Red Hat Data Grid 8.1

Upgrading Data Grid

Upgrade Data Grid to 8.1

Last Updated: 2021-02-08

Red Hat Data Grid 8.1 Upgrading Data Grid

Upgrade Data Grid to 8.1

Legal Notice

Abstract

Find out about changes in Data Grid 8.1 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

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 8.1 Documentation
- Data Grid 8.1 Component Details
- Supported Configurations for Data Grid 8.1
- Data Grid 8 Feature Support
- Data Grid Deprecated Features and Functionality

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.

1.3. MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 2. PERFORMING ROLLING UPGRADES FOR DATA GRID SERVERS

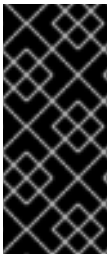
Perform rolling upgrades of your Data Grid clusters to change between versions without downtime or data loss. Rolling upgrades migrate both your Data Grid servers and your data to the target version over Hot Rod.

2.1. 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.

2.1.1. Remote Cache Stores for Rolling Upgrades

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

```
<persistence passivation="false"> 1
  <remote-store xmlns="urn:infinispan:config:store:remote:11.0"
                cache="myDistCache" 2
                protocol-version="2.5" 3
                hotrod-wrapping="true" 4
                raw-values="true" 5
                segmented="false"> 6
    <remote-server host="127.0.0.1" port="11222"/> 7
  </remote-store>
</persistence>
```

1. Disables passivation. Remote cache stores for rolling upgrades must disable passivation.
2. Matches the name of a cache in the source cluster. Target clusters load data from this cache using the remote cache store.
3. 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.
4. Ensures that entries are wrapped in a suitable format for the Hot Rod protocol.
5. Stores data in the remote cache store in raw format. This ensures that clients can use data directly from the remote cache store.
6. Disables segmentation for the remote cache store. You should enable segmentation for remote cache stores only if the number of segments in the target cluster matches the number of segements for the cache in the source cluster.
7. Points to the location of the source cluster.

Reference

- Remote cache store configuration schema
- RemoteStore
- RemoteStoreConfigurationBuilder

2.2. 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 **POST** 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:

   ```
   POST /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 **POST** request:

```
POST /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 3. 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.

3.1. 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.

3.2. 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:

```
$ sha256sum redhat-datagrid-$version-server-patch.zip
```

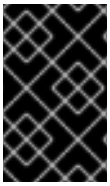
2. Compare with the **MD5** or **SHA-256** checksum value on the Data Grid **Software Details** page.

3.3. 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.

```
$ 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.

```
[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:

```
[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.

```
[disconnected]> patch describe path/to/mypatch.zip
```

```
Red Hat Data Grid patch target=$target_version(my test patch)  source=$source_version
created=$timestamp
```

- **$target_version** is the Data Grid server version from which the patch was created.
- **$source_version** is one or more Data Grid server versions to which you can apply the patch. You can apply patches to Data Grid servers that match the **$source_version** only. Attempting to apply patches to other versions results in the following exception:

```
java.lang.IllegalStateException: The supplied patch cannot be applied to
`$source_version`
```

3.4. 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

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

- **$target_version** displays the Data Grid version that the patch installed.
- **$source_version** displays the Data Grid version before you installed the patch.

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**.

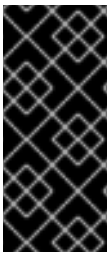
TIP

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
```

3.5. 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.

```
[disconnected]> quit
```

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

```
$ 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 4. 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.

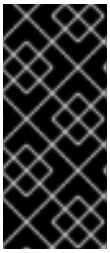
4.1. CACHE STORE MIGRATOR

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.

4.2. 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 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">
```

```
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.infinispan.example</groupId>
    <artifactId>jdbc-migrator-example</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
      <dependency>
        <groupId>org.infinispan</groupId>
        <artifactId>infinispan-tools</artifactId>
      </dependency>
      <!-- Additional dependencies -->
    </dependencies>

    <build>
      <plugins>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>1.2.1</version>
          <executions>
            <execution>
              <goals>
                <goal>java</goal>
              </goals>
            </execution>
          </executions>
          <configuration>
            <mainClass>org.infinispan.tools.store.migrator.StoreMigrator</mainClass>
            <arguments>
              <argument>path/to/migrator.properties</argument>
            </arguments>
          </configuration>
        </plugin>
      </plugins>
    </build>
  </project>
```

4.3. 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:

```
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**.

a. Prepend all configuration properties with **target.** as in the following example:

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

4.3.1. Store Migrator Properties

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

Table 4.1. Cache Store Type Property

Property	Description	Required/Optional
type	Specifies the type of cache store type for a source or target. **.type=JDBC_STRING** **.type=JDBC_BINARY** **.type=JDBC_MIXED** **.type=LEVELDB** **.type=ROCKSDB** **.type=SINGLE_FILE_STORE** **.type=SOFT_INDEX_FILE_STORE** **.type=JDBC_MIXED**	Required

Table 4.2. Common Properties

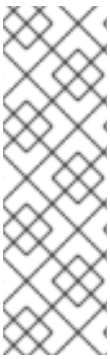
Property	Description	Example Value	Required/Optional
cache_name	Names the cache that the store backs.	**.cache_name=myCache**	Required

Property	Description	Example Value	Required/Optional
segment_count	Specifies the number of segments for target cache stores that can use segmentation. The number of segments must match **clustering.hash.numSegments** 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.	**.segment_count=256**	Optional

Table 4.3. JDBC Properties

Property	Description	Required/Optional
dialect	Specifies the dialect of the underlying database.	Required
version	Specifies the marshaller version for source cache stores. Set one of the following values: * **8** for Data Grid 7.2.x * **9** for Data Grid 7.3.x * **10** Data Grid 8.x	Required for source stores only. For example: **source.version=9**
marshaller.class	Specifies a custom marshaller class.	Required if using custom marshallers.
marshaller.externalizers	Specifies a comma-separated list of custom **AdvancedExternalizer** implementations to load in this format: **[id]:<Externalizer class>**	Optional

Property	Description	Required/Optional
connection_pool.connection_url	Specifies the JDBC connection URL.	Required
connection_pool.driver_class	Specifies the class of the JDBC driver.	Required
connection_pool.username	Specifies a database username.	Required
connection_pool.password	Specifies a password for the database username.	Required
db.major_version	Sets the database major version.	Optional
db.minor_version	Sets the database minor version.	Optional
db.disable_upsert	Disables database upsert.	Optional
db.disable_indexing	Specifies if table indexes are created.	Optional
table.string.table_name_prefix	Specifies additional prefixes for the table name.	Optional
table.string.<id|data|timestamp>.name	Specifies the column name.	Required
table.string.<id|data|timestamp>.type	Specifies the column type.	Required
key_to_string_mapper	Specifies the **TwoWayKey2StringMapper** class.	Optional



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.postrgesql.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 4.4. RocksDB Properties

Property	Description	Required/Optional
location	Sets the database directory.	Required
compression	Specifies the compression type to use.	Optional

```
# 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 4.5. SingleFileStore Properties

Property	Description	Required/Optional
location	Sets the directory that contains the cache store **.dat** file.	Required

```
# 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 4.6. SoftIndexFileStore Properties

Property	Description	Value
Required/Optional	**location**	Sets the database directory.
Required	**index_location**	Sets the database index directory.

```
# 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
```

4.4. 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:
  mvn exec:java