Red Hat Data Grid 8.0

Upgrading Data Grid

Data Grid Documentation

Last Updated: 2020-06-02
Abstract

Find out about changes in Data Grid 8.0 that affect migration from previous versions and then complete the steps to upgrade deployments and migrate your data.
# Table of Contents

## CHAPTER 1. RED HAT DATA GRID
- 1.1. DATA GRID DOCUMENTATION .................................................. 3
- 1.2. DATA GRID DOWNLOADS ....................................................... 3

## CHAPTER 2. MIGRATING TO DATA GRID 8.0 ...................................... 4
- 2.1. DATA GRID 8.0 SERVER ......................................................... 4
- 2.2. DATA GRID CACHES .......................................................... 6
- 2.3. CREATING CACHES .......................................................... 7
- 2.4. CACHE HEALTH STATUS ..................................................... 8
- 2.5. MARSHALLING CAPABILITIES .............................................. 9
- 2.6. DATA GRID CONFIGURATION ............................................. 9
- 2.7. PERSISTENCE ................................................................. 11
- 2.8. REST API ................................................................. 12
- 2.9. HOT ROD CLIENT AUTHENTICATION .................................... 12
- 2.10. JAVA DISTRIBUTIONS AVAILABLE IN MAVEN ..................... 12
- 2.11. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (EAP) MODULES 12
- 2.12. DEPRECATED FEATURES AND FUNCTIONALITY ..................... 12
  - 2.12.1. Deprecations .......................................................... 13
  - 2.12.2. Removed Features and Functionality ............................. 13

## CHAPTER 3. PERFORMING ROLLING UPGRADES FOR DATA GRID SERVERS .. 15
- 3.1. SETTING UP TARGET CLUSTERS .......................................... 15
  - 3.1.1. Remote Cache Stores for Rolling Upgrades ....................... 15
- 3.2. SYNCHRONIZING DATA TO TARGET CLUSTERS ...................... 16

## CHAPTER 4. PATCHING DATA GRID SERVER INSTALLATIONS .................... 18
- 4.1. DATA GRID SERVER PATCHES ............................................. 18
- 4.2. DOWNLOADING SERVER PATCHES ....................................... 18
- 4.3. CREATING SERVER PATCHES ............................................. 19
- 4.4. INSTALLING SERVER PATCHES .......................................... 20
- 4.5. ROLLING BACK SERVER PATCHES ...................................... 21

## CHAPTER 5. MIGRATING DATA BETWEEN CACHE STORES .......................... 23
- 5.1. CACHE STORE MIGRATOR ................................................... 23
- 5.2. GETTING THE STORE MIGRATOR ....................................... 23
- 5.3. CONFIGURING THE STORE MIGRATOR .................................. 24
  - 5.3.1. Store Migrator Properties ........................................... 25
- 5.4. MIGRATING CACHE STORES ............................................. 29
CHAPTER 1. RED HAT DATA GRID

Data Grid is a high-performance, distributed in-memory data store.

**Schemaless data structure**
Flexibility to store different objects as key-value pairs.

**Grid-based data storage**
Designed to distribute and replicate data across clusters.

**Elastic scaling**
Dynamically adjust the number of nodes to meet demand without service disruption.

**Data interoperability**
Store, retrieve, and query data in the grid from different endpoints.

1.1. DATA GRID DOCUMENTATION

Documentation for Data Grid is available on the Red Hat customer portal.

- Data Grid 8.0 Documentation
- Data Grid 8.0 Component Details
- Supported Configurations for Data Grid 8.0

1.2. DATA GRID DOWNLOADS

Access the Data Grid Software Downloads on the Red Hat customer portal.

**NOTE**
You must have a Red Hat account to access and download Data Grid software.
CHAPTER 2. MIGRATING TO DATA GRID 8.0

Review changes in Data Grid 8.0 that affect migration from previous releases.

2.1. DATA GRID 8.0 SERVER

As of 8.0, Data Grid server is no longer based on Red Hat JBoss Enterprise Application Platform (EAP) and is re-designed to be lightweight and more secure with much faster start times.

Data Grid servers use $RHDG_HOME/server/conf/infinispan.xml for configuration.

Data store configuration

You configure how Data Grid stores your data through cache definitions. By default, Data Grid servers include a Cache Manager configuration that lets you create, configure, and manage your cache definitions.

```
<cache-container name="default">
  <transport cluster="${infinispan.cluster.name}" stack="${infinispan.cluster.stack:tcp}" node-name="${infinispan.node.name:}"/>
</cache-container>
```

1. Creates a Cache Manager named "default".
2. Exports Cache Manager statistics through the metrics endpoint.
3. Adds a JGroups cluster transport that allows Data Grid servers to automatically discover each other and form clusters.
4. Uses the default TCP stack for cluster traffic.

In the preceding configuration, there are no cache definitions. When you start 8.0 server, it instantiates the default Cache Manager so you can create cache definitions at runtime through the CLI, REST API, or from remote Hot Rod clients.

NOTE

Data Grid server no longer provides a domain mode as in previous versions that were based on EAP. However, Data Grid server provides a default configuration with clustering capabilities so your data is replicated across all nodes.

Server configuration

Data Grid 8.0 extends infinispan.xml with a server element that defines configuration specific to Data Grid servers.

```
<server>
  <interfaces>
    <interface name="public">
      <inet-address value="${infinispan.bind.address:127.0.0.1}"/>
    </interface>
  </interfaces>
</server>
```
CHAPTER 2. MIGRATING TO DATA GRID 8.0

Creates a default public interface that uses the 127.0.0.1 loopback address.

Creates a default socket binding that binds the public interface to port 11222.

Creates a socket binding for the Memcached connector. Note that the Memcached endpoint is now deprecated.

Defines a default security realm that uses property files to define credentials and RBAC settings.

Exposes the Hot Rod and REST endpoints at 127.0.0.1:11222.

**IMPORTANT**

The REST endpoint handles administrative operations that the Data Grid command line interface (CLI) and console use. For this reason, you should never disable the REST endpoint.

<table>
<thead>
<tr>
<th>Table 2.1. Cheat Sheet</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>7.x</strong></td>
</tr>
<tr>
<td>./standalone.sh -c clustered.xml</td>
</tr>
<tr>
<td>./standalone.sh</td>
</tr>
</tbody>
</table>
### 2.2. DATA GRID CACHES

Except for the Cache service on OpenShift, Data Grid provides empty cache containers by default. When you start Data Grid 8.0 it instantiates a Cache Manager so you can create caches at runtime.

#### Use custom UDP/TCP addresses as follows:
- `Djgroups.udp.address=172.18.1.13`
- `Djgroups.tcp.address=172.18.1.1`

#### Enable JMX as follows:

```
<cache-container name="default"
    statistics="true"> 1
    <jmx enabled="true" /> 2
...</n```

1. Enables statistics for the Cache Manager. This is the default.
2. Exports JMX MBeans.

---

**Reference**

- [Getting Started with Data Grid Servers](#)
- [Network Interfaces: Server Guide](#)
- [Socket Bindings: Server Guide](#)
- [Endpoints: Server Guide](#)
- [Defining Property Realms: Server Guide](#)
- [Security: Server Guide](#)
- [Cluster Transport: Configuration Guide](#)
- [Creating Caches with the CLI](#)
- [Creating Caches through the REST API](#)
In Data Grid 8.0, cache definitions that you create through the `CacheContainerAdmin` API are permanent to ensure that they survive cluster restarts.

```java
.administration()
 .withFlags(AdminFlag.VOLATILE) 1
 .getOrCreateCache("myTemporaryCache", "org.infinispan.DIST_SYNC"); 2
```

1 includes the VOLATILE flag that changes the default behavior and creates temporary caches.

2 returns a cache named "myTemporaryCache" or creates one using the DIST_SYNC configuration template.

**NOTE**

`AdminFlag.PERMANENT` is enabled by default to ensure that cache definitions survive restarts. You must separately add persistent storage to Data Grid for data to survive restarts, for example:

```java
ConfigurationBuilder b = new ConfigurationBuilder();
b.persistence()
 .addSingleFileStore()
 .location("/tmp/myDataStore")
 .maxEntries(5000);
```

### Cache Configuration Templates

Get the list of cache configuration templates as follows:

- Use Tab auto-completion with the CLI:

  ```
  [//containers/default]> create cache --template=
  ```

- Use the REST API:

  ```
  GET 127.0.0.1:11222/rest/v2/cache-managers/default/cache-configs/templates
  ```

### 2.3. CREATING CACHES

Add cache definitions to Data Grid to configure how it stores your data.

**Library Mode**

The following example initializes the Cache Manager and creates a cache definition named "myDistributedCache" that uses the distributed, synchronous cache mode:

```java
GlobalConfigurationBuilder global = GlobalConfigurationBuilder.defaultClusteredBuilder();
DefaultCacheManager cacheManager = new DefaultCacheManager(global.build());
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.clustering().cacheMode(CacheMode.DIST_SYNC);
```

```java
cacheManager.defineConfiguration("myDistributedCache", builder.build());
```
You can also use the `getOrCreate()` method to create your cache definition or return it if it already exists, for example:

```java
Data Grid Server
Remotely create caches at runtime as follows:

- Use the CLI.
  To create a cache named "myCache" with the `DIST_SYNC` cache template, run the following:

  ```bash
  //containers/default]$ create cache --template=org.infinispan.DIST_SYNC
  name=myDistributedCache
  ```

- Use the REST API.
  To create a cache named "myCache", use the following `POST` invocation and include the cache definition in the request payload in XML or JSON format:

  ```
  POST /rest/v2/caches/myCache
  ```

- Use Hot Rod clients.

  ```java
  import org.infinispan.client.hotrod.RemoteCacheManager;
  import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
  import org.infinispan.client.hotrod.impl.ConfigurationProperties;
  import org.infinispan.commons.api.CacheContainerAdmin.AdminFlag;
  import org.infinispan.commons.configuration.XMLStringConfiguration;

  // Create a configuration for a locally running server.
  ConfigurationBuilder builder = new ConfigurationBuilder();
  builder.addServer().host("127.0.0.1").port(11222);
  manager = new RemoteCacheManager(builder.build());
  }

  private void createTemporaryCacheWithTemplate() {
    manager.administration()
      //Override the default and create a volatile cache that
      //does not survive cluster restarts.
      .withFlags(AdminFlag.VOLATILE)
      //Create a cache named myTemporaryCache that uses the
      //distributed, synchronous cache template
      //or return it if it already exists.
      .getOrCreateCache("myTemporaryCache", "org.infinispan.DIST_SYNC");
  }

  For more examples of creating caches with a Hot Rod Java client, see the Data Grid tutorials.

2.4. CACHE HEALTH STATUS

Data Grid now returns one of the following for cache health:
**2.5. MARSHALLING CAPABILITIES**

As of this release, the default marshaller for Data Grid is ProtoStream, which marshals data as Protocol Buffers, a language-neutral, backwards compatible format.

To use ProtoStream, Data Grid requires serialization contexts that contain:

- `.proto` schemas that provide a structured representation of your Java objects as Protobuf message types.
- Marshaller implementations to encode your Java objects to Protobuf format.

Data Grid provides direct integration with ProtoStream libraries and can generate everything you need to initialize serialization contexts.

**IMPORTANT**

Cache stores in previous versions of Data Grid store data in a binary format that is not compatible with ProtoStream marshlers. You must use the `StoreMigrator` utility to migrate your data.

- Data Grid Library Mode does not include JBoss Marshalling by default. You must add the `infinispan-jboss-marshalling` dependency to your classpath.
- Data Grid servers do support JBoss Marshalling but clients must declare the marshaller to use, as in the following Hot Rod client configuration:
  ```java
  .marshaller("org.infinispan.jboss.marshalling.core.JBossUserMarshaller");
  ```
- Spring integration does not yet support the default ProtoStream marshaller. For this reason you should use the Java Serialization Marshaller.
- To use the Java Serialization Marshaller, you must add classes to the deserialization whitelist.

**Reference**

- Data Grid Marshalling
- ProtoStream Marshalling
- Creating Context Initializers
- JBoss Marshalling
- Data Grid Spring Boot Starter
- Java Serialization Marshaller
- Adding Java Classes to Deserialization White Lists

**2.6. DATA GRID CONFIGURATION**
New and Modified Elements and Attributes

- **stack** adds support for inline JGroups stack definitions.
- **stack.combine** and **stack.position** attributes let you override and modify JGroups stack definitions.
- **metrics** lets you configure how Data Grid exports metrics that are compatible with the Eclipse MicroProfile Metrics API.
- **context-initializer** lets you specify a **SerializationContextInitializer** implementation that initializes a Protostream-based marshaller for user types.
- **key-transformers** lets you register transformers that convert custom keys to String for indexing with Lucene.
- **statistics** now defaults to "false".

Deprecated Elements and Attributes

The following elements and attributes are now deprecated:

- **address-count** attribute for the **off-heap** element.
- **protocol** attribute for the **transaction** element.
- **duplicate-domains** attribute for the **jmx** element.
- **advanced-externalizer**
- **custom-interceptors**
- **state-transfer-executor**
- **transaction-protocol**

**NOTE**

Refer to the Configuration Schema for possible replacements or alternatives.

Removed Elements and Attributes

The following elements and attributes were deprecated in a previous release and are now removed:

- **deadlock-detection-spin**
- **compatibility**
- **write-skew**
- **versioning**
- **data-container**
- **eviction**
- **eviction-thread-policy**
2.7. PERSISTENCE

In comparison with some previous versions of Data Grid, such as 7.1, there are changes to cache store configurations. Cache store definitions must:

- Be contained within `persistence` elements.
- Include an `xmlns` namespace.

As of this release, cache store configuration:

- Defaults to `segmented="true"` if the cache store implementation supports segmentation.
- Removes the `singleton` attribute for the `store` element. Use `shared=true` instead.

JDBC String-Based cache stores use connections factories based on Agroal to connect to databases. It is no longer possible to use `c3p0.properties` and `hikari.properties` files.

Likewise, JDBC String-Based cache store configuration that use segmentation, which is now the default, must include the `segmentColumnName` and `segmentColumnType` parameters.

**MySQL Example**

```java
builder.table()
    .tableNamePrefix("ISPN")
    .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
    .dataColumnName("DATA_COLUMN").dataColumnType("VARBINARY(1000)")
    .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT")
    .segmentColumnName("SEGMENT_COLUMN").segmentColumnType("INTEGER")
```

**PostgreSQL Example**

```java
builder.table()
    .tableNamePrefix("ISPN")
    .idColumnName("ID_COLUMN").idColumnType("VARCHAR(255)")
    .dataColumnName("DATA_COLUMN").dataColumnType("BYTEA")
    .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT")
    .segmentColumnName("SEGMENT_COLUMN").segmentColumnType("INTEGER");
```

Reference

- Data Grid Configuration Schema
- Setting Up Persistent Storage
- Segmented Cache Stores
- JDBC String-Based Cache Stores
2.8. REST API

Previous versions of the Data Grid REST API were v1, which is now replaced by REST API v2.

The default context path is now `127.0.0.1:11222/rest/v2/`. You must update any clients or scripts to use REST API v2.

Reference
  - Data Grid REST API

2.9. HOT ROD CLIENT AUTHENTICATION

Hot Rod clients now use **SCRAM-SHA-512** as the default authentication mechanism instead of **DIGEST-MD5**.

**NOTE**

If you use property security realms, you must use the **PLAIN** authentication mechanism.

Reference
  - Configuring Authentication Mechanisms for Hot Rod Java Clients

2.10. JAVA DISTRIBUTIONS AVAILABLE IN MAVEN

Data Grid no longer provides Java artifacts outside the Maven repository, with the exception of the Data Grid server distribution. For information on adding required dependencies for the Data Grid Library, Hot Rod Java client, and utilities such as **StoreMigrator**, see the relevant documentation.

Reference
  - Data Grid Library Mode
  - Hot Rod Java Client Guide
  - Getting the Store Migrator

2.11. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (EAP) MODULES

Data Grid no longer provides modules for applications running on EAP. Instead, EAP will provide direct integration with Data Grid in a future release.

However, until EAP provides functionality for handling the **infinispan** subsystem, you must package Data Grid 8.0 artifacts in your EAP deployments.

2.12. DEPRECATED FEATURES AND FUNCTIONALITY

Support for deprecated functionality is not available beyond the release in which it is deprecated.
2.12.1. Deprecations

Data Grid 8.0 deprecates the following features and functionality:

**Memcached Endpoint Connector**

As of this release, Data Grid no longer supports the Memcached endpoint. The Memcached connector is deprecated and planned for removal in a future release.

**NOTE**

If you have a use case or requirement for the Memcached connector, contact your Red Hat support team to discuss requirements for a future Data Grid implementation of the Memcached connector.

**JBoss Marshalling**

JBoss Marshalling is a Serialization-based marshalling library and was the default marshaller in previous Data Grid versions. You should not use serialization-based marshalling with Data Grid but instead use Protostream, which is a high-performance binary wire format that ensures backwards compatibility.

**Externalizers**

The following interfaces and annotations are now deprecated:

- `org.infinispan.commons.marshall.AdvancedExternalizer`
- `org.infinispan.commons.marshall.Externalizer`
- `@SerializeWith`

**NOTE**

Data Grid ignores `AdvancedExternalizer` implementations when persisting data unless you use JBoss Marshalling.

**Total Order Transaction Protocol**

The `org.infinispan.transaction.TransactionProtocol#TOTAL_ORDER` protocol is deprecated. Use the default 2PC protocol instead.

**Lucene Directory**

The functionality to use Data Grid as a shared, in-memory index for Hibernate Search queries is now deprecated.

**Custom Interceptors**

The functionality to create custom interceptors with the `AdvancedCache` interface is now deprecated.

2.12.2. Removed Features and Functionality
Data Grid 8.0 no longer includes the following features and functionality that was either deprecated in a previous release or replaced with new components:

- Uberjars (replaced with Maven dependencies and individual JAR files)
- EAP Modules (replaced by the EAP Infinispan subsystem)
- Cassandra Cache Store
- Apache Spark Connector
- Apache Hadoop Connector
- Apache Camel component: `jboss-datagrid.camel-library` is replaced by the `camel-infinispan` component in Red Hat Fuse 7.3 and later.
- REST Cache Store
- REST API v1 (replaced by REST API v2)
- Compatibility Mode
- Distributed Execution
- CLI Cache Loader
- LevelDB Cache Store
- `infinispan-cloud` (replaced by default configuration in `infinispan-core`)
- `org.infinispan.atomic` package
- `getBulk()` methods in the `RemoteCache` API for Hot Rod clients
- JDBC PooledConnectionFactory via C3P0 and HikariCP connection pools
- OSGI support
- `infinispan.server.hotrod.workerThreads` system property
- JON Plugin
CHAPTER 3. PERFORMING ROLLING UPGRADES FOR DATA GRID SERVERS

Perform rolling upgrades of your Data Grid clusters to change between versions without downtime or data loss. Rolling upgrades migrate both your Data Grid servers and your data to the target version over Hot Rod.

3.1. SETTING UP TARGET CLUSTERS

Create a cluster that runs the target Data Grid version and uses a remote cache store to load data from the source cluster.

Prerequisites

- Install a Data Grid cluster with the target upgrade version.

**IMPORTANT**

Ensure the network properties for the target cluster do not overlap with those for the source cluster. You should specify unique names for the target and source clusters in the JGroups transport configuration. Depending on your environment you can also use different network interfaces and specify port offsets to keep the target and source clusters separate.

Procedure

1. Add a RemoteCacheStore on the target cluster for each cache you want to migrate from the source cluster.
   
   Remote cache stores use the Hot Rod protocol to retrieve data from remote Data Grid clusters. When you add the remote cache store to the target cluster, it can lazily load data from the source cluster to handle client requests.

2. Switch clients over to the target cluster so it starts handling all requests.
   
   a. Update client configuration with the location of the target cluster.
   
   b. Restart clients.

3.1.1. Remote Cache Stores for Rolling Upgrades

You must use specific remote cache store configuration to perform rolling upgrades, as follows:

```
<persistence passivation="false">
  <remote-store xmlns="urn:infinispan:config:store:remote:10.1"
    cache="myDistCache"
    protocol-version="2.5"
    hotrod-wrapping="true"
    raw-values="true">
    <remote-server host="127.0.0.1" port="11222"/>
  </remote-store>
</persistence>
```
Disables passivation. Remote cache stores for rolling upgrades must disable passivation.

Matches the name of a cache in the source cluster. Target clusters load data from this cache using the remote cache store.

Matches the Hot Rod protocol version of the source cluster. 2.5 is the minimum version and is suitable for any upgrade paths. You do not need to set another Hot Rod version.

Ensures that entries are wrapped in a suitable format for the Hot Rod protocol.

Stores data in the remote cache store in raw format. This ensures that clients can use data directly from the remote cache store.

Points to the location of the source cluster.

Reference

- Remote cache store configuration schema
- RemoteStore
- RemoteStoreConfigurationBuilder

3.2. SYNCHRONIZING DATA TO TARGET CLUSTERS

When your target cluster is running and handling client requests using a remote cache store to load data on demand, you can synchronize data from the source cluster to the target cluster.

This operation reads data from the source cluster and writes it to the target cluster. Data migrates to all nodes in the target cluster in parallel, with each node receiving a subset of the data. You must perform the synchronization for each cache in your Data Grid configuration.

Procedure

1. Start the synchronization operation for each cache in your Data Grid configuration that you want to migrate to the target cluster.
   Use the Data Grid REST API and invoke GET requests with the ?action=sync-data parameter.
   For example, to synchronize data in a cache named "myCache" from a source cluster to a target cluster, do the following:

   ```
   GET /v2/caches/myCache?action=sync-data
   ```

   When the operation completes, Data Grid responds with the total number of entries copied to the target cluster.

   Alternatively, you can use JMX by invoking synchronizeData(migratorName=hotrod) on the RollingUpgradeManager MBean.

2. Disconnect each node in the target cluster from the source cluster.
   For example, to disconnect the "myCache" cache from the source cluster, invoke the following GET request:

   ```
   GET /v2/caches/myCache?action=disconnect-source
   ```
To use JMX, invoke `disconnectSource(migratorName=hotrod)` on the `RollingUpgradeManager` MBean.

**Next steps**

After you synchronize all data from the source cluster, the rolling upgrade process is complete. You can now decommission the source cluster.
CHAPTER 4. PATCHING DATA GRID SERVER INSTALLATIONS

Install and manage patches for Data Grid server installations.

You can apply patches to multiple Data Grid servers with different versions to upgrade to a desired target version. However, patches do not take effect if Data Grid servers are running. For this reason you install patches while servers are offline. If you want to upgrade Data Grid clusters without downtime, create a new cluster with the target version and perform a rolling upgrade to that version instead of patching.

4.1. DATA GRID SERVER PATCHES

Data Grid server patches are .zip archives that contain artifacts that you can apply to your $RHDG_HOME directory to fix issues and add new features.

Patches also provide a set of rules for Data Grid to modify your server installation. When you apply patches, Data Grid overwrites some files and removes others, depending on if they are required for the target version.

However, Data Grid does not make any changes to configuration files that you have created or modified when applying a patch. Server patches do not modify or replace any custom configuration or data.

4.2. DOWNLOADING SERVER PATCHES

Download patches that you can apply to Data Grid servers.

Procedure

1. Access the Red Hat customer portal.
2. Download the appropriate Data Grid server patch from the software downloads section.
3. Open a terminal window and navigate to $RHDG_HOME.
4. Start the CLI.
   
   ```
   $ bin/cli.sh
   [disconnected]>
   ```
5. Describe the patch file you downloaded.

   ```
   [disconnected]>$ patch describe /path/to/redhat-datagrid-$version-server-patch.zip
   Red Hat Data Grid patch target=$target_version source=$source_version
   created=$timestamp
   ```

   - $target_version is the Data Grid version that applies when you install the patch on a server.
   - $source_version is one or more Data Grid server versions where you can install the patch.

Verification

Use the checksum to verify the integrity of your download.

1. Run the md5sum or sha256sum command with the downloaded patch as the argument, for example:
2. Compare with the **MD5** or **SHA-256** checksum value on the Data Grid **Software Details** page.

### 4.3. CREATING SERVER PATCHES

You can create patches for Data Grid servers from an existing server installation.

You can create patches for Data Grid servers starting from 8.0.1. You can patch 8.0 GA servers with 8.0.1. However you cannot patch 7.3.x or earlier servers with 8.0.1 or later.

You can also create patches that either upgrade or downgrade the Data Grid server version. For example, you can create a patch from version 8.0.1 and use it to upgrade version 8.0 GA or downgrade a later version.

**IMPORTANT**

Red Hat supports patched server deployments only with patches that you download from the Red Hat customer portal. Red Hat does not support server patches that you create yourself.

**Procedure**

1. Navigate to `$RHDG_HOME` for a Data Grid server installation that has the target version for the patch you want to create.

2. Start the CLI.

   ```bash
   $ bin/cli.sh
   [disconnected]>
   ```

3. Use the `patch create` command to generate a patch archive and include the `-q` option with a meaningful qualifier to describe the patch.

   ```bash
   [disconnected]>$ patch create -q "this is my test patch" path/to/mypatch.zip \    path/to/target/server/home path/to/source/server/home
   ```

   The preceding command generates a `.zip` archive in the specified directory. Paths are relative to `$RHDG_HOME` for the target server.

**TIP**

Create single patches for multiple different Data Grid versions, for example:

```bash
[disconnected]>$ patch create -q "this is my test patch" path/to/mypatch.zip \    path/to/target/server/home \    path/to/source/server1/home path/to/source/server2/home
```

Where `server1` and `server2` are different Data Grid versions where you can install "mypatch.zip".

4. Describe the generated patch archive.

   ```bash
   [disconnected]>$ patch describe path/to/mypatch.zip
   ```
**4.4. INSTALLING SERVER PATCHES**

Apply patches to Data Grid servers to upgrade or downgrade an existing version.

**Prerequisites**

- Download a server patch for the target version.

**Procedure**

1. Navigate to `$RHDG_HOME` for the Data Grid server you want to patch.

2. Stop the server if it is running.

   **NOTE**
   
   If you patch a server while it is running, the version changes take effect after restart. If you do not want to stop the server, create a new cluster with the target version and perform a rolling upgrade to that version instead of patching.

3. Start the CLI.

   ```
   $ bin/cli.sh
   [disconnected]>
   ```

4. Install the patch.

   ```
   [disconnected]>$ patch install path/to/patch.zip
   ```

5. Start the server to verify the patch is installed.
$ bin/server.sh
...
ISPN080001: Red Hat Data Grid Server $version

If the patch is installed successfully $version matches $target_version.

TIP

Use the --server option to install patches in a different $RHDG_HOME directory, for example:

[disconnected]> patch install path/to/patch.zip --server=path/to/server/home

4.5. ROLLING BACK SERVER PATCHES

Remove patches from Data Grid servers by rolling them back and restoring the previous Data Grid version.

IMPORTANT

If a server has multiple patches installed, you can roll back the last installed patch only.

Rolling back patches does not revert configuration changes you make to Data Grid server. Before you roll back patches, you should ensure that your configuration is compatible with the version to which you are rolling back.

Procedure

1. Navigate to $RHDG_HOME for the Data Grid server installation you want to roll back.

2. Stop the server if it is running.

3. Start the CLI.

   $ bin/cli.sh
   [disconnected]>

4. List the installed patches.

   [disconnected]> patch ls

   Red Hat Data Grid patch target=$target_version source=$source_version
   created=$timestamp installed=$timestamp

   • $target_version is the Data Grid server version after the patch was applied.
   • $source_version is the version for Data Grid server before the patch was applied. Rolling back the patch restores the server to this version.

5. Roll back the last installed patch.

   [disconnected]> patch rollback

6. Quit the CLI.
7. Start the server to verify the patch is rolled back to the previous version.

```bash
$ bin/server.sh
...
ISPN080001: Data Grid Server $version
```

If the patch is rolled back successfully, $version matches $source_version.

**TIP**

Use the `--server` option to rollback patches in a different $RHDG_HOME directory, for example:

```
[disconnected]> patch rollback --server=path/to/server/home
```
CHAPTER 5. MIGRATING DATA BETWEEN CACHE STORES

Data Grid provides a Java utility for migrating persisted data between cache stores.

In the case of upgrading Data Grid, functional differences between major versions do not allow backwards compatibility between cache stores. You can use StoreMigrator to convert your data so that it is compatible with the target version.

For example, upgrading to Data Grid 8.0 changes the default marshaller to Protostream. In previous Data Grid versions, cache stores use a binary format that is not compatible with the changes to marshalling. This means that Data Grid 8.0 cannot read from cache stores with previous Data Grid versions.

In other cases Data Grid versions deprecate or remove cache store implementations, such as JDBC Mixed and Binary stores. You can use StoreMigrator in these cases to convert to different cache store implementations.

5.1. CACHE STORE MIGRATOR

Data Grid provides the StoreMigrator.java utility that recreates data for the latest Data Grid cache store implementations.

StoreMigrator takes a cache store from a previous version of Data Grid as source and uses a cache store implementation as target.

When you run StoreMigrator, it creates the target cache with the cache store type that you define using the EmbeddedCacheManager interface. StoreMigrator then loads entries from the source store into memory and then puts them into the target cache.

StoreMigrator also lets you migrate data from one type of cache store to another. For example, you can migrate from a JDBC String-Based cache store to a Single File cache store.

IMPORTANT

StoreMigrator cannot migrate data from segmented cache stores to:

- Non-segmented cache store.
- Segmented cache stores that have a different number of segments.

5.2. GETTING THE STORE MIGRATOR

StoreMigrator is available as part of the Data Grid tools library, infinispan-tools, and is included in the Maven repository.

Procedure

- Configure your pom.xml for StoreMigrator as follows:

```xml
<project version="1.0" encoding="UTF-8">  
  <project xmlns="http://maven.apache.org/POM/4.0.0"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
    http://maven.apache.org/xsd/maven-4.0.0.xsd">  
```


5.3. CONFIGURING THE STORE MIGRATOR

Set properties for source and target cache stores in a `migrator.properties` file.

**Procedure**

1. Create a `migrator.properties` file.

2. Configure the source cache store in `migrator.properties`.
   a. Prepend all configuration properties with `source`. as in the following example:

   ```xml
   source.type=SOFT_INDEX_FILE_STORE
   source.cache_name=myCache
   source.location=/path/to/source/sifs
   ```

3. Configure the target cache store in `migrator.properties`. 

Red Hat Data Grid 8.0 Upgrading Data Grid
a. Prepend all configuration properties with `target`, as in the following example:

```plaintext
target.type=SINGLE_FILE_STORE
target.cache_name=myCache
target.location=/path/to/target/sfs.dat
```

### 5.3.1. Store Migrator Properties

Configure source and target cache stores in a `StoreMigrator` properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>type</strong></td>
<td>Specifies the type of cache store type for a source or target.</td>
<td>Required</td>
</tr>
<tr>
<td>.type=JDBC_STRING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type=JDBC_BINARY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type=JDBC_MIXED</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type=LEVELDB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type=ROCKSDB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type=SINGLE_FILE_STORE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type=SOFT_INDEX_FILE_STORE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.type=JDBC_MIXED</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 5.2. Common Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example Value</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cache_name</strong></td>
<td>Names the cache that the store backs.</td>
<td><code>.cache_name=myCache</code></td>
<td>Required</td>
</tr>
</tbody>
</table>
### segment_count
- **Description**: Specifies the number of segments for target cache stores that can use segmentation.
- **Example Value**: `.segment_count=256`
- **Required/Optional**: Optional

The number of segments must match `clustering.hash.num Segments` in the Data Grid configuration.

In other words, the number of segments for a cache store must match the number of segments for the corresponding cache. If the number of segments is not the same, Data Grid cannot read data from the cache store.

### Table 5.3. JDBC Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example Value</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dialect</strong></td>
<td>Specifies the dialect of the underlying database.</td>
<td></td>
<td>Required</td>
</tr>
<tr>
<td><strong>version</strong></td>
<td>Specifies the marshaller version for source cache stores. Set one of the following values:</td>
<td>For example: <code>source.version=9</code></td>
<td>Required for source stores only.</td>
</tr>
<tr>
<td></td>
<td>* 8 for Data Grid 7.2.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* 9 for Data Grid 7.3.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>* 10 Data Grid 8.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>marshaller.class</strong></td>
<td>Specifies a custom marshaller class.</td>
<td></td>
<td>Required if using custom marshellers.</td>
</tr>
<tr>
<td><strong>marshaller.externalizers</strong></td>
<td>Specifies a comma-separated list of custom AdvancedExternalizer implementations to load in this format: <code>[id]:&lt;Externalizer class&gt;</code></td>
<td></td>
<td>Optional</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Required/Optional</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td><code>connection_pool.connection_url</code></td>
<td>Specifies the JDBC connection URL.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td><code>connection_pool.driver_class</code></td>
<td>Specifies the class of the JDBC driver.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td><code>connection_pool.username</code></td>
<td>Specifies a database username.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td><code>connection_pool.password</code></td>
<td>Specifies a password for the database username.</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td><code>db.major_version</code></td>
<td>Sets the database major version.</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td><code>db.minor_version</code></td>
<td>Sets the database minor version.</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td><code>db.disable_upsert</code></td>
<td>Disables database upsert.</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td><code>db.disable_indexing</code></td>
<td>Specifies if table indexes are created.</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td><code>table.string.table_name_prefix</code></td>
<td>Specifies additional prefixes for the table name.</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>`table.string.&lt;id</td>
<td>data</td>
<td>timestamp&gt;.name`</td>
<td>Specifies the column name.</td>
</tr>
<tr>
<td>`table.string.&lt;id</td>
<td>data</td>
<td>timestamp&gt;.type`</td>
<td>Specifies the column type.</td>
</tr>
<tr>
<td><code>key_to_string_mapper</code></td>
<td>Specifies the <code>TwoWayKey2StringMapper</code> class.</td>
<td>Optional</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

To migrate from Binary cache stores in older Data Grid versions, change `table.string.*` to `table.binary.*` in the following properties:

- `source.table.binary.table_name_prefix`
- `source.table.binary.<id|data|timestamp>.name`
- `source.table.binary.<id|data|timestamp>.type`

# Example configuration for migrating to a JDBC String-Based cache store

```
target.type=STRING
```
target.cache_name=myCache
target.dialect=POSTGRES
target.marshaller.class=org.example.CustomMarshaller
target.marshaller.externalizers=25:Externalizer1,org.example.Externalizer2
target.connection_pool.connection_url=jdbc:postgresql:postgres
target.connection_pool.driver_class=org.postgresql.Driver
target.connection_pool.username=postgres
target.connection_pool.password=redhat
target.db.major_version=9
target.db.minor_version=5
target.db.disable_upsert=false
target.db.disable_indexing=false
target.table.string.table_name_prefix=tablePrefix
target.table.string.id.name=id_column
target.table.string.data.name=datum_column
target.table.string.timestamp.name=timestamp_column
target.table.string.id.type=VARCHAR
target.table.string.data.type=bytea
target.table.string.timestamp.type=BIGINT
target.key_to_string_mapper=org.infinispan.persistence.keymappers.
DefaultTwoWayKey2StringMapper

Table 5.4. RocksDB Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Sets the database directory.</td>
<td>Required</td>
</tr>
<tr>
<td>compression</td>
<td>Specifies the compression type to use.</td>
<td>Optional</td>
</tr>
</tbody>
</table>

# Example configuration for migrating from a RocksDB cache store.
source.type=ROCKSDB
source.cache_name=myCache
source.location=/path/to/rocksdb/database
source.compression=SNAPPY

Table 5.5. SingleFileStore Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>Sets the directory that contains the cache store .dat file.</td>
<td>Required</td>
</tr>
</tbody>
</table>

# Example configuration for migrating to a Single File cache store.
target.type=SINGLE_FILE_STORE
target.cache_name=myCache
target.location=/path/to/sfs.dat

Table 5.6. SoftIndexFileStore Properties
### Example configuration for migrating to a Soft-Index File cache store.
```yaml
# Example configuration for migrating to a Soft-Index File cache store.
target.type=SOFT_INDEX_FILE_STORE
target.cache_name=myCache
target.location=path/to/sifs/database
target.location=path/to/sifs/index
```

### 5.4. MIGRATING CACHE STORES

Run **StoreMigrator** to migrate data from one cache store to another.

#### Prerequisites
- Get **infinispan-tools.jar**.
- Create a **migrator.properties** file that configures the source and target cache stores.

#### Procedure
- If you build **infinispan-tools.jar** from source, do the following:
  1. Add **infinispan-tools.jar** and dependencies for your source and target databases, such as JDBC drivers, to your classpath.
  2. Specify **migrator.properties** file as an argument for **StoreMigrator**.
- If you pull **infinispan-tools.jar** from the Maven repository, run the following command:
  ```bash
  mvn exec:java
  ```