Red Hat Data Grid 8.1

Data Grid Library Mode

Data Grid を組み込みライブラリーとして実行

Last Updated: 2023-12-15

Red Hat Data Grid 8.1 Data Grid Library Mode

Data Grid を組み込みライブラリーとして実行

法律上の通知

概要

依存関係を設定し、Data Grid をカスタムアプリケーションの組み込みライブラリーとして実行します。

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。

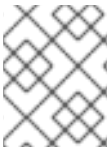
DATA GRID のドキュメント

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 8.1 ドキュメント
- Data Grid 8.1 コンポーネントの詳細
- Data Grid 8.1 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 をご覧ください。

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 8.1 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デプロイメントします。
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>11.0.9.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 ライブラリーモードでの DATA GRID のインストール

Data Grid を組み込みライブラリーとしてプロジェクトに追加します。

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>
```

第3章 DATA GRID を組み込みライブラリーとして実行

Data Grid を組み込みデータストアとしてプロジェクトで実行する方法を学びます。

手順

- デフォルトの Cache Manager を初期化し、以下のようにキャッシュ定義を追加します。

```
GlobalConfigurationBuilder global = GlobalConfigurationBuilder.defaultClusteredBuilder();
DefaultCacheManager cacheManager = new DefaultCacheManager(global.build());
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.clustering().cacheMode(CacheMode.DIST_SYNC);
cacheManager.administration().withFlags(CacheContainerAdmin.AdminFlag.VOLATILE).getOrCreateC
ache("myCache", builder.build());
```

上記のコードは、デフォルトのクラスター化された Cache Manager を初期化します。Cache Manager にはキャッシュ定義が含まれ、キャッシュライフサイクルを制御します。

Data Grid はデフォルトのキャッシュ定義を提供しないため、デフォルトの Cache Manager を初期化した後に、少なくとも 1 つのキャッシュインスタンスを追加する必要があります。この例では、**ConfigurationBuilder** クラスを使用して、分散型同期キャッシュモードを使用するキャッシュ定義を作成します。次に、クラスターのすべてのノードで "myCache" という名前のキャッシュを作成するか、すでに存在する場合はこれを返す **getOrCreateCache()** メソッドを呼び出します。

次のステップ

キャッシュが作成された実行中の Cache Manager ができたので、必要に応じて、キャッシュ定義を追加したり、データをキャッシュに入れたり、Data Grid を設定したりできます。

参照

- プログラムによる Data Grid の設定
- org.infinispan.configuration.global.GlobalConfigurationBuilder
- org.infinispan.manager.EmbeddedCacheManager
- org.infinispan.Cache

第4章 SETTING UP DATA GRID CLUSTERS

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. デフォルトスタックの使用開始

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

1. **infinispan-core-11.0.9.Final-redhat-00001.jar** ファイル内の **default-configs** ディレクトリーで、デフォルトの JGroups スタック **default-jgroups-*.xml** を見つけます。
2. 次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">
    <transport cluster="${infinispan.cluster.name}"
               stack="udp" 1
               node-name="${infinispan.node.name:}"/>
  </cache-container>
</infinispan>
```

1 クラストランスポートには **default-jgroups-udp.xml** を使用します。

   - **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")
        .addProperty("configurationFile", "default-jgroups-udp.xml") 1
        .build();
```

1 クラストランスポートに **default-jgroups-udp.xml** スタックを使用します。

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
```

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)

4.1.1. デフォルトの JGroups スタック

クラストートランスポートを設定するデフォルトの JGroups スタックについて説明します。

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-ec2.xml	**ec2**	トランスポートに TCP を使用し、検出に **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。
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-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。
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。

参照

- JGroups Protocols

4.1.2. クラストートラフィックの TCP および UDP ポート

Data Grid は、クラスト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	プロトコル	説明
7800	TCP/UDP	JGroups クラスターバインドポート

デフォルトのポート	プロトコル	説明
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。

4.2. 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。

```
<infinispan>
  <jgroups>
    <stack name="my-stack"> 1
    </stack>
  </jgroups>
</infinispan>
```

1 "my-stack" という名前のカスタム JGroups スタックを作成します。

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。

```
<infinispan>
  <jgroups>
    <stack name="my-stack" extends="tcp"> 1
    </stack>
  </jgroups>
</infinispan>
```

1 デフォルトの TCP スタックから継承します。

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。
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。

```
<jgroups>
  <stack name="my-stack" extends="tcp">
    <TCPGOSSIP initial_hosts="${jgroups.tunnel.gossip_router_hosts:localhost[12001]}"
          stack.combine="REPLACE"
          stack.position="MPING" /> (1)
    <FD_SOCK stack.combine="REMOVE"/> (2)
    <VERIFY_SUSPECT timeout="2000"/> (3)
    <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" /> (4)
  </stack>
</jgroups>
```

(1) **MPING** の代わりに **TCPGOSSIP** プロトコルを検出メカニズムとして使用します。

(2) **FD_SOCK** プロトコルをスタックから削除します。

(3) **VERIFY_SUSPECT** プロトコルのタイムアウト値を変更します。

(4) **SYM_ENCRYPT** プロトコルをスタックの **VERIFY_SUSPECT** プロトコルの後に追加します。

5. スタック名を **transport** 設定の **stack** 属性の値として指定します。

```
<infinispan>
  <jgroups>
    <stack name="my-stack" extends="tcp">
      ...
    </stack>
    <cache-container name="default" statistics="true">
      <transport cluster="${infinispan.cluster.name}"
                 stack="my-stack" (1)
                 node-name="${infinispan.node.name:}"/>
    </cache-container>
  </jgroups>
</infinispan>
```

(1) クラスタートランスポートに "my-stack" を使用するように Data Grid を設定します。

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)

4.2.1. 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。
REMOVE	スタックからプロトコルを削除します。

4.3. JGROUPS システムプロパティーの使用

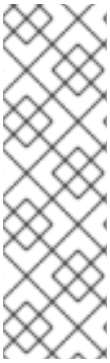
起動時にシステムプロパティーを Data Grid に渡して、クラスターのトランスポートを調整します。

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。

4.3.1. JGroups スタックのシステムプロパティー

JGroups クラストートランスポートスタックを設定するシステムプロパティーを設定します。

システムプロパティー	説明	Default Value	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**	オプション
jgroups.bind.port	ソケットのバインドポート。	**7800**	オプション
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**	オプション
jgroups.mcast_port	マルチキャストソケットのポート。	**46655**	オプション
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	オプション
jgroups.thread_pool.min_threads	スレッドプールの最小スレッド数	0	オプション
jgroups.thread_pool.max_threads	スレッドプールの最大スレッド数	200	オプション
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	オプション

システムプロパティー	説明	Default Value	必須/オプション
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	オプション

Amazon EC3

以下のシステムプロパティーは **default-jgroups-ec2.xml** のみに適用されます。

システムプロパティー	説明	Default Value	必須/オプション
jgroups.s3.access_key	S3 バケットの Amazon S3 アクセスキー。	デフォルト値はありません。	オプション
jgroups.s3.secret_access_key	S3 バケットに使用される Amazon S3 シークレットキー。	デフォルト値はありません。	オプション
jgroups.s3.bucket	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。	オプション

Kubernetes

以下のシステムプロパティーは **default-jgroups-kubernetes.xml** のみに適用されます。

システムプロパティー	説明	Default Value	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

Google Cloud Platform

以下のシステムプロパティーは **default-jgroups-google.xml** のみに適用されます。

システムプロパティー	説明	Default Value	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

参照

- JGroups System Properties
- JGroups Protocol List

4.4. 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>
 <jgroups> 1
  <stack name="prod"> 2
   <TCP bind_port="7800" port_range="30" recv_buf_size="20000000"
send_buf_size="640000"/>
   <MPING bind_addr="127.0.0.1" 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="${jgroups.udp.ip_ttl:2}"/>
   <MERGE3 />
   <FD_SOCK />
   <FD_ALL timeout="3000" interval="1000" timeout_check_interval="1000" />
   <VERIFY_SUSPECT timeout="1000" />
   <pbcast.NAKACK2 use_mcast_xmit="false" xmit_interval="100"
xmit_table_num_rows="50"
                xmit_table_msgs_per_row="1024" xmit_table_max_compaction_time="30000"
/>
   <UNICAST3 xmit_interval="100" xmit_table_num_rows="50"
xmit_table_msgs_per_row="1024"
            xmit_table_max_compaction_time="30000" />
   <pbcast.STABLE stability_delay="200" desired_avg_gossip="2000" max_bytes="1M" />
   <pbcast.GMS print_local_addr="false" join_timeout="${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">
  <transport stack="prod" /> 3
  ...
 </cache-container>
</infinispan>
```

1 1 つ以上の JGroups スタック定義を含めます。

2 "prod" という名前のカスタム JGroups スタックを定義します。

3 クラスタートランスポートに "prod" を使用するように Data Grid を設定します。

4.5. 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>
    <stack-file name="prod-tcp" path="prod-jgroups-tcp.xml"/> 1
  </jgroups>
  <cache-container default-cache="replicatedCache">
    <transport stack="prod-tcp" /> 2
    <replicated-cache name="replicatedCache"/>
  </cache-container>
  ...
</infinispan>
```

1. "prod-jgroups-tcp.xml" 定義を使用する "prod-tcp" という名前のスタックを作成します。
2. クラスタートランスポートに "prod-tcp" を使用するように Data Grid を設定します。

4.6. 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。

参照資料

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

4.6.1. 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

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

参照

- JGroups PING

4.6.2. 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

```
<config>
  <TCP bind_port="7800" />
  <TCPPING timeout="3000"
           initial_hosts="${jgroups.tcpping.initial_hosts:hostname1[port1],hostname2[port2]}"
           port_range="0" 1
           num_initial_members="3"/>
  ...
</config>
```

1 検出の信頼性を高めるために、Red Hat は **port-range=0** を推奨します。

参照

- JGroups TCPPING

4.6.3. 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

```
<config>
  <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}"/>
  ...
</config>
```

参照

- JGroups MPING

4.6.4. 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

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

参照

- JGroups Gossip Router

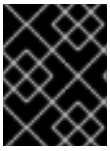
4.6.5. JDBC_PING

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 クラスターから離脱すると、そのノードの IP アドレスが共有データベースから削除されます。

JDBC_PING 設定の例

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



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。

参照

- JDBC_PING
- JDBC_PING Wiki

4.6.6. 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

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

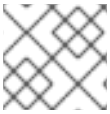
参照

- 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)

4.7. 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。

参照

JGroups JChannel

4.8. 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。

4.8.1. Data Grid クラスターのセキュリティー

クラス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、ノードがクラスター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。

4.8.2. 非対称暗号化を使用したクラスタートランスポートの設定

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>
    <stack name="encrypt-tcp" extends="tcp"> (1)
      <SSL_KEY_EXCHANGE keystore_name="mykeystore.jks" (2)
                        keystore_password="changeit" (3)
                        stack.combine="INSERT_AFTER"
                        stack.position="VERIFY_SUSPECT"/> (4)
      <ASYM_ENCRYPT asym_keylength="2048" (5)
                    asym_algorithm="RSA" (6)
                    change_key_on_coord_leave = "false" (7)
                    change_key_on_leave = "false" (8)
                    use_external_key_exchange = "true" (9)
                    stack.combine="INSERT_AFTER"
                    stack.position="SSL_KEY_EXCHANGE"/> (10)
    </stack>
  </jgroups>
  <cache-container name="default" statistics="true">
    <transport cluster="${infinispan.cluster.name}"
               stack="encrypt-tcp" (11)
               node-name="${infinispan.node.name:}"/>
  </cache-container>
</infinispan>
```

(1) Data Grid のデフォルト TCP スタックを拡張する "encrypt-tcp" という名前のセキュアな JGroups スタックを作成します。

(2) ノードが証明書認証を実行するために使用するキーストアに名前を付けます。

(3) キーストアのパスワードを指定します。

(4) **stack.combine** 属性と **stack.position** 属性を使用して、デフォルトの TCP スタックの **VERIFY_SUSPECT** プロトコルの後に **SSL_KEY_EXCHANGE** を挿入します。

(5) コーディネーターノードが生成する秘密鍵の長さを指定します。デフォルト値は **2048** です。

6 コーディネーターノードが秘密鍵の生成に使用する暗号化エンジンを指定します。デフォルト値は **RSA** です。

7 コーディネーターノードが変更されたときに新しい秘密鍵を生成して配布するように Data Grid を設定します。

8 ノードが離脱するときに新しい秘密鍵を生成して配布するように Data Grid を設定します。

9 証明書認証に **SSL_KEY_EXCHANGE** プロトコルを使用するように Data Grid ノードを設定します。

10 **stack.combine** 属性と **stack.position** 属性を使用して、デフォルトの TCP スタックの **SSL_KEY_EXCHANGE** プロトコルの後に **ASYM_ENCRYPT** を挿入します。

11 セキュアな JGroups スタックを使用するように Data Grid クラスターを設定します。

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
```

参照資料

この手順の **ASYM_ENCRYPT** の設定例は、一般的に使用されるパラメーターを示しています。利用可能なパラメーターの完全なセットについては、JGroups のドキュメントを参照してください。

- JGroups 4 Manual
- JGroups 4.2 Schema

4.8.3. 対称暗号化を使用したクラストートランスポートの設定

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. 次の例のように、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>
      <stack name="encrypt-tcp" extends="tcp"> (1)
        <SYM_ENCRYPT keystore_name="myKeystore.p12" (2)
                     keystore_type="PKCS12" (3)
                     store_password="changeit" (4)
                     key_password="changeit" (5)
                     alias="myKey" (6)
                     stack.combine="INSERT_AFTER"
                     stack.position="VERIFY_SUSPECT"/> (7)
      </stack>
   </jgroups>
   <cache-container name="default" statistics="true">
     <transport cluster="${infinispan.cluster.name}"
                stack="encrypt-tcp" (8)
                node-name="${infinispan.node.name:}"/>
   </cache-container>
</infinispan>
```

(1) Data Grid のデフォルト TCP スタックを拡張する "encrypt-tcp" という名前のセキュアな JGroups スタックを作成します。

(2) ノードが秘密鍵を取得するキーストアに名前を付けます。

(3) キーストアのタイプを指定します。JGroups はデフォルトで JCEKS を使用します。

(4) キーストアのパスワードを指定します。

(5) 秘密鍵のパスワードを指定します。

(6) 秘密鍵のエイリアスを指定します。

(7) **stack.combine** 属性と **stack.position** 属性を使用して、デフォルトの TCP スタックの **VERIFY_SUSPECT** プロトコルの後に **SYM_ENCRYPT** を挿入します。

(8) セキュアな JGroups スタックを使用するように Data Grid クラスターを設定します。

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
```

参照資料

この手順の **SYM_ENCRYPT** の設定例は、一般的に使用されるパラメーターを示しています。利用可能なパラメーターの完全なセットについては、JGroups のドキュメントを参照してください。

- JGroups 4 Manual
- JGroups 4.2 Schema