Red Hat Data Grid 8.0

Data Grid のアップグレード

Data Grid のドキュメント

Last Updated: 2023-03-12

Red Hat Data Grid 8.0 Data Grid のアップグレード

Data Grid のドキュメント

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

以前のバージョンからの移行に影響がある Data Grid 8.0 の変更を確認し、デプロイメントをアップグレードしてデータを移行する手順を実行します。

目次

第1章 RED HAT DATA GRID

Data Grid は、高性能の分散型インメモリーデータストアです。

スキーマレスデータ構造
　さまざまなオブジェクトをキーと値のペアとして格納する柔軟性があります。
グリッドベースのデータストレージ
　クラスター間でデータを分散および複製するように設計されています。
エラスティックスケーリング
　サービスを中断することなく、ノードの数を動的に調整して要件を満たします。
データの相互運用性
　さまざまなエンドポイントからグリッド内のデータを保存、取得、およびクエリーします。

1.1. DATA GRID のドキュメント

Data Grid のドキュメントは、Red Hat カスタマーポータルで入手できます。

- Data Grid 8.0 ドキュメント
- Data Grid 8.0 コンポーネントの詳細
- Data Grid 8.0 でサポートされる設定

1.2. DATA GRID のダウンロード

Red Hat カスタマーポータルで Data Grid Software Downloads にアクセスします。

注記

Data Grid ソフトウェアにアクセスしてダウンロードするには、Red Hat アカウントが必要です。

第2章 DATA GRID 8 0 への移行

以前のリリースからの移行に影響する Data Grid 8.0 の変更を確認します。

2.1. DATA GRID 8.0 サーバー

8.0 より、Data Grid サーバーは Red Hat JBoss Enterprise Application Platform (EAP) をベースとしなくなり、起動時間がはるかに速く、軽量化され、よりセキュアになりました。

Data Grid サーバーは、設定に **$RHDG_HOME/server/conf/infinispan.xml** を使用します。

データストアの設定

キャッシュ定義を使用して Data Grid がデータを保存する方法を設定します。デフォルトでは、Data Grid サーバーには、キャッシュ定義の作成、設定、および管理を可能にする Cache Manager 設定が含まれています。

```
<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 default という名前の Cache Manager を作成します。

2 **metrics** エンドポイントを介して Cache M anager の統計情報をエクスポートします。

3 Data Grid サーバーが自動的に相互のクラスターを検出し、フォームクラスターを検出できるようにする JGroups クラスタートランスポートを追加します。

4 クラスタートラフィックにデフォルトの TCP スタックを使用します。

上記の設定では、キャッシュ定義はありません。8.0 サーバーを起動すると、デフォルトの Cache Manager がインスタンス化されるので、CLI、REST API、またはリモート Hot Rod クライアントから実行時にキャッシュ定義を作成することができます。

注記

Data Grid サーバーは、EAP をベースとした以前のバージョンと同じようにドメインモードを提供しなくなりました。ただし、Data Grid サーバーはクラスターリング機能を備えたデフォルト設定を提供し、データをすべてのノードにレプリケートします。

サーバー設定

Data Grid 8.0 では、**infinispan.xml** に Data Grid サーバーに固有の設定を定義する **server** 要素を追加しています。

```
<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) **127.0.0.1** ループバックアドレスを使用するデフォルトのパブリックインターフェイスを作成します。

(2) パブリックインターフェイスをポート **11222** にバインドするデフォルトのソケットバインディングを作成します。

(3) Memcached コネクターのソケットバインディングを作成します。Memcached エンドポイントが非推奨になりました。

(4) プロパティーファイルを使用して認証情報および RBAC 設定を定義するデフォルトのセキュリティーレルムを定義します。

(5) **127.0.0.1:11222** で Hot Rod と REST のエンドポイントを公開します。

重要

REST エンドポイントは、Data Grid コマンドラインインターフェイス (CLI) とコンソールが使用する管理操作を処理します。このため、REST エンドポイントを無効にしないでください。

表2.1 チートシート

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

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

- カスタム UDP/TCP アドレスを以下のように使用します。
  -Djgroups.udp.address=172.18.1.13
  -Djgroups.tcp.address=172.18.1.1
- 以下のように JMX を有効にします。

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

(1) キャッシュマネージャーの統計を有効にします。これがデフォルトです。

(2) JMX MBean をエクスポートします。

参照資料

- 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 キャッシュ

OpenShift の Cache サービスを除き、Data Grid はデフォルトで空のキャッシュコンテナーを提供します。Data Grid 8.0 を起動すると、キャッシュマネージャーがインスタンス化されるため、実行時にキャッシュを作成できます。

Data Grid 8 0 では、**CacheContainerAdmin** API を介して作成するキャッシュ定義は、クラスターの再起動後も存続することを保証するために永続的です。

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

1 デフォルトの動作を変更し、一時的なキャッシュを作成する **VOLATILE** フラグが含まれます。

2 myTemporaryCache という名前のキャッシュを返すか、**DIST_SYNC** 設定テンプレートを使用して作成します。

注記

AdminFlag.PERMANENT はデフォルトで有効になっており、キャッシュ定義が再起動後も存続するようになっています。データが再起動後も存続するためには、データグリッドに永続ストレージを個別に追加する必要があります。次に例を示します。

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

キャッシュ設定テンプレート

以下のようにキャッシュ設定テンプレートの一覧を取得します。

- CLI で **Tab** のオートコンプリートを使用します。

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

- REST API を使用します。

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

2.3. キャッシュの作成

キャッシュ定義を Data Grid に追加し、データの格納方法を設定します。

ライブラリーモード

以下の例では、Cache Manager を初期化し、分散された同期キャッシュモードを使用する myDistributedCache という名前のキャッシュ定義を作成します。

```
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());
```

getOrCreate() メソッドを使用してキャッシュ定義を作成するか、すでに存在する場合はこれを返すこともできます。以下に例を示します。

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

Data Grid Server

以下のようにランタイム時にキャッシュをリモートで作成します。

- CLI の使用
  DIST_SYNC キャッシュテンプレートで myCache という名前のキャッシュを作成するには、以下を実行します。

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

- REST API を使用します。
  myCache という名前のキャッシュを作成するには、以下の **POST** 呼び出しを使用し、XML または JSON 形式のリクエストペイロードにキャッシュ定義を追加します。

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

- Hot Rod クライアントを使用します。

  ```
  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");
        }
  ```

Hot Rod Java クライアントでキャッシュを作成する例は、Data Grid チュートリアルを参照してください。

2.4. キャッシュヘルスステータス

これで、Data Grid はキャッシュの可用性のために以下のいずれかを返すようになりました。

HEALTHY はキャッシュが想定どおりに動作していることを意味します。
HEALTHY_REBALANCING は、キャッシュがリバランス状態であるが、想定どおりに動作していることを意味します。
DEGRADED は、キャッシュが期待どおりに動作しておらず、トラブルシューティングが必要な可能性があることを示します。

2.5. マーシャリング機能

本リリースでは、Data Grid のデフォルトのマーシャラーは ProtoStream です。これは、言語に依存しない後方互換性形式である Protocol Buffers としてデータをマーシャリングします。

ProtoStream を使用するには、Data Grid には以下を含むシリアル化コンテキストが必要です。

- Java オブジェクトの構造化表現を Protobuf メッセージタイプとして提供する **.proto** スキーマ。
- Java オブジェクトを Protobuf 形式にエンコードするための Marshaller の実装。

Data Grid は ProtoStream ライブラリーとの直接統合を提供し、シリアル化コンテキストを初期化するために必要なものをすべて生成できます。

重要

キャッシュストアの Data Grid は、ProtoStream マーシャラーと互換性のないバイナリー形式でデータを格納します。データを移行するには **StoreMigrator** ユーティリティーを使用する必要があります。

- Data Grid ライブラリーモードには、デフォルトで JBoss マーシャリングが含まれていません。**infinispan-jboss-marshalling** 依存関係をクラスパスに追加します。
- Data Grid サーバーは JBoss マーシャリングをサポートしますが、クライアントは以下の Hot Rod クライアント設定のように使用するマーシャラーを宣言する必要があります。
  .marshaller("org.infinispan.jboss.marshalling.core.JBossUserMarshaller");
- Spring インテグレーションは、デフォルトの ProtoStream マーシャラーをサポートしていません。このため、Java Serialization Marshaller を使用する必要があります。
- Java Serialization Marshaller を使用するには、クラスを非シリアル化ホワイトリストに追加する必要があります。

参照資料

- Data Grid Marshalling
- ProtoStream Marshalling
- Creating Context Initializers

- JBoss Marshalling
- Data Grid Spring Boot スターター
- Java Serialization Marshaller
- Adding Java Classes to Deserialization White Lists

2.6. DATA GRID の設定:

新規および変更された要素と属性

- **stack** は、インライン JGroups スタック定義のサポートを追加します。
- **stack.combine** 属性と **stack.position** 属性を使用すると、JGroups スタック定義をオーバーライドおよび変更できます。
- **metric** 使用すると、Data Grid が Eclipse Micro Profile Metrics API と互換性のあるメトリックをエクスポートする方法を設定できます。
- **context-initializer** を使用すると、ユーザータイプの Protostream ベースのマーシャラーを初期化する **SerializationContextInitializer** 実装を指定できます。
- **key-transformers** を使用すると、Lucene でインデックスを作成するためにカスタムキーを文字列に変換するトランスフォーマーを登録できます。
- **statistics** はデフォルトで false になりました。

非推奨の要素と属性

次の要素と属性は非推奨になりました。

- **off-heap** 要素の **address-count** 属性。
- **transaction** 要素の **protocol** 属性。
- **jmx** 要素の **duplicate-domains** 属性。
- **advanced-externalizer**
- **custom-interceptors**
- **state-transfer-executor**
- **transaction-protocol**

注記

代替可能な項目については、Configuration Schema を参照してください。

削除された要素と属性

次の要素と属性は以前のリリースで非推奨になり、現在は削除されています。

- **deadlock-detection-spin**

- **compatibility**
- **write-skew**
- **versioning**
- **data-container**
- **eviction**
- **eviction-thread-policy**

参照資料

- Data Grid 設定スキーマ
- Data Grid Configuration Guide

2.7. 永続性

7.1 などの以前のバージョンの Data Grid と比較すると、キャッシュストア設定が変更されています。キャッシュストアの定義には、以下が必要です。

- **persistence** 要素内に含まれます。
- **xlmns** namespace を含めます。

本リリースでは、キャッシュストアの設定は以下のようになります。

- キャッシュストアの実装がセグメントをサポートする場合は、デフォルトで **segmented="true"** に設定されます。
- **store** 要素の **singleton** 属性を削除します。代わりに **shared=true** を使用してください。

JDBC String ベースのキャッシュストアは、Agroal をベースとした接続ファクトリーを使用してデータベースに接続します。**c3p0.properties** および **hikari.properties** ファイルを使用できなくなりました。

同様に、デフォルトのセグメンテーションを使用する JDBC String-Based キャッシュストア設定には、**segmentColumnName** および **segmentColumnType** パラメーターが含まれている必要があります。

MySQL の例

```
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");
```

参照資料

- Data Grid 設定スキーマ
- 永続ストレージのセットアップ
- セグメント化されたキャッシュストア
- JDBC 文字列ベースのキャッシュストア

2.8. REST API

以前のバージョンの Data Grid REST API は v1 であり、これは REST API v2 に置き換えられました。

デフォルトのコンテキストパスは **127.0.0.1:11222/rest/v2/** になります。REST API v2 を使用するには、クライアントまたはスクリプトを更新する必要があります。

参照資料

- Data Grid REST API

2.9. HOT ROD クライアント認証

Hot Rod クライアントは、**DIGEST-MD5** ではなく **SCRAM-SHA-512** をデフォルトの認証メカニズムとして使用するようになりました。

注記

プロパティーセキュリティーレルムを使用する場合は、**PLAIN** 認証メカニズムを使用する必要があります。

参照資料

- Configuring Authentication Mechanisms for Hot Rod Java Clients

2.10. MAVEN で利用可能な JAVA ディストリビューション

Data Grid は、Data Grid サーバーのディストリビューションを除いて、Maven リポジトリー外に Java アーティファクトを提供しなくなりました。Data Grid Library、Hot Rod Java クライアント、および **StoreMigrator** などのユーティリティーに必要な依存関係を追加する方法は、関連するドキュメントを参照してください。

参照資料

- Data Grid Library Mode
- Hot Rod Java クライアントガイド
- Store Migrator の取得

2.11. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM (EAP) モジュール

Data Grid は、EAP で実行されるアプリケーションのモジュールを提供しなくなりました。代わりに、EAP は今後のリリースで Data Grid との直接統合を提供します。

ただし、EAP が **infinispan** サブシステムを処理する機能を提供するまで、EAP デプロイメントに Data Grid 8.0 アーティファクトをパッケージ化する必要があります。

2.12. 非推奨の機能

非推奨となったリリース以降では、非推奨の機能のサポートは利用できません。

重要

Red Hat では、新規デプロイメントでの非推奨の機能の追加、有効化、設定は推奨していません。

2.12.1. 非推奨

Data Grid 8.0 では、以下の機能とが非推奨になりました。

Memcached エンドポイントコネクター

本リリースでは、Data Grid は Memcached エンドポイントをサポートしなくなりました。Memcached コネクターは非推奨であり、今後のバージョンで削除される予定です。

注記

Memcached コネクターにユースケースまたは要件がある場合は、Red Hat サポートチームに問い合わせ、Memcached コネクターの今後の Data Grid 実装の要件を確認してください。

JBoss Marshalling

JBoss Marshalling はシリアル化ベースのマーシャリングライブラリーであり、以前の Data Grid バージョンではデフォルトのマーシャラーでした。Data Grid ではシリアル化ベースのマーシャリングを使用するべきではなく、代わりに Protostream を使用してください。こちらは、後方互換性を保証する高パフォーマンスのバイナリーネットワーク形式です。

externalizers

以下のインターフェイスおよびアノテーションが非推奨になりました。

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

注記

Data Grid は、JBoss Marshalling を使用せずにデータを永続化したときに **AdvancedExternalizer** 実装を無視します。

合計受注トランザクションプロトコル

org.infinispan.transaction.TransactionProtocol#TOTAL_ORDER プロトコルは非推奨になりました。代わりにデフォルトの 2PC プロトコルを使用してください。

Lucene ディレクトリー

Data Grid を、Hibernate Search クエリーの共有のインメモリーインデックスとして使用する機能は非推奨になりました。

カスタムインターセプター

AdvancedCache インターフェイスを使用してカスタムインターセプターを作成する機能が非推奨になりました。

2.12.2. 削除された機能

Data Grid 8.0 には、以前のリリースで非推奨となった以下の機能、または新しいコンポーネントに置き換えられた機能が含まれていません。

- Uberjars(Maven 依存関係および個別の JAR ファイルに置き換え)
- EAP モジュール (EAP Infinispan サブシステムによって置き換え)
- Cassandra キャッシュストア
- Apache Spark コネクター
- Apache Hadoop コネクター
- Red Hat Fuse 7.3 以降では Apache Camel コンポーネント **jboss-datagrid-7.3-camel-library** に代わる **camel-infinispan** コンポーネントが提供されます。
- REST キャッシュストア
- REST API v1(REST API v2 に置き換え)
- 互換性モード
- 分散実行
- CLI キャッシュローダー
- LevelDB キャッシュストア
- **infinispan-cloud** (**infinispan-core** のデフォルト設定)
- **org.infinispan.atomic** パッケージ
- Hot Rod クライアントの **RemoteCache** API での **getBulk()** メソッド
- C3P0 および HikariCP 接続プールを介した JDBC PooledConnectionFactory
- OSGI サポート
- **infinispan.server.hotrod.workerThreads** システムプロパティー
- JON プラグイン

第3章 DATA GRID サーバーのローリングアップグレードの実行

Data Grid クラスターのローリングアップグレードを実行して、ダウンタイムやデータの損失なしにバージョン間で変更します。ローリングアップグレードは、Hot Rod 経由で Data Grid サーバーおよびデータの両方をターゲットバージョンに移行します。

3.1. ターゲットクラスターの設定

ターゲット Data Grid バージョンを実行し、リモートキャッシュストアを使用してソースクラスターからデータを読み込むクラスターを作成します。

前提条件

- ターゲットアップグレードバージョンとともに Data Grid クラスターをインストールします。

重要

ターゲットクラスターのネットワークプロパティーはソースクラスターのネットワークプロパティーが重複していないことを確認します。JGroups トランスポート設定でターゲットおよびソースクラスターの一意の名前を指定する必要があります。環境に応じて、異なるネットワークインターフェイスを使用し、ターゲットクラスターとソースクラスターを分離するためにポートオフセットを指定することもできます。

手順

1. ソースクラスターから移行する各キャッシュについて、ターゲットクラスターに **aRemoteCacheStore** を追加します。
   リモートキャッシュストアは Hot Rod プロトコルを使用して、リモート Data Grid クラスターからデータを取得します。リモートキャッシュストアをターゲットクラスターに追加する場合は、ソースクラスターからデータをレイジーに読み込み、クライアント要求を処理します。

2. すべての要求の処理を開始するために、クライアントをターゲットクラスターに切り替えます。

   a. クライアント設定をターゲットクラスターの場所で更新します。

   b. クライアントを再起動します。

3.1.1. ローリングアップグレードのリモートキャッシュストア

以下のようにローリングアップグレードを実行するには、特定のリモートキャッシュストア設定を使用する必要があります。

```
<persistence passivation="false"> 1
  <remote-store xmlns="urn:infinispan:config:store:remote:10.1"
                cache="myDistCache" 2
                protocol-version="2.5" 3
                hotrod-wrapping="true" 4
                raw-values="true"> 5
    <remote-server host="127.0.0.1" port="11222"/> 6
  </remote-store>
</persistence>
```

1. パッシベーションを無効にします。ローリングアップグレードのリモートキャッシュストアは、パッシベーションを無効にする必要があります。
2. ソースクラスター内のキャッシュの名前と一致します。ターゲットクラスターは、リモートキャッシュストアを使用してこのキャッシュからデータを読み込みます。
3. ソースクラスターの Hot Rod プロトコルバージョンと一致します。**2.5** は最小バージョンであり、アップグレードパスに適しています。別の Hot Rod でバージョンを設定する必要はありません。
4. エントリーが Hot Rod プロトコルに適した形式でラップされるようにします。
5. データを raw 形式でリモートキャッシュストアに保存します。これにより、クライアントはリモートキャッシュストアから直接データを使用できるようになります。
6. ソースクラスターの場所を参照します。

参照資料

- リモートキャッシュストア設定スキーマ
- RemoteStore
- RemoteStoreConfigurationBuilder

3.2. ターゲットクラスターへのデータの同期

ターゲットクラスターがリモートキャッシュストアを使用してクライアント要求を実行し、オンデマンドでデータを読み込む場合は、ソースクラスターからターゲットクラスターにデータを同期できます。

この操作はソースクラスターからデータを読み取り、ターゲットクラスターに書き込みます。データは、ターゲットクラスターのすべてのノードに並行して移行され、各ノードはデータのサブセットを受け取ります。Data Grid 設定で、各キャッシュの同期を実行する必要があります。

手順

1. ターゲットクラスターに移行する Data Grid 設定の各キャッシュの同期操作を開始します。Data Grid REST API を使用し、**?action=sync- data** パラメーターで **GET** リクエストを呼び出します。たとえば、myCache という名前のキャッシュ内のデータをソースクラスターからターゲットクラスターに同期するには、以下を実行します。

```
GET /v2/caches/myCache?action=sync-data
```

   操作が完了すると、Data Grid はターゲットクラスターにコピーされたエントリーの合計数で応答します。

   または、**RollingUpgradeManager** MBean で **synchronizeData(migratorName=hotrod)** を呼び出すことで JMX を使用できます。

2. ターゲットクラスター内の各ノードをソースクラスターから切断します。たとえば、ソースクラスターから myCache キャッシュを切断するには、以下の **POST** 要求を呼び出します。

```
GET /v2/caches/myCache?action=disconnect-source
```

JMX を使用するには、**RollingUpgradeManager** MBean で **disconnectSource(migratorName=hotrod)** を呼び出します。

次のステップ

ソースクラスターからすべてのデータを同期した後、ローリングアップグレードプロセスが完了しました。ソースクラスターの使用を停止できるようになりました。

第4章 キャッシュストア間のデータの移行

Data Grid は、キャッシュストア間で永続化されたデータを移行するための Java ユーティリティーを提供します。

Data Grid をアップグレードする場合、メジャーバージョン間の機能相違点は、キャッシュストア間の後方互換性を許可しません。**StoreMigrator** を使用してデータを変換し、ターゲットバージョンとの互換性を持つことができます。

たとえば、Data Grid 8.0 にアップグレードすると、デフォルトのマーシャラーが Protostream に変更されます。以前の Data Grid バージョンでは、キャッシュストアはバイナリー形式を使用し、マーシャリングする変更との互換性がありません。つまり、Data Grid 8.0 は、以前の Data Grid バージョンでキャッシュストアから読み込むことができません。

他の場合は、Data Grid のバージョンが、JDBC Mixed および Binary ストアなどのキャッシュストア実装を非推奨または削除します。このような場合は、**StoreMigrator** を使用して異なるキャッシュストア実装に変換できます。

4.1. キャッシュストアマイグレーション

Data Grid は、最新の Data Grid キャッシュストア実装のデータを再作成する **StoreMigrator.java** ユーティリティーを提供します。

StoreMigrator は以前のバージョンの Data Grid のキャッシュストアを取得し、キャッシュストア実装をターゲットとして使用します。

StoreMigrator を実行すると、**EmbeddedCacheManager** インターフェイスを使用して定義したキャッシュストアタイプでターゲットキャッシュが作成されます。**StoreMigrator** は、ソースストアからメモリーにエントリーを読み込み、それらをターゲットキャッシュに配置します。

StoreMigrator を使用すると、あるタイプのキャッシュストアから別のストアにデータを移行することもできます。たとえば、JDBC String ベースのキャッシュストアから Single File キャッシュストアに移行することができます。

重要

StoreMigrator は、セグメント化されたキャッシュストアから以下にデータを移行できません。

- 非セグメント化されたキャッシュストア。
- セグメント数が異なるセグメント化されたキャッシュストア。

4.2. STORE MIGRATOR の取得

StoreMigrator は、Data Grid ツールライブラリー **infinispan-tools** の一部として利用でき、Maven リポジトリーに含まれます。

手順

- **StoreMigrator** の **pom.xml** を以下のように設定します。

```
<?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>
```

4.3. ストア移行の設定

ソースおよびターゲットのキャッシュストアのプロパティーを **migrator.properties** ファイルに設定します。

手順

1. **migrator.properties** ファイルを作成します。
2. ソースキャッシュストアを **migrator.properties** に設定します。
   a. 以下の例にあるように、すべての設定プロパティーの先頭に **source.** を追加します。

```
source.type=SOFT_INDEX_FILE_STORE
source.cache_name=myCache
source.location=/path/to/source/sifs
```

3. **migrator.properties** でターゲットキャッシュストアを設定します。
   a. 以下の例のように、すべての設定プロパティーの先頭に **target.** を付けます。

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

4.3.1. 移行プロパティーの保存

ソースおよびターゲットのキャッシュストアを **StoreMigrator** プロパティーで設定します。

表4.1 キャッシュストアタイププロパティー

プロパティー	説明	必須/オプション
type	ソースまたはターゲットのキャッシュストアタイプのタイプを指定します。 **.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**	必須

表4.2 一般的なプロパティー

プロパティー	説明	値の例	必須/オプション
cache_name	ストアがバックアップするキャッシュに名前を付けます。	**.cache_name=myCache**	必須

プロパティー	説明	値の例	必須/オプション
segment_count	セグメンテーションを使用できるターゲットキャッシュストアのセグメント数を指定します。 セグメント数は、Data Grid 設定の **clustering.hash.numSegments** と一致する必要があります。 つまり、キャッシュストアのセグメント数は、対応するキャッシュのセグメント数と一致する必要があります。セグメントの数が同一でない場合、Data Grid はキャッシュストアからデータを読み込めません。	**.segment_count=256**	Optional

表4.3 JDBC プロパティー

プロパティー	説明	必須/オプション
dialect	基礎となるデータベースのダイアレクトを指定します。	必須
version	ソースキャッシュストアのマーシャラーバージョンを指定します。以下のいずれかの値を設定します。 * Data Grid 7.2.x の場合は **8** * Data Grid 7.3.x の場合は **9** * Data Grid 8.x の場合は **10**	ソースストアにのみ必要です。 例: **source.version=9**
marshaller.class	カスタムマーシャラークラスを指定します。	カスタムマーシャラーを使用する場合に必要です。
marshaller.externalizers	**[id]:<Externalizer class>** 形式で読み込むカスタム **AdvancedExternalizer** 実装のコンマ区切りリストを指定します。	Optional

プロパティー	説明	必須/オプション
connection_pool.connection_url	JDBC 接続 URL を指定します。	必須
connection_pool.driver_class	JDBC ドライバーのクラスを指定します。	必須
connection_pool.username	データベースユーザー名を指定します。	必須
connection_pool.password	データベースユーザー名のパスワードを指定します。	必須
db.major_version	データベースのメジャーバージョンを設定します。	Optional
db.minor_version	データベースのマイナーバージョンを設定します。	Optional
db.disable_upsert	データベース upsert を無効にします。	Optional
db.disable_indexing	テーブルインデックスが作成されるかどうかを指定します。	Optional
table.string.table_name_prefix	テーブル名の追加接頭辞を指定します。	Optional
table.string.<id|data|timestamp>.name	列名を指定します。	必須
table.string.<id|data|timestamp>.type	列タイプを指定します。	必須
key_to_string_mapper	**TwoWayKey2StringMapper** クラスを指定します。	Optional

注記

Binary キャッシュストアから古い Data Grid バージョンの移行には、以下のプロパティーで **table.string.*** を **table.binary.*** に変更します。

- **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
```

表4.4 RocksDB プロパティー

プロパティー	説明	必須/オプション
location	データベースディレクトリーを設定します。	必須
圧縮	使用する圧縮タイプを指定します。	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
```

表4.5 SingleFileStore プロパティー

プロパティー	説明	必須/オプション
location	キャッシュストア **.dat** ファイルが含まれるディレクトリーを設定します。	必須

```
# 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
```

表4.6 SoftIndexFileStore プロパティー

プロパティー	説明	値
必須/オプション	**location**	データベースディレクトリーを設定します。
必須	**index_location**	データベースインデックスディレクトリーを設定します。

```
# 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. キャッシュストアの移行

StoreMigrator を実行して、あるキャッシュストアから別のキャッシュストアにデータを移行します。

前提条件

- **infinispan-tools.jar** を取得します。
- ソースおよびターゲットのキャッシュストアを設定する **migrator.properties** ファイルを作成します。

手順

- ソースから **infinispan-tools.jar** をビルドする場合は、以下を実行します。
  1. JDBC ドライバーなどのソースおよびターゲットのデータベースの **infinispan-tools.jar** および依存関係をクラスパスに追加します。
  2. **migrator.properties** ファイルを **StoreMigrator** の引数として指定します。
- Maven リポジトリーから **infinispan-tools.jar** をプルする場合は、以下のコマンドを実行します。
  mvn exec:java