Red Hat Data Grid 8.2

Upgrading Data Grid

Upgrade Data Grid to 8.2

Last Updated: 2023-11-23

Red Hat Data Grid 8.2 Upgrading Data Grid

Upgrade Data Grid to 8.2

Legal Notice

Abstract

Upgrade Data Grid clusters from one 8.x version to the next. 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 3

DATA GRID DOCUMENTATION 4

DATA GRID DOWNLOADS 5

MAKING OPEN SOURCE MORE INCLUSIVE 6

CHAPTER 1. DATA GRID 8 UPGRADE NOTES 7
1.1. UPGRADING TO DATA GRID 8.2 7
Upgrading deployments with Single File cache stores 7
Upgrade from 8.1 at a minimum 7
Migrating ProtoStream marshaller configuration 7

CHAPTER 2. PERFORMING ROLLING UPGRADES FOR DATA GRID SERVERS 9
2.1. SETTING UP TARGET CLUSTERS 9
2.1.1. Remote Cache Stores for Rolling Upgrades 9
2.2. SYNCHRONIZING DATA TO TARGET CLUSTERS 10

CHAPTER 3. MIGRATING DATA BETWEEN CACHE STORES 12
3.1. CACHE STORE MIGRATOR 12
3.2. GETTING THE STORE MIGRATOR 12
3.3. CONFIGURING THE STORE MIGRATOR 13
3.3.1. Store Migrator Properties 14
3.4. MIGRATING CACHE STORES 18

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 8.2 Documentation
- Data Grid 8.2 Component Details
- Supported Configurations for Data Grid 8.2
- 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 8.2

Read the following information to ensure a successful upgrade from previous versions of Data Grid 8 to 8.2:

Upgrading deployments with Single File cache stores

When upgrading Data Grid to 8.2.0, caches that include a **SingleFileStore** persistence configuration can encounter an issue that leads to data corruption.

This issue affects upgrades to Data Grid 8.2.0 only. As of Data Grid 8.2.1 this issue no longer occurs during upgrade.

If you have already upgraded from an earlier version to 8.2.0, you should do the following as soon as possible:

1. Back up any **$RHDG_HOME/server/data/*.dat** files.
2. Upgrade to Data Grid 8.2.1 or later.

After successful upgrade, Data Grid recovers any corrupted data and restores the Single File Store on first start.

Cross-site replication state transfer

For caches that backup to other clusters via cross-site replication, you should perform a state transfer after upgrading to 8.2.

From the Infinispan CLI use the **site push-site-state** command as follows:

```
[//containers/default]> site push-site-state --cache=cacheName --site=NYC
```

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 Data Grid 8.2 because user serialization contexts are separated from Data Grid serialization contexts in 8.2. To overcome this incompatibility issue, Data Grid 8.2 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.

Migrating ProtoStream marshaller configuration

Data Grid 8.2 upgrades the ProtoStream library that provides marshalling capabilities. As part of the upgrade process from Data Grid 8.1 you should also review ProtoStream migration details to avoid any data compatibility issues that might arise from differences in how ProtoStream encodes entries as Protobuf.

In addition the **MessageMarshaller** API and the **ProtoSchemaBuilder** annotation are deprecated in the ProtoStream API. You should migrate any serialization context initializers in Data Grid 8.1 to the **AutoProtoSchemaBuilder** annotation as part of the upgrade to Data Grid 8.2.

Additional resources

- ProtoStream annotations

- Creating serialization context initializers
- Migrating applications to the AutoProtoSchemaBuilder annotation

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:

```
<!-- Remote cache stores for rolling upgrades must disable passivation. -->
<persistence passivation="false">
  <!-- The value of the cache attribute matches the name of a cache in the source cluster. Target clusters load data from this cache using the remote cache store. -->
  <!-- The "protocol-version" attribute matches the Hot Rod protocol version of the source cluster. 2.5 is the minimum version and is suitable for any upgrade path. -->
  <!-- You should enable segmentation for remote cache stores only if the number of segments in the target cluster matches the number of segments for the cache in the source cluster. -->
  <remote-store xmlns="urn:infinispan:config:store:remote:12.1"
                cache="myDistCache"
                protocol-version="2.5"
```

```
            hotrod-wrapping="true"
            raw-values="true"
            segmented="false">
    <!-- Configures authentication and encryption according to the security realm of the source
cluster. -->
    <security>
      <authentication server-name="infinispan">
        <digest username="admin"
            password="changeme"
            realm="default"/>
      </authentication>
    </security>
    <!-- Points to the location of the source cluster. -->
    <remote-server host="127.0.0.1" port="11222"/>
  </remote-store>
</persistence>
```

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. 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 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.

3.2. 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>
```

3.3. 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
```

3.3.1. Store Migrator Properties

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** 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 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 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