Red Hat Data Grid 8.0

Data Grid Migration Guide

Data Grid Documentation

Last Updated: 2020-06-02

Red Hat Data Grid 8.0 Data Grid Migration Guide

Data Grid Documentation

Legal Notice

Abstract

Migrate to Data Grid 8.0 from previous versions.

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 8.0 Documentation
- Data Grid 8.0 Component Details
- Supported Configurations for Data Grid 8.0

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.

CHAPTER 2. MIGRATING TO DATA GRID 8.0

Review changes in Data Grid 8.0 that affect migration from previous releases.

2.1. DATA GRID 8.0 SERVER

As of 8.0, Data Grid server is no longer based on Red Hat JBoss Enterprise Application Platform (EAP) and is re-designed to be lightweight and more secure with much faster start times.

Data Grid servers use **$RHDG_HOME/server/conf/infinispan.xml** for configuration.

Data store configuration

You configure how Data Grid stores your data through cache definitions. By default, Data Grid servers include a Cache Manager configuration that lets you create, configure, and manage your cache definitions.

```
<cache-container name="default" 1
                 statistics="true"> 2
  <transport cluster="${infinispan.cluster.name}" 3
             stack="${infinispan.cluster.stack:tcp}" 4
             node-name="${infinispan.node.name:}"/>
</cache-container>
```

1. Creates a Cache Manager named "default".
2. Exports Cache Manager statistics through the **metrics** endpoint.
3. Adds a JGroups cluster transport that allows Data Grid servers to automatically discover each other and form clusters.
4. Uses the default TCP stack for cluster traffic.

In the preceding configuration, there are no cache definitions. When you start 8.0 server, it instantiates the default Cache Manager so you can create cache definitions at runtime through the CLI, REST API, or from remote Hot Rod clients.

NOTE

Data Grid server no longer provides a domain mode as in previous versions that were based on EAP. However, Data Grid server provides a default configuration with clustering capabilities so your data is replicated across all nodes.

Server configuration

Data Grid 8.0 extends **infinispan.xml** with a **server** element that defines configuration specific to Data Grid servers.

```
<server>
  <interfaces>
    <interface name="public">
      <inet-address value="${infinispan.bind.address:127.0.0.1}"/> 1
    </interface>
```

```
  </interfaces>

  <socket-bindings default-interface="public"
                   port-offset="${infinispan.socket.binding.port-offset:0}">
    <socket-binding name="default"
                    port="${infinispan.bind.port:11222}"/> (2)
    <socket-binding name="memcached"
                    port="11221"/> (3)
  </socket-bindings>

  <security>
    <security-realms>
      <security-realm name="default"> (4)
        <properties-realm groups-attribute="Roles">
          <user-properties path="users.properties" relative-to="infinispan.server.config.path" plain-
text="true"/>
          <group-properties path="groups.properties" relative-to="infinispan.server.config.path" />
        </properties-realm>
      </security-realm>
    </security-realms>
  </security>

  <endpoints socket-binding="default" security-realm="default"> (5)
    <hotrod-connector name="hotrod"/>
    <rest-connector name="rest"/>
  </endpoints>
</server>
```

1. Creates a default public interface that uses the **127.0.0.1** loopback address.
2. Creates a default socket binding that binds the public interface to port **11222**.
3. Creates a socket binding for the Memcached connector. Note that the Memcached endpoint is now deprecated.
4. Defines a default security realm that uses property files to define credentials and RBAC settings.
5. Exposes the Hot Rod and REST endpoints at **127.0.0.1:11222**.

IMPORTANT

The REST endpoint handles administrative operations that the Data Grid command line interface (CLI) and console use. For this reason, you should never disable the REST endpoint.

Table 2.1. Cheat Sheet

7.x	8.x
./standalone.sh -c clustered.xml	**./server.sh**
./standalone.sh	**./server.sh -c infinispan-local.xml**

7.x	8.x
-Djboss.default.multicast.address=234.99.54.20	**-Djgroups.mcast_addr=234.99.54.20**
-Djboss.bind.address=172.18.1.13	**-Djgroups.bind.address=172.18.1.13**
-Djboss.default.jgroups.stack=udp	**-j udp**

- Use custom UDP/TCP addresses as follows:
  -Djgroups.udp.address=172.18.1.13
  -Djgroups.tcp.address=172.18.1.1
- Enable JMX as follows:

```
<cache-container name="default"
                 statistics="true"> 1
  <jmx enabled="true" /> 2
  ...
```

1 Enables statistics for the Cache Manager. This is the default.

2 Exports JMX MBeans.

Reference

- Getting Started with Data Grid Servers
- Network Interfaces: Server Guide
- Socket Bindings: Server Guide
- Endpoints: Server Guide
- Defining Property Realms: Server Guide
- Security: Server Guide
- Cluster Transport: Configuration Guide
- Creating Caches with the CLI
- Creating Caches through the REST API

2.2. DATA GRID CACHES

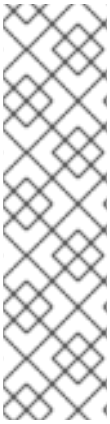
Except for the Cache service on OpenShift, Data Grid provides empty cache containers by default. When you start Data Grid 8.0 it instantiates a Cache Manager so you can create caches at runtime.

In Data Grid 8.0, cache definitions that you create through the **CacheContainerAdmin** API are permanent to ensure that they survive cluster restarts.

```
.administration()
  .withFlags(AdminFlag.VOLATILE) 1
  .getOrCreateCache("myTemporaryCache", "org.infinispan.DIST_SYNC"); 2
```

1 includes the **VOLATILE** flag that changes the default behavior and creates temporary caches.

2 returns a cache named "myTemporaryCache" or creates one using the **DIST_SYNC** configuration template.



NOTE

AdminFlag.PERMANENT is enabled by default to ensure that cache definitions survive restarts. You must separately add persistent storage to Data Grid for data to survive restarts, for example:

```
ConfigurationBuilder b = new ConfigurationBuilder();
b.persistence()
  .addSingleFileStore()
  .location("/tmp/myDataStore")
  .maxEntries(5000);
```

Cache Configuration Templates

Get the list of cache configuration templates as follows:

- Use **Tab** auto-completion with the CLI:

```
[//containers/default]> create cache --template=
```

- Use the REST API:

```
GET 127.0.0.1:11222/rest/v2/cache-managers/default/cache-configs/templates
```

2.3. CREATING CACHES

Add cache definitions to Data Grid to configure how it stores your data.

Library Mode

The following example initializes the Cache Manager and creates a cache definition named "myDistributedCache" that uses the distributed, synchronous cache mode:

```
GlobalConfigurationBuilder global = GlobalConfigurationBuilder.defaultClusteredBuilder();
    DefaultCacheManager cacheManager = new DefaultCacheManager(global.build());
    ConfigurationBuilder builder = new ConfigurationBuilder();
    builder.clustering().cacheMode(CacheMode.DIST_SYNC);
cacheManager.defineConfiguration("myDistributedCache", builder.build());
```

You can also use the **getOrCreate()** method to create your cache definition or return it if it already exists, for example:

```
cacheManager.administration().getOrCreateCache("myDistributedCache", builder.build());
```

Data Grid Server

Remotely create caches at runtime as follows:

- Use the CLI.
  To create a cache named "myCache" with the **DIST_SYNC** cache template, run the following:

  ```
  [//containers/default]> create cache --template=org.infinispan.DIST_SYNC
  name=myDistributedCache
  ```

- Use the REST API.
  To create a cache named "myCache", use the following **POST** invocation and include the cache definition in the request payload in XML or JSON format:

  ```
  POST /rest/v2/caches/myCache
  ```

- Use Hot Rod clients.

  ```
  import org.infinispan.client.hotrod.RemoteCacheManager;
  import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
  import org.infinispan.client.hotrod.impl.ConfigurationProperties;
  import org.infinispan.commons.api.CacheContainerAdmin.AdminFlag;
  import org.infinispan.commons.configuration.XMLStringConfiguration;

  // Create a configuration for a locally running server.
  ConfigurationBuilder builder = new ConfigurationBuilder();
        builder.addServer().host("127.0.0.1").port(11222);

        manager = new RemoteCacheManager(builder.build());
    }

    ...

    private void createTemporaryCacheWithTemplate() {
          manager.administration()
                //Override the default and create a volatile cache that
                //does not survive cluster restarts.
                .withFlags(AdminFlag.VOLATILE)
                //Create a cache named myTemporaryCache that uses the
                //distributed, synchronous cache template
                //or return it if it already exists.
                .getOrCreateCache("myTemporaryCache", "org.infinispan.DIST_SYNC");
      }
  ```

For more examples of creating caches with a Hot Rod Java client, see the Data Grid tutorials.

2.4. CACHE HEALTH STATUS

Data Grid now returns one of the following for cache health:

HEALTHY means a cache is operating as expected.
HEALTHY_REBALANCING means a cache is in the rebalancing state but otherwise operating as expected.
DEGRADED indicates a cache is not operating as expected and possibly requires troubleshooting.

2.5. MARSHALLING CAPABILITIES

As of this release, the default marshaller for Data Grid is ProtoStream, which marshalls data as Protocol Buffers, a language-neutral, backwards compatible format.

To use ProtoStream, Data Grid requires serialization contexts that contain:

- **.proto** schemas that provide a structured representation of your Java objects as Protobuf message types.
- Marshaller implementations to encode your Java objects to Protobuf format.

Data Grid provides direct integration with ProtoStream libraries and can generate everything you need to initialize serialization contexts.

IMPORTANT

Cache stores in previous versions of Data Grid store data in a binary format that is not compatible with ProtoStream marshallers. You must use the **StoreMigrator** utility to migrate your data.

- Data Grid Library Mode does not include JBoss Marshalling by default. You must add the **infinispan-jboss-marshalling** dependency to your classpath.
- Data Grid servers do support JBoss Marshalling but clients must declare the marshaller to use, as in the following Hot Rod client configuration:
.marshaller("org.infinispan.jboss.marshalling.core.JBossUserMarshaller");
- Spring integration does not yet support the default ProtoStream marshaller. For this reason you should use the Java Serialization Marshaller.
- To use the Java Serialization Marshaller, you must add classes to the deserialization whitelist.

Reference

- Data Grid Marshalling
- ProtoStream Marshalling
- Creating Context Initializers
- JBoss Marshalling
- Data Grid Spring Boot Starter
- Java Serialization Marshaller
- Adding Java Classes to Deserialization White Lists

2.6. DATA GRID CONFIGURATION

New and Modified Elements and Attributes

- **stack** adds support for inline JGroups stack definitions.
- **stack.combine** and **stack.position** attributes let you override and modify JGroups stack definitions.
- **metrics** lets you configure how Data Grid exports metrics that are compatible with the Eclipse MicroProfile Metrics API.
- **context-initializer** lets you specify a **SerializationContextInitializer** implementation that initializes a Protostream-based marshaller for user types.
- **key-transformers** lets you register transformers that convert custom keys to String for indexing with Lucene.
- **statistics** now defaults to "false".

Deprecated Elements and Attributes

The following elements and attributes are now deprecated:

- **address-count** attribute for the **off-heap** element.
- **protocol** attribute for the **transaction** element.
- **duplicate-domains** attribute for the **jmx** element.
- **advanced-externalizer**
- **custom-interceptors**
- **state-transfer-executor**
- **transaction-protocol**

NOTE

Refer to the Configuration Schema for possible replacements or alternatives.

Removed Elements and Attributes

The following elements and attributes were deprecated in a previous release and are now removed:

- **deadlock-detection-spin**
- **compatibility**
- **write-skew**
- **versioning**
- **data-container**
- **eviction**
- **eviction-thread-policy**

Reference

- Data Grid Configuration Schema
- Data Grid Configuration Guide

2.7. PERSISTENCE

In comparison with some previous versions of Data Grid, such as 7.1, there are changes to cache store configurations. Cache store definitions must:

- Be contained within **persistence** elements.
- Include an **xlmns** namespace.

As of this release, cache store configuration:

- Defaults to **segmented="true"** if the cache store implementation supports segmentation.
- Removes the **singleton** attribute for the **store** element. Use **shared=true** instead.

JDBC String-Based cache stores use connections factories based on Agroal to connect to databases. It is no longer possible to use **c3p0.properties** and **hikari.properties** files.

Likewise, JDBC String-Based cache store configuration that use segmentation, which is now the default, must include the **segmentColumnName** and **segmentColumnType** parameters.

MySQL Example

```
builder.table()
      .tableNamePrefix("ISPN")
      .idColumnName("ID_COLUMN").idColumnType(“VARCHAR(255)”)
      .dataColumnName("DATA_COLUMN").dataColumnType(“VARBINARY(1000)”)
      .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType(“BIGINT”)
      .segmentColumnName("SEGMENT_COLUMN").segmentColumnType("INTEGER")
```

PostgreSQL Example

```
builder.table()
      .tableNamePrefix("ISPN")
      .idColumnName("ID_COLUMN").idColumnType(“VARCHAR(255)”)
      .dataColumnName("DATA_COLUMN").dataColumnType(“BYTEA”)
      .timestampColumnName("TIMESTAMP_COLUMN").timestampColumnType("BIGINT”)
      .segmentColumnName("SEGMENT_COLUMN").segmentColumnType("INTEGER");
```

Reference

- Data Grid Configuration Schema
- Setting Up Persistent Storage
- Segmented Cache Stores
- JDBC String-Based Cache Stores

2.8. REST API

Previous versions of the Data Grid REST API were v1, which is now replaced by REST API v2.

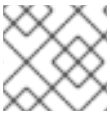
The default context path is now **127.0.0.1:11222/rest/v2/**. You must update any clients or scripts to use REST API v2.

Reference

- Data Grid REST API

2.9. HOT ROD CLIENT AUTHENTICATION

Hot Rod clients now use **SCRAM-SHA-512** as the default authentication mechanism instead of **DIGEST-MD5**.



NOTE

If you use property security realms, you must use the **PLAIN** authentication mechanism.

Reference

- Configuring Authentication Mechanisms for Hot Rod Java Clients

2.10. JAVA DISTRIBUTIONS AVAILABLE IN MAVEN

Data Grid no longer provides Java artifacts outside the Maven repository, with the exception of the Data Grid server distribution. For information on adding required dependencies for the Data Grid Library, Hot Rod Java client, and utilities such as **StoreMigrator**, see the relevant documentation.

Reference

- Data Grid Library Mode
- Hot Rod Java Client Guide
- Getting the Store Migrator

2.11. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (EAP) MODULES

Data Grid no longer provides modules for applications running on EAP. Instead, EAP will provide direct integration with Data Grid in a future release.

However, until EAP provides functionality for handling the **infinispan** subsystem, you must package Data Grid 8.0 artifacts in your EAP deployments.

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

CHAPTER 4. DEPRECATED FEATURES AND FUNCTIONALITY

Support for deprecated functionality is not available beyond the release in which it is deprecated.

IMPORTANT

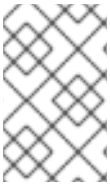
Red Hat does not recommend including, enabling, or configuring deprecated functionality in new deployments.

4.1. DEPRECATIONS

Data Grid 8.0 deprecates the following features and functionality:

Memcached Endpoint Connector

As of this release, Data Grid no longer supports the Memcached endpoint. The Memcached connector is deprecated and planned for removal in a future release.



NOTE

If you have a use case or requirement for the Memcached connector, contact your Red Hat support team to discuss requirements for a future Data Grid implementation of the Memcached connector.

JBoss Marshalling

JBoss Marshalling is a Serialization-based marshalling library and was the default marshaller in previous Data Grid versions. You should not use serialization-based marshalling with Data Grid but instead use Protostream, which is a high-performance binary wire format that ensures backwards compatibility.

Externalizers

The following interfaces and annotations are now deprecated:

- **org.infinispan.commons.marshall.AdvancedExternalizer**
- **org.infinispan.commons.marshall.Externalizer**
- **@SerializeWith**

NOTE

Data Grid ignores **AdvancedExternalizer** implementations when persisting data unless you use JBoss Marshalling.

Total Order Transaction Protocol

The **org.infinispan.transaction.TransactionProtocol#TOTAL_ORDER** protocol is deprecated. Use the default 2PC protocol instead.

Lucene Directory

The functionality to use Data Grid as a shared, in-memory index for Hibernate Search queries is now deprecated.

Custom Interceptors

The functionality to create custom interceptors with the **AdvancedCache** interface is now deprecated.

4.2. REMOVED FEATURES AND FUNCTIONALITY

Data Grid 8.0 no longer includes the following features and functionality that was either deprecated in a previous release or replaced with new components:

- Uberjars (replaced with Maven dependencies and individual JAR files)
- EAP Modules (replaced by the EAP Infinispan subsystem)
- Cassandra Cache Store
- Apache Spark Connector
- Apache Hadoop Connector
- Apache Camel component: **jboss-datagrid-camel-library** is replaced by the **camel-infinispan** component in Red Hat Fuse 7.3 and later.
- REST Cache Store
- REST API v1 (replaced by REST API v2)
- Compatibility Mode
- Distributed Execution
- CLI Cache Loader
- LevelDB Cache Store
- **infinispan-cloud** (replaced by default configuration in **infinispan-core**)
- **org.infinispan.atomic** package
- **getBulk()** methods in the **RemoteCache** API for Hot Rod clients
- JDBC PooledConnectionFactory via C3P0 and HikariCP connection pools
- OSGI support
- **infinispan.server.hotrod.workerThreads** system property
- JON Plugin