Red Hat Data Grid 8.2

Upgrading Data Grid

Upgrade Data Grid to 8.2
Red Hat Data Grid 8.2 Upgrading Data Grid

Upgrade Data Grid to 8.2
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. PATCHING DATA GRID SERVER INSTALLATIONS** ................................................................. 12
  3.1. DATA GRID SERVER PATCHES .......................................................................................................... 12
  3.2. DOWNLOADING SERVER PATCHES .................................................................................................... 12
  3.3. INSTALLING SERVER PATCHES ......................................................................................................... 13
  3.4. ROLLING BACK SERVER PATCHES .................................................................................................... 14

**CHAPTER 4. MIGRATING DATA BETWEEN CACHE STORES** ..................................................................... 16
  4.1. CACHE STORE MIGRATOR .................................................................................................................. 16
  4.2. GETTING THE STORE MIGRATOR ..................................................................................................... 16
  4.3. CONFIGURING THE STORE MIGRATOR ............................................................................................ 17
    4.3.1. Store Migrator Properties .......................................................................................................... 18
  4.4. MIGRATING CACHE STORES ............................................................................................................. 22
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 and then 8.2.

User serialization contexts are separated from Data Grid serialization contexts in Data Grid 8.2, which results in binary incompatibility with previous versions.

To overcome this incompatibility issue, Data Grid 8.2 automatically converts any existing persistent cache stores from Data Grid 8.1 at startup. However, Data Grid 8.2 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:

```xml
<persistence passivation="false">

<!-- Remote cache stores for rolling upgrades must disable passivation. -->

<!-- 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"
>
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 Server installations with different versions to upgrade to a desired target version. However, patches do not take effect if Data Grid Server is running. 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.

The Data Grid command line interface (CLI) includes a patch create command that lets you create custom patches. However this command is intended for use only for Infinispan Server, or community, deployments.

IMPORTANT

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

3.2. DOWNLOADING SERVER PATCHES

Download patches that you can apply to Data Grid Server installations.

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:

   ```bash
   $ 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. INSTALLING SERVER PATCHES

Apply patches to Data Grid Server 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 installation you want to patch.

2. Stop the Data Grid 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.

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

4. Install the patch.

   ```bash
   [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

3.4. ROLLING BACK SERVER PATCHES

Remove patches from Data Grid Server 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.

$ 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
<?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"/>
```
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`. 
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**

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

**Table 4.2. Common Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example Value</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cache_name</code></td>
<td>Names the cache that the store backs.</td>
<td><code>.cache_name=myCache</code></td>
<td>Required</td>
</tr>
</tbody>
</table>
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 4.3. JDBC Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Example Value</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialect</td>
<td>Specifies the dialect of the underlying database.</td>
<td></td>
<td>Required</td>
</tr>
<tr>
<td>version</td>
<td>Specifies the marshaller version for source cache stores. Set one of the following values:</td>
<td><code>source.version=9</code></td>
<td>Required for source stores only. For example: source.version=9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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>marshaller.class</td>
<td>Specifies a custom marshaller class.</td>
<td></td>
<td>Required if using custom marshellers.</td>
</tr>
<tr>
<td>marshaller.externalizers</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 TwoWayKey2StringMapper 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

```plaintext```
target.type=STRING
```plaintext```
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 4.4. 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.

```java
source.type=ROCKSDB
source.cache_name=myCache
source.location=/path/to/rocksdb/database
source.compression=SNAPPY
```

### Table 4.5. 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.

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

### Table 4.6. SoftIndexFileStore Properties
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required/Optional</td>
<td>location</td>
<td>Sets the database directory.</td>
</tr>
<tr>
<td>Required</td>
<td>index_location</td>
<td>Sets the database index directory.</td>
</tr>
</tbody>
</table>

```bash
# 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:
  ```bash
  mvn exec:java
  ```