

The `RpcManager` component manages all remote calls to remote cache instances in the cluster.



NOTE

The `RpcManager` component is only available in clustered mode.

Table D.28. Attributes

Name	Description	Type	Writable
averageReplicationTime	The average time spent in the transport layer, in milliseconds.	long	No

Name	Description	Type	Writable
committedViewAsString	Retrieves the committed view.	String	No
pendingViewAsString	Retrieves the pending view.	String	No
replicationCount	Number of successful replications.	long	No
replicationFailures	Number of failed replications.	long	No
successRatio	Successful replications as a ratio of total replications.	String	No
successRatioFloatingPoint	Successful replications as a ratio of total replications in numeric double format.	double	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

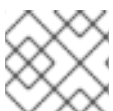
Table D.29. Operations

Name	Description	Signature
resetStatistics	Resets statistics gathered by this component.	void resetStatistics()
setStatisticsEnabled	Whether statistics should be enabled or disabled (true/false)	void setStatisticsEnabled(boolean enabled)

D.19. STATETRANSFERMANAGER

`org.infinispan.statetransfer.StateTransferManager`

The `StateTransferManager` component handles state transfer in Red Hat JBoss Data Grid.



NOTE

The `StateTransferManager` component is only available in clustered mode.

Table D.30. Attributes

Name	Description	Type	Writable
joinComplete	If true, the cluster topology is updated to include the node. If false, the node is not yet joined to the cluster.	boolean	No
stateTransferInProgress	Checks whether there is a pending inbound state transfer on this node.	boolean	No
rebalancingStatus	Checks if there is a state transfer in progress for the cache.	String	No

D.20. STATISTICS

`org.infinispan.interceptors.CacheMgmtInterceptor`

This component handles general statistics such as timings, hit/miss ratio, etc.

Table D.31. Attributes

Name	Description	Type	Writable
averageReadTime	Average number of milliseconds for a read operation on the cache.	long	No
averageWriteTime	Average number of milliseconds for a write operation in the cache.	long	No
elapsedTime	Number of seconds since cache started.	long	No
evictions	Number of cache eviction operations.	long	No
hitRatio	Percentage hit/(hit+miss) ratio for the cache.	double	No
hits	Number of cache attribute hits.	long	No
misses	Number of cache attribute misses.	long	No

Name	Description	Type	Writable
numberOfEntries	Number of entries currently in the cache.	int	No
readWriteRatio	Read/writes ratio for the cache.	double	No
removeHits	Number of cache removal hits.	long	No
removeMisses	Number of cache removals where keys were not found.	long	No
stores	Number of cache attribute PUT operations.	long	No
timeSinceReset	Number of seconds since the cache statistics were last reset.	long	No
averageRemoveTime	Average number of milliseconds for a remove operation in the cache	long	No

Table D.32. Operations

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

D.21. TRANSACTIONS

`org.infinispan.interceptors.TxInterceptor`

The Transactions component manages the cache's participation in JTA transactions.

Table D.33. Attributes

Name	Description	Type	Writable
commits	Number of transaction commits performed since last reset.	long	No

Name	Description	Type	Writable
prepares	Number of transaction prepares performed since last reset.	long	No
rollbacks	Number of transaction rollbacks performed since last reset.	long	No
statisticsEnabled	Enables or disables the gathering of statistics by this component.	boolean	Yes

Table D.34. Operations

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

D.22. TRANSPORT

`org.infinispan.server.core.transport.NettyTransport`

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

Table D.35. Attributes

Name	Description	Type	Writable
hostName	Returns the host to which the transport binds.	String	No
idleTimeout	Returns the idle timeout.	String	No
numberOfGlobalConnections	Returns a count of active connections in the cluster. This operation will make remote calls to aggregate results, so latency may have an impact on the speed of calculation for this attribute.	Integer	false
numberOfLocalConnections	Returns a count of active connections this server.	Integer	No

Name	Description	Type	Writable
numberWorkerThreads	Returns the number of worker threads.	String	No
port	Returns the port to which the transport binds.	String	receiveBufferSize
Returns the receive buffer size.	String	No	sendBufferSize
Returns the send buffer size.	String	No	totalBytesRead
Returns the total number of bytes read by the server from clients, including both protocol and user information.	String	No	totalBytesWritten
Returns the total number of bytes written by the server back to clients, including both protocol and user information.	String	No	tcpNoDelay

D.23. XSITEADMIN

`org.infinispan.xsite.XSiteAdminOperations`

The XSiteAdmin component exposes tooling for backing up data to remote sites.

Table D.36. Operations

Name	Description	Signature
bringSiteOnline	Brings the given site back online on all the cluster.	String bringSiteOnline(String p0)
amendTakeOffline	Amends the values for 'TakeOffline' functionality on all the nodes in the cluster.	String amendTakeOffline(String p0, int p1, long p2)
getTakeOfflineAfterFailures	Returns the value of the 'afterFailures' for the 'TakeOffline' functionality.	String getTakeOfflineAfterFailures(String p0)

Name	Description	Signature
getTakeOfflineMinTimeToWait	Returns the value of the 'minTimeToWait' for the 'TakeOffline' functionality.	String getTakeOfflineMinTimeToWait(String p0)
setTakeOfflineAfterFailures	Amends the values for 'afterFailures' for the 'TakeOffline' functionality on all the nodes in the cluster.	String setTakeOfflineAfterFailures(String p0, int p1)
setTakeOfflineMinTimeToWait	Amends the values for 'minTimeToWait' for the 'TakeOffline' functionality on all the nodes in the cluster.	String setTakeOfflineMinTimeToWait(String p0, long p1)
siteStatus	Check whether the given backup site is offline or not.	String siteStatus(String p0)
status	Returns the status(offline/online) of all the configured backup sites.	String status()
takeSiteOffline	Takes this site offline in all nodes in the cluster.	String takeSiteOffline(String p0)
pushState	Starts the cross-site state transfer to the site name specified.	String pushState(String p0)
cancelPushState	Cancel the cross-site state transfer to the site name specified.	String cancelPushState(String p0)
getSendingSiteName	Returns the site name that is pushing state to this site.	String getSendingSiteName()
cancelReceiveState	Restores the site to the normal state. It is used when the link between the sites is broken during the state transfer.	String cancelReceiveState(String p0)
getPushStateStatus	Returns the status of completed and running cross-site state transfer.	String getPushStateStatus()
clearPushStateStatus	Clears the status of completed cross-site state transfer.	String clearPushStateStatus()

APPENDIX E. CONFIGURATION RECOMMENDATIONS

E.1. TIMEOUT VALUES

Table E.1. Timeout Value Recommendations for JBoss Data Grid

Timeout Value	Parent Element	Default Value	Recommended Value
distributedSyncTimeout	transport	240,000 (4 minutes)	Same as default
lockAcquisitionTimeout	locking	10,000 (10 seconds)	Same as default
cacheStopTimeout	transaction	30,000 (30 seconds)	Same as default
completedTxTimeout	transaction	60,000 (60 seconds)	Same as default
replTimeout	sync	15,000 (15 seconds)	Same as default
timeout	stateTransfer	240,000 (4 minutes)	Same as default
timeout	backup	10,000 (10 seconds)	Same as default
flushLockTimeout	async	1 (1 millisecond)	Same as default. Note that this value applies to asynchronous cache stores, but not asynchronous caches.
shutdownTimeout	async	25,000 (25 seconds)	Same as default. Note that this value applies to asynchronous cache stores, but not asynchronous caches.
pushStateTimeout	singletonStore	10,000 (10 seconds)	Same as default.
backup	replicationTimeout	10,000 (10 seconds)	remoteCallTimeout

APPENDIX F. PERFORMANCE RECOMMENDATIONS

F.1. CONCURRENT STARTUP FOR LARGE CLUSTERS

When starting a large number of instances, each managing a large number of caches, in parallel this may take a while as rebalancing attempts to distribute the data evenly as each node joins the cluster. To limit the number of rebalancing attempts made during the initial startup of the cluster disable rebalancing temporarily by following the below steps:

1. Start the first node in the cluster.
2. Set JMX attribute `jboss.infinispan/CacheManager/"clustered"/LocalTopologyManager/rebalancingEnabled` to `false`, as seen in [LocalTopologyManager](#).
3. Start the remaining nodes in the cluster.
4. Re-enable the JMX attribute `jboss.infinispan/CacheManager/"clustered"/LocalTopologyManager/rebalancingEnabled` by setting this value back to `true`, as seen in [LocalTopologyManager](#).

APPENDIX G. REFERENCES

G.1. ABOUT CONSISTENCY

Consistency is the policy that states whether it is possible for a data record on one node to vary from the same data record on another node.

For example, due to network speeds, it is possible that a write operation performed on the master node has not yet been performed on another node in the store, a strong consistency guarantee will ensure that data which is not yet fully replicated is not returned to the application.

G.2. ABOUT CONSISTENCY GUARANTEE

Despite the locking of a single owner instead of all owners, Red Hat JBoss Data Grid's consistency guarantee remains intact. Consider the following situation:

1. If Key **K** is hashed to nodes **{A, B}** and transaction **TX1** acquires a lock for **K** on, for example, node **A** and
2. If another cache access occurs on node **B**, or any other node, and **TX2** attempts to lock **K**, this access attempt fails with a timeout because the transaction **TX1** already holds a lock on **K**.

This lock acquisition attempt always fails because the lock for key **K** is always deterministically acquired on the same node of the cluster, irrespective of the transaction's origin.

G.3. ABOUT JBOSS CACHE

Red Hat JBoss Cache is a tree-structured, clustered, transactional cache that can also be used in a standalone, non-clustered environment. It caches frequently accessed data in-memory to prevent data retrieval or calculation bottlenecks that occur while enterprise features such as Java Transactional API (JTA) compatibility, eviction and persistence are provided.

JBoss Cache is the predecessor to Infinispan and Red Hat JBoss Data Grid.

G.4. ABOUT RELAY2

The *RELAY* protocol bridges two remote clusters by creating a connection between one node in each site. This allows multicast messages sent out in one site to be relayed to the other and vice versa.

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

The *RELAY2* protocol works similarly to *RELAY* but with slight differences. Unlike *RELAY*, the *RELAY2* protocol:

- connects more than two sites.
- connects sites that operate autonomously and are unaware of each other.
- offers both unicasts and multicast routing between sites.

G.5. ABOUT RETURN VALUES

Values returned by cache operations are referred to as return values. In Red Hat JBoss Data Grid, these return values remain reliable irrespective of which cache mode is employed and whether synchronous or asynchronous communication is used.

G.6. ABOUT RUNNABLE INTERFACES

A Runnable Interface (also known as a Runnable) declares a single `run()` method, which executes the active part of the class' code. The Runnable object can be executed in its own thread after it is passed to a thread constructor.

G.7. ABOUT TWO PHASE COMMIT (2PC)

A Two Phase Commit protocol (2PC) is a consensus protocol used to atomically commit or roll back distributed transactions. It is successful when faced with cases of temporary system failures, including network node and communication failures, and is therefore widely utilized.

G.8. ABOUT KEY-VALUE PAIRS

A key-value pair (KVP) is a set of data consisting of a key and a value.

- A key is unique to a particular data entry. It consists of entry data attributes from the related entry.
- A value is the data assigned to and identified by the key.

G.9. REQUESTING A FULL BYTE ARRAY

How can I request the Red Hat JBoss Data Grid return a full byte array instead of partial byte array contents?

As a default, JBoss Data Grid only partially prints byte arrays to logs to avoid unnecessarily printing large byte arrays. This occurs when either:

- JBoss Data Grid caches are configured for lazy deserialization. Lazy deserialization is not available in JBoss Data Grid's Remote Client-Server mode.
- A *Memcached* or *Hot Rod* server is run.

In such cases, only the first ten positions of the byte array display in the logs. To display the complete contents of the byte array in the logs, pass the `-Dinfinispan.arrays.debug=true` system property at start up.

Partial Byte Array Log

```
2010-04-14 15:46:09,342 TRACE [ReadCommittedEntry] (HotRodWorker-1-1)
Updating entry
(key=CacheKey{data=ByteArray{size=19, hashCode=1b3278a,
array=[107, 45, 116, 101, 115, 116, 82, 101, 112, 108, ..]}}
removed=false valid=true changed=true created=true
value=CacheValue{data=ByteArray{size=19,
array=[118, 45, 116, 101, 115, 116, 82, 101, 112, 108, ..]},
version=281483566645249}]
```

And here's a log message where the full byte array is shown:

```
2010-04-14 15:45:00,723 TRACE [ReadCommittedEntry] (Incoming-2,Infinispan-
```

```
Cluster,eq-6834) Updating entry
(key=CacheKey{data=ByteArray{size=19, hashCode=6cc2a4,
array=[107, 45, 116, 101, 115, 116, 82, 101, 112, 108, 105, 99, 97, 116,
101, 100, 80, 117, 116]}}
removed=false valid=true changed=true created=true
value=CacheValue{data=ByteArray{size=19,
array=[118, 45, 116, 101, 115, 116, 82, 101, 112, 108, 105, 99, 97, 116,
101, 100, 80, 117, 116]},
version=281483566645249}]
```