Red Hat Data Grid 8.3

Java アプリケーションへの Data Grid の埋め込み

Data Grid を使用した組み込みキャッシュの作成

Last Updated: 2023-12-06

Red Hat Data Grid 8.3 Java アプリケーションへの Data Grid の埋め込み

Data Grid を使用した組み込みキャッシュの作成

法律上の通知

概要

Data Grid を Java プロジェクトに追加し、アプリケーションで組み込みキャッシュを使用します。

目次

RED HAT DATA GRID

Data Grid は、高性能の分散型インメモリーデータストアです。

スキーマレスデータ構造

さまざまなオブジェクトをキーと値のペアとして格納する柔軟性があります。

グリッドベースのデータストレージ

クラスター間でデータを分散および複製するように設計されています。

エラスティックスケーリング

サービスを中断することなく、ノードの数を動的に調整して要件を満たします。

データの相互運用性

さまざまなエンドポイントからグリッド内のデータを保存、取得、およびクエリーします。

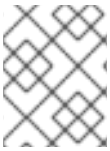
DATA GRID のドキュメント

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 8.3 ドキュメント
- Data Grid 8.3 コンポーネントの詳細
- Data Grid 8.3 でサポートされる設定
- 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 プロパティー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、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ を参照してください。

第1章 DATA GRID MAVEN リポジトリーの設定

Data Grid Java ディストリビューションは Maven から入手できます。

顧客ポータルから Data Grid Maven リポジトリーをダウンロードするか、パブリック Red Hat Enterprise Maven リポジトリーから Data Grid 依存関係をプルできます。

1.1. DATA GRID MAVEN リポジトリーのダウンロード

パブリック Red Hat Enterprise Maven リポジトリーを使用しない場合は、ローカルファイルシステム、Apache HTTP サーバー、または Maven リポジトリーマネージャーに Data Grid Maven リポジトリーをダウンロードし、インストールします。

手順

1. Red Hat カスタマーポータルにログインします。
2. Software Downloads for Data Grid に移動します。
3. Red Hat Data Grid 8.3 Maven リポジトリーをダウンロードします。
4. アーカイブされた Maven リポジトリーをローカルファイルシステムに展開します。
5. **README.md** ファイルを開き、適切なインストール手順に従います。

1.2. RED HAT MAVEN リポジトリーの追加

Red Hat GA リポジトリーを Maven ビルド環境に組み込み、Data Grid アーティファクトおよび依存関係を取得します。

手順

- Red Hat GA リポジトリーを Maven 設定ファイル (通常は **~/.m2/settings.xml**) に追加するか、プロジェクトの **pom.xml** ファイルに直接追加します。

```
<repositories>
  <repository>
    <id>redhat-ga-repository</id>
    <name>Red Hat GA Repository</name>
    <url>https://maven.repository.redhat.com/ga/</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>redhat-ga-repository</id>
    <name>Red Hat GA Repository</name>
    <url>https://maven.repository.redhat.com/ga/</url>
  </pluginRepository>
</pluginRepositories>
```

参照資料

- Red Hat Enterprise Maven Repository

1.3. DATA GRID POM の設定

Maven は、プロジェクトオブジェクトモデル (POM) ファイルと呼ばれる設定ファイルを使用して、プロジェクトを定義し、ビルドを管理します。POM ファイルは XML 形式であり、モジュールとコンポーネントの依存関係、ビルドの順序、および結果となるプロジェクトのパッケージ化と出力のターゲットを記述します。

手順

1. プロジェクト **pom.xml** を開いて編集します。
2. 正しい Data Grid バージョンで **version.infinispan** プロパティーを定義します。
3. **dependencyManagement** セクションに **infinispan-bom** を含めます。
   BOM(Bill of Materials) は、依存関係バージョンを制御します。これにより、バージョンの競合が回避され、プロジェクトに依存関係として追加する Data Grid アーティファクトごとにバージョンを設定する必要がなくなります。
4. **pom.xml** を保存して閉じます。

以下の例は、Data Grid のバージョンと BOM を示しています。

```
<properties>
  <version.infinispan>13.0.10.Final-redhat-00001</version.infinispan>
</properties>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-bom</artifactId>
      <version>${version.infinispan}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

次のステップ

必要に応じて、Data Grid アーティファクトを依存関係として **pom.xml** に追加します。

第2章 組み込みキャッシュの作成

Data Grid は、プログラムを使用して Cache Manager と組み込みキャッシュライフサイクルの両方を制御できる **EmbeddedCacheManager** API を提供します。

2.1. DATA GRID のプロジェクトへの追加

Data Grid をプロジェクトに追加して、アプリケーションで組み込みキャッシュを作成します。

前提条件

- Maven リポジトリーから Data Grid アーティファクトを取得するようにプロジェクトを設定します。

手順

- 以下のように、**infinispan-core** アーティファクトを **pom.xml** の依存関係として追加します。

```
<dependencies>
 <dependency>
  <groupId>org.infinispan</groupId>
  <artifactId>infinispan-core</artifactId>
 </dependency>
</dependencies>
```

2.2. 組み込みキャッシュの設定

Data Grid は、キャッシュマネージャーと、埋め込みキャッシュを設定する **ConfigurationBuilder** API を制御する **GlobalConfigurationBuilder** API を提供します。

前提条件

- **infinispan-core** アーティファクトを **pom.xml** の依存関係として追加します。

手順

1. デフォルトのキャッシュマネージャーを初期化し、埋め込みキャッシュを追加できます。
2. **ConfigurationBuilder** API を使用して、埋め込みキャッシュを 1 つ以上追加します。
3. クラスターのすべてのノードで組み込みキャッシュを作成するか、すでに存在するキャッシュを返す **getOrCreateCache()** メソッドを呼び出します。

```
// Set up a clustered cache manager.
GlobalConfigurationBuilder global = GlobalConfigurationBuilder.defaultClusteredBuilder();
// Initialize the default cache manager.
DefaultCacheManager cacheManager = new DefaultCacheManager(global.build());
// Create a distributed cache with synchronous replication.
ConfigurationBuilder builder = new ConfigurationBuilder();
                builder.clustering().cacheMode(CacheMode.DIST_SYNC);
// Obtain a volatile cache.
```

```
Cache<String, String> cache =
cacheManager.administration().withFlags(CacheContainerAdmin.AdminFlag.VOLATILE).getOrCreateC
ache("myCache", builder.build());
```

関連資料

- EmbeddedCacheManager
- EmbeddedCacheManager Configuration
- org.infinispan.configuration.global.GlobalConfiguration
- org.infinispan.configuration.cache.ConfigurationBuilder

第3章 DATA GRID 統計および JMX 監視の有効化および設定

Data Grid は、JMX MBean をエクスポートしたり、Cache Manager およびキャッシュ統計を提供できます。

3.1. 組み込みキャッシュでの統計の有効化

キャッシュマネージャーおよび埋め込みキャッシュの統計をエクスポートするように Data Grid を設定します。

手順

1. Data Grid 設定を開いて編集します。
2. **statistics="true"** 属性または **.statistics(true)** メソッドを追加します。
3. Data Grid 設定を保存して閉じます。

組み込みキャッシュの統計

XML

```
<infinispan>
  <cache-container statistics="true">
    <distributed-cache statistics="true"/>
    <replicated-cache statistics="true"/>
  </cache-container>
</infinispan>
```

GlobalConfigurationBuilder

```
GlobalConfigurationBuilder global =
GlobalConfigurationBuilder.defaultClusteredBuilder().cacheContainer().statistics(true);
DefaultCacheManager cacheManager = new DefaultCacheManager(global.build());

Configuration builder = new ConfigurationBuilder();
builder.statistics().enable();
```

3.2. DATA GRID メトリクスの設定

Data Grid は、MicroProfile Metrics API と互換性のあるメトリクスを生成します。

- ゲージは、書き込み操作または JVM アップタイムの平均数 (ナノ秒) などの値を指定します。
- ヒストグラムは、読み取り、書き込み、削除の時間などの操作実行時間の詳細を提供します。

デフォルトでは、Data Grid は統計を有効にするとゲージを生成しますが、ヒストグラムを生成するように設定することもできます。

手順

1. Data Grid 設定を開いて編集します。

2. **metrics** 要素またはオブジェクトをキャッシュコンテナーに追加します。
3. **gauges** 属性またはフィールドを使用してゲージを有効または無効にします。
4. **histograms** 属性またはフィールドでヒストグラムを有効または無効にします。
5. クライアント設定を保存して閉じます。

メトリクスの設定

XML

```
<infinispan>
  <cache-container statistics="true">
    <metrics gauges="true"
             histograms="true" />
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "statistics" : "true",
      "metrics" : {
        "gauges" : "true",
        "histograms" : "true"
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    statistics: "true"
    metrics:
      gauges: "true"
      histograms: "true"
```

GlobalConfigurationBuilder

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()
  //Computes and collects statistics for the Cache Manager.
  .statistics().enable()
  //Exports collected statistics as gauge and histogram metrics.
  .metrics().gauges(true).histograms(true)
  .build();
```

検証

埋め込みキャッシュの場合は、必要な MicroProfile API およびプロバイダー JAR をクラスパスに追加して Data Grid メトリクスをエクスポートする必要があります。

関連情報

- Eclipse MicroProfile Metrics

3.3. JMX MBEAN の登録

Data Grid は、統計の収集と管理操作の実行に使用できる JMX MBean を登録できます。統計を有効にする必要もあります。そうしないと、Data Grid は JMX MBean のすべての統計属性に **0** 値を提供します。

手順

1. Data Grid 設定を開いて編集します。
2. **jmx** 要素またはオブジェクトをキャッシュコンテナーに追加し、**enabled** 属性またはフィールドの値として **true** を指定します。
3. **domain** 属性またはフィールドを追加し、必要に応じて JMX MBean が公開されるドメインを指定します。
4. クライアント設定を保存して閉じます。

JMX の設定

XML

```
<infinispan>
  <cache-container statistics="true">
    <jmx enabled="true"
         domain="example.com"/>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "statistics" : "true",
      "jmx" : {
        "enabled" : "true",
        "domain" : "example.com"
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    statistics: "true"
    jmx:
      enabled: "true"
      domain: "example.com"
```

GlobalConfigurationBuilder

```
GlobalConfiguration global = GlobalConfigurationBuilder.defaultClusteredBuilder()
   .jmx().enable()
   .domain("org.mydomain");
```

3.3.1. JMX リモートポートの有効化

一意のリモート JMX ポートを提供し、JMXServiceURL 形式の接続を介して Data Grid MBean を公開します。

次のいずれかの方法を使用して、リモート JMX ポートを有効にできます。

- Data Grid サーバーセキュリティーレルムの 1 つに対する認証を必要とするリモート JMX ポートを有効にします。
- 標準の Java 管理設定オプションを使用して、手動でリモート JMX ポートを有効にします。

前提条件

- 認証付きのリモート JMX の場合、デフォルトのセキュリティーレルムを使用してユーザーロールを定義します。ユーザーが JMX リソースにアクセスするには、読み取り/書き込みアクセス権を持つ **controlRole** または読み取り専用アクセス権を持つ **monitorRole** が必要です。

手順

次のいずれかの方法を使用して、リモート JMX ポートを有効にして Data Grid サーバーを起動します。

- ポート **9999** を介してリモート JMX を有効にします。

```
bin/server.sh --jmx 9999
```

警告

SSL を無効にしてリモート JMX を使用することは、本番環境向けではありません。

- 起動時に以下のシステムプロパティーを Data Grid サーバーに渡します。

```
bin/server.sh -Dcom.sun.management.jmxremote.port=9999 -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false
```

警告

認証または SSL なしでリモート JMX を有効にすることは安全ではなく、どのような環境でも推奨されません。認証と SSL を無効にすると、権限のないユーザーがサーバーに接続し、そこでホストされているデータにアクセスできるようになります。

関連情報

- セキュリティーレルムの作成

3.3.2. Data Grid MBean

Data Grid は、管理可能なリソースを表す JMX MBean を公開します。

org.infinispan:type=Cache

キャッシュインスタンスに使用できる属性および操作。

org.infinispan:type=CacheManager

Data Grid キャッシュやクラスターのヘルス統計など、Cache Manager で使用できる属性および操作。

使用できる JMX MBean の詳細な一覧および説明、ならびに使用可能な操作および属性については、**Data Grid JMX Components**のドキュメントを参照してください。

関連情報

- Data Grid JMX Components

3.3.3. カスタム MBean サーバーでの MBean の登録

Data Grid には、カスタム MBeanServer インスタンスに MBean を登録するのに使用できる **MBeanServerLookup** インターフェイスが含まれています。

前提条件

- **getMBeanServer()** メソッドがカスタム MBeanServer インスタンスを返すように **MBeanServerLookup** の実装を作成します。
- JMX MBean を登録するように Data Grid を設定します。

手順

1. Data Grid 設定を開いて編集します。

2. **mbean-server-lookup** 属性またはフィールドをキャッシュマネージャーの JMX 設定に追加します。
3. **MBeanServerLookup** 実装の完全修飾名 (FQN) を指定します。
4. クライアント設定を保存して閉じます。

JMX MBean サーバールックアップの設定

XML

```
<infinispan>
  <cache-container statistics="true">
    <jmx enabled="true"
         domain="example.com"
         mbean-server-lookup="com.example.MyMBeanServerLookup"/>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "statistics" : "true",
      "jmx" : {
        "enabled" : "true",
        "domain" : "example.com",
        "mbean-server-lookup" : "com.example.MyMBeanServerLookup"
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    statistics: "true"
    jmx:
      enabled: "true"
      domain: "example.com"
      mbeanServerLookup: "com.example.MyMBeanServerLookup"
```

GlobalConfigurationBuilder

```
GlobalConfiguration global = GlobalConfigurationBuilder.defaultClusteredBuilder()
   .jmx().enable()
   .domain("org.mydomain")
   .mBeanServerLookup(new com.acme.MyMBeanServerLookup());
```

第4章 DATA GRID クラスタートランスポートの設定

Data Grid には、ノードがクラスターに自動的に参加および離脱できるように、トランスポート層が必要です。また、トランスポート層により、Data Grid ノードはネットワーク上でデータを複製または分散し、リバランスや状態遷移などの操作を実施することができます。

4.1. デフォルトの JGROUPS スタック

Data Grid は、**infinispan-core-13.0.10.Final-redhat-00001.jar** ファイル内の **default-configs** ディレクトリーに、デフォルトの JGroups スタックファイル **default-jgroups-*.xml** を提供します。

File name	スタック名	説明
default-jgroups-udp.xml	**udp**	トランスポートに UDP を使用し、検出に UDP マルチキャストを使用します。(100 ノードを超える) 大規模なクラスター、またはレプリケートされたキャッシュまたは無効化モードを使用している場合に適しています。オープンソケットの数を最小限に抑えます。
default-jgroups-tcp.xml	**tcp**	トランスポートには TCP を使用し、検出には **UDP** マルチキャストを使用する **MPING** プロトコルを使用します。TCP はポイントツーポイントプロトコルとして UDP よりも効率的であるため、分散キャッシュを使用している**場合にのみ**、小規模なクラスター (100 ノード未満) に適しています。
default-jgroups-kubernetes.xml	**kubernetes**	トランスポートに TCP を使用し、検出に **DNS_PING** を使用します。UDP マルチキャストが常に利用できるとは限らない Kubernetes および Red Hat OpenShift ノードに適しています。
default-jgroups-ec2.xml	**ec2**	トランスポートに TCP を使用し、検出に **NATIVE_S3_PING** を使用します。UDP マルチキャストが利用できない Amazon EC2 ノードに適しています。追加の依存関係が必要です。
default-jgroups-google.xml	**google**	トランスポートに TCP を使用し、検出に **GOOGLE_PING2** を使用します。UDP マルチキャストが利用できない Google Cloud Platform ノードに適しています。追加の依存関係が必要です。
default-jgroups-azure.xml	**azure**	トランスポートに TCP を使用し、検出に **AZURE_PING** を使用します。UDP マルチキャストが利用できない Microsoft Azure ノードに適しています。追加の依存関係が必要です。

関連情報

- JGroups Protocols

4.2. クラスター検出プロトコル

Data Grid は、ノードがネットワーク上でお互いを自動的に見つけてクラスターを形成できるようにするさまざまなプロトコルをサポートしています。

Data Grid が使用できる 2 種類の検出メカニズムがあります。

- ほとんどのネットワークで機能する汎用検出プロトコルで、外部サービスに依存しません。
- Data Grid クラスターのトポロジー情報を保存し、取得するために外部サービスに依存する検出プロトコル。
  たとえば、DNS_PING プロトコルは DNS サーバーレコードで検出を実行します。

注記

ホスト型プラットフォームで Data Grid を実行するには、個別のクラウドプロバイダーが課すネットワーク制約に適合する検出メカニズムを使用する必要があります。

関連情報

- JGroups Discovery Protocols
- JGroups cluster transport configuration for Data Grid 8.x (Red Hat ナレッジベースの記事)

4.2.1. PING

PING または UDPPING は、UDP プロトコルで動的なマルチキャストを使用する一般的な JGroups 検出メカニズムです。

結合時に、ノードは IP マルチキャストアドレスに PING 要求を送信し、Data Grid クラスターにある他のノードを検出します。各ノードは、コーディネーターノードのアドレスとその独自のアドレスが含まれるパケットで PING リクエストに応答します。C はコーディネーターのアドレスで、A は自分のアドレスです。ノードが PING 要求に応答すると、結合ノードは新しいクラスターのコーディネーターノードになります。

PING 設定の例

```
<PING num_discovery_runs="3"/>
```

関連情報

- JGroups PING

4.2.2. TCPPING

TCPPING は、クラスターメンバーの静的アドレスリストを使用する汎用 JGroups 検索メカニズムです。

TCPPING を使用すると、ノードが相互に動的に検出できるようにするのではなく、JGroups スタックの一部として Data Grid クラスター内の各ノードの IP アドレスまたはホスト名を手動で指定します。

TCPPING 設定の例

```
<TCP bind_port="7800" />
```

```
<TCPPING timeout="3000"
         initial_hosts="${jgroups.tcpping.initial_hosts:hostname1[port1],hostname2[port2]}"
          port_range="0"
          num_initial_members="3"/>
```

関連情報

- JGroups TCPPING

4.2.3. MPING

MPING は IP マルチキャストを使用して Data Grid クラスターの初期メンバーシップを検出します。

MPING を使用して TCPPING 検出を TCP スタックに置き換え、初期ホストの静的リストの代わりに、検出にマルチキャストを使用できます。ただし、UDP スタックで MPING を使用することもできます。

MPING 設定の例

```
<MPING mcast_addr="${jgroups.mcast_addr:228.6.7.8}"
       mcast_port="${jgroups.mcast_port:46655}"
       num_discovery_runs="3"
       ip_ttl="${jgroups.udp.ip_ttl:2}"/>
```

関連情報

- JGroups MPING

4.2.4. TCPGOSSIP

gossip ルーターは、Data Grid クラスターが他のノードのアドレスを取得できるネットワーク上の集中的な場所を提供します。

以下のように、Gosssip ルーターのアドレス (**IP:PORT**) を Data Grid ノードに挿入します。

1. このアドレスをシステムプロパティーとして JVM に渡します (例: **-DGossipRouterAddress="10.10.2.4[12001]"**)。
2. JGroups 設定ファイルのそのシステムプロパティーを参照します。

Gossip ルーター設定の例

```
<TCP bind_port="7800" />
<TCPGOSSIP timeout="3000"
           initial_hosts="${GossipRouterAddress}"
           num_initial_members="3" />
```

関連情報

- JGroups Gossip Router

4.2.5. JDBC_PING

JDBC_PING は共有データベースを使用して Data Grid クラスターに関する情報を保存します。このプロトコルは、JDBC 接続を使用できるすべてのデータベースをサポートします。

ノードは IP アドレスを共有データベースに書き込むため、ノードに結合してネットワーク上の Data Grid クラスターを検索できます。ノードが Data Grid クラスターのままにすると、共有データベースから IP アドレスを削除します。

JDBC_PING 設定の例

```
<JDBC_PING connection_url="jdbc:mysql://localhost:3306/database_name"
           connection_username="user"
           connection_password="password"
           connection_driver="com.mysql.jdbc.Driver"/>
```

重要

適切な JDBC ドライバーをクラスパスに追加して、Data Grid が JDBC_PING を使用できるようにします。

関連情報

- JDBC_PING
- JDBC_PING Wiki

4.2.6. DNS_PING

JGroups DNS_PING は DNS サーバーをクエリーし、OKD や Red Hat OpenShift などの Kubernetes 環境で Data Grid クラスターメンバーを検出します。

DNS_PING 設定の例

```
<dns.DNS_PING dns_query="myservice.myproject.svc.cluster.local" />
```

関連情報

- JGroups DNS_PING
- DNS for Services and Pods (DNS エントリーを追加するための Kubernetes ドキュメント)

4.2.7. クラウド検出プロトコル

Data Grid には、クラウドプロバイダーに固有の検出プロトコル実装を使用するデフォルトの JGroups スタックが含まれています。

検出プロトコル	デフォルトのスタックファイル	アーティファクト	バージョン
NATIVE_S3_PING	**default-jgroups-ec2.xml**	**org.jgroups.aws.s3:native-s3-ping**	**1.0.0.Final**

検出プロトコル	デフォルトのスタックファイル	アーティファクト	バージョン
GOOGLE_PING2	**default-jgroups-google.xml**	**org.jgroups.google:jgroups-google**	**1.0.0.Final**
AZURE_PING	**default-jgroups-azure.xml**	**org.jgroups.azure:jgroups-azure**	**1.3.0.Final**

クラウド検出プロトコルの依存関係の提供
NATIVE_S3_PING、**GOOGLE_PING2**、または **AZURE_PING** の Cloud Discovery プロトコルを使用するには、依存するライブラリーを Data Grid に提供する必要があります。

手順

- アーティファクト依存関係をプロジェクトの **pom.xml** に追加します。

続いて、JGroups スタックファイルの一部として、またはシステムプロパティーを使用して、クラウド検出プロトコルを設定できます。

関連情報

- JGroups NATIVE_S3_PING
- JGroups GOOGLE_PING2
- JGroups AZURE_PING

4.3. デフォルトの JGROUPS スタックの使用

Data Grid は JGroups プロトコルスタックを使用するため、ノードは専用のクラスターチャネルに相互に送信できるようにします。

Data Grid は、**UDP** プロトコルおよび **TCP** プロトコルに事前設定された JGroups スタックを提供します。これらのデフォルトスタックは、ネットワーク要件向け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際の開始点として使用することができます。

手順

デフォルトの JGroups スタックの 1 つを使用するには、以下のいずれかを行います。

- **infinispan.xml** ファイルの **stack** 属性を使用します。

```
<infinispan>
  <cache-container default-cache="replicatedCache">
    <!-- Use the default UDP stack for cluster transport. -->
    <transport cluster="${infinispan.cluster.name}"
               stack="udp"
               node-name="${infinispan.node.name:}"/>
  </cache-container>
</infinispan>
```

- **addProperty()** メソッドを使用して JGroups スタックファイルを設定します。

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder().transport()
      .defaultTransport()
      .clusterName("qa-cluster")
      //Uses the default-jgroups-udp.xml stack for cluster transport.
      .addProperty("configurationFile", "default-jgroups-udp.xml")
      .build();
```

検証

Data Grid は、以下のメッセージをログに記録して、使用するスタックを示します。

```
[org.infinispan.CLUSTER] ISPN000078: Starting JGroups channel cluster with stack udp
```

関連情報

- JGroups cluster transport configuration for Data Grid 8.x (Red Hat ナレッジベースの記事)

4.4. JGROUPS スタックのカスタマイズ

プロパティーを調整してチューニングし、ネットワーク要件に対応するクラスタートランスポート設定を作成します。

Data Grid は、設定を容易にするためにデフォルトの JGroups スタックを拡張する属性を提供します。他のプロパティーを組み合わせてデフォルトスタックからプロパティーの継承、削除、置き換えを行うことができます。

手順

1. **infinispan.xml** ファイルに新しい JGroups スタック宣言を作成します。
2. **extends** 属性を追加し、プロパティーを継承する JGroups スタックを指定します。
3. **stack.combine** 属性を使用して、継承されたスタックに設定されたプロトコルのプロパティーを変更します。
4. **stack.position** 属性を使用して、カスタムスタックの場所を定義します。
5. スタック名を **transport** 設定の **stack** 属性の値として指定します。
   たとえば、以下のようにデフォルトの TCP スタックで Gosssip ルーターと対称暗号化を使用して評価できます。

```
<infinispan>
  <jgroups>
    <!-- Creates a custom JGroups stack named "my-stack". -->
    <!-- Inherits properties from the default TCP stack. -->
    <stack name="my-stack" extends="tcp">
      <!-- Uses TCPGOSSIP as the discovery mechanism instead of MPING -->
      <TCPGOSSIP initial_hosts="${jgroups.tunnel.gossip_router_hosts:localhost[12001]}"
            stack.combine="REPLACE"
            stack.position="MPING" />
      <!-- Removes the FD_SOCK protocol from the stack. -->
      <FD_SOCK stack.combine="REMOVE"/>
      <!-- Modifies the timeout value for the VERIFY_SUSPECT protocol. -->
      <VERIFY_SUSPECT timeout="2000"/>
```

```
    <!-- Adds SYM_ENCRYPT to the stack after VERIFY_SUSPECT. -->
    <SYM_ENCRYPT sym_algorithm="AES"
                 keystore_name="mykeystore.p12"
                 keystore_type="PKCS12"
                 store_password="changeit"
                 key_password="changeit"
                 alias="myKey"
                 stack.combine="INSERT_AFTER"
                 stack.position="VERIFY_SUSPECT" />
  </stack>
  <cache-container name="default" statistics="true">
   <!-- Uses "my-stack" for cluster transport. -->
   <transport cluster="${infinispan.cluster.name}"
              stack="my-stack"
              node-name="${infinispan.node.name:}"/>
  </cache-container>
 </jgroups>
</infinispan>
```

6. Data Grid ログをチェックして、スタックを使用していることを確認します。

```
[org.infinispan.CLUSTER] ISPN000078: Starting JGroups channel cluster with stack my-stack
```

参照資料

- JGroups cluster transport configuration for Data Grid 8.x (Red Hat ナレッジベースの記事)

4.4.1. 継承属性

JGroups スタックを拡張すると、継承属性により、拡張しているスタックでプロトコルやプロパティーを調整できます。

- **stack.position** は、変更するプロトコルを指定します。
- **stack.combine** は、次の値を使用して JGroups スタックを拡張します。

値	説明
COMBINE	プロトコルプロパティーをオーバーライドします。
REPLACE	プロトコルを置き換えます。
INSERT_AFTER	別のプロトコルの後にプロトコルをスタックに追加します。挿入ポイントとして指定するプロトコルには影響しません。 JGroups スタックのプロトコルは、スタック内の場所を基にして相互に影響します。**NAKACK2** がセキュリティーで保護されるように、たとえば、**SYM_ENCRYPT** プロトコルまたは **ASYM_ENCRYPT** プロトコル後に **NAKACK2** などのプロトコルを置く必要があります。

値	説明
INSERT_BEFORE	別のプロトコルの前にプロトコルをスタックに挿入します。挿入ポイントとして指定するプロトコルに影響します。
REMOVE	スタックからプロトコルを削除します。

4.5. JGROUPS システムプロパティーの使用

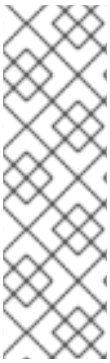
起動時にシステムプロパティーを Data Grid に渡して、クラスターのトランスポートを調整します。

手順

- **-D<property-name>=<property-value>** 引数を使用して JGroups システムプロパティーを必要に応じて設定します。

たとえば、以下のようにカスタムバインドポートと IP アドレスを設定します。

```
java -cp ... -Djgroups.bind.port=1234 -Djgroups.bind.address=192.0.2.0
```



注記

クラスター化された Red Hat JBoss EAP アプリケーションに Data Grid クラスターを組み込むと、JGroups システムプロパティーは競合したり、互いに上書きしたりする可能性があります。

たとえば、Data Grid クラスターまたは Red Hat JBoss EAP アプリケーションのいずれかに一意のバインドアドレスを設定しないでください。この場合、Data Grid と Red Hat JBoss EAP アプリケーションの両方が JGroups のデフォルトプロパティーを使用し、同じバインドアドレスを使用してクラスターを形成しようとします。

4.5.1. クラストートランスポートプロパティー

以下のプロパティーを使用して JGroups クラストートランスポートをカスタマイズします。

システムプロパティー	説明	デフォルト値	必須/オプション
jgroups.bind.address	クラストートランスポートのバインドアドレス。	**SITE_LOCAL**	任意
jgroups.bind.port	ソケットのバインドポート。	**7800**	任意
jgroups.mcast_addr	マルチキャストの IP アドレス (検出およびクラスター間の通信の両方)。IP アドレスは、IP マルチキャストに適した有効なクラス D アドレスである必要があります。	**228.6.7.8**	任意

システムプロパティー	説明	デフォルト値	必須/オプション
jgroups.mcast_port	マルチキャストソケットのポート。	**46655**	任意
jgroups.ip_ttl	IP マルチキャストパケットの Time-to-live (TTL)この値は、パケットが破棄される前にパケットが作成できるネットワークホップの数を定義します。	2	任意
jgroups.thread_pool.min_threads	スレッドプールの最小スレッド数	0	任意
jgroups.thread_pool.max_threads	スレッドプールの最大スレッド数	200	任意
jgroups.join_timeout	結合リクエストが正常に実行されるまで待機する最大時間 (ミリ秒単位)。	2000	任意
jgroups.thread_dumps_threshold	スレッドダンプがログに記録される前にスレッドプールが満杯である必要がある回数。	10000	任意

関連情報

- JGroups system properties
- JGroups protocol list

4.5.2. クラウド検出プロトコルのシステムプロパティー

以下のプロパティーを使用して、ホストされたプラットフォームの JGroups 検出プロトコルを設定します。

4.5.2.1. Amazon EC2

NATIVE_S3_PING を設定するためのシステムプロパティー。

システムプロパティー	説明	デフォルト値	必須/オプション
jgroups.s3.region_name	Amazon S3 リージョンの名前。	デフォルト値はありません。	任意
jgroups.s3.bucket_name	Amazon S3 バケットの名前。名前は存在し、一意でなければなりません。	デフォルト値はありません。	任意

4.5.2.2. Google Cloud Platform

GOOGLE_PING2 を設定するためのシステムプロパティー。

システムプロパティー	説明	デフォルト値	必須/オプション
jgroups.google.bucket_name	Google Compute Engine バケットの名前。名前は存在し、一意でなければなりません。	デフォルト値はありません。	必須

4.5.2.3. Azure

AZURE_PING のシステムプロパティー。

システムプロパティー	説明	デフォルト値	必須/オプション
jboss.jgroups.azure_ping.storage_account_name	Azure ストレージアカウントの名前。名前は存在し、一意でなければなりません。	デフォルト値はありません。	必須
jboss.jgroups.azure_ping.storage_access_key	Azure ストレージアクセスキーの名前。	デフォルト値はありません。	必須
jboss.jgroups.azure_ping.container	ping 情報を格納するコンテナーの有効な DNS 名。	デフォルト値はありません。	必須

4.5.2.4. OpenShift

DNS_PING のシステムプロパティー。

システムプロパティー	説明	デフォルト値	必須/オプション
jgroups.dns.query	クラスターメンバーを返す DNS レコードを設定します。	デフォルト値はありません。	必須

4.6. インライン JGROUPS スタックの使用

完全な JGroups スタックの定義を **infinispan.xml** ファイルに挿入することができます。

手順

- カスタム JGroups スタック宣言を **infinispan.xml** ファイルに埋め込みます。

```
<infinispan>
  <!-- Contains one or more JGroups stack definitions. -->
  <jgroups>
    <!-- Defines a custom JGroups stack named "prod". -->
    <stack name="prod">
      <TCP bind_port="7800" port_range="30" recv_buf_size="20000000"
send_buf_size="640000"/>
      <MPING break_on_coord_rsp="true"
             mcast_addr="${jgroups.mping.mcast_addr:228.2.4.6}"
             mcast_port="${jgroups.mping.mcast_port:43366}"
             num_discovery_runs="3"
             ip_ttl="${jgroups.udp.ip_ttl:2}"/>
      <MERGE3 />
      <FD_SOCK />
      <FD_ALL timeout="3000" interval="1000" timeout_check_interval="1000" />
      <VERIFY_SUSPECT timeout="1000" />
      <pbcast.NAKACK2 use_mcast_xmit="false" xmit_interval="200"
xmit_table_num_rows="50"
                      xmit_table_msgs_per_row="1024" xmit_table_max_compaction_time="30000"
/>
      <UNICAST3 conn_close_timeout="5000" xmit_interval="200" xmit_table_num_rows="50"
              xmit_table_msgs_per_row="1024" xmit_table_max_compaction_time="30000" />
      <pbcast.STABLE desired_avg_gossip="2000" max_bytes="1M" />
      <pbcast.GMS print_local_addr="false" join_timeout="${jgroups.join_timeout:2000}" />
      <UFC max_credits="4m" min_threshold="0.40" />
      <MFC max_credits="4m" min_threshold="0.40" />
      <FRAG3 />
    </stack>
  </jgroups>
  <cache-container default-cache="replicatedCache">
    <!-- Uses "prod" for cluster transport. -->
    <transport cluster="${infinispan.cluster.name}"
           stack="prod"
           node-name="${infinispan.node.name:}"/>
  </cache-container>
</infinispan>
```

4.7. 外部 JGROUPS スタックの使用

infinispan.xml ファイルでカスタム JGroups スタックを定義する外部ファイルを参照します。

手順

1. カスタム JGroups スタックファイルをアプリケーションクラスパスに配置します。
   または、外部スタックファイルを宣言する際に絶対パスを指定することもできます。
2. **stack-file** 要素を使用して、外部スタックファイルを参照します。

```
<infinispan>
  <jgroups>
    <!-- Creates a "prod-tcp" stack that references an external file. -->
    <stack-file name="prod-tcp" path="prod-jgroups-tcp.xml"/>
  </jgroups>
  <cache-container default-cache="replicatedCache">
    <!-- Use the "prod-tcp" stack for cluster transport. -->
    <transport stack="prod-tcp" />
    <replicated-cache name="replicatedCache"/>
  </cache-container>
  <!-- Cache configuration goes here. -->
</infinispan>
```

TransportConfigurationBuilder クラスで **addProperty()** メソッドを使用して、以下のようにカスタム JGroups スタックファイルを指定することもできます。

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder().transport()
        .defaultTransport()
        .clusterName("prod-cluster")
        //Uses a custom JGroups stack for cluster transport.
        .addProperty("configurationFile", "my-jgroups-udp.xml")
        .build();
```

この例では、**my-jgroups-udp.xml** は、以下のようなカスタムプロパティーで UDP スタックを参照します。

カスタム UDP スタックの例

```
<config xmlns="urn:org:jgroups"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:org:jgroups http://www.jgroups.org/schema/jgroups-4.2.xsd">
    <UDP bind_addr="${jgroups.bind_addr:127.0.0.1}"
         mcast_addr="${jgroups.udp.mcast_addr:192.0.2.0}"
         mcast_port="${jgroups.udp.mcast_port:46655}"
         tos="8"
         ucast_recv_buf_size="20000000"
         ucast_send_buf_size="640000"
         mcast_recv_buf_size="25000000"
         mcast_send_buf_size="640000"
         max_bundle_size="64000"
         ip_ttl="${jgroups.udp.ip_ttl:2}"
         enable_diagnostics="false"
         thread_naming_pattern="pl"
         thread_pool.enabled="true"
         thread_pool.min_threads="2"
         thread_pool.max_threads="30"
```

```
        thread_pool.keep_alive_time="5000" />
    <!-- Other JGroups stack configuration goes here. -->
</config>
```

関連情報

- org.infinispan.configuration.global.TransportConfigurationBuilder

4.8. カスタム JCHANNELS の使用

以下の例のように、カスタム JGroups JChannels を構築します。

```
GlobalConfigurationBuilder global = new GlobalConfigurationBuilder();
JChannel jchannel = new JChannel();
// Configure the jchannel as needed.
JGroupsTransport transport = new JGroupsTransport(jchannel);
global.transport().transport(transport);
new DefaultCacheManager(global.build());
```

注記

Data Grid は、すでに接続されているカスタム JChannels を使用できません。

関連情報

- JGroups JChannel

4.9. クラスタートランスポートの暗号化

ノードが暗号化されたメッセージと通信できるように、クラスタートランスポートを保護します。また、有効なアイデンティティーを持つノードのみが参加できるように、証明書認証を実行するように Data Grid クラスターを設定することもできます。

4.9.1. JGroups 暗号化プロトコル

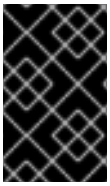
クラスタートラフィックのセキュリティーを保護するには、Data Grid ノードを設定し、シークレットキーで JGroups メッセージペイロードを暗号化します。

Data Grid ノードは、以下のいずれかから秘密鍵を取得できます。

- コーディネーターノード (非対称暗号化)
- 共有キーストア (対称暗号化)

コーディネーターノードからの秘密鍵の取得

非対称暗号化は、Data Grid 設定の JGroups スタックに **ASYM_ENCRYPT** プロトコルを追加して対称暗号化を設定します。これにより、Data Grid クラスターはシークレットキーを生成して配布できます。



重要

非対称暗号化を使用する場合は、ノードが証明書認証を実行し、シークレットキーを安全に交換できるようにキーストアを提供する必要もあります。これにより、中間者 (MitM) 攻撃からクラスターが保護されます。

非対称暗号化は、以下のようにクラストラフィックのセキュリティーを保護します。

1. Data Grid クラスターの最初のノードであるコーディネーターノードは、秘密鍵を生成します。
2. 参加ノードは、コーディネーターとの証明書認証を実行して、相互に ID を検証します。
3. 参加ノードは、コーディネーターノードに秘密鍵を要求します。その要求には、参加ノードの公開鍵が含まれています。
4. コーディネーターノードは、秘密鍵を公開鍵で暗号化し、参加ノードに返します。
5. 参加ノードは秘密鍵を復号してインストールします。
6. ノードはクラスターに参加し、秘密鍵でメッセージを暗号化および復号化します。

共有キーストアからの秘密鍵の取得

対称暗号化は、Data Grid 設定の JGroups スタックに **SYM_ENCRYPT** プロトコルを追加して対称暗号化を設定します。これにより、Data Grid クラスターは、指定したキーストアから秘密鍵を取得できます。

1. ノードは、起動時に Data Grid クラスパスのキーストアから秘密鍵をインストールします。
2. ノードはクラスターに参加し、秘密鍵でメッセージを暗号化および復号化します。

非対称暗号化と対称暗号化の比較

証明書認証を持つ **ASYM_ENCRYPT** は、**SYM_ENCRYPT** と比較して、暗号化の追加の層を提供します。秘密鍵のコーディネーターノードへのリクエストを暗号化するキーストアを提供します。Data Grid は、そのシークレットキーを自動的に生成し、クラストラフィックを処理し、秘密鍵の生成時に指定します。たとえば、ノードが離れる場合に新規のシークレットキーを生成するようにクラスターを設定できます。これにより、ノードが証明書認証を回避して古いキーで参加できなくなります。

一方、**SYM_ENCRYPT** は **ASYM_ENCRYPT** よりも高速です。ノードがクラスターコーディネーターとキーを交換する必要がないためです。**SYM_ENCRYPT** への潜在的な欠点は、クラスターのメンバーシップの変更時に新規シークレットキーを自動的に生成するための設定がないことです。ユーザーは、ノードがクラストラフィックを暗号化するのに使用するシークレットキーを生成して配布する必要があります。

4.9.2. 非対称暗号化を使用したクラストランスポートのセキュア化

Data Grid クラスターを設定し、JGroups メッセージを暗号化するシークレットキーを生成して配布します。

手順

1. Data Grid がノードの ID を検証できるようにする証明書チェーンでキーストアを作成します。
2. クラスター内の各ノードのクラスパスにキーストアを配置します。
   Data Grid Server の場合は、$RHDG_HOME ディレクトリーにキーストアを配置します。

3. 以下の例のように、**SSL_KEY_EXCHANGE** プロトコルおよび **ASYM_ENCRYPT** プロトコルを Data Grid 設定の JGroups スタックに追加します。

```
<infinispan>
 <jgroups>
  <!-- Creates a secure JGroups stack named "encrypt-tcp" that extends the default TCP stack. -->
  <stack name="encrypt-tcp" extends="tcp">
   <!-- Adds a keystore that nodes use to perform certificate authentication. -->
   <!-- Uses the stack.combine and stack.position attributes to insert SSL_KEY_EXCHANGE into the default TCP stack after VERIFY_SUSPECT. -->
   <SSL_KEY_EXCHANGE keystore_name="mykeystore.jks"
                     keystore_password="changeit"
                     stack.combine="INSERT_AFTER"
                     stack.position="VERIFY_SUSPECT"/>
   <!-- Configures ASYM_ENCRYPT -->
   <!-- Uses the stack.combine and stack.position attributes to insert ASYM_ENCRYPT into the default TCP stack before pbcast.NAKACK2. -->
   <!-- The use_external_key_exchange = "true" attribute configures nodes to use the `SSL_KEY_EXCHANGE` protocol for certificate authentication. -->
   <ASYM_ENCRYPT asym_keylength="2048"
                 asym_algorithm="RSA"
                 change_key_on_coord_leave = "false"
                 change_key_on_leave = "false"
                 use_external_key_exchange = "true"
                 stack.combine="INSERT_BEFORE"
                 stack.position="pbcast.NAKACK2"/>
  </stack>
 </jgroups>
 <cache-container name="default" statistics="true">
  <!-- Configures the cluster to use the JGroups stack. -->
  <transport cluster="${infinispan.cluster.name}"
             stack="encrypt-tcp"
             node-name="${infinispan.node.name:}"/>
 </cache-container>
</infinispan>
```

検証

Data Grid クラスターを起動した際、以下のログメッセージは、クラスターがセキュアな JGroups スタックを使用していることを示しています。

```
[org.infinispan.CLUSTER] ISPN000078: Starting JGroups channel cluster with stack <encrypted_stack_name>
```

Data Grid ノードは **ASYM_ENCRYPT** を使用している場合のみクラスターに参加でき、コーディネーターノードからシークレットキーを取得できます。それ以外の場合は、次のメッセージが Data Grid ログに書き込まれます。

```
[org.jgroups.protocols.ASYM_ENCRYPT] <hostname>: received message without encrypt header from <hostname>; dropping it
```

関連情報

- JGroups 4 Manual

- JGroups 4.2 Schema

4.9.3. 対称暗号化を使用したクラストートランスポートのセキュア化

指定したキーストアからの秘密鍵を使用して JGroups メッセージを暗号化するように Data Grid クラスターを設定します。

手順

1. シークレットキーが含まれるキーストアを作成します。
2. クラスター内の各ノードのクラスパスにキーストアを配置します。
   Data Grid Server の場合は、$RHDG_HOME ディレクトリーにキーストアを配置します。
3. Data Grid 設定の JGroups スタックに **SYM_ENCRYPT** プロトコルを追加します。

```
<infinispan>
  <jgroups>
    <!-- Creates a secure JGroups stack named "encrypt-tcp" that extends the default TCP stack. -->
    <stack name="encrypt-tcp" extends="tcp">
      <!-- Adds a keystore from which nodes obtain secret keys. -->
      <!-- Uses the stack.combine and stack.position attributes to insert SYM_ENCRYPT into the
default TCP stack after VERIFY_SUSPECT. -->
      <SYM_ENCRYPT keystore_name="myKeystore.p12"
                   keystore_type="PKCS12"
                   store_password="changeit"
                   key_password="changeit"
                   alias="myKey"
                   stack.combine="INSERT_AFTER"
                   stack.position="VERIFY_SUSPECT"/>
    </stack>
  </jgroups>
  <cache-container name="default" statistics="true">
    <!-- Configures the cluster to use the JGroups stack. -->
    <transport cluster="${infinispan.cluster.name}"
               stack="encrypt-tcp"
               node-name="${infinispan.node.name:}"/>
  </cache-container>
</infinispan>
```

検証

Data Grid クラスターを起動した際、以下のログメッセージは、クラスターがセキュアな JGroups スタックを使用していることを示しています。

```
[org.infinispan.CLUSTER] ISPN000078: Starting JGroups channel cluster with stack
<encrypted_stack_name>
```

Data Grid ノードは、**SYM_ENCRYPT** を使用し、共有キーストアからシークレットキーを取得できる場合に限りクラスターに参加できます。それ以外の場合は、次のメッセージが Data Grid ログに書き込まれます。

```
[org.jgroups.protocols.SYM_ENCRYPT] <hostname>: received message without encrypt header from
<hostname>; dropping it
```

関連情報

- JGroups 4 Manual
- JGroups 4.2 Schema

4.10. クラスタートラフィックの TCP および UDP ポート

Data Grid は、クラスタートランスポートメッセージに以下のポートを使用します。

デフォルトのポート	Protocol	説明
7800	TCP/UDP	JGroups クラスターバインドポート
46655	UDP	JGroups マルチキャスト

クロスサイトレプリケーション
Data Grid は、JGroups RELAY2 プロトコルに以下のポートを使用します。

7900
OpenShift で実行している Data Grid クラスターの向け。

7800
ノード間のトラフィックに UDP を使用し、クラスター間のトラフィックに TCP を使用する場合。

7801
ノード間のトラフィックに TCP を使用し、クラスター間のトラフィックに TCP を使用する場合。