Red Hat Data Grid 8.3

Upgrading Data Grid

Upgrade Data Grid to 8.3

Last Updated: 2023-11-24

Red Hat Data Grid 8.3 Upgrading Data Grid

Upgrade Data Grid to 8.3

Legal Notice

Abstract

Upgrade Data Grid clusters from one 8.x version to another. You can perform rolling upgrades to avoid downtime or offline upgrades during which Data Grid converts data for compatibility.

Table of Contents

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.

DATA GRID DOCUMENTATION

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

- Data Grid 8.3 Documentation
- Data Grid 8.3 Component Details
- Supported Configurations for Data Grid 8.3
- Data Grid 8 Feature Support
- Data Grid Deprecated Features and Functionality

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.

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 1. DATA GRID 8 UPGRADE NOTES

Review the details in this section before upgrading from one Data Grid 8 version to another.

1.1. UPGRADING TO DATA GRID 8.3

Read the following information to ensure a successful upgrade from previous versions of Data Grid 8 to 8.3:

Upgrading deployments with file-based cache stores

As of Data Grid 8.3, file-based cache stores default to soft index, **SoftIndexFileStore**, instead of single-file, **SingleFileStore**. In Data Grid 8.2 and earlier, **SingleFileStore** was the default for file-based cache stores.

Any **file-store** configuration that has a schema version of **13.0** or later is automatically migrated to a **SoftIndexFileStore** after upgrading.

If you are upgrading to Data Grid 8.3 from a previous version, and your caches include any configuration with the **soft-index-file-store** element, you should convert that configuration to use the **file-store** element instead. See the migration documentation for details about schema changes and modifying your Data Grid cache configuration.

Additional resources

- Migrating to Data Grid 8

Upgrading Data Grid Server deployments with multiple endpoint configuration

As of Data Grid Server 8.3 you configure endpoints with security realms and Hot Rod or REST connectors with **endpoint** elements. The **endpoints** element wraps multiple **endpoint** elements.

If you are upgrading Data Grid Server from 8.2 or earlier, and your configuration includes any child elements for the **endpoints** element, you must migrate your configuration to use the **endpoint** element.

Additional resources

- Migrating to Data Grid 8

Upgrade from 8.1 at a minimum

If you are upgrading from 8.0, you must first upgrade to 8.1. Persistent data in Data Grid 8.0 is not binary compatible with later versions. To overcome this incompatibility issue, Data Grid 8.2 and later automatically converts existing persistent cache stores from Data Grid 8.1 at cluster startup. However, Data Grid does not convert cache stores from Data Grid 8.0.

CHAPTER 2. PERFORMING ROLLING UPGRADES FOR DATA GRID SERVER CLUSTERS

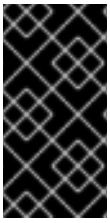
Perform rolling upgrades of your Data Grid clusters to change between versions without downtime or data loss and migrate data over the Hot Rod protocol.

2.1. SETTING UP TARGET DATA GRID CLUSTERS

Create a cluster that uses the Data Grid version to which you plan to upgrade and then connect the source cluster to the target cluster using a remote cache store.

Prerequisites

- Install Data Grid Server nodes with the desired version for your target cluster.



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 port offsets to separate the target and source clusters.

Procedure

1. Create a remote cache store configuration, in JSON format, that allows the target cluster to connect to the source cluster.
   Remote cache stores on the target cluster use the Hot Rod protocol to retrieve data from the source cluster.

```
{
  "remote-store": {
    "cache": "myCache",
    "shared": true,
    "raw-values": true,
    "security": {
      "authentication": {
        "digest": {
          "username": "username",
          "password": "changeme",
          "realm": "default"
        }
      }
    },
    "remote-server": [
      {
        "host": "127.0.0.1",
        "port": 12222
      }
    ]
  }
}
```

2. Use the Data Grid Command Line Interface (CLI) or REST API to add the remote cache store configuration to the target cluster so it can connect to the source cluster.

   - CLI: Use the **migrate cluster connect** command on the target cluster.

     ```
     [//containers/default]> migrate cluster connect -c myCache --file=remote-store.json
     ```

   - REST API: Invoke a POST request that includes the remote store configuration in the payload with the **rolling-upgrade/source-connection** method.

     ```
     POST /v2/caches/myCache/rolling-upgrade/source-connection
     ```

3. Repeat the preceding step for each cache that you want to migrate.

4. 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.

Additional resources

- Remote cache store configuration schema

2.2. SYNCHRONIZING DATA TO TARGET CLUSTERS

When you set up a target Data Grid cluster and connect it to a source cluster, the target cluster can handle client requests using a remote cache store and load data on demand. To completely migrate data to the target cluster, so you can decommission the source cluster, you can synchronize data. 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 that you want to migrate to the target cluster.

Prerequisites

- Set up a target cluster with the appropriate Data Grid version.

Procedure

1. Start synchronizing each cache that you want to migrate to the target cluster with the Data Grid Command Line Interface (CLI) or REST API.

   - CLI: Use the **migrate cluster synchronize** command.

     ```
     migrate cluster synchronize -c myCache
     ```

   - REST API: Use the **?action=sync-data** parameter with a POST request.

     ```
     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.

2. Disconnect each node in the target cluster from the source cluster.

- CLI: Use the **migrate cluster disconnect** command.

```
migrate cluster disconnect -c myCache
```

- REST API: Invoke a DELETE request.

```
DELETE /v2/caches/myCache/rolling-upgrade/source-connection
```

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. 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.

3.1. 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 RocksDB 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.

3.2. GETTING THE CACHE 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>
```

3.3. CONFIGURING THE CACHE 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
source.version=<version>
```

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
```

3.3.1. Configuration properties for the cache store migrator

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

Table 3.1. 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 3.2. Common Properties

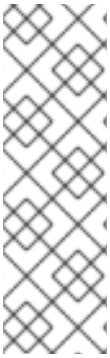
Property	Description	Example Value	Required/Optional
cache_name	Names the cache that the store backs.	**.cache_name=myCache**	Required

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 3.3. 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** for Data Grid 8.0.x * **11** for Data Grid 8.1.x * **12** for Data Grid 8.2.x * **13** for Data Grid 8.3.x	Required for source stores only.
marshaller.class	Specifies a custom marshaller class.	Required if using custom marshallers.

Property	Description	Required/Optional
marshaller.externalizers	Specifies a comma-separated list of custom **AdvancedExternalizer** implementations to load in this format: **[id]:<Externalizer class>**	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 3.4. 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 3.5. 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 3.6. 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
```

3.4. MIGRATING DATA GRID 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**