



Red Hat Data Grid 8.5

Data Grid 8 への移行

デプロイメントとアプリケーションの Data Grid 8 への移行

Red Hat Data Grid 8.5 Data Grid 8 への移行

デプロイメントとアプリケーションの Data Grid 8 への移行

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

以前のバージョンから Red Hat Data Grid 8 に移行します。本ガイドでは、Data Grid 8 デプロイメントにおけるアーキテクチャー上の相違点の概要に加えて、これらのバージョン間での変更点について詳しく説明します。

目次

RED HAT DATA GRID	3
DATA GRID のドキュメント	4
DATA GRID のダウンロード	5
多様性を受け入れるオープンソースの強化	6
第1章 DATA GRID 8	7
1.1. DATA GRID 8 への移行	7
1.2. 移行パス	7
1.3. コンポーネントのダウンロード	7
第2章 DATA GRID SERVER デプロイメントの移行	10
2.1. DATA GRID SERVER 8	10
2.2. DATA GRID SERVER の設定	10
2.3. DATA GRID SERVER 8.2 設定スキーマへの変更	13
2.4. DATA GRID SERVER 8.3 設定スキーマへの変更	15
2.5. DATA GRID SERVER 8.4 設定スキーマへの変更	16
2.6. DATA GRID SERVER 8.5 設定スキーマへの変更	17
2.7. DATA GRID SERVER のエンドポイントとネットワーク設定	18
2.8. DATA GRID SERVER のセキュリティー	21
2.9. DATA GRID SERVER エンドポイントの分離	25
2.10. DATA GRID SERVER 共有データソース	27
2.11. DATA GRID SERVER の JMX とメトリクス	27
2.12. DATA GRID SERVER のチートシート	28
第3章 DATA GRI 設定の移行	31
3.1. DATA GRID キャッシュの設定	31
3.2. エビクション設定	38
3.3. 有効期限の設定	42
3.4. 永続キャッシュストア	43
3.5. DATA GRID クラスタートランスポート	47
3.6. DATA GRID 認証	49
第4章 DATA GRID 8 API への移行	54
4.1. REST API	54
4.2. クエリー API	54
第5章 アプリケーションの DATA GRID 8 への移行	62
5.1. DATA GRID 8 でのマーシャリング	62
5.2. AUTOPROTOSCHEMABUILDER アノテーションへのアプリケーションの移行	65
第6章 RED HAT OPENSIFT での DATA GRID クラスターの移行	72
6.1. OPENSIFT の DATA GRID	72
6.2. OPENSIFT 上の DATA GRID 8.2	73
6.3. OPENSIFT 上の DATA GRID 8.3	74
6.4. OPENSIFT 上の DATA GRID 8.4	74
6.5. OPENSIFT 上の DATA GRID 8.5	74
第7章 キャッシュストア間のデータの移行	75
7.1. キャッシュストアマイグレーター	75
7.2. キャッシュストアマイグレーターの設定	75
7.3. DATA GRID キャッシュストアの移行	81

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ルリング

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。

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 8.5 ドキュメント](#)
- [Data Grid 8.5 コンポーネントの詳細](#)
- [Data Grid 8.5 でサポートされる構成](#)
- [Data Grid 8 機能のサポート](#)
- [Data Grid で非推奨の機能](#)

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。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、用語の置き換えは、今後の複数のリリースにわたって段階的に実施されます。詳細は、[弊社の CTO、Chris Wright のメッセージ](#) を参照してください。

第1章 DATA GRID 8

簡単な概要といくつかの基本事項を見て、Data Grid 8 への移行を始めましょう。

1.1. DATA GRID 8 への移行

Data Grid 8 では、サーバーデプロイメント用のまったく新しいアーキテクチャーなど、以前の Data Grid バージョンからの大幅な変更が導入されています。

これにより、既存の環境では移行の特定の側面がより困難になりますが、データグリッドチームは、これらの変更がデプロイメントの複雑さと管理オーバーヘッドを削減することでユーザーに利益をもたらすと考えています。

以前のバージョンと比較して、Data Grid 8 に移行すると、次のことが可能になります。

- コンテナプラットフォーム用に構築されたクラウドネイティブデザイン。
- メモリーフットプリントが軽くなり、全体的なリソース使用量が少なくなります。
- より速い開始時間。
- 攻撃対象領域を小さくすることでセキュリティを強化します。
- Red Hat テクノロジーおよびソリューションとのより良い統合。

また、Data Grid 8 は、実績のある信頼できるオープンソーステクノロジーから構築された、可能な限り最高のメモリー内データストレージ機能を提供し続けます。

1.2. 移行パス

本ドキュメントは、Data Grid 7.3 から Data Grid 8 への移行に焦点を当てていますが、7.0.1 以降の 7.x バージョンにも引き続き適用できます。

Data Grid 6 からの移行を計画している場合、このドキュメントでは必要なものがすべて網羅されていない可能性があります。移行する前に、デプロイメントに固有のアドバイスについて Red Hat サポートに連絡する必要があります。

いつものように、このドキュメントを改善することでお手伝いできるかどうかお知らせください。

1.3. コンポーネントのダウンロード

Data Grid 8 の使用を開始するには、次のいずれかを行います。

- ベアメタルまたはその他のホスト環境に Data Grid をインストールする場合は、Red Hat カスタマーポータルからコンポーネントをダウンロードします。
- OpenShift で実行している場合は、Data Grid Operator サブスクリプションを作成します。

次の情報は、ベアメタルデプロイメントで利用可能なコンポーネントのダウンロードについて説明しています。これは、以前のバージョンの Data Grid とは異なります。

以下も参照してください。

- [Data Grid on OpenShift Migration](#)

- [データグリッド 8 でサポートされている設定](#)

Maven リポジトリ

Data Grid 8 は、以下のコンポーネントの Red Hat カスタマーポータルからの個別のダウンロードを提供しなくなりました。

- 以前のバージョンではライブラリーモードと呼ばれていた、カスタムアプリケーションで埋め込みキャッシュを作成するための Data Grid コアライブラリー。
- Hot Rod Java クライアント。
- **StoreMigrator** などのユーティリティー。

これらのコンポーネントをダウンロードとして利用できるようにする代わりに、Data Grid は Maven リポジトリを介して Java アーティファクトを提供します。この変更は、Maven を使用して依存関係を一元管理できることを意味します。これにより、プロジェクト間の依存関係をより適切に制御できます。

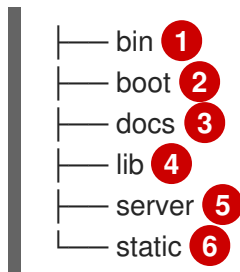
顧客ポータルから Data Grid Maven リポジトリをダウンロードするか、パブリック Red Hat Enterprise Maven リポジトリから Data Grid 依存関係をプルできます。両方の方法の説明は、データグリッドのドキュメントに記載されています。

- [Data Grid Maven リポジトリの設定](#)

データグリッドサーバー

Data Grid Server は、ダウンロードしてホストファイルシステムに抽出できるアーカイブとして配布されます。

アーカイブ配布には、次の最上位フォルダーが含まれています。



- 1 データグリッドサーバーとデータグリッドコマンドラインインターフェイス (CLI) を起動および管理するためのスクリプト。
- 2 ブートライブラリー。
- 3 Data Grid Server の設定と実行に役立つリソース。
- 4 Data Grid Server のランタイムライブラリー。このディレクトリーは、カスタムコードライブラリーではなく、内部コードのみを対象としていることに注意してください。
- 5 Data Grid Server インスタンスのルートディレクトリー。
- 6 Data Grid Console コンソールの静的リソース。

server ディレクトリーは、Data Grid Server インスタンスのルートディレクトリーで、カスタムコードライブラリー、設定ファイル、およびデータのサブディレクトリーが含まれています。

ファイルシステムとディストリビューションの内容の詳細については、[データグリッドサーバーガイド](#)を参照してください。

- [Data Grid サーバーファイルシステム](#)
- [Data Grid Server README](#)

Source code

各 Data Grid リリースのコンパイルされていないソースコード。

第2章 DATA GRID SERVER デプロイメントの移行

このセクションの詳細を確認して、Data Grid サーバーの正常な移行を計画および準備します。

2.1. DATA GRID SERVER 8

Data Grid Server 8 は次のとおりです。

- 最新のシステムアーキテクチャー向けに設計されています。
- コンテナプラットフォーム用に構築されています。
- Quarkus を使用したネイティブイメージのコンパイル用に最適化されています。

クラウドネイティブアーキテクチャーへの移行は、Data Grid Server 8 が Red Hat JBoss Enterprise Application Platform (EAP) に基づいていないことを意味します。代わりに、Data Grid Server 8 は [Netty](#) プロジェクトのクライアント/サーバーフレームワークに基づいています。

EAP との統合によって提供された機能の多くは、Data Grid 8 に関連しなくなったか、変更されたため、この変更は以前のバージョンからの移行に影響します。

たとえば、サーバー設定の複雑さは以前のリリースと比較して大幅に軽減されていますが、既存の設定を新しいスキーマに適合させる必要があります。また、Data Grid 8 は、はるかにきめ細かい設定を実現できた以前のバージョンよりも、サーバー設定に関する規則を提供します。さらに、Data Grid Server はドメインモードを利用して設定を一元管理しなくなりました。

Data Grid チームは、これらの設定変更により、既存のクラスターを Data Grid 8 に移行するためお客様に追加の労力がかかることを認識しています。

Red Hat OpenShift などのコンテナオーケストレーションプラットフォームを使用して、Red Hat Ansible などの自動化エンジンとともにデータグリッドクラスターをプロビジョニングおよび管理し、データグリッド設定を管理することを推奨します。これらのテクノロジーは、データグリッドに固有のソリューションではなく、より汎用的で複数の異種システムに適しているという点で、より高い柔軟性を提供します。

Data Grid 8 への移行に関しては、Red Hat Ansible のようなソリューションが大規模な設定のデプロイメントに役立つことは注目に値します。ただし、そのツールは、既存のデータグリッド設定の実際の移行を必ずしも支援するとは限りません。

2.2. DATA GRID SERVER の設定

Data Grid は、利用可能なコンピューティングリソースをインテリジェントかつ効率的に利用できるようにするスケーラブルなデータレイヤーを提供します。Data Grid Server のデプロイメントでこれを実現するために、設定は動的と静的の2つのレイヤーに分けられます。

動的設定

動的設定は変更可能であり、実行時にキャッシュを作成し、クラスターにノードを追加したり、クラスターからノードを削除したりすると変更されます。

Data Grid Server クラスターをデプロイした後、Data Grid CLI、Data Grid Console、または Hot Rod および REST エンドポイントを介してキャッシュを作成します。Data Grid Server は、ノード間で分散されるクラスター状態の一部として、これらのキャッシュを永続的に保存します。参加している各ノードは、変更が発生すると Data Grid Server がすべてのノード間で自動的に同期する完全なクラスター状態を受け取ります。

静的設定

静的設定は不変であり、実行時に変更されません。

静的設定は、クラスタートランスポート、認証と暗号化、共有データソースなどの基盤となるメカニズムを設定するときに定義します。

デフォルトでは、Data Grid Server は静的設定に `$RHDG_HOME/server/conf/infinispan.xml` を使用します。

設定のルート要素は **infinispan** であり、次の2つの基本スキーマを宣言します。

```
<infinispan
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:infinispan:config:14.0 https://infinispan.org/schemas/infinispan-config-14.0.xsd
                      urn:infinispan:server:14.0 https://infinispan.org/schemas/infinispan-server-14.0.xsd"
  xmlns="urn:infinispan:config:14.0"
  xmlns:server="urn:infinispan:server:14.0">
```

- **urn:infinispan:config** スキーマは、キャッシュコンテナなどのコア Infinispan 機能の設定を検証します。
- **urn:infinispan:server** スキーマは、Data Grid Server の設定を検証します。

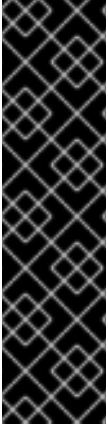
キャッシュコンテナの設定

cache-container 要素を使用して、キャッシュライフサイクルを管理するメカニズムを提供する **CacheManager** インターフェイスを設定します。

```
<!-- Creates a cache manager named "default" that exports statistics. -->
<cache-container name="default"
  statistics="true">
  <!-- Defines cluster transport properties, including the cluster name. -->
  <!-- Uses the default TCP stack for inter-cluster communication. -->
  <transport cluster="{infinispan.cluster.name}"
    stack="{infinispan.cluster.stack:tcp}"
    node-name="{infinispan.node.name:}"/>
</cache-container>
```

cache-container 要素は、次の設定要素も保持できます。

- キャッシュマネージャーの **security**。
- MicroProfile 互換性メトリクスの **metrics**。
- JMX の監視と管理のための **jmx**。



重要

以前のバージョンでは、データグリッド設定で複数の **cache-container** 要素を定義して、さまざまなエンドポイントでキャッシュコンテナを公開できました。

Data Grid 8 では、Data Grid CLI とコンソールがクラスターごとに1つのキャッシュマネージャーしか処理できないため、複数のキャッシュコンテナを設定しないでください。ただし、必要に応じて、キャッシュコンテナの名前をデフォルトよりも環境にとって意味のある名前に変更できます。

キャッシュマネージャーが相互に干渉しないように、マルチテナンシーを実現するには、個別の Data Grid クラスターを使用する必要があります。

Server configuration

server 要素を使用して、基盤となる Data Grid Server メカニズムを設定します。

```
<server>
  <interfaces>
    <interface name="public"> 1
      <inet-address value="{infinispan.bind.address:127.0.0.1}"/> 2
    </interface>
  </interfaces>

  <socket-bindings default-interface="public" 3
    port-offset="{infinispan.socket.binding.port-offset:0}"> 4
    <socket-binding name="default" 5
      port="{infinispan.bind.port:11222}"/> 6
    <socket-binding name="memcached" port="11221"/> 7
  </socket-bindings>

  <security>
    <security-realms> 8
      <security-realm name="default"> 9
        <server-identities> 10
          <ssl>
            <keystore path="application.keystore" 11
              keystore-password="password"
              alias="server"
              key-password="password"
              generate-self-signed-certificate-host="localhost"/>
          </ssl>
        </server-identities>
        <properties-realm groups-attribute="Roles"> 12
          <user-properties path="users.properties" 13
            relative-to="infinispan.server.config.path"
            plain-text="true"/> 14
          <group-properties path="groups.properties" 15
            relative-to="infinispan.server.config.path" />
        </properties-realm>
      </security-realm>
    </security-realms>
  </security>
```



```
<endpoints socket-binding="default" security-realm="default" /> 16
```

```
</server>
```

- 1 サーバーをネットワーク上で利用できるようにする "public" という名前の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に **127.0.0.1** ループバックアドレスを使用します。
- 3 パブリックインターフェイスを、Data Grid Server エンドポイントが着信クライアント接続をリスンするネットワークポートにバインドします。
- 4 ネットワークポートのオフセットを **0** に指定します。
- 5 default という名前のソケットバインディングを作成します。
- 6 ソケットバインディング用のポート **11222** を指定します。
- 7 ポート **11221** に Memcached コネクターのソケットバインディングを作成します。
- 8 エンドポイントをネットワーク侵入から保護するセキュリティーレلمを定義します。
- 9 "default" という名前のセキュリティーレلمを作成します。
- 10 ID 検証用に SSL/TLS キーストアを設定します。
- 11 サーバー証明書を含むキーストアを指定します。
- 12 プロパティーファイルを使用して、ユーザーをロールにマップするユーザーとグループを定義するように、デフォルトのセキュリティーレلمを設定します。
- 13 データグリッドユーザーを含むプロパティーファイルに名前を付けます。
- 14 **users.properties** ファイルの内容をプレーンテキストとして保存することを指定します。
- 15 データグリッドユーザーをロールにマップするプロパティーファイルに名前を付けます。
- 16 Hot Rod および REST コネクターでエンドポイントを設定します。

この例では、Data Grid 8.2 のデフォルトである暗黙の **hotrod-connector** と **rest-connector** 要素を示しています。

8.0 および 8.1 の Data Grid Server 設定は、明示的に宣言された Hot Rod および REST コネクターを使用します。

関連情報

- [Data Grid Server ガイド](#)
- [Data Grid Server リファレンス](#)

2.3. DATA GRID SERVER 8.2 設定スキーマへの変更

以前の 7.x バージョンでは、Data Grid Server 用の個別のスキーマはありませんでした。このトピックでは、8.1 から 8.2 までの Data Grid Server 設定スキーマへの変更をリスト表示します。

セキュリティー認証

Data Grid 8.2 以降、サーバー設定では、デフォルトで承認が有効になり、ロールと権限に基づいてユーザーアクセスが制限されます。

```
<cache-container name="default" statistics="true">
  <transport cluster="{infinispan.cluster.name:cluster}"
    stack="{infinispan.cluster.stack:tcp}"
    node-name="{infinispan.node.name:}" />
  <security>
    <authorization/> ❶
  </security>
</cache-container>
```

- ❶ サーバーの管理と運用、およびキャッシュマネージャーのライフサイクルの承認を有効にします。 **authorization** エレメントを削除して、セキュリティー許可を無効にすることができます。

クライアントトラストストア

Data Grid 8.2 以降、クライアントトラストストアをサーバー ID 設定に追加し、 **truststore-realm** 要素を使用して証明書を検証できます。

8.1

```
<security-realm name="default">
  <server-identities>
    <ssl>
      <keystore path="server.pfx"
        keystore-password="password" alias="server"/>
    </ssl>
  </server-identities>
  <truststore-realm path="trust.pfx" password="secret"/>
</security-realm>
```

8.2

```
<security-realm name="default">
  <server-identities>
    <ssl>
      <keystore path="server.pfx"
        keystore-password="password" alias="server"/>
      <truststore path="trust.pfx" password="secret"/> ❶
    </ssl>
  </server-identities>
  <truststore-realm/> ❷
</security-realm>
```

- ❶ クライアント証明書を保持するトラストストアを指定します。
- ❷ サーバー設定に **truststore-realm** 要素を含める場合、トラストストアにはすべてのクライアントの公開証明書が含まれている必要があります。 **truststore-realm** 要素を含めない場合には、トラストストアに必要なのは証明書チェーンのみです。

エンドポイントコネクター

Data Grid 8.2 以降、**hotrod-connector** と **rest-connector** 要素は、デフォルトの **endpoints** 設定で自動的に設定されます。

```
<endpoints socket-binding="default" security-realm="default"/>
```

変更された要素と属性

- **path**、**provider**、**keystore-password**、および **relative-to** 属性が、**truststore-realm** 要素から削除されます。
- **name** 属性が **truststore-realm** 要素に追加されます。

新しい要素と属性

- **security** 要素に追加された **credential-stores** 子要素
credential-stores 要素には、**credential-store**、**clear-text-credential**、および **credential-reference** の子要素も含まれます。
以下は、デフォルトでサーバー設定に含まれています。

```
<credential-stores>
  <credential-store name="credentials" path="credentials.pfx">
    <clear-text-credential clear-text="secret"/>
  </credential-store>
</credential-stores>
```

- **endpoints** 要素に追加された **ip-filter**、**accept**、**reject** 子要素
- **hotrod-connector** および **rest-connector** 要素に追加された **security-realm** 属性
- **cache-max-size** と **cache-lifespan** が **security-realm** 要素に追加され、ID キャッシュのサイズとエントリーの寿命を設定します。
- クライアント証明書を検証するためのトラストストアを指定するために **ssl** 要素に追加された **truststore** 子要素

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。

- **authentication** 要素の **security-realm** 属性
- **encryption** 要素の **security-realm** 属性

削除された要素と属性

Data Grid 8.2 では要素や属性は削除されません。

2.4. DATA GRID SERVER 8.3 設定スキーマへの変更

このトピックでは、8.2 から 8.3 までの Data Grid Server 設定スキーマへの変更をリスト表示します。

スキーマの変更

- **urn:infinispan:server** namespace の **endpoints** 要素は、繰り返される要素ではなくなり、0 以上の **endpoint** 要素のラッパーになります。

Data Grid Server 8.2

-

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

Data Grid Server 8.3

```
<endpoints>
  <endpoint socket-binding="public" security-realm="application-realm" admin="false">
    <hotrod-connector/>
    <rest-connector/>
  </endpoint>
  <endpoint socket-binding="private" security-realm="management-realm">
    <hotrod-connector/>
    <rest-connector/>
  </endpoint>
</endpoints>
```

変更された要素と属性

Data Grid 8.3 では、要素や属性は変更されていません。

新しい要素と属性

- **socket-binding** および **security-realm** を使用する **endpoint** 要素により、**endpoints** 要素内に含まれる複数のエンドポイント設定を定義できます。
- **security-realm-distributed** 要素を使用して、複数のセキュリティーレームを統合します。
- **security-realm** 要素の **default-realm** 属性は、別のレームを指定しない限り、最初に宣言されたセキュリティーレームであるデフォルトのセキュリティーレームを指定します。
- **security-realm** 要素の **name** 属性は、同じタイプの複数のレームを論理的に分割します。
- **hotrod-connector** 要素の **network-prefix-override** 属性は、ホストシステムがインターフェイス用に提供するネットマスクを使用するか、IANA プライベートアドレス規則に従うネットマスクで上書きするかどうかを設定します。
- **sasl** 要素の **policy** 属性は、利用可能なメカニズムのセットをフィルターするポリシーをリスト表示します。
- **ldap-realm** 要素の **client-ssl-context** 属性は、SSL/TLS 接続のクライアントを検証するトラストストアを提供するレームを指定します。

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。

- **regex-principal-transformer** 要素の **name** 属性が無視されるようになりました。
- TLS/SSL サーバーアイデンティティの **keystore** 要素の **keystore-password** 属性は非推奨になりました。代わりに **password** 属性を使用してください。

削除された要素と属性

Data Grid 8.3 では、要素や属性は削除されていません。

2.5. DATA GRID SERVER 8.4 設定スキーマへの変更

このトピックでは、8.3 から 8.4 までの Data Grid Server 設定スキーマへの変更を一覧表示します。

スキーマの変更

Data Grid 8.4 ではスキーマの変更は行われません。

変更された要素と属性

データソースの接続プールを設定する以下の属性には、デフォルト値が含まれるようになりました。

- **max-size** のデフォルト値は **2147483647** です。これは、プール内の接続数に制限がないことを意味します。
- **Min-size** のデフォルトは **0** です。これは、起動時にプールを空にできることを意味します。
- **initial-size** はデフォルトで **0** に設定されます。これは、最初は接続が作成されないことを意味します。

データソース接続プールを設定するための以下の属性では、デフォルト値が **0** に設定されています。0 の場合は、これらの機能は無効になります。

- **background-validation**
- **validate-on-acquisition**
- **leak-detection**
- **idle-removal**

新しい要素と属性

- **RESP-connector** 要素は、Data Grid の RESP エンドポイントを有効にします。
- **connection-property** 要素の新しい **maxOccurs** 属性は、この要素を実行できる最大回数を指定します。**maxOccurs** のデフォルト値は **unbounded** です。
- **masked-credential complexType**: 認証情報キーストアにマスクされたパスワードを追加します。マスク属性は、マスクされたパスワードを **MASKED_VALUE;SALT;ITERATION** 形式で指定します。
- **command-credential** は、認証情報キーストアのパスワードを提供する外部コマンドを実行します。**command** 属性は、外部コマンドを指定します。

非推奨の要素と属性

Data Grid 8.4 では非推奨になった要素や属性はありません。

削除された要素と属性

- **protocol-connector** 要素の **worker-threads** 属性が削除されました。
- **security-realm-filesystem** 要素が削除されました。

2.6. DATA GRID SERVER 8.5 設定スキーマへの変更

このトピックでは、8.4 から 8.5 までの Data Grid Server 設定スキーマへの変更を一覧表示します。

変更された要素と属性

- **ldap-name-rewriter** 要素の名前が **name-rewriter** に変更されました。

新しい要素と属性

- 証拠デコーダーのタイプを **x500-subject-evidence-decoder** または **x509-subject-alt-name-evidence-decoder** に指定できる **security-evidence-decoder** 要素
- **memcached-connector** の **authentication** 要素
- **memcached-connector** の **encryption** 要素
- **memcached-connector** の **protocol** 属性を使用すると、使用する Memcached プロトコルを設定できます。
- **memcached-connector** の **security-realm** 属性を使用すると、認証、キャッシュ認可、および暗号化に使用するセキュリティーレルムを定義できます。
- **rest-connector** の **compression-level** 属性値のデフォルトが **6** になりました。
- **rest-connector** の **compression-threshold** 属性を使用すると、レスポンス本体のサイズがしきい値を超えた場合にレスポンス本体の圧縮を設定できます。値は負でない数値である必要があります。**0** はすべての応答の圧縮を有効にします。
- **rest-connector** の **require-ssl-client-auth** 属性はオプションではなくなりました。
- **security-realm** 内の **evidence-decoder** 要素
- **security-realm** 内の **aggregate-realm** 要素
- **security-realm** の **cache-lifespan** 属性のデフォルトが **-1** ではなく **60000** になりました。
- **security-realm-properties** の **groups-attribute** 属性のデフォルト値が **Roles** になりました。
- **name-rewriter** の **case-principal-transformer** 要素
- **name-rewriter** の **common-name-principal-transformer** 要素

非推奨の要素と属性

Data Grid 8.5 では非推奨になった要素や属性はありません。

削除された要素と属性

Data Grid 8.5 では要素と属性は削除されませんでした。

2.7. DATA GRID SERVER のエンドポイントとネットワーク設定

このセクションでは、以前のバージョンから移行する場合の Data Grid Server エンドポイントとネットワーク設定について説明します。

Data Grid 8 は、単一のネットワークインターフェイスとポートを使用してネットワーク上のエンドポイントを公開することにより、サーバーエンドポイントの設定を簡素化します。

2.7.1. インターフェイス

インターフェイスは、エンドポイントをネットワークの場所にバインドします。

Data Grid Server 7.x ネットワークインターフェイスの設定

Data Grid 7.x では、サーバー設定でさまざまなインターフェイスを使用して、管理アクセスと管理アクセスをキャッシュアクセスから分離していました。

```
<interfaces>
  <interface name="management">
    <inet-address value="\${jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="\${jboss.bind.address:127.0.0.1}"/>
  </interface>
</interfaces>
```

Data Grid Server 8 ネットワークインターフェイスの設定

Data Grid 8 には、管理アクセスと管理アクセス、およびキャッシュアクセス用のすべてのクライアント接続用に1つのネットワークインターフェイスがあります。

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

2.7.2. ソケットバインディング

ソケットバインディングは、エンドポイントがクライアント接続をリッスンするポートにネットワークインターフェイスをマップします。

Data Grid Server 7.x ソケットバインディングの設定

Data Grid 7.x では、サーバー設定は、管理コンソール用の **9990** やネイティブ管理プロトコル用のポート **9999** など、管理と管理に一意のポートを使用していました。古いバージョンでは、外部 Hot Rod アクセス用の **11222** や REST 用の **8080** など、エンドポイントごとに一意のポートも使用されていました。

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="\${jboss.socket.binding.port-offset:0}">
  <socket-binding name="management-http" interface="management"
port="\${jboss.management.http.port:9990}"/>
  <socket-binding name="management-https" interface="management"
port="\${jboss.management.https.port:9993}"/>
  <socket-binding name="hotrod" port="11222"/>
  <socket-binding name="hotrod-internal" port="11223"/>
  <socket-binding name="hotrod-multi-tenancy" port="11224"/>
  <socket-binding name="memcached" port="11211"/>
  <socket-binding name="rest" port="8080"/>
  ...
</socket-binding-group>
```

Data Grid Server 8 の単一ポート設定

Data Grid 8 は、単一のポートを使用してサーバーへのすべての接続を処理します。Hot Rod クライアント、REST クライアント、Data Grid CLI、および Data Grid Console はすべて、ポート **11222** を使用します。

```
<socket-bindings default-interface="public"
port-offset="\${infinispan.socket.binding.port-offset:0}">
  <socket-binding name="default" port="\${infinispan.bind.port:11222}"/>
  <socket-binding name="memcached" port="11211"/>
</socket-bindings>
```

2.7.3. エンドポイント

エンドポイントは、リモートクライアント接続をリッスンし、Hot Rod や HTTP (REST) などのプロトコルを介して要求を処理します。



注記

Data Grid CLI は、すべてのキャッシュおよび管理操作に REST エンドポイントを使用します。

Data Grid Server 7.x エンドポイントサブシステム

Data Grid 7.x では、**endpoint** サブシステムを使用して、Hot Rod および REST エンドポイントのコネクターを設定できます。

```
<subsystem xmlns="urn:infinispan:server:endpoint:9.4">
  <hotrod-connector socket-binding="hotrod" cache-container="local">
    <topology-state-transfer lazy-retrieval="false" lock-timeout="1000" replication-timeout="5000"/>
  </hotrod-connector>
  <rest-connector socket-binding="rest" cache-container="local">
    <authentication security-realm="ApplicationRealm" auth-method="BASIC"/>
  </rest-connector>
</subsystem>
```

Data Grid Server 8 エンドポイント設定

Data Grid 8 は、**endpoint** サブシステムを **endpoint** 要素に置き換えます。**hotrod-connector** と **rest-connector** の設定要素と属性は、以前のバージョンと同じです。

Data Grid 8.2 以降、デフォルトの **endpoints** 設定は、以下のように暗黙的な Hot Rod および REST コネクターを使用します。

```
<endpoints socket-binding="default" security-realm="default"/>
```

Data Grid Server 8.0 から 8.2

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

Data Grid Server 8.3 以降では、セキュリティーレルムを使用してエンドポイントを設定し、**endpoint** 要素を使用して Hot Rod または REST コネクターを設定します。**endpoints** 要素は、複数の **endpoint** 設定のラッパーになりました。

Data Grid Server 8.3 以降

```
<endpoints>
  <endpoint socket-binding="public" security-realm="application-realm" admin="false">
    <hotrod-connector/>
    <rest-connector/>
  </endpoint>
  <endpoint socket-binding="private" security-realm="management-realm">
    <hotrod-connector/>
```



```
<rest-connector/>
</endpoint>
</endpoints>
```

関連情報

- [Data Grid Server ガイド](#)

2.8. DATA GRID SERVER のセキュリティー

Data Grid Server のセキュリティーは、認証と暗号化を設定して、ネットワーク攻撃を防ぎ、データを保護します。

2.8.1. セキュリティーレルム

Data Grid 8 では、セキュリティーレルムは暗黙の設定オプションを提供します。これは、以前のバージョンほど多くの設定を提供する必要がないことを意味します。たとえば、Kerberos レルムを定義すると、Kerberos 機能を利用できます。トラストストアを追加すると、証明書認証を取得します。

Data Grid 7.x には、2つのデフォルトのセキュリティーレルムがありました。

- **ManagementRealm** は管理 API を保護します。
- **ApplicationRealm** は、エンドポイントとリモートクライアント接続を保護します。

一方、Data Grid 8 は、Hot Rod および REST エンドポイントに使用できる複数の異なるセキュリティーレルムを定義できる **security** 要素を提供します。

```
<security>
  <security-realms>
    ...
  </security-realms>
</security>
```

サポートされているセキュリティーレルム

- プロパティーレルムは、プロパティーファイル **users.properties** と **groups.properties** を使用して、データグリッドにアクセスできるユーザーとグループを定義します。
- LDAP レルムは、OpenLDAP、Red Hat Directory Server、Apache Directory Server、Microsoft Active Directory などの LDAP サーバーに接続して、ユーザーを認証し、メンバーシップ情報を取得します。
- トラストストアレルムは、データグリッドへのアクセスが許可されているすべてのクライアントの公開証明書を含むキーストアを使用します。
- トークンレルムは外部サービスを使用してトークンを検証し、Red Hat SSO などの RFC-7662 (OAuth2 トークンイントロスペクション) と互換性のあるプロバイダーを必要とします。

2.8.2. サーバー ID

サーバー ID は、証明書チェーンを使用して、データグリッドサーバー ID をリモートクライアントに証明します。

Data Grid 8 は、以前のバージョンと同じ設定を使用して SSL ID を定義しますが、使いやすさが向上しています。

- セキュリティーレームに SSLID が含まれている場合、Data Grid はそのセキュリティーレームを使用するエンドポイントの暗号化を自動的に有効にします。
- テストおよび開発環境の場合、Data Grid には、起動時にキーストアを自動的に生成する **generate-self-signed-certificate-host** 属性が含まれています。

```
<security-realm name="default">
  <server-identities>
    <ssl>
      <keystore path="..."
        relative-to="..."
        keystore-password="..."
        alias="..."
        key-password="..."
        generate-self-signed-certificate-host="..."/>
    </ssl>
  </server-identities>
  ...
</security-realm>
```

2.8.3. エンドポイント認証メカニズム

Hot Rod および REST エンドポイントは、SASL または HTTP メカニズムを使用してクライアント接続を認証します。

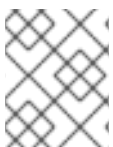
Data Grid 8 は、Data Grid 7.x 以前と同じ **authentication** 要素を **hotrod-connector** および **rest-connector** 設定に使用します。

```
<hotrod-connector name="hotrod">
  <authentication>
    <sasl mechanisms="..." server-name="..."/>
  </authentication>
</hotrod-connector>
<rest-connector name="rest">
  <authentication>
    <mechanisms="..." server-principal="..."/>
  </authentication>
</rest-connector>
```

以前のバージョンとの主な違いの1つは、Data Grid 8 がエンドポイントの追加の認証メカニズムをサポートしていることです。

Hot Rod SASL 認証メカニズム

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ー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。

認証メカニズム	説明	関連する詳細
PLAIN	プレーンテキスト形式の認証情報を使用します。 PLAIN 認証は、暗号化された接続でのみ使用する必要があります。	Basic HTTP メカニズムに似ています。
DIGEST *	ハッシュアルゴリズムとナンス値を使用します。ホットロッドコネクタは、強度の順に、 DIGEST-MD5 、 DIGEST-SHA 、 DIGEST-SHA-256 、 DIGEST-SHA-384 、および DIGEST-SHA-512 ハッシュアルゴリズムをサポートします。	Digest HTTP メカニズムに似ています。
SCRAM *	ハッシュアルゴリズムとナンス値に加えてソルト値を使用します。ホットロッドコネクタは、 SCRAM-SHA 、 SCRAM-SHA-256 、 SCRAM-SHA-384 、および SCRAM-SHA-512 ハッシュアルゴリズムを強度順にサポートします。	Digest HTTP メカニズムに似ています。
GSSAPI	Kerberos チケットを使用し、Kerberos ドメインコントローラーが必要です。対応する Kerberos サーバー ID をレルム設定に追加する必要があります。ほとんどの場合、ユーザーメンバーシップ情報を提供するために ldap-realm も指定します。	SPNEGO HTTP メカニズムに似ています。
GS2-KRB5	Kerberos チケットを使用し、Kerberos ドメインコントローラーが必要です。対応する Kerberos サーバー ID をレルム設定に追加する必要があります。ほとんどの場合、ユーザーメンバーシップ情報を提供するために ldap-realm も指定します。	SPNEGO HTTP メカニズムに似ています。
EXTERNAL	クライアント証明書を使用します。	CLIENT_CERT HTTP メカニズムに似ています。
OAuthBEARER	OAuth トークンを使用し、 token-realm 設定が必要です。	BEARER_TOKEN HTTP メカニズムに似ています。

HTTP (REST) 認証メカニズム

認証メカニズム	説明	関連する詳細
BASIC	プレーンテキスト形式の認証情報を使用します。暗号化された接続でのみ BASIC 認証を使用する必要があります。	HTTP Basic HTTP 認証方式に対応し、 PLAIN SASL メカニズムと同様です。
DIGEST	ハッシュアルゴリズムとナンス値を使用します。REST コネクタは、 SHA-512 、 SHA-256 、および MD5 ハッシュアルゴリズムをサポートします。	Digest HTTP 認証スキームに対応し、 DIGEST-* SASL メカニズムに似ています。
SPNEGO	Kerberos チケットを使用し、Kerberos ドメインコントローラーが必要です。対応する Kerberos サーバー ID をレルム設定に追加する必要があります。ほとんどの場合、ユーザーメンバーシップ情報を提供するために ldap-realm も指定します。	Negotiate HTTP 認証スキームに対応し、 GSSAPI および GS2-KRB5SASL メカニズムに類似しています。
BEARER_TOKEN	OAuth トークンを使用し、 token-realm 設定が必要です。	Bearer HTTP 認証スキームに対応し、 OAUTHBEARERSASL メカニズムに似ています。
CLIENT_CERT	クライアント証明書を使用します。	EXTERNAL SASL メカニズムに似ています。

2.8.4. EAP アプリケーションの認証

EAP アプリケーションクラスパスの **hotrod-client.properties** にクレデンシャルを追加して、次の方法でデータグリッドで認証できるようになりました。

- リモートキャッシュコンテナ (**remote-cache-container**)
- リモートストア (**remote-store**)
- EAP モジュール

2.8.5. ロギング

Data Grid は、JBossLogManager に基づいていた以前のバージョンのロギングサブシステムの代わりに Apache Log4j2 を使用します。

デフォルトでは、Data Grid はログメッセージを次のディレクトリーに書き込みます。
\$RHDG_HOME/\${infinispan.server.root}/log

server.log はデフォルトのログファイルです。

アクセスログ

以前のバージョンでは、Data Grid にキャッシュのセキュリティーログを監査するためのロガーが含まれていました。

```
<authorization audit-logger="org.infinispan.security.impl.DefaultAuditLogger">
```

Data Grid 8 は、この監査ロガーを提供しなくなりました。

ただし、Hot Rod エンドポイントと REST エンドポイントのログカテゴリーを使用できます。

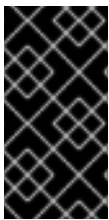
- `org.infinispan.HOTROD_ACCESS_LOG`
- `org.infinispan.REST_ACCESS_LOG`

関連情報

- [Data Grid Server ガイド](#)

2.9. DATA GRID SERVER エンドポイントの分離

以前のバージョンから移行する場合、既存の設定に一致するように、Data Grid エンドポイントに異なるネットワークロケーションを作成できます。ただし、Data Grid アーキテクチャーが変更され、すべてのクライアント接続に単一のポートを使用するようになったため、以前のバージョンのすべてのオプションを使用できるわけではありません。



重要

Data Grid CLI やコンソールなどの管理ツールは REST API を使用します。Data Grid CLI とコンソールを無効にせずに、エンドポイント設定から REST API を削除することはできません。同様に、REST エンドポイントを分離して、キャッシュアクセスと管理アクセスに異なるポートまたはソケットバインディングを使用することはできません。

手順

1. REST エンドポイントと Hot Rod エンドポイントに別々のネットワークインターフェイスを定義します。
たとえば、Hot Rod エンドポイントを外部に公開するための "public" インターフェイスと、アクセスが制限されているネットワークロケーション上の REST エンドポイントを公開するための "private" インターフェイスを定義します。

```
<interfaces>
  <interface name="public">
    <inet-address value="${infinispan.bind.address:198.51.100.0}"/>
  </interface>
  <interface name="private">
    <inet-address value="${infinispan.bind.address:192.0.2.0}"/>
  </interface>
</interfaces>
```

この設定により、以下が作成されます。

- **198.51.100.0** IP アドレスを持つ "public" インターフェイス。
 - **192.0.2.0** IP アドレスを持つ "private" インターフェイス。
2. 次の例のように、エンドポイントに個別のソケットバインディングを設定します。

```
<socket-bindings default-interface="private"
```

```

        port-offset="${infinispan.socket.binding.port-offset:0}">
<socket-binding name="default"
    port="${infinispan.bind.port:8080}"/>
<socket-binding name="hotrod"
    interface="public"
    port="11222"/>
</socket-bindings>

```

この例では、

- プライベートインターフェイスをソケットバインディングのデフォルトとして設定します。
- ポート **8080** を使用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。
- パブリックインターフェイスとポート **11222** を使用する hotrod ソケットバインディングを作成します。

3. 次に、エンドポイント用に個別のセキュリティーレームを作成します。

```

<security>
  <security-realms>
    <security-realm name="truststore">
      <server-identities>
        <ssl>
          <keystore path="server.p12"
            relative-to="infinispan.server.config.path"
            keystore-password="secret"
            alias="server"/>
        </ssl>
      </server-identities>
      <truststore-realm path="trust.p12"
        relative-to="infinispan.server.config.path"
        keystore-password="secret"/>
    </security-realm>
    <security-realm name="kerberos">
      <server-identities>
        <kerberos keytab-path="http.keytab"
          principal="HTTP/localhost@INFINISPAN.ORG"
          required="true"/>
      </server-identities>
    </security-realm>
  </security-realms>
</security>

```

この例では、

- トラストストアのセキュリティーレームを設定します。
- Kerberos セキュリティーレームを設定します。

4. 次のようにエンドポイントを設定します。

```

<endpoints>
  <endpoint socket-binding="default"
    security-realm="kerberos">

```

```

<hotrod-connector/>
<rest-connector/>
</endpoint>
<endpoint socket-binding="hotrod"
  security-realm="truststore">
  <hotrod-connector/>
  <rest-connector/>
</endpoint>
</endpoints>

```

5. Data Grid Server を起動します。
ログには、エンドポイントがクライアント接続を受け入れるネットワークの場所を示す次のメッセージが含まれます。

```

[org.infinispan.SERVER] ISPN080004: Protocol HotRod listening on 198.51.100.0:11222
[org.infinispan.SERVER] ISPN080004: Protocol SINGLE_PORT listening on 192.0.2.0:8080
[org.infinispan.SERVER] ISPN080034: Server '<hostname>' listening on http://192.0.2.0:8080

```

次のステップ

1. <http://192.0.2.0:8080> の任意のブラウザから Data Grid Console にアクセスします
2. カスタムの場所で接続するように Data Grid CLI を設定します。次に例を示します。

```
$ bin/cli.sh -c http://192.0.2.0:8080
```

2.10. DATA GRID SERVER 共有データソース

Data Grid 7.x JDBC キャッシュストアは、**PooledConnectionFactory** を使用してデータベース接続を取得できます。

Data Grid 8 を使用すると、サーバー設定でマネージドデータソースを作成して、JDBC キャッシュストアを使用したデータベース接続の接続プールとパフォーマンスを最適化できます。

データソース設定は、次の2つのセクションで設定されています。

- データベースへの接続方法を定義する **connection factory**。
- 接続をプールして再利用する方法を定義し、Agroal に基づく **connection pool**。

最初にサーバー設定でデータソース接続ファクトリーと接続プールを定義し、次にそれを JDBC キャッシュストア設定に追加します。

JDBC キャッシュストアの移行の詳細については、このドキュメントの[キャッシュストアの移行セクション](#)を参照してください。

関連情報

- [Data Grid Server ガイド](#)

2.11. DATA GRID SERVER の JMX とメトリクス

Data Grid 8 は、Prometheus などのメトリクスツールと統合するために JMX と **/metrics** エンドポイントの両方を介してメトリクスを公開します。

`/metrics` エンドポイントは以下を提供します。

- JVM の稼働時間やキャッシュ操作の平均秒数などの値を返すゲージ。
- 読み取り、書き込み、および削除操作にかかる時間をパーセンタイルで示すヒストグラム。

以前のバージョンでは、Prometheus メトリクスは、ネイティブでサポートされるのではなく、JMX メトリクスをマップするエージェントによって収集されていました。

以前のバージョンの Data Grid は、JBoss Operations Network (JON) プラグインを使用して、メトリクスを取得し、操作を実行していました。Data Grid 8 は JON プラグインを使用しなくなりました。

Data Grid 8 は、JMX と Prometheus のメトリクスをキャッシュマネージャーとキャッシュレベルの設定に分離します。

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

1 キャッシュマネージャーの統計を有効にします。これはデフォルトになります。

2 すべての統計と操作を含む JMX MBean をエクスポートします。

```
<distributed-cache name="mycache" statistics="true" /> 1
```

1 キャッシュの統計を有効にします。

関連情報

- [Data Grid Server ガイド](#)

2.12. DATA GRID SERVER のチートシート

次のコマンドと例は、Data Grid Server を操作するためのクイックリファレンスとして使用してください。

サーバーインスタンスの起動

- Linux

```
$ bin/server.sh
```

- Microsoft Windows

```
$ bin\server.bat
```

CLI の開始

- Linux

```
$ bin/cli.sh
```


- Microsoft Windows

```
$ bin\cli.bat
```

ユーザーの作成

- Linux

```
$ bin/cli.sh user create myuser -p "qwer1234!"
```

- Microsoft Windows

```
$ bin\cli.bat user create myuser -p "qwer1234!"
```

サーバーインスタンスの停止

- 単一サーバーインスタンス

```
[//containers/default]> shutdown server $hostname
```

- クラスター全体

```
[//containers/default]> shutdown cluster
```

使用可能なコマンドオプションのリスト表示

-h フラグを使用して、サーバーを実行するために使用可能なコマンドオプションをリスト表示します。

- Linux

```
$ bin/server.sh -h
```

- Microsoft Windows

```
$ bin\server.bat -h
```

7.x から 8 のリファレンス

7.x	8.x
<code>./standalone.sh -c clustered.xml</code>	<code>./server.sh</code>
<code>./standalone.sh</code>	<code>./server.sh -c infinispan-local.xml</code>
<code>-Djboss.default.multicast.address=234.99.54.20</code>	<code>-Djgroups.mcast_addr=234.99.54.20</code>
<code>-Djboss.bind.address=172.18.1.13</code>	<code>-Djgroups.bind.address=172.18.1.13</code>
<code>-Djboss.default.jgroups.stack=udp</code>	<code>-j udp</code>

関連情報

- [Data Grid Server ガイド](#)

第3章 DATA GRI 設定の移行

Data Grid 8 への移行に影響するデータグリッド設定の変更を見つけます。

3.1. DATA GRID キャッシュの設定

Data Grid 8 は、デフォルトで空のキャッシュコンテナを提供します。Data Grid 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。

ただし、以前のバージョンと比較すると、すぐに使用できるデフォルトのキャッシュはありません。

Data Grid 8 では、**CacheContainerAdmin** API を介して作成するキャッシュは、クラスタ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。

永続的なキャッシュ

```
.administration()
  .withFlags(AdminFlag.PERMANENT) ❶
  .getOrCreateCache("myPermanentCache", "org.infinispan.DIST_SYNC");
```

- ❶ **AdminFlag.PERMANENT** はデフォルトで有効になっており、キャッシュが再起動後も存続するようになっています。

キャッシュを作成するときにこのフラグを設定する必要はありません。ただし、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);
```

揮発性キャッシュ

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

- ❶ データグリッドの再起動時にキャッシュが失われるように **VOLATILE** フラグを設定します。
- ❷ "myTemporaryCache" という名前のキャッシュを返すか、**DIST_SYNC** テンプレートを使用してキャッシュを作成します。

Data Grid 8 は、推奨設定でキャッシュを作成するために使用できるサーバーインストール用のキャッシュテンプレートを提供します。

使用可能なキャッシュテンプレートのリストは、次のように取得できます。

- 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
```

3.1.1. キャッシュエンコーディング

リモートキャッシュを作成するときは、キーと値の MediaType を設定する必要があります。MediaType を設定すると、データのストレージ形式が保証されます。

キャッシュをエンコードするには、設定で MediaType を指定します。他の要件がない限り、ProtoStream を使用する必要があります。これは、言語に依存しない下位互換性のある形式でデータを格納します。

```
<encoding media-type="application/x-protostream"/>
```

エンコーディングを使用した分散キャッシュ設定

```
<infinispan>
  <cache-container>
    <distributed-cache name="myCache" mode="SYNC">
      <encoding media-type="application/x-protostream"/>
      ...
    </distributed-cache>
  </cache-container>
</infinispan>
```

リモートキャッシュをエンコードしない場合、Data Grid Server は次のメッセージをログに記録します。

```
WARN (main) [org.infinispan.encoding.impl.StorageConfigurationManager] ISPN000599:
Configuration for cache 'mycache' does not define the encoding for keys or values. If you use
operations that require data conversion or queries, you should configure the cache with a specific
MediaType for keys or values.
```

将来のバージョンでは、データ変換が行われる操作にキャッシュエンコーディングが必要になります。たとえば、キャッシュのインデックス作成とデータコンテナの検索、リモートタスクの実行、Hot Rod および REST エンドポイントからのさまざまな形式でのデータの読み取りと書き込み、リモートフィルター、コンバーター、リスナーの使用などです。

3.1.2. キャッシュヘルスステータス

Data Grid 7.x には、クラスターのヘルスステータスとクラスター内のキャッシュを返すヘルスチェック API が含まれています。

Data Grid 8 は Health API も提供します。組み込みおよびサーバーインストールの場合、次の MBean を使用して JMX 経由で Health API にアクセスできます。

```
org.infinispan:type=CacheManager,name="default",component=CacheContainerHealth
```

Data Grid Server は、REST エンドポイントと Data Grid Console を介して Health API も公開します。

表3.1ヘルスステータス

7.x	8.x	説明
HEALTHY	HEALTHY	キャッシュが期待どおりに動作していることを示します。
Rebalancing	HEALTHY_REBALANCING	キャッシュがリバランス状態にあることを示しますが、それ以外場合は想定どおりに動作します。
Unhealthy	DEGRADED	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。
該当なし	FAILED	8.2 で追加され、提供された設定でキャッシュを開始できなかったことを示します。

関連情報

- [Data Grid キャッシュの設定](#)

3.1.3. Data Grid 8.1 設定スキーマへの変更

このトピックでは、8.0 から 8.1 までの Data Grid 設定スキーマへの変更をリスト表示します。

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 MicroProfile Metrics API と互換性のあるメトリクス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**

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**

3.1.4. Data Grid 8.2 設定スキーマへの変更

このトピックでは、8.1 から 8.2 までの Data Grid 設定スキーマへの変更をリスト表示します。

変更された要素と属性

- **white-list** が **allow-list** に変更されました
- **role** は、定義されたユーザーロールの **role** とセキュリティ認証の権限のサブ要素になりました。
- **context-initializer** は、**SerializationContextInitializer** の自動登録用に更新されます。設定に **context-initializer** 要素が含まれていない場合、**java.util.ServiceLoader** メカニズムは、クラスパス上のすべての **SerializationContextInitializer** 実装を自動的に検出し、それらをロードします。
- **minOccurs** 属性のデフォルト値は、**indexed-entity** 要素に対して **0** から **1** に変更されます。

新しい要素と属性

- 名前/値のトランスポートプロパティを渡すことができる **transport** 要素に追加された **property** 属性
- アクセス制御リスト (ACL) キャッシュのサイズとタイムアウトを設定するために、**security** 要素に追加された **cache-size** 属性と **cache-timeout** 属性
- **indexing** 要素に追加された **index-reader**、**index-writer**、および **index-merge** の子要素
- ストレージオプションを指定する **indexing** 要素に追加された **storage** 属性
- インデックスにファイルシステムストレージを使用するときにディレクトリーを指定する **indexing** 要素に追加された **path** 属性
- ノードがエントリーのバイアスを取得できるタイミングを制御する、**scattered-cache** 要素に追加された **bias-acquisition** 属性

- ミリ秒単位で、ノードが取得したバイアスを維持できる期間を指定する、**scattered-cache** 要素に追加された **bias-lifespan** 属性
- クロスサイトレプリケーションとの競合を解決するためのアルゴリズムを指定する、**backups** 要素に追加された **merge-policy** 属性
- **backup** の **state-transfer** 要素に追加された **mode** 属性
mode 属性は、クロスサイトレプリケーション状態の転送を手動で行うか自動で行うかを設定します。
- **INSERT_ABOVE**、**INSERT_BEFORE**、および **INSERT_BELOW** 属性を継承して JGroups のスタックを拡張するための **stack.combine** 属性に追加します。

非推奨の要素と属性

Data Grid 8.2 では非推奨の要素や属性はありません。

削除された要素と属性

Data Grid 8.2 では要素や属性は削除されません。

3.1.5. Data Grid 8.3 設定スキーマへの変更

このトピックでは、8.2 から 8.3 までの Data Grid 設定スキーマへの変更をリスト表示します。

スキーマの変更

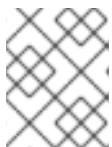
- **urn:infinispan:config:store:soft-index** namespace is no longer available.

変更された要素と属性

- **urn:infinispan:config** namespace の **file-store** 要素は、デフォルトで **soft-index** ファイル キャッシュストアを使用します。
- **single-file-store** 要素は **urn:infinispan:config** namespace に含まれますが、非推奨になりました。

新しい要素と属性

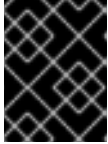
- **index** 要素と **data** 要素を使用して、Data Grid が **file-store** 要素でファイルベースのキャッシュストアのインデックスとデータを格納する方法を設定できるようになりました。
- **file-store** 要素の **open-files-limit** と **compaction-threshold** 属性
- クロスサイト通信にグローバルクラスター名を定義できるようにする **remote-sites** 要素および **remote-site** 要素に追加された **cluster** 属性



注記

cluster 属性で指定するグローバルクラスター名は、すべてのサイトで同じである必要があります。

- **currentNumberOfEntries** 統計を使用してデータセットを計算できるようにするために、**metrics** 要素に **accurate-size** 属性が追加されました。



重要

Data Grid 8.3 以降、**currentNumberOfEntries** 統計は、実行するのにコストのかかる操作であるため、デフォルトで **-1** の値を返します。

- 最大アイドル有効期限を持つクラスター化キャッシュ内のエントリーのタイムスタンプを更新する方法を制御する **expiration** 要素に追加された **touch** 属性デフォルト値は **SYNC** で、属性は同期レプリケーションを使用するキャッシュにのみ適用されます。非同期レプリケーションを使用するキャッシュのタイムスタンプは非同期で更新されます。
- 有効期限の値を割り当てる **strong-counter** に追加される **lifespan** 属性 (ミリ秒単位)。デフォルト値は **-1** で、強力な一貫性のあるカウンターが期限切れにならないことを意味します。



注記

強力なカウンターの **lifespan** 属性は、現在テクノロジープレビューとして提供されています。

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。

- **single-file-store** 要素
- **file-store** 要素の **max-entries** および **path** 属性

削除された要素と属性

Data Grid スキーマでは、以下の要素および属性が利用できなくなりました。

- **transport** 要素の **remote-command-executor** 属性
- **distributed-cache** 要素の **capacity** 属性

3.1.6. Data Grid 8.4 設定スキーマへの変更

このトピックでは、8.3 から 8.4 までの Data Grid 設定スキーマへの変更を一覧表示します。

スキーマの変更

新しい要素と属性

- **default-max-results** 属性が **クエリ** 要素に追加されました。これにより、クエリによって返される結果の数を制限できます。インデックス付き、インデックスなし、およびハイブリッドクエリに適用されます。
- **start-mode** 属性を使用すると、キャッシュの開始時に Data Grid が実行する操作を定義できます。オプションは **purge**、**reindex**、**auto**、または **none** です。デフォルト値は **none** です。
- **raft-members** 属性を使用すると、スペースで区切られた raft メンバーのリストを定義できます。

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。

- **scattered-cache** 要素が非推奨に

削除された要素と属性

Data Grid スキーマでは、以下の要素および属性が利用できなくなりました。

- **fetch-state** ストアプロパティは使用できなくなりました。使用中の xml 設定からこの属性を削除できます。

3.1.7. Data Grid 8.5 設定スキーマへの変更

このトピックでは、8.4 から 8.5 までの Data Grid 設定スキーマへの変更を一覧表示します。

新しい要素と属性

- **tracing** 要素が **cache-container** に追加され、トレーシングを設定できるようになりました。トレースは OpenTelemetry コレクターによって収集できます。
- **authorization** に **group-only-mapping** 属性が追加されました。この属性を使用して、**principal-to-role mapping** がグループプリンシパルにのみ適用されるか、ユーザープリンシパルにも適用されるかを指定します。デフォルト値 **true** は、**principal-to-role mapping** をグループプリンシパルにのみ適用します。両方のプリンシパルタイプにマッピングを適用するには、値を **false** に設定します。
- ロールの説明を定義できる **description** 属性が **roles** に追加されました。
- **serialization** に **schema-compatibility** 属性が追加され、スキーマの更新時に実行される互換性検証を指定できるようになりました。
- **unclean-shutdown-action** 属性が **global-state** に追加されました。これにより、永続的なグローバル状態でダングリングロックファイルが見つかった場合に実行されるアクションを定義でき、ノードの不正なシャットダウンを示します。デフォルト値は **FAIL** です。
- **indexing** に **index-sharding** 要素が追加されました。シャーディングは、インデックスデータをシャードと呼ばれる複数の小さなインデックスに分割するプロセスです。シャーディングにより、大量のデータを処理する際のパフォーマンスが向上します。シャーディングはデフォルトで無効になっています。
- **indexing** に **indexing-mode** 要素が追加され、キャッシュ操作がインデックスに伝播される方法を定義できるようになりました。デフォルトでは、キャッシュへのすべての変更はインデックスにすぐに適用されます。
- **indexing** に **tracing** 要素が追加され、トレーシングを設定できるようになりました。トレースは OpenTelemetry コレクターによって収集できます。
- **cache** に **aliases** 属性が追加され、キャッシュに 0 個以上のエイリアス名を定義できるようになりました。
- キャッシュが統計情報を収集するかどうかを定義する **statistics** 属性が **cache** に追加されました。最適なパフォーマンスを得るには、統計情報収集を無効にしておきます。

非推奨の要素と属性

このリリースでは非推奨はありません。

削除された要素と属性

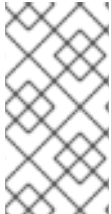
Data Grid スキーマでは、以下の要素および属性が利用できなくなりました。

- **scattered-cache** 要素が削除されました。
- **property** 要素が **cache** から削除されました。
- **auto-config** 要素が **indexing** から削除されました。

- **statistics-available** 属性が **indexing** から削除されました。
- **connection-interval** 属性が **persistence** から削除されました。

3.2. エビクション設定

Data Grid 8 は、以前のバージョンと比較して、エビクション設定を簡素化します。ただし、エビクション設定は、データグリッドのさまざまなバージョン間で多数の変更が加えられているため、移行が簡単ではない可能性があります。



注記

Data Grid 7.2 以降、**memory** 要素が設定内の **eviction** 要素に置き換わります。このセクションでは、**memory** 要素のみを使用したエビクション設定について説明します。**eviction** 要素を使用する設定の移行については、Data Grid 7.2 のドキュメントを参照してください。

3.2.1. ストレージタイプ

Data Grid では、次のオプションを使用して、エンタリーをメモリーに保存する方法を制御できます。

- オブジェクトを JVM ヒープメモリーに格納します。
- バイトをネイティブメモリー (オフヒープ) に格納します。
- バイトを JVM ヒープメモリーに格納します。

Data Grid 8 の変更

以前の 7.x バージョンおよび 8.0 では、**object**、**binary**、および **off-heap** 要素を使用してストレージタイプを設定していました。

Data Grid 8.1 以降では、**storage** 属性を使用して、オブジェクトを JVM ヒープメモリーに格納するか、バイトとしてオフヒープメモリーに格納します。

JVM ヒープメモリーにバイトを格納するには、**encoding** 要素を使用して、データのバイナリーストレージ形式を指定します。

Data Grid 7.x	Data Grid 8
<code><memory><object /></memory></code>	<code><memory /></code>
<code><memory><off-heap /></memory></code>	<code><memory storage="OFF_HEAP" /></code>
<code><memory><binary /></memory></code>	<code><encoding media-type="..." /></code>

Data Grid 8 のオブジェクトストレージ

デフォルトでは、Data Grid 8.1 はオブジェクトストレージ (JVM ヒープ) を使用します。

```
<distributed-cache>
  <memory />
</distributed-cache>
```

`storage="HEAP"` を明示的に設定して、データをオブジェクトとして JVM ヒープメモリーに格納することもできます。

```
<distributed-cache>
  <memory storage="HEAP" />
</distributed-cache>
```

Data Grid 8 のオフヒープストレージ

データをバイトとしてネイティブメモリーに保存するには、`storage` 属性の値として `"OFF_HEAP"` を設定します。

```
<distributed-cache>
  <memory storage="OFF_HEAP" />
</distributed-cache>
```

オフヒープアドレス数

以前のバージョンでは、`offheap` の `address-count` 属性を使用すると、衝突を回避するためにハッシュマップで使用可能なポインタの数を指定できます。Data Grid 8.1 では、`address-count` は使用されなくなり、オフヒープメモリーは衝突を回避するために動的にサイズ変更されます。

Data Grid 8 のバイナリーストレージ

`encoding` 要素を使用して、キャッシュエントリーのバイナリーストレージ形式を指定します。

```
<distributed-cache>
  <!--Configure MediaType for entries with binary formats.-->
  <encoding media-type="application/x-protostream"/>
  <memory ... />
</distributed-cache>
```



注記

この変更の結果、Data Grid は、`byte[]` と混合されたプリミティブと String を格納しなくなり、`byte[]` のみを格納します。

3.2.2. エビクシヨンスきい値

エビクシヨンを使用すると、コンテナが設定されたしきい値より大きくなったときにエントリーを削除することで、Data Grid がデータコンテナのサイズを制御できます。

Data Grid 7.x および 8.0 では、キャッシュ内のエントリーの最大制限を定義する 2 つのエビクシヨンタイプを指定します。

- **COUNT** は、キャッシュ内のエントリーの数を測定します。
- **MEMORY** は、キャッシュ内のすべてのエントリーが占めるメモリーの量を測定します。

設定した設定に応じて、メモリーの数または合計量のいずれかが最大値を超えると、Data Grid は未使用のエントリーを削除します。

Data Grid 7.x および 8.0 も長いようなデータコンテナのサイズを規定する `size` 属性を使用します。設定するストレージの種類に応じて、エントリーの数またはメモリーの量が `size` 属性の値を超えると、エビクシヨンが発生します。

Data Grid 8.1 では、**size** 属性は **COUNT** および **MEMORY** とともに非推奨になりました。代わりに、次の 2 つの方法のいずれかでデータコンテナの最大サイズを設定します。

- **max-count** 属性を持つエントリーの総数。
- **max-size** 属性を持つメモリの最大量 (バイト単位)。

エントリーの総数に基づくエビクション

```
<distributed-cache>
  <memory max-count="..." />
</distributed-cache>
```

最大メモリー量に基づくエビクション

```
<distributed-cache>
  <memory max-size="..." />
</distributed-cache>
```

3.2.3. エビクションストラテジー

エビクション戦略は、Data Grid がエビクションを実行する方法を制御します。

Data Grid 7.x および 8.0 では、**strategy** 属性を使用して次のエビクション戦略のいずれかを設定できます。

ストラテジー	説明
NONE	Data Grid はエントリーを削除しません。これは、エビクションを設定しない限り、デフォルト設定です。
REMOVE	Data Grid は、キャッシュが設定されたサイズを超えないように、メモリーからエントリーを削除します。これは、エビクションを設定するときのデフォルト設定です。
MANUAL	Data Grid は削除を実行しません。削除は、 evict() メソッドを Cache API から呼び出すことによって手動で行われます。
EXCEPTION	Data Grid は、設定されたサイズを超える場合、キャッシュに新しいエントリーを書き込みません。キャッシュに新しいエントリーを書き込む代わりに、Data Grid は ContainerFullException 出力します。

Data Grid 8.1 では、以前のバージョンと同じ戦略を使用できます。ただし、**strategy** 属性は **when-full** 属性に置き換えられます。

```
<distributed-cache>
  <memory when-full="<eviction_strategy>" />
</distributed-cache>
```

エビクショナルゴリズム

Data Grid 7.2 では、エビクショナルゴリズムを設定する機能は、Low Inter-Reference Recency Set (LIRS) とともに非推奨になりました。

バージョン 7.2 以降、Data Grid には、TinyLFU と呼ばれる Least Frequently Used (LFU) キャッシュ置換アルゴリズムのバリエーションを実装する Caffeine キャッシングライブラリーが含まれています。オフヒープストレージの場合、Data Grid は Least Recently Used (LRU) アルゴリズムのカスタム実装を使用します。

3.2.4. エビクシヨン設定の比較

異なる Data Grid バージョン間でエビクシヨン設定を比較します。

オブジェクトストレージとエントリー数の排除

7.2 から 8.0

```
<memory>
  <object size="1000000" eviction="COUNT" strategy="REMOVE"/>
</memory>
```

8.1

```
<memory max-count="1MB" when-full="REMOVE"/>
```

オブジェクトストレージとメモリー量の排除

7.2 から 8.0

```
<memory>
  <object size="1000000" eviction="MEMORY" strategy="MANUAL"/>
</memory>
```

8.1

```
<memory max-size="1MB" when-full="MANUAL"/>
```

バイナリーストレージとエントリー数の排除

7.2 から 8.0

```
<memory>
  <binary size="500000000" eviction="MEMORY" strategy="EXCEPTION"/>
</memory>
```

8.1

```
<cache>
  <encoding media-type="application/x-protostream"/>
  <memory max-size="500 MB" when-full="EXCEPTION"/>
</cache>
```

バイナリーストレージとメモリー量の排除

7.2 から 8.0

```
<memory>  
  <binary size="500000000" eviction="COUNT" strategy="MANUAL"/>  
</memory>
```

8.1

```
<memory max-count="500 MB" when-full="MANUAL"/>
```

オフヒープストレージとエントリー数の排除

7.2 から 8.0

```
<memory>  
  <off-heap size="100000000" eviction="COUNT"/>  
</memory>
```

8.1

```
<memory storage="OFF_HEAP" max-count="10MB"/>
```

オフヒープストレージとメモリー量の排除

7.2 から 8.0

```
<memory>  
  <off-heap size="1000000000" eviction="MEMORY"/>  
</memory>
```

8.1

```
<memory storage="OFF_HEAP" max-size="1GB"/>
```

関連情報

- [Configuring Data Grid caches](#)
- [RHDG 7.3 以降の新しいエビクションポリシー TinyLFU \(Red Hat ナレッジベース\)](#)
- [Data Grid 7.2 の製品ドキュメント](#)

3.3. 有効期限の設定

有効期限は、ライフスパンまたは最大アイドル時間に基づいてキャッシュからエントリーを削除します。

設定を Data Grid 7.x から 8 に移行する場合、有効期限のために行う必要のある変更はありません。設定は同じままです。

ライフスパンの満了

```
<expiration lifespan="1000" />
```

最大アイドル有効期限

```
<expiration max-idle="1000" interval="120000" />
```

Data Grid 7.2 以前の場合、クラスター化されたキャッシュで **max-idle** を使用すると、技術的な制限があり、パフォーマンスが低下しました。

Data Grid 7.3 以降、クライアントが **max-idle** 有効期限値を持つエントリーを読み取ると、Data Grid はクラスター化されたキャッシュ内のすべての所有者にタッチコマンドを送信します。これにより、エントリーの相対アクセス時間がクラスター全体で同じになります。

Data Grid 8 は、クラスター間で **max-idle** の有効期限について同じタッチコマンドを送信します。ただし、**max-idle** の使用を開始する前に、考慮すべき技術的な考慮事項がいくつかあります。有効期限の仕組みの詳細と、タッチコマンドがクラスター化されたキャッシュのパフォーマンスにどのように影響するかを確認するには、**Configuring Data Grid caches** を参照してください。

関連情報

- [Configuring Data Grid caches](#)

3.4. 永続キャッシュストア

Data Grid 7.x と比較すると、Data Grid 8 のキャッシュストア設定にいくつかの変更があります。

永続性 SPI

Data Grid 8.1 では、キャッシュストア用の **NonBlockingStore** インターフェイスが導入されています。**NonBlockingStore** SPI は、呼び出し元のスレッドをブロックしてはならないメソッドを公開します。

データグリッドを永続データソースに接続するキャッシュストアは、**NonBlockingStore** インターフェイスを実装します。

ブロッキング操作を使用するカスタムキャッシュストア実装の場合、Data Grid はそれらの操作を処理するための **BlockingManager** ユーティリティークラスを提供します。

NonBlockingStore インターフェイスの導入により、以下のインターフェイスは非推奨になります。

- **CacheLoader**
- **CacheWriter**
- **AdvancedCacheLoader**
- **AdvancedCacheWriter**

カスタムキャッシュストア

Data Grid 8 では、以前のバージョンと同様に、**store** 要素を使用してカスタムキャッシュストアを設定できます。

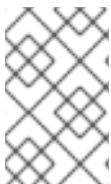
次の変更が適用されます。

- **singleton** 属性が削除されます。代わりに **shared=true** を使用してください。
- **segmented** 属性が追加され、デフォルトで **true** になります。

セグメント化されたキャッシュストア

Data Grid 8 の時点で、キャッシュストア設定はデフォルトで **segmented="true"** になり、次のキャッシュストア要素に適用されます。

- **store**
- **file-store**
- **string-keyed-jdbc-store**
- **jpa-store**
- **remote-store**
- **rocksdb-store**
- **soft-index-file-store**



注記

Data Grid 8.3 では、キャッシュ設定の **file-store** 要素によりソフトインデックスのファイルベースストアが作成されます。詳細は、[ファイルベースのキャッシュストアがデフォルト soft index に設定される](#) を参照してください。

単一ファイルキャッシュストア

シングルファイルキャッシュストアの **relative-to** 属性は、Data Grid 8 削除されています。キャッシュストア設定にこの属性が含まれている場合、Data Grid はそれを無視し、**path** 属性のみを使用してストアの場所を設定します。

JDBC キャッシュストア

JDBC キャッシュストアには、**xmlns** 名前空間宣言を含める必要があります。これは一部の Data Grid 7.x バージョンでは必要ありませんでした。

```
<persistence>
  <string-keyed-jdbc-store xmlns="urn:infinispan:config:store:jdbc:14.0" shared="true">
    ...
  </string-keyed-jdbc-store>
</persistence>
```

JDBC 接続ファクトリー

Data Grid 7.x JDBC キャッシュストアは、次の **ConnectionFactory** 実装を使用してデータベース接続を取得できます。

- **ManagedConnectionFactory**
- **SimpleConnectionFactory**
- **PooledConnectionFactory**

Data Grid 8 は、Red Hat JBoss EAP と同じ Agroal に基づく接続ファクトリーを使用してデータベースに接続するようになりました。**c3p0.properties** および **hikari.properties** ファイルを使用できなくなりました。



注記

Data Grid 8.3 JDBC 接続ファクトリーは **org.infinispan.persistence.jdbc.common.configuration** パッケージに含まれます。

セグメンテーション

今デフォルトでセグメンテーションを可能に JDBC 文字列ベースのキャッシュストアの設定は、次のプログラムの例のように、**segmentColumnName** と **segmentColumnType** パラメーターを含める必要があります。

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");
```

ライトビハインド

Write Behind モードの **thread-pool-size** 属性は、Data Grid 8 で削除されています。

キャッシュストアとローダーの削除

Data Grid 7.3 は、Data Grid 8 で使用できなくなった次のキャッシュストアとローダーを非推奨にします。

- Cassandra キャッシュストア
- REST キャッシュストア
- LevelDB キャッシュストア
- CLI キャッシュローダー

キャッシュストアマイグレーター

以前のバージョンのデータグリッドのキャッシュストアは、データグリッド 8 と互換性のないバイナリ形式でデータを保存します。

StoreMigrator ユーティリティを使用して、永続キャッシュストア内のデータを Data Grid 8 に移行します。

3.4.1. ファイルベースのキャッシュストアがデフォルト **soft index** に設定される

キャッシュ設定に **file-store** 永続性を含めると、単一ファイルキャッシュストア **SingleFileStore** ではなく、soft index ファイルベースのキャッシュストア **SoftIndexFileStore** が作成されるようになりました。Data Grid 8.2 以前では、**SingleFileStore** はファイルベースのキャッシュストアのデフォルトでした。

Data Grid 8.3 に移行またはアップグレードする場合、**file-store** 設定はサーバーの起動時に **SoftIndexFileStore** に自動的に変換されます。新しいストアとの互換性を確保するため、設定が **SoftIndexFileStore** に変換されると、その設定を変更せずに **SingleFileStore** に戻すことはできません。

3.4.1.1. 宣言型設定

Data Grid 8.2 以前

```
<persistence>
  <soft-index-file-store xmlns="urn:infinispan:config:soft-index:12.1">
    <index path="testCache/index" />
    <data path="testCache/data" />
  </soft-index-file-store>
</persistence>
```

Data Grid 8.3 以降

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

3.4.1.2. プログラムによる設定

Data Grid 8.2 以前

```
ConfigurationBuilder b = new ConfigurationBuilder();
b.persistence()
  .addStore(SoftIndexFileStoreConfigurationBuilder.class)
  .indexLocation("testCache/index");
  .dataLocation("testCache/data");
```

Data Grid 8.3 以降

```
ConfigurationBuilder b = new ConfigurationBuilder();
b.persistence()
  .addSoftIndexFileStore()
  .indexLocation("testCache/index")
  .dataLocation("testCache/data");
```

3.4.1.3. Data Grid 8.3 での単一ファイルキャッシュストアの使用

SingleFileStore キャッシュストアは、Data Grid 8.3 以降で設定できますが、Red Hat ではこれを推奨していません。**SoftIndexFileStore** キャッシュストアを使用するとスケーラビリティが向上するため、こちらを使用するようにしてください。

宣言的

```
<persistence passivation="false">
  <single-file-store shared="false"
    preload="true">
```

```

        fetch-state="true"
        read-only="false"/>
</persistence>

```

プログラマティック

```

ConfigurationBuilder b = new ConfigurationBuilder();
b.persistence()
    .addSingleFileStore();

```

3.5. DATA GRID クラスタートランスポート

Data Grid は、JGroups テクノロジーを使用して、クラスター化されたノード間の通信を処理します。

JGroups スタック設定要素と属性は、以前のデータグリッドバージョンから大幅に変更されていません。

以前のバージョンと同様に、Data Grid は、ネットワーク要件に最適化されたカスタムクラスタートランスポート設定を構築するための開始点として使用できる事前設定済みの JGroups スタックを提供します。同様に、Data Grid は、外部 XML ファイルで定義された JGroups スタックを **infinispan.xml** に追加する機能を提供します。

Data Grid 8 は、クラスタートランスポート設定を容易にするための使いやすさの向上をもたらしました。

- インラインスタックを使用すると、**jgroups** 要素を使用して **infinispan.xml** 内で直接 JGroups スタックを設定できます。
- **infinispan.xml** 内で JGroups スキーマを宣言します。
- UDP および TCP プロトコル用に事前設定された JGroups スタック
- JGroups スタックを拡張して、特定のプロトコルとプロパティを調整できるようにする継承属性

```

<infinispan
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:infinispan:config:14.0 https://infinispan.org/schemas/infinispan-config-14.0.xsd
                    urn:infinispan:server:14.0 https://infinispan.org/schemas/infinispan-server-14.0.xsd
                    urn:org:jgroups http://www.jgroups.org/schema/jgroups-4.2.xsd" ❶
  xmlns="urn:infinispan:config:14.0"
  xmlns:server="urn:infinispan:server:14.0">

  <jgroups> ❷
    <stack name="xsite" extends="udp"> ❸
      <relay.RELAY2 site="LON" xmlns="urn:org:jgroups"/>
      <remote-sites default-stack="tcp">
        <remote-site name="LON"/>
        <remote-site name="NYC"/>
      </remote-sites>
    </stack>
  </jgroups>

```

```
<cache-container ...>
...
</infinispan>
```

- 1 **infinispan.xml** 内で JGroups 4.2 スキーマを宣言します。
- 2 カスタムスタック定義を含む JGroups 要素を追加します。
- 3 クロスサイトレプリケーション用の JGroups プロトコルスタックを定義します。

3.5.1. トランスポートセキュリティー

以前のバージョンと同様に、Data Grid 8 は JGroups SYM_ENCRYPT および ASYM_ENCRYPT プロトコルを使用してクラスター通信を暗号化します。

Data Grid では、キーストアとトラストストアを TLS サーバーアイデンティティーとして含むセキュリティーレームを使用して、クラスタートランスポートをセキュアにすることもできます。以下に例を示します。

```
<cache-container>
<transport server:security-realm="tls-transport"/>
</cache-container>
```

ノード認証

Data Grid 7.x では、JGroups SASL プロトコルにより、ノードは組み込みサーバーとリモートサーバーの両方のインストールでセキュリティーレームに対して認証できます。

Data Grid 8 以降、セキュリティーレームに対してノード認証を設定することはできません。同様に、Data Grid 8 は、クラスター化されたノードの認証に JGroups AUTH プロトコルを使用することを推奨していません。

ただし、組み込みデータグリッドのインストールでは、JGroups クラスタートランスポートに **jgroups** 要素の一部として SASL 設定が含まれます。以前のバージョンと同様に、SASL 設定は、ノード認証に必要な特定の情報を取得するために、**CallbackHandlers** などの JAAS 概念に依存しています。

3.5.2. 再送信要求

Data Grid 8.2 は、デフォルトの JGroups スタック内の UNICAST3 および NAKACK2 プロトコルの再送信要求の設定を次のように変更します。

- **xmit_interval** プロパティーの値が 100 ミリ秒から 200 ミリ秒に増加します。
- **max_xmit_req_size** プロパティーは、UDP で最大 8500、TCP で最大 64000 ではなく、再送信要求ごとに最大 500 メッセージを設定するようになりました。

Data Grid 8 への移行の一環として、これらの推奨設定を使用するようにカスタム JGroups スタック設定を調整する必要があります。

関連情報

- [Data Grid Server ガイド](#)
- [Using Embedded Data Grid Caches](#)

- [Data Grid セキュリティーガイド](#)

3.6. DATA GRID 認証

Data Grid は、ロールベースのアクセス制御 (RBAC) を使用してデータへのアクセスを制限し、クラスターの暗号化を使用してノード間の通信を保護します。

ロールと権限

Data Grid 8.2 は、RBAC に使用できる一連のデフォルトのユーザーと権限を提供しますが、次の変更が加えられています。

- **ClusterRoleMapper** は、Data Grid がセキュリティープリンシパルを承認ロールに関連付けるために使用するデフォルトのメカニズムです。
- 新しい **MONITOR** 権限により、ユーザーは Data Grid 統計にアクセスできます。
- ユーザーがキャッシュやカウンターなどのリソースを作成および削除するために必要な新しい **CREATE** 権限。



注記

CREATE は、Data Grid 8.1 以前で Protobuf スキーマとサーバースクリプトを作成および削除するためにユーザーが必要とする `__schema_manager` および `__script_manager` ロールを置き換えます。

Data Grid 8.2 に移行するときは、Data Grid 8.1 以前で `__schema_manager` および `__script_manager` のロールを持っていたユーザーに、**deployer** ロールを割り当てる必要があります。次のようにコマンドラインインターフェイス (CLI) を使用します。

```
[[/containers/default]> user roles grant --roles=deployer <user>
```

Cache Manager のパーミッション

表3.2 Data Grid 8.1

パーミッション	機能	説明
設定	defineConfiguration	新しいキャッシュ設定を定義します。
LISTEN	addListener	キャッシュマネージャーに対してリスナーを登録します。
ライフサイクル	stop	キャッシュマネージャーを停止します。
ALL	-	すべてのキャッシュマネージャーのアクセス許可が含まれます。

表3.3 Data Grid 8.2

パーミッション	機能	説明
設定	defineConfiguration	新しいキャッシュ設定を定義します。
LISTEN	addListener	キャッシュマネージャーに対してリスナーを登録します。
ライフサイクル	stop	キャッシュマネージャーを停止します。
CREATE	createCache, removeCache	キャッシュ、カウンター、スキーマ、スクリプトなどのコンテナリソースを作成および削除することができます。
MONITOR	getStats	JMX 統計および metrics エンドポイントへのアクセスを許可します。
ALL	-	すべてのキャッシュマネージャーのアクセス許可が含まれます。

キャッシュ権限

表3.4 Data Grid 8.1

パーミッション	機能	説明
READ	get, contains	キャッシュからエントリーを取得します。
WRITE	put, putIfAbsent, replace, remove, evict	キャッシュ内のデータの書き込み、置換、削除、エビクト。
EXEC	distexec, streams	キャッシュに対するコードの実行を許可します。
LISTEN	addListener	キャッシュに対してリスナーを登録します。
BULK_READ	keySet, values, entrySet, query	一括取得操作を実行します。
BULK_WRITE	clear, putAll	一括書き込み操作を実行します。
ライフサイクル	start, stop	キャッシュを開始および停止します。

パーミッション	機能	説明
ADMIN	getVersion, addInterceptor*, removeInterceptor, getInterceptorChain, getEvictionManager, getComponentRegistry, getDistributionManager, getAuthorizationManager, evict, getRpcManager, getCacheConfiguration, getCacheManager, getInvocationContextContainer, setAvailability, getDataContainer, getStats, getXAResource	基盤となるコンポーネントと内部構造へのアクセスを許可します。
ALL	-	すべてのキャッシュパーミッションが含まれます。
ALL_READ	-	READ パーミッションと BULK_READ パーミッションを組み合わせます。
ALL_WRITE	-	WRITE パーミッションと BULK_WRITE パーミッションを組み合わせます。

表3.5 Data Grid 8.2

パーミッション	機能	説明
READ	get, contains	キャッシュからエントリーを取得します。
WRITE	put, putIfAbsent, replace, remove, evict	キャッシュ内のデータの書き込み、置換、削除、エビクト。
EXEC	distexec, streams	キャッシュに対するコードの実行を許可します。
LISTEN	addListener	キャッシュに対してリスナーを登録します。
BULK_READ	keySet, values, entrySet, query	一括取得操作を実行します。
BULK_WRITE	clear, putAll	一括書き込み操作を実行します。

パーミッション	機能	説明
ライフサイクル	start, stop	キャッシュを開始および停止します。
ADMIN	getVersion, addInterceptor*, removeInterceptor, getInterceptorChain, getEvictionManager, getComponentRegistry, getDistributionManager, getAuthorizationManager, evict, getRpcManager, getCacheConfiguration, getCacheManager, getInvocationContextContainer, setAvailability, getDataContainer, getStats, getXAResource	基盤となるコンポーネントと内部構造へのアクセスを許可します。
MONITOR	getStats	JMX 統計および metrics エンドポイントへのアクセスを許可します。
ALL	-	すべてのキャッシュパーミッションが含まれます。
ALL_READ	-	READ パーミッションと BULK_READ パーミッションを組み合わせます。
ALL_WRITE	-	WRITE パーミッションと BULK_WRITE パーミッションを組み合わせます。

キャッシュマネージャーの承認

Data Grid 8.2 以降では、次のように、**authentication** 要素を **cache-container** セキュリティー設定に含めることができます。

```
<infinispan>
  <cache-container name="secured">
    <security>
      <authorization/> ①
    </security>
  </cache-container>
</infinispan>
```

- ① デフォルトのロールと権限でキャッシュマネージャーのセキュリティ認証を有効にします。

次のようにグローバル認証設定を定義することもできます。


```

<infinispan>
  <cache-container default-cache="secured" name="secured">
    <security>
      <authorization> ❶
        <identity-role-mapper /> ❷
        <role name="admin" permissions="ALL" /> ❸
        <role name="reader" permissions="READ" />
        <role name="writer" permissions="WRITE" />
        <role name="supervisor" permissions="READ WRITE EXEC"/>
      </authorization>
    </security>
  </cache-container>
</infinispan>

```

- ❶ キャッシュマネージャーのライフサイクルを制御するには、ユーザー権限が必要です。
- ❷ ロールにプリンシパルをマップ **PrincipalRoleMapper** の実装を指定します。
- ❸ 一連のロールと関連する権限を定義します。

暗黙的なキャッシュ認証

Data Grid 8 は、キャッシュが **cache-container** から承認設定を継承できるようにすることで使いやすさを向上させるため、各キャッシュのロールとパーミッションを明示的に設定する必要はありません。

```

<local-cache name="secured">
  <security>
    <authorization/> ❶
  </security>
</local-cache>

```

- ❶ キャッシュコンテナで定義されたロールと権限を使用します。

Data Grid 8.2 以降、設定に **authorization** 要素を含めると、カスタムグローバル権限のセットを定義しない限り、デフォルトのロールと権限を使用して、そのキャッシュへのアクセスが制限されます。

関連情報

- [Data Grid セキュリティーガイド](#)

第4章 DATA GRID 8 API への移行

Data Grid 8 への移行に影響を与える Data Grid API への変更を見つけます。

API の非推奨と削除

このセクションの詳細に加えて、API の非推奨と削除も確認する必要があります。

[データグリッドの非推奨の機能と機能](#) (Red Hat ナレッジベース) を参照してください。

4.1. REST API

Data Grid 7.x は、Data Grid 8 で REST API v2 に置き換えられた REST API v1 を使用していました。

REST API v2 のデフォルトのコンテキストパスは `<server_hostname>:11222/rest/v2/` です。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。

performAsync ヘッダーも REST エンドポイントから削除されました。REST エンドポイントで非同期操作を実行するクライアントは、ブロックを回避するために、自分の側で要求と応答を管理する必要があります。

REST 操作の **PUT**、**POST**、および **DELETE** メソッドは、リクエストがリソースを返さない場合、**200** ではなくステータス **204** (コンテンツなし) を返すようになりました。

関連情報

- [Data Grid REST API](#)

4.1.1. 8.3 における REST API の変更

Data Grid 8.3 には、以下の REST API への変更が含まれています。

キャッシュの再インデックス設定

Data Grid キャッシュのインデックスを再作成するための **mass-index** 操作は非推奨になりました。以下の例のように、代わりに **reindex** を使用するようにクライアントを更新します。

```
/v2/caches/<cacheName>/search/indexes?action=reindex
```

ローリングアップグレード操作

以下の操作が非推奨になりました。

```
POST /v2/caches/<cacheName>?action=disconnect-source
```

代わりに **source-connection** 操作を使用してください。

```
DELETE /v2/caches/<cacheName>/rolling-upgrade/source-connection
```

4.2. クエリー API

Data Grid 8 は、より使いやすく、より軽量なデザインの更新された Query API をもたらします。Data Grid 7.x と比較して、分散キャッシュ内の値全体を検索すると、より効率的なクエリーパフォーマンスが得られ、より良い結果が得られます。



注記

Data Grid 8 Query API はかなりのリファクタリングを経ているため、現在非推奨となっているいくつかの機能と機能リソースがあります。

このトピックでは、以前のバージョンから移行するときに設定に加える必要のある変更
に焦点を当てます。これらの変更には、廃止されたすべてのインターフェイス、メソ
ッド、またはその他の設定を削除する計画を含める必要があります。

非推奨の機能の完全なリストについては、[Data Grid の非推奨と削除](#) (Red Hat ナレッジ
ベース) を参照してください。

Data Grid キャッシュのインデックス作成

Data Grid Lucene Directory、**InfinispanIndexManager** および **AffinityIndexManager** インデックスマ
ネージャー、および Hibernate Search 用の Infinispan Directory プロバイダーは、8.0 で非推奨になり、
8.1 で削除されました。

auto-config 属性は 8.1 で非推奨になり、削除される予定です。

インデックスモード設定を設定する **index()** メソッドは非推奨になりました。設定でインデックス作成
を有効にすると、Data Grid はインデックス作成を管理するための最良の方法を自動的に選択します。



重要

いくつかのインデックス設定値はサポートされなくなり、それらを含めると致命的な設
定エラーが発生します。

設定に次の変更を加える必要があります。

- **.indexing().index(Index.NONE)** を **indexing().enabled(false)** に変更
- 他のすべての列挙値を次のように変更します。 **indexing().enabled(true)**

宣言的に、設定に他のインデックス設定要素が含まれている場合は、**enabled="true"** を指定する必要
はありません。ただし、プログラムでインデックスを設定する場合は、**enabled()** メソッドを呼び出す
必要があります。同様に、JSON 形式のデータグリッド設定では、インデックスを明示的に有効にする
必要があります。次に例を示します。

```
"indexing": {
  "enabled": "true"
  ...
},
```

インデックス付きタイプ

インデックス設定ですべてのインデックス付きタイプを宣言する必要があります。宣言されていないタ
イプがインデックス付きキャッシュで使用される場合、Data Grid は警告メッセージをログに記録しま
す。この要件は、Java クラスと Protobuf タイプの両方に適用されます。

Data Grid 8 でのインデックス作成の有効化

- 宣言的に

```
<distributed-cache name="my-cache">
  <indexing>
```

```

<indexed-entities>
  <indexed-entity>com.acme.query.test.Car</indexed-entity>
  <indexed-entity>com.acme.query.test.Truck</indexed-entity>
</indexed-entities>
</indexing>
</distributed-cache>

```

- プログラムで

```

import org.infinispan.configuration.cache.*;

ConfigurationBuilder config=new ConfigurationBuilder();
config.indexing().enable().addIndexedEntity(Car.class).addIndexedEntity(Truck.class);

```

Querying values in caches

org.infinispan.query.SearchManager インターフェイスは Data Grid 8 で非推奨になり、Lucene および Hibernate Search のネイティブオブジェクトをサポートしなくなりました。

削除されたメソッド

- Lucene クエリーを受け取る **.getQuery()** メソッド。代わりに、**org.infinispan.query.Search** エントリーポイントから Ickle クエリーを取得する別のメソッドを使用してください。同様に、**.getQuery()** の呼び出し時に、複数のターゲットエンティティクラスを指定できなくなりました。Ickle クエリー文字列は、代わりにエンティティを提供します。
- Hibernate Search クエリーを直接構築する **.buildQueryBuilderForClass()**。代わりに Ickle クエリーを使用してください。

org.infinispan.query.CacheQuery インターフェイスも非推奨になりました。代わりに、**org.infinispan.query.dsl.Query** インターフェイスを **Search.getQueryFactory()** メソッドから取得する必要があります。

org.infinispan.query.dsl.Query のインスタンスはクエリー結果をキャッシュしなくなり、**list()** などのメソッドを呼び出すときにクエリーを再実行することができることに注意してください。

エンティティマッピング

今後は、すべての場合において、**@SortableField** で並べ替えが必要なフィールドにアノテーションを付ける必要があります。

関連情報

- [Data Grid Query API](#)
- [Data Grid の非推奨と削除](#)

4.2.1. 8.2 での Query API の変更

Data Grid は、Hibernate および Apache Lucene ライブラリーをアップグレードして、Query API のパフォーマンスおよび機能を改善します。このアップグレードの一環として、Data Grid は新しいインデックス作成機能を導入し、いくつかの Hibernate および Lucene アノテーションを削除します。

クエリーの統計

Data Grid 8.2 は、以下のようにキャッシュ設定で統計を宣言的に有効にした場合にのみ、クエリーおよびインデックスの統計を公開します。

```
<replicated-cache name="myReplicatedCache" statistics="true">
  <!-- Cache configuration goes here. -->
</replicated-cache>
```

JMX 経由でのクエリーおよびインデックスの統計を有効にすることはできなくなりました。

Data Grid キャッシュのインデックス作成

インデックス型の宣言

Data Grid 8.1 では、インデックス設定で宣言されていないタイプが許可されていました。Data Grid 8.2 以降、設定ですべてのインデックス付きタイプを宣言する必要があります。この要件は、Java クラスと Protobuf タイプの両方に適用されます。インデックス付きタイプの宣言の詳細については、8.1 移行の詳細を参照してください。

インデックスマネージャー

Data Grid 8.2 は、デフォルトのインデックスマネージャーとして **near-real-time** を使用し、設定を必要としなくなりました。

- Data Grid 8.1:

```
<indexing>
  <property name="default.indexmanager">near-real-time</property>
</indexing>
```

- Data Grid 8.2:

```
<indexing enabled="true"/>
```

インデックスリーダーおよびライター

Data Grid 8.2 では、インデックスを作成するための内部コンポーネントであるインデックスリーダーとインデックスライターが導入されています。

設定を適応させるには、次のことを行う必要があります。

1. **property** 要素または **.addProperty()** メソッドを使用するインデックス設定を削除します。
2. 次のいずれかの方法でインデックス作成動作を設定します。
 - 宣言的: **<index-reader>** および **<index-writer>** 要素を追加します。
 - プログラム: **builder.indexing().reader()** と **builder.indexing().writer()** メソッドを追加します。

リーダーの更新

8.2 で追加された **refresh-interval** 属性を使用して、インデックスリーダーの更新期間を設定します。

- Data Grid 8.1:

```
<indexing>
  <property name="default.reader.async_refresh_period_ms">1000</property>
</indexing>
```

- Data Grid 8.2:

```
<indexing>
  <index-reader refresh-interval="1000"/>
</indexing>
```

ライターのコミット間隔

8.2 で追加された **commit-interval** 属性を使用して、インデックスライターがインデックスストレージにコミットする間隔を設定します。Data Grid 8.2 では、インデックス作成はデフォルトで非同期であり、**default.worker.execution** プロパティは使用されなくなりました。

- Data Grid 8.1:

```
<indexing>
  <property name="default.worker.execution">async</property>
  <property name="default.index_flush_interval">500</property>
</indexing>
```

- Data Grid 8.2:

```
<indexing>
  <index-writer commit-interval="500"/>
</indexing>
```

Lucene インデックスチューニングプロパティ

Data Grid 8.2 は、**ram-buffer-size** 属性と、インデックスを調整するためのプロパティを置き換える **factor** 属性と **max-size** 属性を持つ **index-merge** 要素を追加します。

- Data Grid 8.1:

```
<indexing>
  <property name="default.indexwriter.merge_factor">30</property>
  <property name="default.indexwriter.merge_max_size">1024</property>
  <property name="default.indexwriter.ram_buffer_size">256</property>
</indexing>
```

- Data Grid 8.2:

```
<indexing>
  <index-writer ram-buffer-size="256">
    <index-merge factor="30" max-size="1024"/>
  </index-writer>
</indexing>
```

インデックスストレージ

Data Grid 8.2 には、以前のバージョンの **property** 要素設定を置き換える **storage** 属性が含まれています。**storage** 属性を使用すると、インデックスを JVM ヒープに格納するかホストファイルシステムに格納するかを設定できます。

ファイルシステムストレージ

- Data Grid 8.1:

```
<indexing>
```

```
<property name="default.directory_provider">filesystem</property>
<property name="default.indexBase">${java.io.tmpdir}/baseDir</property>
</indexing>
```

- Data Grid 8.2:

```
<indexing storage="filesystem" path="${java.io.tmpdir}/baseDir"/>
```

JVM ヒープストレージ

- Data Grid 8.1:

```
<indexing>
<property name="default.directory_provider">local-heap</property>
</indexing>
```

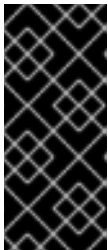
- Data Grid 8.2:

```
<indexing storage="local-heap">
</indexing>
```

インデックスプロパティの適応

インデックス設定を Data Grid 8.2 に移行するときは、次の変更も行う必要があります。

- **lucene_version** プロパティを削除します。



重要

Data Grid 8.2 で古い Lucene バージョンで作成したインデックスを使用しないでください。

インデックス設定を調整した後、Data Grid を初めて起動するときにインデックスを再構築して、Data Grid 8.2 への移行を完了する必要があります。

- **default.sharding_strategy.nbr_of_shards** プロパティを削除します。
このプロパティは、置き換えなしで非推奨になりました。
- **infinispan.query.lucene.max-boolean-clauses** プロパティを削除します。
Data Grid 8.2 以降、これを JVM プロパティとして設定する必要があります。

Hibernate および Lucene アノテーション

@Field、@Indexed、@SortableField などの Hibernate および Lucene アノテーションの移行については、[Hibernate Search Migration Guide](#) の [Annotation mapping](#) セクションを参照してください。

関連情報

- [Data Grid Query API](#)
- [Data Grid の非推奨と削除](#)
- [Hibernate Search 移行ガイド: アノテーションマッピング](#)

4.2.2. 8.3 での Query API の変更

Data Grid 8.3 は **IndexedQueryMode** パラメーターを削除します。Data Grid は、キャッシュのクエリーに最適なモードを自動的に検出し、以前のバージョンでは任意のパラメーターを無視していました。

関連情報

- [Data Grid キャッシュのクエリー](#)
- [Data Grid Query API](#)
- [Data Grid の非推奨と削除](#)

4.2.3. 8.4 での Query API の変更

Data Grid のネイティブアノテーション

Data Grid 8.4 では、**@Indexed**、**@Basic**、**@Decimal**、**@Keyword**、**@Text**、および **@Embedded** という新しいインデックスアノテーションが導入されています。各アノテーションは、エンティティーのインデックス付け方法を詳しく記述するために使用できる一連の属性をサポートしています。

これらの新しいアノテーションは、Hibernate Search アノテーションに代わるものとなりました。これにより、Java クラスにリモートキャッシュ用の **@ProtoDoc** アノテーションを付ける必要がなくなりました。すべてのアノテーションは、生成された **.proto** ファイルにコメントとしてコピーされます。

次の表は、Hibernate Search 5 (HS5) アノテーションと Data Grid ネイティブアノテーション間のフィールドのマッピングをまとめたものです。

HS5 アノテーション	インデックス属性	Data Grid のネイティブアノテーション	説明
@Field(index=Index.YES)	searchable	@Basic、@Decimal、@Keyword、@Text	以前はインデックス作成対象とマークされていたフィールドが searchable になりました。
@Field(store = Store.YES)	projectable = true	@Basic、@Decimal、@Keyword、@Text	以前は格納対象とマークされていたフィールドが projectable になりました。
type String && @Field(analyze = Analyze.YES)	analyzer = "<definition>"	@Text	アナライザー定義でマークされた文字列フィールドは、インデックス作成中に引き続き分析されます。
@Field(analyze = Analyze.NO) && (@Field(store = Store.YES) OR @Field(sortable = Sortable.YES))	sortable = true	@Basic、@Decimal、@Keyword	分析されなかったものの、インデックスに格納されるか、sortable とマークされていたフィールドは、sortable になりました。

HS5 アノテーション	インデックス属性	Data Grid のネイティブ アノテーション	説明
該当なし	aggregable = true	@Basic、@Decimal、@Keyword	Hibernate Search 5 アノテーションを使用して集計操作を実行することはできませんでした。
該当なし	normalizer = "lowercase"	@Keyword	プロセス中にデータが失われる可能性があるため、分析または正規化されたフィールドをマッピングすることはできません。

クエリー効率

default-max-results キャッシュプロパティを使用して、クエリーインスタンスに対して返される結果の数を制限できます。**default-max-results** のデフォルト値は 100 です。クエリーによって返される結果の数を制限すると、明示的な制限が設定されていないクエリーのパフォーマンスが大幅に向上します。

関連情報

- [Data Grid キャッシュのクエリー](#)
- [Data Grid Query API](#)

第5章 アプリケーションの DATA GRID 8 への移行

5.1. DATA GRID 8 でのマーシャリング

マーシャリング機能は、Data Grid 8 で大幅にリファクタリングされ、内部オブジェクトとユーザーオブジェクトを分離します。

Data Grid が内部クラスのマーシャリングを処理するようになったため、埋め込みキャッシュまたはリモートキャッシュを使用してマーシャラーを設定するときに、これらの内部クラスを処理する必要がなくなりました。

5.1.1. ProtoStream マーシャリング

デフォルトでは、Data Grid 8 は ProtoStream API を使用して、言語に依存しない下位互換性のある形式である Protocol Buffers としてデータをマーシャリングします。

Protobuf エンコーディングはスキーマ定義のフォーマットであり、現在多くのアプリケーションのデフォルト標準であり、Data Grid 7 のデフォルトであった JBoss Marshalling と比較してデータをトランスコードする際の柔軟性が高くなっています。

ProtoStream マーシャラーは Protobuf 形式に基づいているため、Data Grid は最初に Java オブジェクトに変換せずに他のエンコーディングに変換できます。JBoss Marshalling を使用する場合、他の形式に変換する前に、キーと値を Java オブジェクトに変換する必要があります。

Data Grid 8 への移行の一環として、Java クラスに ProtoStream マーシャリングの使用を開始する必要があります。

高レベルから、ProtoStream マーシャラーを使用するには、ProtoStream プロセッサーを使用して **SerializationContextInitializer** 実装を生成します。まず、**@Proto** アノテーションを Java クラスに追加してから、Data Grid が提供する ProtoStream プロセッサーを使用して、以下を含むシリアル化コンテキストを生成します。

- 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。

埋め込みキャッシュとリモートキャッシュのどちらを使用するかに応じて、Data Grid は **SerializationContextInitializer** 実装を自動的に登録できます。

ネストされた ProtoStream アノテーション

Data Grid 8.2 は ProtoStream 4.4.0.Final にアップグレードします。これには、場合によっては移行が必要です。

以前のバージョンでは、ProtoStream API は、メッセージが最上位としてのみ生成されたメッセージタイプを正しくネストしませんでした。

永続キャッシュストアに Protobuf でエンコードされたエントリーがある場合は、ProtoStream アノテーションがトップレベルになるように Java クラスを変更する必要があります。これにより、永続化されたメッセージのネストが Java クラスのネストと一致することが保証されます。一致しない場合、データの非互換性の問題が発生する可能性があります。

たとえば、次のようなネストされた Java クラスがある場合:

```
class OuterClass {
```

```
class InnerClass {
    @ProtoField(1)
    int someMethod() {
    }
}
```

InnerClass が **OuterClass** の子ではなくなるように、クラスを調整する必要があります。

```
class InnerClass {
    @ProtoField(1)
    int someMethod() {
    }
}
```

Data Grid サーバーとのマーシャリング

リモートキャッシュには Protobuf エンコーディングのみを使用し、カスタムタイプには ProtoStream マーシャラーを組み合わせて使用する必要があります。

JBoss マーシャリングなどの他のマーシャラー実装では、Data Grid CLI、Data Grid Console、または Ickle クエリーと互換性のない異なるキャッシュエンコーディングを使用する必要があります。

キャッシュストアと ProtoStream

Data Grid 7.x では、キャッシュストアに永続化するデータは、Data Grid 8 の ProtoStream マーシャラーと互換性がありません。データを Data Grid 7.x キャッシュストアから Data Grid 8 キャッシュストアに移行するには、**StoreMigrator** ユーティリティを使用する必要があります。

5.1.2. 代替マーシャラーの実装

Data Grid は、ProtoStream の代替マーシャラー実装を提供し、古いバージョンからの移行を容易にします。これらの代替マーシャラーは、ProtoStream マーシャリングに移行する際の暫定的な解決策としてのみ使用する必要があります。



注記

新しいプロジェクトの場合、将来のアップグレードや移行に関する問題を回避するために、ProtoStream マーシャリングのみを使用することを強く推奨します。

デシリアライズ許可リスト

Red Hat では、多様性を受け入れる用語の使用への取り組みに努めており、Java クラスのシリアル化を設定する際に使用するホワイトリストという用語を許可リストに変更しています。

Data Grid 8.1

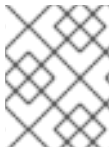
```
<cache-container>
  <serialization>
    <white-list>
      <class>org.infinispan.test.data.Person</class>
      <regex>org.infinispan.test.data.*</regex>
    </white-list>
  </serialization>
</cache-container>
```

Data Grid 8.2

```
<cache-container>
  <serialization>
    <allow-list>
      <class>org.infinispan.test.data.Person</class>
      <regex>org.infinispan.test.data.*</regex>
    </allow-list>
  </serialization>
</cache-container>
```

JBoss marshalling

Data Grid 7 では、JBoss Marshalling がデフォルトのマーシャラーです。Data Grid 8 では、ProtoStream マーシャリングがデフォルトです。



注記

Java シリアル化を使用するクライアント要件がある場合は、JBoss Marshalling の代わりに **JavaSerializationMarshaller** を使用する必要があります。

Data Grid 8 への移行中に一時的な解決策として JBoss Marshalling を使用する必要がある場合は、以下を実行します。

埋め込みキャッシュ

1. **infinispan-jboss-marshalling** 依存関係をクラスパスに追加します。
2. 次に、**JBossUserMarshaller** を使用するようにデータグリッドを設定します。

```
<serialization marshaller="org.infinispan.jboss.marshalling.core.JBossUserMarshaller"/>
```

3. Data Grid が逆シリアル化を許可するクラスのリストにクラスを追加します。

リモートキャッシュ

Data Grid Server は JBoss Marshalling をサポートしておらず、**infinispan-jboss-marshalling** モジュールがクラスパス上にある場合、**GenericJBossMarshaller** は自動的に設定されなくなりました。

次のように、JBoss Marshalling を使用するように Hot Rod Java クライアントを設定する必要があります。

- **RemoteCacheManager**

```
.marshaller("org.infinispan.jboss.marshalling.common.GenericJBossMarshaller");
```

- **hotrod-client.properties**

```
infinispan.client.hotrod.marshaller = GenericJBossMarshaller
```

関連情報

- [キャッシュのエンコードとマーシャリング](#)

5.2. AUTOPROTOSCHEMABUILDER アノテーションへのアプリケーションの移行

以前のバージョンの Data Grid は、ProtoStream API の **MessageMarshaller** インターフェイスを使用してマーシャリングを設定します。

MessageMarshaller API と **ProtoSchemaBuilder** アノテーションはどちらも、ProtoStream4.3.4 に対応する Data Grid 8.1.1 で非推奨になりました。

MessageMarshaller インターフェイスの使用には、以下のいずれかが含まれます。

- Protobuf スキーマを手動で作成します。
- **ProtoSchemaBuilder** アノテーションを Java クラスに追加してから、Protobuf スキーマを生成します。

ただし、ProtoStream マーシャリングを設定するためのこれらの手法は、Data Grid 8.1.1 以降で使用可能な **AutoProtoSchemaBuilder** アノテーションほど効率的で信頼性が高くありません。**AutoProtoSchemaBuilder** アノテーションを Java クラスに追加し、Protobuf スキーマと関連するマーシャラーを含む **SerializationContextInitializer** 実装を生成するだけです。

Red Hat は、**AutoProtoSchemaBuilder** アノテーションの使用を開始して、ProtoStream マーシャラーから最良の結果を取得することを推奨します。

次のコード例は、アプリケーションを **MessageMarshaller** API から **AutoProtoSchemaBuilder** アノテーションに移行する方法を示しています。

5.2.1. 基本的な MessageMarshaller の実装

この例には、デフォルト以外のタイプを使用するいくつかのフィールドが含まれています。コードジェネレーターはデフォルトで **int** 型を使用するため、**text** フィールドの順序は異なり、**fixed32** フィールドは生成された Protobuf スキーマタイプと競合します。

SimpleEntry.java

```
public class SimpleEntry {
    private String description;
    private Collection<String> text;
    private int intDefault;
    private Integer fixed32;

    // public Getter, Setter, equals and hashCode methods omitted for brevity
}
```

SimpleEntryMarshaller.java

```
import org.infinispan.protostream.MessageMarshaller;

public class SimpleEntryMarshaller implements MessageMarshaller<SimpleEntry> {

    @Override
    public void writeTo(ProtoStreamWriter writer, SimpleEntry testEntry) throws IOException {
        writer.writeString("description", testEntry.getDescription());
        writer.writeInt("intDefault", testEntry.getIntDefault());
    }
}
```

```

writer.writeInt("fix32", testEntry.getFixed32());
writer.writeCollection("text", testEntry.getText(), String.class);
}

@Override
public SimpleEntry readFrom(MessageMarshaller.ProtoStreamReader reader) throws IOException {
    SimpleEntry x = new SimpleEntry();

    x.setDescription(reader.readString("description"));
    x.setIntDefault(reader.readInt("intDefault"));
    x.setFixed32(reader.readInt("fix32"));
    x.setText(reader.readCollection("text", new LinkedList<String>(), String.class));

    return x;
}
}

```

結果の Protobuf スキーマ

```

syntax = "proto2";

package example;

message SimpleEntry {
    required string description = 1;
    optional int32 intDefault = 2;
    optional fixed32 fix32 = 3;
    repeated string text = 4;
}

```

AutoProtoSchemaBuilder アノテーションに移行

SimpleEntry.java

```

import org.infinispan.protostream.annotations.ProtoField;
import org.infinispan.protostream.descriptors.Type;

public class SimpleEntry {

    private String description;
    private Collection<String> text;
    private int intDefault;
    private Integer fixed32;

    @ProtoField(number = 1)
    public String getDescription() {...}

    @ProtoField(number = 4, collectionImplementation = LinkedList.class)
    public Collection<String> getText() {...}

    @ProtoField(number = 2, defaultValue = "0")
    public int getIntDefault() {...}

    @ProtoField(number = 3, type = Type.FIXED32)
    public Integer getFixed32() {...}
}

```

```
// public Getter, Setter, equals and hashCode methods and convenient constructors omitted for
brevity
}
```

SimpleEntryInitializer.java

```
import org.infinispan.protostream.GeneratedSchema;
import org.infinispan.protostream.annotations.AutoProtoSchemaBuilder;

@AutoProtoSchemaBuilder(includeClasses = { SimpleEntry.class }, schemaFileName =
"simple.proto", schemaFilePath = "proto", schemaPackageName = "example")
public interface SimpleEntryInitializer extends GeneratedSchema {
}
```

重要な所見

- **Field 2** は、以前のバージョンの ProtoStream マーシャラーがチェックしなかった **int** として定義されています。
- Java **int** フィールドは null 許容ではないため、ProtoStream プロセッサは失敗します。Java **int** フィールドは、**required** または **defaultValue** で初期化する必要があります。Java アプリケーションの観点からは、**int** フィールドは "0" で初期化されるため、put 操作で設定されるため、影響を与えることなく **defaultValue** を使用できます。**required** への変更は、常に存在する場合、保存されたデータの観点からは問題ではありませんが、さまざまなクライアントで問題が発生する可能性があります。
- 互換性を確立するため、**Field 3** は、**Type.FIXED32** に明示的に設定する必要があります。
- テキストコレクションは、結果の Protobuf スキーマに対して正しい順序で設定する必要があります。

重要

Protobuf スキーマのテキストコレクションの順序は、移行の前後で同じである必要があります。同様に、移行中に **fixed32** タイプを設定する必要があります。

そうでない場合、クライアントアプリケーションは次の例外を出力し、起動に失敗する可能性があります。

```
Exception ( ISPN004034: Unable to unmarshall bytes )
```

また、キャッシュされたデータに不完全または不正確な結果が表示される場合もあります。

5.2.2. カスタムタイプを使用した MessageMarshaller の実装

このセクションでは、ProtoStream がネイティブに処理しないフィールドを含む **MessageMarshaller** 実装の移行例を示します。

次の例では **BigInteger** クラスを使用していますが、データグリッドアダプターやカスタムクラスを含め、すべてのクラスに適用されます。



注記

BigInteger クラスは不変であるため、引数のないコンストラクターはありません。

CustomTypeEntry.java

```
import java.math.BigInteger;

public class CustomTypeEntry {

    final String description;
    final BigInteger bigInt;

    // public Getter, Setter, equals and hashCode methods and convenient constructors omitted for
    // brevity
}
```

CustomTypeEntryMarshaller.java

```
import org.infinispan.protostream.MessageMarshaller;

public class CustomTypeEntryMarshaller implements MessageMarshaller<CustomTypeEntry> {

    @Override
    public void writeTo(ProtoStreamWriter writer, CustomTypeEntry testEntry) throws IOException {
        writer.writeString("description", testEntry.description);
        writer.writeString("bigInt", testEntry.bigInt.toString());
    }

    @Override
    public CustomTypeEntry readFrom(MessageMarshaller.ProtoStreamReader reader) throws
    IOException {
        final String desc = reader.readString("description");
        final BigInteger blnt = new BigInteger(reader.readString("bigInt"));

        return new CustomTypeEntry(desc, blnt);
    }
}
```

CustomTypeEntry.proto

```
syntax = "proto2";

package example;

message CustomTypeEntry {
    required string description = 1;
    required string bigInt = 2;
}
```

アダプタークラスを使用して移行されたコード

ProtoAdapter アノテーションを使用して、**MessageMarshaller** 実装で作成した Protobuf スキーマと互換性のある Protobuf スキーマを生成する方法で **CustomType** クラスをマーシャリングできます。

このアプローチでは、次のことができます。

- **CustomTypeEntry** クラスにアノテーションを追加してはなりません。
- **@ProtoAdapter** アノテーションを使用して Protobuf スキーマとマーシャラーの生成方法を制御する **CustomTypeEntryAdapter** クラスを作成します。
- **@AutoProtoSchemaBuilder** アノテーションとともに **CustomTypeEntryAdapter** クラスを含めます。



注記

AutoProtoSchemaBuilder アノテーションは **CustomTypeEntry** クラスを参照しないため、そのクラスに含まれるアノテーションはすべて無視されます。

次の例が示す **CustomTypeEntry** クラスの ProtoStream アノテーションを含む **CustomTypeEntryAdapter** クラス:

CustomTypeEntryAdapter.java

```
import java.math.BigInteger;

import org.infinispan.protostream.annotations.ProtoAdapter;
import org.infinispan.protostream.annotations.ProtoFactory;
import org.infinispan.protostream.annotations.ProtoField;

@ProtoAdapter(CustomTypeEntry.class)
public class CustomTypeEntryAdapter {

    @ProtoFactory
    public CustomTypeEntry create(String description, String bigInt) {
        return new CustomTypeEntry(description, new BigInteger(bigInt));
    }

    @ProtoField(number = 1, required = true)
    public String getDescription(CustomTypeEntry t) {
        return t.description;
    }

    @ProtoField(number = 2, required = true)
    public String getBigInt(CustomTypeEntry t) {
        return t.bigInt.toString();
    }
}
```

次の例が示す **CustomTypeEntryAdapter** クラスを参照する **AutoProtoSchemaBuilder** アノテーションと **t** もに **SerializationContextInitializer** を示しています。

CustomTypeEntryInitializer.java

```
import org.infinispan.protostream.GeneratedSchema;
import org.infinispan.protostream.annotations.AutoProtoSchemaBuilder;

@AutoProtoSchemaBuilder(includeClasses = { CustomTypeEntryAdapter.class },
```

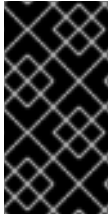
```

schemaFileName = "custom.proto",
schemaFilePath = "proto",
schemaPackageName = "example")
public interface CustomTypeAdapterInitializer extends GeneratedSchema { }

```

アダプタークラスなしで移行されたコード

アダプタークラスを作成する代わりに、ProtoStream アノテーションを **CustomTypeEntry** クラスに直接追加できます。



重要

この例では、生成された Protobuf スキーマは、**BigInteger** が別個のメッセージであるため、**MessageMarshaller** インターフェイスを介して追加されたキャッシュ内のデータと互換性がありません。アダプターフィールドに同じ文字列が書き込まれていても、データのマーシャリングを解除することはできません。

次の例が示す直接 ProtoStream アノテーションを含む **CustomTypeEntry** クラス:

CustomTypeEntry.java

```

import java.math.BigInteger;

public class CustomTypeEntry {

    @ProtoField(number = 1)
    final String description;
    @ProtoField(number = 2)
    final BigInteger bigInt;

    @ProtoFactory
    public CustomTypeEntry(String description, BigInteger bigInt) {
        this.description = description;
        this.bigInt = bigInt;
    }

    // public Getter, Setter, equals and hashCode methods and convenient constructors omitted for
    // brevity
}

```

以下の例は、**CustomTypeEntry** および **BigIntegerAdapter** クラスを参照する **AutoProtoSchemaBuilder** アノテーションと共に **SerializationContextInitializer** を示しています。

CustomTypeEntryInitializer.java

```

import org.infinispan.protostream.GeneratedSchema;
import org.infinispan.protostream.annotations.AutoProtoSchemaBuilder;
import org.infinispan.protostream.types.java.math.BigIntegerAdapter;

@AutoProtoSchemaBuilder(includeClasses = { CustomTypeEntry.class,
    BigIntegerAdapter.class },
    schemaFileName = "customtype.proto",

```

```
    schemaFilePath = "proto",  
    schemaPackageName = "example")  
public interface CustomTypeInitializer extends GeneratedSchema { }
```

前述の **SerializationContextInitializer** 実装から Protobuf スキーマを生成すると、以下の Protobuf スキーマになります。

CustomTypeEntry.proto

```
syntax = "proto2";  
  
package example;  
  
message BigInteger {  
    optional bytes bytes = 1;  
}  
  
message CustomTypeEntry {  
    optional string description = 1;  
    optional BigInteger bigInt = 2;  
}
```

第6章 RED HAT OPENSIFT での DATA GRID クラスターの移行

Red Hat OpenShift で実行されている Data Grid クラスターの移行の詳細を確認してください。

6.1. OPENSIFT の DATA GRID

Data Grid 8 は、運用インテリジェンスを提供し、OpenShift に Data Grid をデプロイするための管理の複雑さを軽減する Data Grid Operator を導入しています。Data Grid 8 では、Data Grid Operator は認証、クライアントキーストア、外部ネットワークアクセス、ロギングなどの Data Grid クラスターのほとんどの設定を処理します。

Data Grid 8.3 では、OpenShift に Data Grid クラスターをデプロイするための Helm チャートが導入されました。Data Grid チャートは、Data Grid Operator が管理するクラスターをデプロイすることができない場合や、Data Grid クラスターを手動で設定、デプロイメント、および管理する必要のあるシナリオの代替手段を提供します。

Data Grid サービスの作成

Data Grid 7.3 では、OpenShift で Data Grid クラスターを作成するための Cache サービスおよび Data Grid サービスが導入されました。

Data Grid 7.3 でこれらのサービスを作成するには、必要に応じてサービステンプレートをインポートしてから、テンプレートパラメーターと環境変数を使用してサービスを設定します。

7.3 でのキャッシュサービスノードの作成

```
$ oc new-app cache-service \
  -p APPLICATION_USER=${USERNAME} \
  -p APPLICATION_PASSWORD=${PASSWORD} \
  -p NUMBER_OF_INSTANCES=3 \
  -p REPLICATION_FACTOR=2
```

7.3 での Data Grid サービスノードの作成

```
$ oc new-app datagrid-service \
  -p APPLICATION_USER=${USERNAME} \
  -p APPLICATION_PASSWORD=${PASSWORD} \
  -p NUMBER_OF_INSTANCES=3
  -e AB_PROMETHEUS_ENABLE=true
```

Data Grid 8 でのサービスの作成

1. Data Grid Operator サブスクリプションを作成します。
2. **Infinispan** カスタムリソース (CR) を作成して、Data Grid クラスターをインスタンス化し、設定します。

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-infinispan
spec:
  replicas: 2
  service:
    type: Cache 1
```

- 1 **spec.service.type** フィールドは、Cache サービスノードまたは Data Grid サービスノードを作成するかどうかを指定します。

6.1.1. コンテナストレージ

Data Grid 7.3 サービスは、**/opt/datagrid/standalone/data** にマウントされたストレージボリュームを使用します。

Data Grid 8 サービスは、**/opt/infinispan/server/data** にマウントされた永続ボリューム要求を使用します。

6.1.2. Data Grid CLI

Data Grid 7.3 では、リモートシェルからのみ CLI にアクセスできます。Data Grid 7.3 CLI 経由で行った変更は Pod にバインドされ、再起動後は維持されませんでした。Data Grid 8 を使用すると、OpenShift でクラスターを使用して管理操作を実行したり、データを操作したりするための完全に機能するメカニズムとして CLI を使用できます。

6.1.3. Data Grid コンソール

Data Grid 7.3 は OpenShift のコンソールをサポートしませんでした。Data Grid 8 では、コンソールを使用して OpenShift で実行されているクラスターを監視し、管理操作を実行し、キャッシュをリモートで作成できます。

6.1.4. Data Grid のカスタマイズ

Data Grid 7.3 では、Source-to-Image(S2I) プロセスおよび **ConfigMap** API を使用して、OpenShift で実行される Data Grid サーバーイメージをカスタマイズできます。

Data Grid 8 では、Red Hat は、Red Hat Container Registry からの Data Grid イメージのカスタマイズをサポートしません。

Data Grid Operator は、OpenShift 上の Data Grid 8 クラスターのデプロイメントおよび管理を処理します。

その結果、カスタムを使用できません。

- 検出プロトコル
- 暗号化メカニズム (SYM_ENCRYPT または ASYM_ENCRYPT)
- 永続的なデータソース

Data Grid 8.0 および 8.1 では、Data Grid Operator では **JAR** ファイルや他のアーティファクトなどのカスタムコードをデプロイすることはできません。Data Grid 8.2 では、永続ボリューム要求 (PVC) を使用して、カスタムコードを Data Grid クラスターで利用可能にすることができます。

6.1.5. デプロイメント設定テンプレート

Data Grid 7.3 で利用可能だったデプロイメント設定テンプレートおよび環境変数は、Data Grid 8 で削除されました。

6.2. OPENSIFT 上の DATA GRID 8.2

このトピックでは、Data Grid Operator を使用した Data Grid 8.1 から 8.2 への移行の詳細について説明します。

Prometheus ServiceMonitor

Data Grid メトリクスを収集するために Prometheus 用に **ServiceMonitor** を作成する必要がなくなりました。OpenShift Container Platform でユーザー定義プロジェクトのモニタリングを有効にすると、Data Grid Operator は Prometheus Operator がインストールされるとこれを自動的に検出し、続いて **ServiceMonitor** を作成します。

6.3. OPENSIFT 上の DATA GRID 8.3

Data Grid Operator を使用した Data Grid 8.3 デプロイメントへの移行要件はありません。

6.4. OPENSIFT 上の DATA GRID 8.4

Data Grid Operator または Data Grid Helm チャートを使用した Data Grid 8.4 デプロイメントの移行要件はありません。

6.5. OPENSIFT 上の DATA GRID 8.5

Data Grid 8.3.1 は Data Grid Operator から削除されたため、Data Grid Operator をアップグレードする前にサーバーバージョンをアップグレードする必要があります。

第7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、デフォルトのマージャーが Protostream に変更になります。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、マーシャリングの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。

7.1. キャッシュストアマイグレーター

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 CLI **migrate store** コマンドを提供します。

ストアマイグレーターは、以前のバージョンの Data Grid のキャッシュストアをソースとして取得し、キャッシュストアの実装をターゲットとして使用します。

ストアマイグレーターを実行すると、**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。ストアマイグレーターは、ソースストアからメモリーにエンTRIESをロードし、それらをターゲットキャッシュに配置します。

ストアマイグレーターを使用すると、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 文字列ベースのキャッシュストアから、SIFS キャッシュストアに移行できます。



重要

ストアマイグレーターは、セグメント化されたキャッシュストアから以下へは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。

7.2. キャッシュストアマイグレーターの設定

migrator.properties ファイルを使用して、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を設定します。

手順

1. **migrator.properties** ファイルを作成します。
2. **migrator.properties** ファイルを使用して、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を設定します。

→ **migrator.properties** 接続種を **CacheMigrator** フォルダに追加し、**migrator.properties** の全設定プロパティに追加します。

- a. **source.** 接頭辞をソースキャッシュストアの全設定プロパティに追加します。

ソースキャッシュストアの例

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



重要

セグメント化されたキャッシュストアからデータを移行するには、**source.segment_count** プロパティを使用してセグメント数も設定する必要があります。セグメント数は、Data Grid 設定の **clustering.hash.numSegments** と一致させる必要があります。キャッシュストアのセグメント数が対応するキャッシュのセグメント数と一致しないと、Data Grid がキャッシュストアからデータを読み取ることができません。

- b. **target.** 接頭辞をターゲットキャッシュストアの全設定プロパティに追加します。

ターゲットキャッシュストアの例

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

7.2.1. キャッシュストアマイグレーターの設定プロパティ

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

表7.1 キャッシュストアタイププロパティ

プロパティ	説明	必須/オプション
-------	----	----------

プロパティ	説明	必須/オプション
type	<p>ソースまたはターゲットのキャッシュストアのタイプを指定します。</p> <p>.type=JDBC_STRING</p> <p>.type=JDBC_BINARY</p> <p>.type=JDBC_MIXED</p> <p>.type=LEVELDB</p> <p>.type=ROCKSDB</p> <p>.type=SINGLE_FILE_STORE</p> <p>.type=SOFT_INDEX_FILE_STORE</p> <p>.type=JDBC_MIXED</p>	必須

表7.2 一般的なプロパティ

プロパティ	説明	値の例	必須/オプション
cache_name	バックアップするキャッシュの名前。	.cache_name=myCache	必須
segment_count	<p>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。</p> <p>セグメント数は、Data Grid 設定の clustering.hash.num Segments と一致させる必要があります。キャッシュストアのセグメント数が対応するキャッシュのセグメント数と一致しないと、Data Grid がキャッシュストアからデータを読み取ることができません。</p>	.segment_count=256	任意
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.allow-list.classes

プロパティ	説明	値の例	必須/オプション
デシリアライズが許可される完全修飾クラス名のコンマ区切りリストを指定します。	任意	marshaller.allow-list.regexps	どのクラスのデシリアライズを許可するかを決定する正規表現のコンマ区切りリストを指定します。
任意	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。	任意

表7.3 JDBC プロパティ

プロパティ	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ジャ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 8.0.x の場合は 10 * Data Grid 8.1.x の場合は 11 * Data Grid 8.2.x の場合は 12 * Data Grid 8.3.x の場合は 13	ソースストアにのみ必要です。
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

プロパティ	説明	必須/オプション
db.disable_upsert	データベース upsert を無効にします。	任意
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。	任意
table.string.table_name_prefix	テーブル名の追加接頭辞を指定します。	任意
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。	任意

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.infinispan.commons.marshall.JavaSerializationMarshaller
target.marshaller.allow-list.classes=org.example.Person,org.example.Animal
target.marshaller.allow-list.regexps="org.another.example.*"
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.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7.4 RocksDB プロパティ

プロパティ	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。	任意

```
# 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7.5 SingleFileStore プロパティ

プロパティ	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7.6 SoftIndexFileStore プロパティ

プロパティ	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
```

7.3. DATA GRID キャッシュストアの移行

ストアマイグレーターを実行して、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

- Data Grid CLI を入手している。
- 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** ファイルを作成している。

手順

- **migrate store -p /path/to/migrator.properties** CLI コマンドを実行します。