Red Hat Data Grid 8.4

Upgrading Data Grid

Upgrade Data Grid to 8.4

Last Updated: 2024-03-26

Red Hat Data Grid 8.4 Upgrading Data Grid

Upgrade Data Grid to 8.4

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 8.4 Documentation
- Data Grid 8.4 Component Details
- Supported Configurations for Data Grid 8.4
- 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 8.4

Read the following information to ensure a successful upgrade from previous versions of Data Grid 8 to 8.4:

Hot Rod client defaults

Data Grid 8.4.6 introduced changes to the properties of the Hot Rod client.

infinispan.client.hotrod.ssl_hostname_validation

A new property, **infinispan.client.hotrod.ssl_hostname_validation** with a default value of **true**. This property enables TLS hostname validation based on RFC 2818 rules. Additionally, setting the **infinispan.client.hotrod.sni_host_name** is now required when hostname validation is enabled.

Table 1.1. Default property changes

Property	Data Grid 8.4	Previous versions
infinispan.client.hotrod.connect_timeout	**2000 ms / 2 seconds**	60000 ms / 60 seconds
infinispan.client.hotrod.socket_timeout	**2000 ms / 2 seconds**	60000 ms / 60 seconds
infinispan.client.hotrod.max_retries	**3**	10
infinispan.client.hotrod.min_evictable_idle_time	**180000 ms / 3 minutes**	1800000 ms / 30 minutes

Improved metrics naming for JGroups and cross-site metrics

In Data Grid 8.4.4, you can enable the **name-as-tags** property for JGroups metrics and cross-site metrics.

Enabling **name-as-tags** simplifies metrics, displaying cluster and site names as tags rather than including them in metric names.

When you set **name-as-tags** to **false**, metrics are named based on the channel, resulting in multiple metrics for the same purpose:

```
# TYPE vendor_jgroups_xsite_frag4_get_number_of_sent_fragments gauge
# HELP vendor_jgroups_xsite_frag4_get_number_of_sent_fragments Number of sent fragments
vendor_jgroups_xsite_frag4_get_number_of_sent_fragments{cluster="xsite",node="..."} 0.0
# TYPE vendor_jgroups_cluster_frag4_get_number_of_sent_fragments gauge
# HELP vendor_jgroups_cluster_frag4_get_number_of_sent_fragments Number of sent fragments
vendor_jgroups_cluster_frag4_get_number_of_sent_fragments{cluster="cluster",node="..."} 2.0
```

When you set **name-as-tags** to **true**, metrics are simplified, and cluster and site names appear as tags:

```
# TYPE vendor_jgroups_frag4_get_number_of_sent_fragments gauge
# HELP vendor_jgroups_frag4_get_number_of_sent_fragments Number of sent fragments
```

```
vendor_jgroups_frag4_get_number_of_sent_fragments{cache_manager="default",cluster="xsite",node="..."} 0.0
vendor_jgroups_frag4_get_number_of_sent_fragments{cache_manager="default",cluster="cluster",node="..."} 2.0
```

In addition to simplified metrics, when you change the cluster name and site name, there is no need to update Grafana dashboards, as the metric names remain consistent.

Migrating from Java 8

As of Data Grid 8.4, Red Hat supports Java 11 and Java 17 for Data Grid Server installations, Hot Rod Java clients, and when using Data Grid for embedded caches in custom applications. Data Grid users must upgrade their applications at least to Java 11. Support for Java 8 was deprecated in Data Grid 8.2 and removed in Data Grid 8.4.

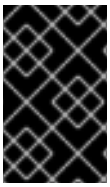
Embedded caches

Red Hat supports Java 11 and Java 17 when using Data Grid for embedded caches in custom applications. Data Grid users must upgrade their applications at least to Java 11.

Remote caches

Red Hat supports Java 11 and Java 17 for Data Grid Server and Hot Rod Java clients.

Hot Rod Java clients running in applications that require Java 8 can continue using older versions of client libraries. Red Hat supports using older Hot Rod Java client versions in combination with the latest Data Grid Server version. However, if you continue using older version of the client you will miss fixes and enhancements.



IMPORTANT

OpenJDK 17 has removed support for the Nashorn JavaScript engine, its APIs, and the **jjs** tool. If your Data Grid Server uses JavaScript to automate tasks, you must install the Nashorn JavaScript engine.

Adding Jakarta EE dependencies

As of version 8.4 Data Grid distributes Jakarta EE 9+ based jars. If your application requires Jakarta specific dependencies, append the artifacts with **-jakarta**, for example:

pom.xml

```
<dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-client-hotrod-jakarta</artifactId>
</dependency>
```

Updating store properties configuration

In Data Grid 8.4, store provided properties no longer override store explicit configuration. The following example illustrates how you should update your configuration.

This example sets the read only configuration of the store to **true**:

example.xml

```
<persistence>
  <file-store>
```

```
    <index path="testCache/index" />
    <data path="testCache/data" />
    <property name="readOnly">true</property>
  </file-store>
</persistence>
```

In the updated example bellow, the **read-only** property is provided to the store itself and does not affect the store explicit configuration.

example.xml

```
<persistence>
  <file-store read-only="true">
    <index path="testCache/index" />
    <data path="testCache/data" />
  </file-store>
</persistence>
```

Additional resources

- Migrating to Data Grid 8

Deprecated features and functionality

In Data Grid 8.4.3, the following features and functionality are deprecated and planned to be removed in future Data Grid releases.

Red Hat will provide support for these features and functionalities during the current release lifecycle, but they will no longer receive enhancements and will be eventually removed. It is recommended to transition to alternative solutions to ensure future compatibility.

Support for Java 11

Support for Java 11 is deprecated and planned to be removed in Data Grid version 8.5. Users of Data Grid 8.5 must upgrade their applications at least to Java 17.

You can continue using older Hot Rod Java client versions in combination with the latest Data Grid Server version. However, if you continue using older version of the client you will miss fixes and enhancements.

Support for Java EE dependencies

Support for Java EE dependencies is deprecated and planned to be removed in Data Grid version 8.5. Transition to Jakarta EE dependencies and APIs to align with the evolving Java enterprise ecosystem.

Support for Spring 5.x and Spring Boot 2.x

Support for Spring Boot 2.x and Spring 5.x is deprecated and planned to be removed in version Data Grid 8.5. Migrate to newer versions of Spring Boot and Spring framework for compatibility with future Data Grid releases.

Support for JCache

Support for JCache (JSR 107) is deprecated and planned to be removed in Data Grid version 8.5. As an alternative use other caching API developments in the Jakarta EE ecosystem.

Deprecation of Data Grid modules for Red Hat JBoss EAP

The Data Grid modules for Red Hat JBoss EAP applications that were distributed as a part of the Data Grid release are deprecated and planned to be removed in Data Grid version 8.5.

JBoss EAP users can use the **infinispan** subsystem that is integrated within the JBoss EAP product release without the need to separately install Data Grid modules.

Scattered cache mode

Scattered cache mode is deprecated and planned to be removed in Data Grid version 8.5. As an alternative to scattered caches, you can use distributed caches instead.

Adding caches to ignore list using the Data Grid Console or REST API

The ability to add caches to the ignore list using the Data Grid Console or REST API, which allows temporarily excluding specific caches from client requests, is deprecated. This feature is planned to be removed in future releases.

Cache service type

The **Cache** service type is deprecated and planned to be removed in Data Grid 8.5. The **Cache** service type was designed to provide a convenient way to create a low-latency data store with minimal configuration. Use the **DataGrid** service type to automate complex operations such as cluster upgrades and data migration.

Testing Data Grid Server on Windows

The support for Data Grid Server on on Windows Server 2019 is deprecated and planned to be removed in Data Grid 8.5. However, the Data Grid team will continue testing C++ Hot Rod client with Windows Server 2019.

Deprecation of the **PrincipalRoleMapperContext** interface

org.infinispan.security.PrincipalRoleMapperContext was deprecated in Data Grid 8.4 and replaced by **org.infinispan.security.AuthorizationMapperContext**.

Removal of the **fetch-state** store property

The **fetch-state** attribute has been deprecated and removed in Data Grid 8.4 without any replacement. You can remove the attribute from your xml configuration.

This change does not affect shared stores that have access to the same data. Local cache stores can use purge on startup to avoid loading stale entries from persistent storage.

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
      }
    ]
  }
}
```

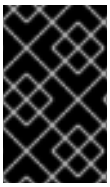
2. Use the Data Grid Command Line Interface (CLI) or REST API to add the remote cache store configuration to the target cluster so it can connect to the source cluster.
   - CLI: Use the **migrate cluster connect** command on the target cluster.

     ```
     [//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 /rest/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.



IMPORTANT

If you need to migrate Indexed caches you must first migrate the internal **___protobuf_metadata** cache so that the .proto schemas defined on the source cluster will also be present on the target cluster.

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 /rest/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 /rest/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 persistent 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.

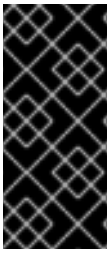
3.1. CACHE STORE MIGRATOR

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. CONFIGURING THE CACHE STORE MIGRATOR

Use the **migrator.properties** file to configure properties for source and target cache stores.

Procedure

1. Create a **migrator.properties** file.
2. Configure properties for source and target cache store using the **migrator.properties** file.
   a. Add the **source.** prefix to all configuration properties for the source cache store.

   Example source cache store

   ```
   source.type=SOFT_INDEX_FILE_STORE
   ```

```
source.cache_name=myCache
source.location=/path/to/source/sifs
source.version=<version>
```

IMPORTANT

For migrating data from segmented cache stores, you must also configure the number of segments using the **source.segment_count** property. The number of segments must match **clustering.hash.numSegments** in your Data Grid configuration. If the number of segments for a cache store does not match the number of segments for the corresponding cache, Data Grid cannot read data from the cache store.

b. Add the **target.** prefix to all configuration properties for the target cache store.

Example target cache store

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

3.2.1. 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 for a source or target cache store. **.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

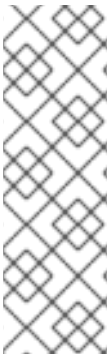
Table 3.2. Common Properties

Property	Description	Example Value	Required/Optional
cache_name	The name of the cache that you want to back up.	**.cache_name=myCache**	Required
segment_count	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. If the number of segments for a cache store does not match the number of segments for the corresponding cache, 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.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.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
```

3.3. MIGRATING DATA GRID CACHE STORES

You can use the **StoreMigrator** to migrate data between cache stores with different Data Grid versions or to migrate data from one type of cache store to another.

Prerequisites

- Have a **infinispan-tools.jar**.
- Have the source and target cache store configured in the **migrator.properties** file.

Procedure

- If you built the **infinispan-tools.jar** from the source code, do the following:
  1. Add **infinispan-tools.jar** to your classpath.
  2. Add dependencies for your source and target databases, such as JDBC drivers to your classpath.
  3. Specify **migrator.properties** file as an argument for **StoreMigrator**.
- If you pulled **infinispan-tools.jar** from the Maven repository, run the following command:

```
mvn exec:java
```