Red Hat Data Grid 8.3

Data Grid Server ガイド

Data Grid Server のデプロイメントのデプロイ、保護、および管理

Last Updated: 2023-12-06

Red Hat Data Grid 8.3 Data Grid Server ガイド

Data Grid Server のデプロイメントのデプロイ、保護、および管理

法律上の通知

Copyright © 2023 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

Data Grid Server デプロイメントをインストールして設定します。

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。

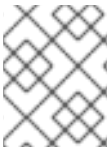
DATA GRID のドキュメント

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 SERVER を使い始める

Data Grid Server を素早く設定し、基本情報を確認します。

Ansible コレクション

オプションで Keycloak キャッシュおよびサイト間のレプリケーション設定が含まれる Ansible コレクションを使用して、Data Grid クラスターのインストールを自動化します。Ansible コレクションでは、インストール時に各サーバーインスタンスの静的設定に Data Grid キャッシュを挿入することもできます。

Data Grid の Ansible コレクション は、Red Hat **Automation Hub** から入手できます。

1.1. DATA GRID SERVER の要件

Data Grid Server には Java 仮想マシンが必要です。サポートされるバージョンの詳細は、Data Grid Supported Configurations を参照してください。

1.2. DATA GRID SERVER ディストリビューションのダウンロード

Data Grid Server ディストリビューションは、Java ライブラリー (**JAR** ファイル) および設定ファイルのアーカイブです。

手順

1. Red Hat カスタマーポータルにアクセスします。
2. ソフトウェアダウンロードセクション から Red Hat Data Grid 8.3 Server をダウンロードします。
3. 以下のように、サーバーダウンロードアーカイブを引数として、**md5sum** または **sha256sum** を実行します。

```
sha256sum jboss-datagrid-${version}-server.zip
```

4. Data Grid **Software Details** ページで **MD5** または **SHA-256** チェックサムの値と比較します。

参照資料

- Data Grid Server README には、サーバーディストリビューションの内容が記載されています。

1.3. DATA GRID SERVER のインストール

ホストシステムに Data Grid Server ディストリビューションをインストールします。

前提条件

- Data Grid Server ディストリビューションアーカイブをダウンロードします。

手順

- 適切なツールを使用して Data Grid Server のアーカイブをホストファイルシステムにデプロイメントします。

```
unzip redhat-datagrid-8.3.1-server.zip
```

作成されたディレクトリーは **$RHDG_HOME** です。

1.4. DATA GRID SERVER の起動

サポートされる任意のホスト上の Java 仮想マシン (JVM) で、Data Grid Server インスタンスを実行します。

前提条件

- サーバーディストリビューションをダウンロードしてインストールします。

手順

1. **$RHDG_HOME** でターミナルを開きます。
2. **server** スクリプトで Data Grid Server インスタンスを起動します。

   Linux

   ```
   bin/server.sh
   ```

   Microsoft Windows

   ```
   bin\server.bat
   ```

以下のメッセージがログに記録される場合は、Data Grid Server は正常に実行されています。

```
ISPN080004: Protocol SINGLE_PORT listening on 127.0.0.1:11222
ISPN080034: Server '...' listening on http://127.0.0.1:11222
ISPN080001: Data Grid Server <version> started in <mm>ms
```

検証

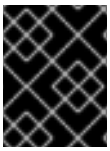
1. 任意のブラウザーで **127.0.0.1:11222/console/** を開きます。
2. プロンプトで認証情報を入力し、Data Grid Console に進みます。

1.5. 起動時に DATA GRID SERVER 設定を渡す

Data Grid Server の起動時にカスタム設定を指定します。

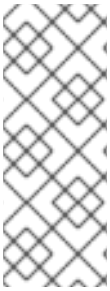
Data Grid Server は、**--server-config** 引数を使用して起動時にオーバーレイする複数の設定ファイルを解析できます。必要な数の設定オーバーレイファイルを任意の順序で使用できます。設定オーバーレイファイル:

- 有効なデータグリッド設定であり、ルート **server** 要素またはフィールドが含まれている必要があります。
- オーバーレイファイルの組み合わせが完全な設定になる限り、完全な設定である必要はありません。



重要

Data Grid Server は、オーバーレイファイル間の競合する設定を検出しません。各オーバーレイファイルは、前述の設定で競合する設定を上書きします。



注記

起動時にキャッシュ設定を Data Grid Server に渡した場合、これらのキャッシュはクラスター全体に動的に作成されません。キャッシュを各ノードに手動で伝播する必要があります。

さらに、起動時に Data Grid Server に渡すキャッシュ設定には、**infinispan** および **cache-container** 要素が含まれている必要があります。

前提条件

- サーバーディストリビューションをダウンロードしてインストールします。
- カスタムサーバー設定を Data Grid Server インストールの **server/conf** ディレクトリーに追加します。

手順

1. **$RHDG_HOME** でターミナルを開きます。
2. 以下のように、**--server-config=** または **-c** 引数を使用して 1 つ以上の設定ファイルを指定します。

```
bin/server.sh -c infinispan.xml -c datasources.yaml -c security-realms.json
```

1.6. DATA GRID ユーザーの作成および変更

Data Grid ユーザーの認証情報を追加し、データへのアクセスを制御するパーミッションを割り当てます。

Data Grid サーバーのインストールは、プロパティーレルムを使用して、Hot Rod エンドポイントおよび REST エンドポイントのユーザーを認証します。これは、Data Grid にアクセスする前に 1 人以上のユーザーを作成する必要があることを意味します。

デフォルトでは、ユーザーはキャッシュにアクセスして Data Grid リソースと対話するためのパーミッションを持つロールも必要です。ユーザーにロールを個別に割り当てたり、ロールパーミッションを持つグループにユーザーを追加したりすることができます。

Data Grid コマンドラインインターフェイス (CLI) の **user** コマンドを使用して、ユーザーを作成し、ロールを割り当てます。

ヒント

CLI セッションから **help user** を実行し、コマンドの詳細を取得します。

1.6.1. 認証情報の追加

Data Grid Console の **admin** ユーザーと、Data Grid 環境を完全に制御する必要があります。このため、初めて認証情報を追加する時に **admin** パーミッションを持つユーザーを作成する必要があります。

手順

1. **$RHDG_HOME** でターミナルを開きます。
2. **user create** コマンドで **admin** ユーザーを作成します。
   - **admin** グループに割り当てられているユーザーを追加します。

     ```
     bin/cli.sh user create myuser -p changeme -g admin
     ```

   - 暗黙的な承認を使用して **admin** パーティションを取得します。

     ```
     bin/cli.sh user create admin -p changeme
     ```

3. 任意のテキストエディターで、**user.properties** および **groups.properties** を開き、ユーザーおよびグループを確認します。

```
$ cat server/conf/users.properties

#$REALM_NAME=default$
#$ALGORITHM=encrypted$
myuser=scram-sha-1\:BYGcIAwvf6b...

$ cat server/conf/groups.properties

myuser=admin
```

1.6.2. ユーザーへのロールの割り当て

ユーザーにロールを割り当て、ユーザーがデータにアクセスし、Data Grid リソースを変更するための適切なパーミッションを持つようにします。

手順

1. **admin** ユーザーで CLI セッションを開始します。

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

2. **deployer** ロールを katie に割り当てます。

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

3. katie のロールをリスト表示します。

   ```
   [//containers/default]> user roles ls katie
   ["deployer"]
   ```

1.6.3. グループへのユーザーの追加

グループを使用すると、複数のユーザーのパーミッションを変更できます。グループにロールを割り当ててから、そのグループにユーザーを追加します。ユーザーは、グループロールからパーミッションを継承します。

手順

1. **admin** ユーザーで CLI セッションを開始します。
2. **user create** コマンドを使用してグループを作成します。
   a. **--groups** 引数を使用して、グループ名として developers を指定します。
   b. グループのユーザー名とパスワードを設定します。
   プロパティーレルムでは、グループは特別なタイプのユーザーで、ユーザー名とパスワードも必要です。

```
[//containers/default]> user create --groups=developers developers -p changeme
```

3. グループをリスト表示します。

```
[//containers/default]> user ls --groups
["developers"]
```

4. **application** ロールを developers グループに割り当てます。

```
[//containers/default]> user roles grant --roles=application developers
```

5. developers グループのロールをリスト表示します。

```
[//containers/default]> user roles ls developers
["application"]
```

6. 必要に応じて、既存のユーザーを一度に 1 人ずつグループに追加します。

```
[//containers/default]> user groups john --groups=developers
```

1.6.4. ユーザーロールとパーミッション

Data Grid には、データにアクセスして Data Grid リソースと対話するためのパーミッションをユーザーに付与するデフォルトのロールのセットが含まれています。

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。

重要

ClusterRoleMapper は、プリンシパル名をロール名に一致させます。**admin** という名前のユーザーは **admin** パーミッションを自動的に取得し、**deployer** という名前のユーザーは **deployer** パーミッションを取得する、というようになります。

ロール	パーミッション	説明
admin	ALL	Cache Manager ライフサイクルの制御など、すべてのパーミッションを持つスーパーユーザー。
deployer	ALL_READ、ALL_WRITE、LISTEN、EXEC、MONITOR、CREATE	**application** パーミッションに加えて、Data Grid リソースを作成および削除できます。
application	ALL_READ、ALL_WRITE、LISTEN、EXEC、MONITOR	**observer** パーミッションに加え、Data Grid リソースへの読み取りおよび書き込みアクセスがあります。また、イベントをリッスンし、サーバータスクおよびスクリプトを実行することもできます。
observer	ALL_READ、MONITOR	**monitor** パーミッションに加え、Data Grid リソースへの読み取りアクセスがあります。
monitor	MONITOR	JMX および **metrics** エンドポイント経由で統計を表示できます。

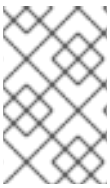
参照資料

- org.infinispan.security.AuthorizationPermission Enumeration
- Data Grid configuration schema reference

1.7. クラスタービューの確認

同じネットワーク上の Data Grid Server インスタンスは、自動的に相互に検出し、クラスターを形成します。

ローカルで実行されている Data Grid Server インスタンスを使用して、デフォルトの **TCP** スタックで **MPING** プロトコルを使用してクラスター検出を監視するには、この手順を実行します。カスタムネットワーク要件のクラスタートランスポートを調整する場合は、Data Grid クラスターの設定に関するドキュメントを参照してください。



注記

この手順は、クラスター検出の原則を示すことを目的としており、実稼働環境を対象としていません。コマンドラインでポートオフセットを指定することは、実稼働用のクラスタートランスポートを設定するための信頼できる方法ではありません。

前提条件

Data Grid Server の 1 つのインスタンスを実行している。

手順

1. **$RHDG_HOME** でターミナルを開きます。
2. root ディレクトリーを **server2** にコピーします。

```
cp -r server server2
```

3. ポートオフセットと **server2** ディレクトリーを指定します。

```
bin/server.sh -o 100 -s server2
```

検証

127.0.0.1:11222/console/cluster-membership のコンソールでクラスターのメンバーシップを表示できます。

Data Grid は、ノードがクラスターに参加する際に、以下のメッセージも記録します。

```
INFO  [org.infinispan.CLUSTER] (jgroups-11,<server_hostname>)
ISPN000094: Received new cluster view for channel cluster:
[<server_hostname>|3] (2) [<server_hostname>, <server2_hostname>]

INFO  [org.infinispan.CLUSTER] (jgroups-11,<server_hostname>)
ISPN100000: Node <server2_hostname> joined the cluster
```

1.8. DATA GRID SERVER のシャットダウン

個別に実行中のサーバーを停止するか、クラスターを正常に停止します。

手順

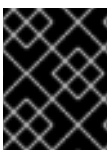
1. Data Grid への CLI 接続を作成します。
2. 次のいずれかの方法で Data Grid Server をシャットダウンします。
   - **shutdown cluster** コマンドを使用して、クラスターのすべてのノードを停止します。以下に例を示します。

```
shutdown cluster
```

     このコマンドは、クラスターの各ノードの **data** フォルダーにクラスターの状態を保存します。キャッシュストアを使用する場合、**shutdown cluster** コマンドはキャッシュのすべてのデータも永続化します。

   - **shutdown server** コマンドおよびサーバーのホスト名を使用して、個々のサーバーインスタンスを停止します。以下に例を示します。

```
shutdown server <my_server01>
```



重要

shutdown server コマンドは、リバランス操作が完了するまで待機しません。これにより、同時に複数のホスト名を指定すると、データが失われる可能性があります。

ヒント

このコマンドの使用方法の詳細については、**help shutdown** を実行してください。

検証

Data Grid は、サーバーをシャットダウンしたときに以下のメッセージをログに記録します。

```
ISPN080002: Data Grid Server stopping
ISPN000080: Disconnecting JGroups channel cluster
ISPN000390: Persisted state, version=<$version> timestamp=YYYY-MM-DDTHH:MM:SS
ISPN080003: Data Grid Server stopped
```

1.8.1. Data Grid クラスターの再起動

シャットダウン後に Data Grid クラスターをオンラインに戻す場合、クラスターが利用できるのを待ってから、ノードの追加または削除、またはクラスター状態の変更を行う必要があります。

shutdown server コマンドでクラスター化ノードをシャットダウンする場合は、各サーバーを逆の順序で再起動する必要があります。
たとえば、**server1** をシャットダウンしてから、**server2** をシャットダウンする場合は、最初に **server2** を起動してから **server1** を起動する必要があります。

shutdown cluster コマンドでクラスターをシャットダウンすると、すべてのノードが再度参加した後にのみ、クラスターは完全に機能するようになります。
ノードは任意の順序で再起動できますが、シャットダウン前に参加していたすべてのノードが実行されるまで、クラスターは DEGRADED 状態のままになります。

1.9. DATA GRID SERVER のインストールディレクトリー構造

Data Grid Server は、**$RHDG_HOME** の下のホストファイルシステムの以下のフォルダーを使用します。

```
├── bin
├── boot
├── docs
├── lib
├── server
└── static
```

ヒント

$RHDG_HOME ディレクトリーの各フォルダーと、ファイルシステムのカスタマイズに使用できるシステムプロパティーに関する詳細は、Data Grid Server README を参照してください。

1.9.1. サーバー root ディレクトリー

bin および **docs** フォルダーのリソースの他に、対話する必要がある **$RHDG_HOME** 下の唯一のフォルダーは、デフォルトで **server** と名付けられたサーバー root ディレクトリーです。

同じ **$RHDG_HOME** ディレクトリーまたは別のディレクトリーに複数のノードを作成できますが、各 Data Grid Server インスタンスには、独自のサーバー root ディレクトリーが必要です。たとえば、5 つのノードのクラスターは、ファイルシステム上に以下のサーバー root ディレクトリーを持つことがで

きます。

```
├── server
├── server1
├── server2
├── server3
└── server4
```

各サーバーの root ディレクトリーには、以下のフォルダーが含まれている必要があります。

```
├── server
│   ├── conf
│   ├── data
│   ├── lib
│   └── log
```

server/conf
Data Grid Server インスタンスの **infinispan.xml** 設定ファイルを保持します。

Data Grid は、設定を 2 つの層に分割します。

動的

データスケーラビリティーの変更可能なキャッシュ設定を作成します。
Data Grid Server は、実行時に作成したキャッシュを、ノード全体に分散されているクラスター状態とともに永続的に保存します。参加している各ノードは、変更が発生するたびに Data Grid Server がすべてのノード間で同期する完全なクラスター状態を受け取ります。

Static

クラスタートランスポート、セキュリティー、共有データソースなど、基盤となるサーバーのメカニズムに対して **infinispan.xml** に設定を追加します。

server/data
Data Grid Server がクラスターの状態を維持するために使用する内部ストレージを提供します。

重要

server/data のコンテンツを直接削除または変更しないでください。

サーバーの実行中に **caches.xml** などのファイルを変更すると、破損が発生する可能性があります。コンテンツを削除すると、誤った状態になる可能性があります。つまり、シャットダウン後にクラスターを再起動できなくなります。

server/lib
カスタムフィルター、カスタムイベントリスナー、JDBC ドライバー、カスタム **ServerTask** 実装などの拡張 **JAR** ファイルが含まれます。

server/log
Data Grid Server のログファイルを保持します。

関連情報

- Data Grid Server README
- What is stored in the <server>/data directory used by a RHDG server（Red Hat ナレッジベース）

第2章 ネットワークインターフェイスおよびソケットバインディング

ネットワークインターフェイスを介して Data Grid Server を公開するには、IP アドレスにバインドします。その後、Data Grid Server がリモートクライアントアプリケーションからの要求を処理できるように、インターフェイスを使用するようにエンドポイントを設定することができます。

2.1. ネットワークインターフェイス

Data Grid Server は、単一の TCP/IP ポートへエンドポイントを多重化し、インバウンドクライアント要求のプロトコルを自動的に検出します。Data Grid Server が、クライアント要求をリッスンするようにネットワークインターフェイスにバインドする方法を設定できます。

インターネットプロトコル (IP) アドレス

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!-- Selects a specific IPv4 address, which can be public, private, or loopback. This is the default network interface for Data Grid Server. -->
  <interfaces>
    <interface name="public">
      <inet-address value="${infinispan.bind.address:127.0.0.1}"/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "inet-address": {
        "value": "127.0.0.1"
      }
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      inetAddress:
        value: "127.0.0.1"
```

ループバックアドレス

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!-- Selects an IP address in an IPv4 or IPv6 loopback address block. -->
  <interfaces>
    <interface name="public">
      <loopback/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "loopback": null
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      loopback: ~
```

非ループバックアドレス

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!-- Selects an IP address in an IPv4 or IPv6 non-loopback address block. -->
  <interfaces>
    <interface name="public">
      <non-loopback/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "non_loopback": null
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      nonLoopback: ~
```

任意のアドレス

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!-- Uses the `INADDR_ANY` wildcard address which means Data Grid Server listens for inbound client requests on all interfaces. -->
  <interfaces>
    <interface name="public">
      <any-address/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "any_address": null
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      anyAddress: ~
```

ローカルリンク

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!-- Selects a link-local IP address in an IPv4 or IPv6 address block. -->
  <interfaces>
    <interface name="public">
      <link-local/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "link_local": null
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      linkLocal: ~
```

サイトローカル

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!-- Selects a site-local (private) IP address in an IPv4 or IPv6 address block. -->
  <interfaces>
    <interface name="public">
      <site-local/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "site_local": null
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      siteLocal: ~
```

2.1.1. 一致およびフォールバックストラテジー

Data Grid Server は、ホストシステム上のネットワークインターフェイスをすべて列挙し、値と一致するインターフェイス、ホスト、または IP アドレスにバインドできます。これには、柔軟性を高めるための正規表現を含むことができます。

ホストの一致

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!-- Selects an IP address that is assigned to a matching host name. -->
  <interfaces>
    <interface name="public">
      <match-host value="my_host_name"/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "match-host": {
        "value": "my_host_name"
      }
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      matchHost:
        value: "my_host_name"
```

インターフェイスの一致

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!--Selects an IP address assigned to a matching network interface. -->
  <interfaces>
    <interface name="public">
      <match-interface value="eth0"/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "match-interface": {
        "value": "eth0"
      }
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      matchInterface:
        value: "eth0"
```

アドレスの一致

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <!-- Selects an IP address that matches a regular expression. -->
  <interfaces>
    <interface name="public">
      <match-address value="132\..*"/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "match-address": {
        "value": "132\\..*"
      }
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      matchAddress:
        value: "127\\..*"
```

- フォールバック

XML

```xml
<server xmlns="urn:infinispan:server:13.0">
  <!-- Includes multiple strategies that Data Grid Server tries in the declared order until it finds a match. -->
  <interfaces>
    <interface name="public">
      <match-host value="my_host_name"/>
      <match-address value="132\..*"/>
      <any-address/>
    </interface>
  </interfaces>
</server>
```

JSON

```
{
  "server": {
    "interfaces": [{
      "name": "public",
      "match-host": {
        "value": "my_host_name"
      },
      "match-address": {
        "value": "132\\..*"
      },
      "any_address": null
    }]
  }
}
```

YAML

```
server:
  interfaces:
    - name: "public"
      matchHost:
        value: "my_host_name"
      matchAddress:
        value: "132\\..*"
      anyAddress: ~
```

2.2. 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コネクターをネットワークインターフェイスおよびポートにマッピングします。デフォルトでは、Data Grid Server には、REST および Hot Rod エンドポイントのポート **11222** で localhost インターフェイス **127.0.0.1** をリッスンするソケットバインディング設定が含まれています。Memcached エンドポイントを有効にすると、デフォルトのソケットバインディングは、ポート **11221** にバインドするように Data Grid Server を設定します。

デフォルトのソケットバインディング

```
<server xmlns="urn:infinispan:server:13.0">
  <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="11221"/>
  </socket-bindings>
</server>
```

設定要素または属性	説明
socket-bindings	Data Grid Server エンドポイントがクライアント接続をバインドおよびリッスンするすべてのネットワークインターフェイスおよびポートが含まれるルート要素。
default-interface	Data Grid Server がデフォルトでリッスンするネットワークインターフェイスを宣言します。
port-offset	Data Grid Server がソケットバインディングのポート宣言に適用するオフセットを指定します。
socket-binding	ネットワークインターフェイスのポートにバインドするように Data Grid Server を設定します。

カスタムソケットバインディング宣言

以下の設定例では、"private" という名前の **interface** 宣言と、Data Grid Server をプライベート IP アドレスにバインドする **socket-binding** 宣言を追加します。

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <interfaces>
    <interface name="public">
      <inet-address value="${infinispan.bind.address:127.0.0.1}"/>
    </interface>
    <interface name="private">
      <inet-address value="10.1.2.3"/>
    </interface>
  </interfaces>

  <socket-bindings default-interface="public"
                   port-offset="${infinispan.socket.binding.port-offset:0}">
    <socket-binding name="private_binding"
                    interface="private"
                    port="49152"/>
  </socket-bindings>
```

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

JSON

```
{
 "server": {
  "interfaces": [{
   "name": "private",
   "inet-address": {
    "value": "10.1.2.3"
   }
  }, {
   "name": "public",
   "inet-address": {
    "value": "127.0.0.1"
   }
  }],
  "socket-bindings": {
   "port-offset": "0",
   "default-interface": "public",
   "socket-binding": [{
    "name": "private_binding",
    "port": "1234",
    "interface": "private"
   }]
  },
  "endpoints": {
   "endpoint": {
    "socket-binding": "private_binding",
    "security-realm": "default"
   }
  }
 }
}
```

YAML

```
server:
  interfaces:
    - name: "private"
      inetAddress:
        value: "10.1.2.3"
    - name: "public"
      inetAddress:
        value: "127.0.0.1"
  socketBindings:
    portOffset: "0"
    defaultInterface: "public"
    socketBinding:
      - name: "private_binding"
        port: "49152"
        interface: "private"
```

```
endpoints:
  endpoint:
    socketBinding: "private_binding"
    securityRealm: "default"
```

2.3. DATA GRID SERVER のバインドアドレスの変更

Data Grid Server は、Hot Rod エンドポイントおよび REST エンドポイントでインバウンドクライアント接続をリッスンするネットワーク IP アドレスにバインドします。IP アドレスを直接 Data Grid Server 設定で指定するか、サーバーインスタンスの起動時に指定できます。

前提条件

- 1 つ以上の Data Grid Server がインストールされている。

手順

以下のいずれかの方法で、Data Grid Server がバインドする IP アドレスを指定します。

- Data Grid Server 設定を開き、**inet-address** 要素の値を設定します。以下に例を示します。

```
<server xmlns="urn:infinispan:server:13.0">
  <interfaces>
    <interface name="custom">
      <inet-address value="${infinispan.bind.address:192.0.2.0}"/>
    </interface>
  </interfaces>
</server>
```

- **-b** オプションまたは **infinispan.bind.address** システムプロパティーを使用します。

  Linux

```
bin/server.sh -b 192.0.2.0
```

  Windows

```
bin\server.bat -b 192.0.2.0
```

2.3.1. すべてのアドレスのリッスン

Data Grid Server 設定のバインドアドレスとして **0.0.0.0** メタアドレスまたは **INADDR_ANY** を指定すると、利用可能なすべてのネットワークインターフェイスで着信クライアント接続をリッスンします。

クライアントのインテリジェンス

すべてのアドレスでリッスンするように Data Grid を設定すると、クラスタートポロジーで Hot Rod クライアントを提供する方法に影響します。Data Grid Server がバインドするインターフェイスが複数ある場合は、各インターフェイスに対して IP アドレスのリストを送信します。

たとえば、各サーバーノードがバインドするクラスターは以下のようになります。

- **10.0.0.0/8** サブネット

- **192.168.0.0/16** サブネット
- **127.0.0.1** ループバック

Hot Rod クライアントは、クライアントが接続するインターフェイスに属するサーバーノードの IP アドレスを受信します。クライアントが **192.168.0.0** に接続する場合、たとえば、**10.0.0.0** でリッスンするノードについては、クラスタートポロジーの詳細を受け取りません。

ネットマスクの上書き

Kubernetes およびその他の環境は、IP アドレスの領域をサブネットに分割し、その異なるサブネットを単一のネットワークとして使用します。たとえば、**10.129.2.100/23** および **10.129.4.100/23** は異なるサブネットにありますが、**10.0.0.0/8** ネットワークに属します。

このため、Data Grid Server は、ホストシステムが提供するネットマスクを、プライベートネットワークと予約ネットワークの IANA 規約に従ったネットマスクで上書きします。

- IPv4: **10.0.0.0/8**、**192.168.0.0/16**、**172.16.0.0/12**、**169.254.0.0/16** および **240.0.0.0/4**
- IPv6: **fc00::/7** および **fe80::/10**

IPv4 の場合は **RFC 1918**、IPv6 の場合は **RFC 4193** および **RFC 3513** を参照してください。

注記

必要に応じて、ホストシステムが Data Grid Server 設定の **network-prefix-override** 属性を持つインターフェイスに提供するネットマスクを使用するように Hot Rod コネクターを設定できます。

関連情報

- Data Grid Server schema reference
- RFC 1918
- RFC 4193
- RFC 3513

2.4. DATA GRID SERVER ポートおよびプロトコル

Data Grid Server は、異なるプロトコルでのクライアントアクセスを許可するネットワークエンドポイントを提供します。

ポート	プロトコル	説明
11222	TCP	Hot Rod および REST
11221	TCP	Memcached(デフォルトでは無効)

単一ポート

Data Grid Server は、1 つの TCP ポート **11222** で複数のプロトコルを公開します。1 つのポートで複数のプロトコルを処理すると、設定が簡素化され、Data Grid クラスターをデプロイする際の管理の複雑

さが軽減されます。また、1 つのポートを使用すると、ネットワーク上の攻撃対象領域が最小限に抑えられるため、セキュリティーも強化されます。

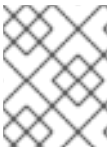
Data Grid Server は、クライアントからの HTTP/1.1、HTTP/2、および Hot Rod プロトコル要求を、さまざまな方法で単一のポートを介して処理します。

HTTP/1.1 アップグレードヘッダー

クライアントリクエストには、**HTTP/1.1 upgrade** ヘッダーフィールドを追加して、Data Grid Server と HTTP/1.1 の接続を開始できます。続いて、クライアントアプリケーションは **Upgrade: protocol** ヘッダーフィールドを送信できます。ここで、**protocol** はサーバーエンドポイントになります。

Application-Layer Protocol Negotiation (ALPN)/Transport Layer Security (TLS)

クライアント要求には、TLS 接続を介してプロトコルをネゴシエートするための Data Grid Server エンドポイントの Server Name Indication (SNI) マッピングが含まれます。



注記

アプリケーションは、ALPN 拡張機能をサポートする TLS ライブラリーを使用する必要があります。Data Grid は、Java 用の WildFly OpenSSL バインディングを使用します。

Hot Rod の自動検出

Hot Rod ヘッダーを含むクライアントリクエストは、自動的に Hot Rod エンドポイントにルーティングされます。

2.4.1. Data Grid トラフィック用のネットワークファイアウォールの設定

ファイアウォールルールを調整して、Data Grid Server とクライアントアプリケーションの間のトラフィックを許可します。

手順

Red Hat Enterprise Linux (RHEL) ワークステーションでは、たとえば、以下のように firewalld を使用してポート **11222** へのトラフィックを許可できます。

```
# firewall-cmd --add-port=11222/tcp --permanent
success
# firewall-cmd --list-ports | grep 11222
11222/tcp
```

ネットワーク全体に適用されるファイアウォールルールを設定するには、nftables ユーティリティーを使用できます。

参照資料

- firewalld の使用および設定
- nftables の使用

2.5. ポートオフセットの指定

同じホストで複数の Data Grid Server インスタンスのポートオフセットを設定します。デフォルトのポートオフセットは **0** です。

手順

Data Grid CLI または **infinispan.socket.binding.port-offset** システムプロパティーで **-o** スイッチを使用して、ポートオフセットを設定します。

たとえば、以下のようにオフセットが **100** のサーバーインスタンスを起動します。デフォルトの設定では、これにより、Data Grid Server がポート **11322** でリッスンします。

Linux

```
bin/server.sh -o 100
```

Windows

```
bin\server.bat -o 100
```

第3章 DATA GRID SERVER エンドポイント

Data Grid Server のエンドポイントは、Hot Rod および REST プロトコルを介して、キャッシュマネージャーへのクライアントアクセスを提供します。

3.1. DATA GRID SERVER エンドポイント

3.1.1. Hot Rod

Hot Rod は、テキストベースのプロトコルと比較して、データへのアクセス時間を短縮し、パフォーマンスを向上するために設計されたバイナリー TCP クライアントサーバープロトコルです。

Data Grid は、Java、C++、C#、Node.js、およびその他のプログラミング言語で Hot Rod クライアントライブラリーを提供します。

トポロジーの状態遷移

Data Grid はトポロジーキャッシュを使用して、クライアントにクラスタービューを提供します。トポロジーキャッシュには、内部 JGroups トランスポートアドレスを公開された Hot Rod エンドポイントにマッピングするエントリーが含まれます。

クライアントが要求を送信すると、Data Grid サーバーは、要求ヘッダーのトポロジー ID をキャッシュからのトポロジー ID と比較します。クライアントに古いトポロジー ID がある場合は、Data Grid サーバーは新しいトポロジービューを送信します。

クラスタートポロジービューを使用すると、Hot Rod クライアントは、ノードがいつ参加および離脱するかを即座に検出できるため、動的な負荷分散とフェイルオーバーが可能になります。

分散キャッシュモードでは、一貫性のあるハッシュアルゴリズムにより、Hot Rod クライアント要求をプライマリー所有者に直接ルーティングすることもできます。

3.1.2. REST

Data Grid は、HTTP クライアントがデータにアクセスし、クラスターを監視および保守し、管理操作を実行できるようにする RESTful インターフェイスを公開します。

標準の HTTP ロードバランサーを使用して、クライアントに負荷分散およびフェイルオーバー機能を提供できます。ただし、HTTP ロードバランサーは静的クラスタービューを維持し、クラスタートポロジーの変更が発生したときに手動で更新する必要があります。

3.1.3. Memcached

Data Grid は、リモートクライアントアクセス用の Memcached テキストプロトコルの実装を提供します。

重要

Memcached エンドポイントは非推奨であり、今後のリリースで削除される予定です。

Data Grid Memcached エンドポイントは、レプリケートおよび分散キャッシュモードを使用したクラスタリングをサポートします。

Cache::Memcached Perl クライアントなどの一部の Memcached クライアント実装は、クラストポロジーの変更時に手動更新を必要とする Data Grid サーバーアドレスの静的リストで負荷分散およびフェイルオーバー検出機能を提供できます。

3.1.4. エンドポイントプロトコルの比較

	Hot Rod	HTTP / REST
トポロジー対応	Y	N
ハッシュ対応	Y	N
暗号化	Y	Y
認証	Y	Y
条件付き操作	Y	Y
バルク操作	Y	N
トランザクション	Y	N
リスナー	Y	N
クエリー	Y	Y
実行	Y	N
クロスサイトフェイルオーバー	Y	N

3.1.5. Hot Rod クライアントの Data Grid Server との互換性

Data Grid Server を使用すると、異なるバージョンの Hot Rod クライアントを接続することができます。たとえば、Data Grid クラスターへの移行またはアップグレードの際に、Hot Rod クライアントのバージョンが Data Grid Server よりも低い Data Grid バージョンになることがあります。

ヒント

Data Grid は、最新の機能およびセキュリティー機能強化の恩恵を受けるために、最新の Hot Rod クライアントバージョンを使用することを推奨しています。

Data Grid 8 以降

Hot Rod プロトコルバージョン 3.x は、Data Grid Server のクライアントに対して、可能な限り高いバージョンを自動的にネゴシエートします。

Data Grid 7.3 以前

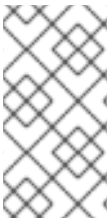
Data Grid Server バージョンよりも高い Hot Rod プロトコルバージョンを使用するクライアントは、**infinispan.client.hotrod.protocol_version** プロパティーを設定する必要があります。

関連情報

- Hot Rod protocol reference
- Connecting Hot Rod clients to servers with different versions (Red Hat ナレッジベース)

3.2. DATA GRID SERVER エンドポイントの設定

Hot Rod および REST エンドポイントがソケットにバインドする方法を制御し、セキュリティーレルム設定を使用します。複数のエンドポイントを設定し、管理機能を無効にすることもできます。



注記

各一意のエンドポイント設定には、Hot Rod コネクターと REST コネクターの両方が含まれている必要があります。Data Grid Server は、**endpoint** 設定に **hotrod-connector** 要素と **rest-connector** 要素またはフィールドを暗黙的に含みます。これらの要素をカスタム設定に追加し、エンドポイントの認証メカニズムを指定する必要があります。

前提条件

- ソケットバインディングとセキュリティーレルムを Data Grid Server 設定に追加します。

手順

1. Data Grid Server 設定を開いて編集します。
2. **endpoints** 要素で複数の **endpoint** 設定をラップします。
3. エンドポイントが **socket-binding** 属性で使用するソケットバインディングを指定します。
4. エンドポイントが **security-realm** 属性で使用するセキュリティーレルムを指定します。
5. 必要に応じて、**admin="false"** 属性を使用して管理者アクセスを無効にします。
   この設定では、ユーザーはエンドポイントから Data Grid Console またはコマンドラインインターフェイス (CLI) にアクセスできません。
6. 変更を設定に保存します。

複数のエンドポイント設定

以下の Data Grid Server 設定は、専用のセキュリティーレルムを持つ個別のソケットバインディングにエンドポイントを作成します。

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <endpoints>
    <endpoint socket-binding="public"
              security-realm="application-realm"
              admin="false">
    </endpoint>
    <endpoint socket-binding="private"
              security-realm="management-realm">
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "endpoints": [{
      "socket-binding": "private",
      "security-realm": "private-realm"
    }, {
      "socket-binding": "public",
      "security-realm": "default",
      "admin": "false"
    }]
  }
}
```

YAML

```
server:
  endpoints:
    - socketBinding: public
      securityRealm: application-realm
      admin: false
    - socketBinding: private
      securityRealm: management-realm
```

関連情報

- ネットワークインターフェイスおよびソケットバインディング

3.3. エンドポイントコネクター

コネクターは、ソケットバインディングおよびセキュリティーレルムを使用するように Hot Rod および REST エンドポイントを設定します。

デフォルトのエンドポイント設定

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

設定要素または属性	説明
endpoints	エンドポイントコネクター設定をラップします。
endpoint	ソケットバインディングおよびセキュリティーレルムを使用するように Hot Rod および REST コネクターを設定する Data Grid Server エンドポイントを宣言します。
hotrod-connector	エンドポイント設定に Hot Rod **endpoint** が含まれます。

設定要素または属性	説明
rest-connector	エンドポイント設定に Hot Rod **endpoint** が含まれます。
memcached-connector	Memcached エンドポイントを設定し、デフォルトでは無効になっています。

関連情報

- Data Grid schema reference

3.4. エンドポイント IP アドレスのフィルタールール

Data Grid Server エンドポイントは、クライアントが IP アドレスに基づいて接続できるかどうかを制御するフィルタールールを使用できます。Data Grid Server は、クライアント IP アドレスの一致を見つけるまで、フィルタリングルールを順番に適用します。

CIDR ブロックは、IP アドレスとそれに関連するネットワークマスクのコンパクトな表現です。CIDR 表記は、IP アドレス、スラッシュ ('/') 文字、および 10 進数を指定します。10 進数は、ネットワークマスクの先頭の 1 ビットのカウントです。この数は、ネットワーク接頭辞の幅 (ビット単位) として考えることもできます。CIDR 表記の IP アドレスは、常に IPv4 または IPv6 の標準仕様によって表されます。

アドレスは、ホスト ID(例: **10.0.0.1/8** など) を含む特定のインターフェイスアドレスを指定できます。もしくは、**10.0.0.0/8** または **10/8** のように 0 のホスト識別子を使用して、ネットワークインターフェイス全体の開始アドレスを指定できます。

以下に例を示します。

- **192.168.100.14/24** は、IPv4 アドレス **192.168.100.14** とその関連付けられたネットワーク接頭辞 **192.168.100.0** を表します。または、先行 1 ビットを 24 個持つサブネットマスク **255.255.255.0** となります。
- IPv4 ブロック **192.168.100.0/22** は、**192.168.100.0** から **192.168.103.255** までの 1024 IPv4 アドレスを表します。
- IPv6 ブロック **2001:db8::/48** は **2001:db8:0:0:0:0:0:0** から **2001:db8:0:ffff:ffff:ffff:ffff:ffff** までの IPv6 アドレスのブロックを表します。
- **::1/128** は IPv6 ループバックアドレスを表します。接頭辞長は 128 で、アドレスのビット数になります。

IP アドレスフィルターの設定

次の設定では、Data Grid Server は **192.168.0.0/16** および **10.0.0.0/8CIDR** ブロックのアドレスからの接続のみを受け入れます。Data Grid Server は、他のすべての接続を拒否します。

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <endpoints>
    <endpoint socket-binding="default" security-realm="default">
      <ip-filter>
        <accept from="192.168.0.0/16"/>
```

```
        <accept from="10.0.0.0/8"/>
        <reject from="/0"/>
      </ip-filter>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "endpoints": {
      "endpoint": {
        "socket-binding": "default",
        "security-realm": "default",
        "ip-filter": {
          "accept-from": ["192.168.0.0/16", "10.0.0.0/8"],
          "reject-from": "/0"
        }
      }
    }
  }
}
```

YAML

```
server:
  endpoints:
    endpoint:
      socketBinding: "default"
      securityRealm: "default"
      ipFilter:
        acceptFrom: ["192.168.0.0/16","10.0.0.0/8"]
        rejectFrom: "/0"
```

3.5. IP アドレスをフィルターするためのルールの検証および変更

クライアントアドレスに基づいて、接続を許可または拒否するように、Data Grid Server エンドポイントで IP アドレスフィルタールールを設定します。

前提条件

- Data Grid コマンドラインインターフェイス (CLI) のインストール

手順

1. Data Grid Server への CLI 接続を作成します。
2. 必要に応じて、IP フィルタールール **server connector ipfilter** コマンドを検査して変更します。
   a. クラスター全体のコネクターでアクティブな IP フィルタリングルールの一覧を表示します。

```
server connector ipfilter ls endpoint-default
```

b. クラスター全体で IP フィルタリングルールを設定します。

注記

このコマンドは、既存のルールを置き換えます。

```
server connector ipfilter set endpoint-default --
rules=ACCEPT/192.168.0.0/16,REJECT/10.0.0.0/8`
```

c. クラスター全体のコネクターですべての IP フィルタリングルールを削除します。

```
server connector ipfilter clear endpoint-default
```

第4章 エンドポイント認証メカニズム

Data Grid Server は、Hot Rod および REST エンドポイントにカスタム SASL および HTTP 認証メカニズムを使用できます。

4.1. DATA GRID SERVER の認証

認証は、エンドポイントへのユーザーアクセスと、Data Grid Console およびコマンドラインインターフェイス (CLI) を制限します。

Data Grid Server には、ユーザー認証を強制するデフォルトのセキュリティーレルムが含まれます。デフォルトの認証は、**server/conf/users.properties** ファイルに保存されているユーザー認証情報とともにプロパティーレルムを使用します。Data Grid Server はデフォルトでセキュリティー認証も有効にするため、**server/conf/groups.properties** ファイルに保存されているパーミッションを持つユーザーを割り当てる必要があります。

ヒント

コマンドラインインターフェイス (CLI) で **user create** コマンドを使用して、ユーザーを追加し、パーミッションを割り当てます。サンプルおよび詳細情報について **user create --help** を実行します。

4.2. DATA GRID SERVER の認証メカニズムの設定

特定の認証メカニズムを使用するように Hot Rod および REST エンドポイントを明示的に設定することができます。認証メカニズムの設定は、セキュリティーレルムのデフォルトメカニズムを明示的に上書きする必要がある場合にのみ必要です。

注記

設定の各 **endpoint** セクションには、**hotrod-connector** および **rest-connector** 要素またはフィールドが含まれている必要があります。たとえば、**hotrod-connector** を明示的に宣言する場合は、認証メカニズムを設定しない場合でも **rest-connector** も宣言する必要があります。

前提条件

- 必要に応じて、Data Grid Server 設定にセキュリティーレルムを追加します。

手順

1. Data Grid Server 設定を開いて編集します。
2. **endpoint** 要素またはフィールドを追加し、**security-realm** 属性で使用するセキュリティーレルムを指定します。
3. **hotrod-connector** 要素またはフィールドを追加して、Hot Rod エンドポイントを設定します。
   a. **authentication** 要素またはフィールドを追加します。
   b. **sasl mechanism** 属性で使用する Hot Rod エンドポイントの SASL 認証メカニズム を指定します。
   c. 該当する場合は、**qop** 属性で SASL 品質の保護設定を指定します。

d. 必要に応じて **server-name** 属性を使用して Data Grid Server アイデンティティーを指定します。

4. **rest-connector** 要素またはフィールドを追加して REST エンドポイントを設定します。

a. **authentication** 要素またはフィールドを追加します。

b. **mechanism** 属性で使用する REST エンドポイントの HTTP 認証メカニズムを指定します。

5. 変更を設定に保存します。

認証メカニズムの設定

以下の設定では、Hot Rod エンドポイントが認証に使用する SASL メカニズムを指定します。

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <endpoints>
    <endpoint socket-binding="default"
              security-realm="my-realm">
      <hotrod-connector>
        <authentication>
          <sasl mechanisms="SCRAM-SHA-512 SCRAM-SHA-384 SCRAM-SHA-256
                            SCRAM-SHA-1 DIGEST-SHA-512 DIGEST-SHA-384
                            DIGEST-SHA-256 DIGEST-SHA DIGEST-MD5 PLAIN"
                server-name="infinispan"
                qop="auth"/>
        </authentication>
      </hotrod-connector>
      <rest-connector>
        <authentication mechanisms="DIGEST BASIC"/>
      </rest-connector>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "endpoints": {
      "endpoint": {
        "socket-binding": "default",
        "security-realm": "my-realm",
        "hotrod-connector": {
          "authentication": {
            "security-realm": "default",
            "sasl": {
              "server-name": "infinispan",
              "mechanisms": ["SCRAM-SHA-512", "SCRAM-SHA-384", "SCRAM-SHA-256", "SCRAM-
SHA-1", "DIGEST-SHA-512", "DIGEST-SHA-384", "DIGEST-SHA-256", "DIGEST-SHA", "DIGEST-
MD5", "PLAIN"],
              "qop": ["auth"]
            }
          }
```

```
        },
        "rest-connector": {
          "authentication": {
            "mechanisms": ["DIGEST", "BASIC"],
            "security-realm": "default"
          }
        }
      }
    }
  }
}
```

YAML

```
server:
  endpoints:
    endpoint:
      socketBinding: "default"
      securityRealm: "my-realm"
      hotrodConnector:
        authentication:
          securityRealm: "default"
          sasl:
            serverName: "infinispan"
            mechanisms:
              - "SCRAM-SHA-512"
              - "SCRAM-SHA-384"
              - "SCRAM-SHA-256"
              - "SCRAM-SHA-1"
              - "DIGEST-SHA-512"
              - "DIGEST-SHA-384"
              - "DIGEST-SHA-256"
              - "DIGEST-SHA"
              - "DIGEST-MD5"
              - "PLAIN"
            qop:
              - "auth"
      restConnector:
        authentication:
          mechanisms:
            - "DIGEST"
            - "BASIC"
          securityRealm: "default"
```

4.2.1. 認証の無効化

ローカル開発環境または分離されたネットワークでは、認証されていないクライアント要求を許可するように Data Grid を設定できます。ユーザー認証を無効にする場合は、Data Grid セキュリティー設定で承認も無効にする必要があります。

手順

1. Data Grid Server 設定を開いて編集します。

2. **endpoints** 要素またはフィールドから **security-realm** 属性を削除します。
3. **cache-container** および各キャッシュ設定の **security** 設定から、**authorization** 要素をすべて削除します。
4. 変更を設定に保存します。

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <endpoints socket-binding="default"/>
</server>
```

JSON

```
{
  "server": {
    "endpoints": {
      "endpoint": {
        "socket-binding": "default"
      }
    }
  }
}
```

YAML

```
server:
  endpoints:
    endpoint:
      socketBinding: "default"
```

4.3. DATA GRID SERVER の認証メカニズム

Data Grid Server は、セキュリティーレルム設定に一致する認証メカニズムでエンドポイントを自動的に設定します。たとえば、Kerberos セキュリティーレルムを追加すると、Data Grid Server は Hot Rod エンドポイントの **GSSAPI** および **GS2-KRB5** 認証メカニズムを有効にします。

ホットローテーションエンドポイント

Data Grid Server は、設定に対応するセキュリティーレルムが含まれている場合に Hot Rod エンドポイントの以下の SASL 認証メカニズムを有効にします。

セキュリティーレルム	SASL 認証メカニズム
プロパティーレルムおよび LDAP レルム	SCRAM-*, DIGEST-*, **SCRAM-***
トークンレルム	OAUTHBEARER
信頼レルム	EXTERNAL

セキュリティーレルム	SASL 認証メカニズム
Kerberos ID	GSSAPI、GS2-KRB5
SSL/TLS ID	PLAIN

REST エンドポイント

Data Grid Server は、設定に対応するセキュリティーレルムが含まれている場合に REST エンドポイントの以下の HTTP 認証メカニズムを有効にします。

セキュリティーレルム	HTTP 認証メカニズム
プロパティーレルムおよび LDAP レルム	DIGEST
トークンレルム	BEARER_TOKEN
信頼レルム	CLIENT_CERT
Kerberos ID	SPNEGO
SSL/TLS ID	BASIC

4.3.1. SASL 認証メカニズム

Data Grid Server は、Hot Rod エンドポイントで以下の SASL 認証メカニズムをサポートします。

認証メカニズム	説明	セキュリティーレルムタイプ	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。	プロパティーレルムおよび LDAP レルム	**BASIC** HTTP 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。ホットロッドコネクターは、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。	プロパティーレルムおよび LDAP レルム	**Digest** HTTP メカニズムに似ています。

認証メカニズム	説明	セキュリティーレルムタイプ	関連する詳細
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ーは、**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。	プロパティーレルムおよび LDAP レルム	**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。	Kerberos レルム	**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。	Kerberos レルム	**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。	トラストストアレルム	**CLIENT_CERTHTTP** 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。	トークンレルム	**BEARER_TOKEN** HTTP メカニズムに似ています。

4.3.2. SASL Quality of Protection (QoP)

SASL メカニズムが整合性およびプライバシー保護 (QoP) 設定をサポートする場合は、**qop** 属性を使用して Hot Rod エンドポイント設定に追加できます。

QoP 設定	説明
auth	認証のみ。
auth-int	整合性保護による認証。
auth-conf	整合性とプライバシー保護による認証。

4.3.3. SASL ポリシー

SASL ポリシーは、Hot Rod 認証メカニズムを細かく制御できます。

ヒント

Data Grid のキャッシュ承認では、ロールおよびパーミッションに基づいてキャッシュへのアクセスを制限します。キャッシュ認証を設定し、**<no-anonymous value=false />** を設定して匿名ログインを許可し、アクセスロジックをキャッシュ承認に委譲します。

ポリシー	説明	デフォルト値
forward-secrecy	セッション間の forward secrecy をサポートする SASL メカニズムのみを使用します。これは、1 つのセッションに分割しても、将来のセッションに分割するための情報が自動的に提供されないことを意味します。	false
pass-credentials	クライアント認証情報が必要な SASL メカニズムのみを使用してください。	false
no-plain-text	単純な受動的攻撃の影響を受けやすい SASL メカニズムは使用しないでください。	false
no-active	アクティブな非辞書攻撃の影響を受けやすい SASL メカニズムは使用しないでください。	false
no-dictionary	受動的な辞書攻撃の影響を受けやすい SASL メカニズムは使用しないでください。	false
no-anonymous	匿名ログインを許可する SASL メカニズムは使用しないでください。	true

SASL ポリシーの設定

以下の設定では、Hot Rod エンドポイントは、すべての SASL ポリシーに準拠する唯一のメカニズムであるため、認証に **GSSAPI** メカニズムを使用します。

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <endpoints>
    <endpoint socket-binding="default"
              security-realm="default">
      <hotrod-connector>
        <authentication>
          <sasl mechanisms="PLAIN DIGEST-MD5 GSSAPI EXTERNAL"
                server-name="infinispan"
                qop="auth"
                policy="no-active no-plain-text"/>
        </authentication>
      </hotrod-connector>
      <rest-connector/>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "endpoints" : {
      "endpoint" : {
        "socket-binding" : "default",
        "security-realm" : "default",
        "hotrod-connector" : {
          "authentication" : {
            "sasl" : {
              "server-name" : "infinispan",
              "mechanisms" : [ "PLAIN","DIGEST-MD5","GSSAPI","EXTERNAL" ],
              "qop" : [ "auth" ],
              "policy" : [ "no-active","no-plain-text" ]
            }
          }
        },
        "rest-connector" : ""
      }
    }
  }
}
```

YAML

```
server:
  endpoints:
    endpoint:
      socketBinding: "default"
      securityRealm: "default"
```

```
    hotrodConnector:
      authentication:
        sasl:
          serverName: "infinispan"
          mechanisms:
            - "PLAIN"
            - "DIGEST-MD5"
            - "GSSAPI"
            - "EXTERNAL"
          qop:
            - "auth"
          policy:
            - "no-active"
            - "no-plain-text"
    restConnector: ~
```

4.3.4. HTTP 認証メカニズム

Data Grid Server は、REST エンドポイントに以下の HTTP 認証メカニズムをサポートします。

認証メカニズム	説明	セキュリティーレルムタイプ	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。	プロパティーレルムおよび LDAP レルム	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 コネクターは、**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。	プロパティーレルムおよび LDAP レルム	**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。	Kerberos レルム	**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。

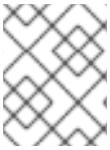
認証メカニズム	説明	セキュリティーレルムタイプ	関連する詳細
BEARER_TOKEN	OAuth トークンを使用し、**token-realm** 設定が必要です。	トークンレルム	**Bearer** HTTP 認証スキームに対応し、**OAUTHBEARER SASL** メカニズムに似ています。
CLIENT_CERT	クライアント証明書を使用します。	トラストストアレルム	**EXTERNAL** SASL メカニズムに似ています。

第5章 セキュリティーレルム

セキュリティーレルムは、ユーザー ID にアクセスおよび検証する環境内のネットワークプロトコルおよびインフラストラクチャーと Data Grid Server デプロイメントを統合します。

5.1. セキュリティーレルムの作成

セキュリティーレルムを Data Grid Server 設定に追加し、デプロイメントへのアクセスを制御します。設定に 1 つ以上のセキュリティーレルムを追加できます。



注記

設定にセキュリティーレルムを追加すると、Data Grid Server は Hot Rod および REST エンドポイントの一致する認証メカニズムを自動的に有効にします。

前提条件

- 必要に応じて、ソケットバインディングを Data Grid Server 設定に追加します。
- キーストアを作成するか、PEM ファイルがあり、TLS/SSL 暗号化でセキュリティーレルムを設定します。
  Data Grid Server は起動時にキーストアを生成することもできます。
- セキュリティーレルム設定に依存するリソースまたはサービスをプロビジョニングします。
  たとえば、トークンレルムを追加する場合は、OAuth サービスをプロビジョニングする必要があります。

この手順では、複数のプロパティーレルムを設定する方法を説明します。開始する前に、ユーザーを追加し、コマンドラインインターフェイス (CLI) でパーミッションを割り当てるプロパティーファイルを作成する必要があります。**user create** コマンドを使用します。

```
user create <username> -p <changeme> -g <role> \
    --users-file=application-users.properties \
    --groups-file=application-groups.properties

user create <username> -p <changeme> -g <role> \
    --users-file=management-users.properties \
    --groups-file=management-groups.properties
```

ヒント

サンプルおよび詳細情報について **user create --help** を実行します。

注記

CLI を使用してプロパティーレルムに認証情報を追加すると、接続しているサーバーインスタンスにのみユーザーが作成されます。プロパティーレルムの認証情報をクラスター内の各ノードに手動で同期する必要があります。

手順

1. Data Grid Server 設定を開いて編集します。

2. 複数のセキュリティーレルムの作成を含めるには、**security** 設定の **security-realms** 要素を使用します。

3. **security-realm** 要素でセキュリティーレルムを追加し、**name** 属性の一意の名前を付けます。

重要

ハイフン (**-**) やアンパサンド (**&**) などの特殊文字をセキュリティーレルム名に追加しないでください。セキュリティーレルム名に特殊文字が含まれていると、Data Grid Server エンドポイントに到達できなくなる可能性があります。

この例に従うには、**ApplicationRealm** という名前のセキュリティーレルムと、**ManagementRealm** という名前の 1 つのセキュリティーレルムを作成します。

4. Data Grid Server の TLS/SSL 識別に **server-identities** 要素を指定して、必要に応じてキーストアを設定します。

5. 以下の要素またはフィールド 1 つを追加して、セキュリティーレルムのタイプを指定します。

- **properties-realm**
- **ldap-realm**
- **token-realm**
- **truststore-realm**

6. 必要に応じて、設定するセキュリティーレルムタイプのプロパティーを指定します。
例に従うには、**user-properties** および **group-properties** 要素またはフィールドの **path** 属性を使用して、CLI で作成した ***.properties** ファイルを指定します。

7. 複数の異なるタイプのセキュリティーレルムを設定に追加する場合は、**distributed-realm** 要素またはフィールドを含めて、Data Grid Server がレルムを相互に組み合わせて使用できるようにします。

8. **security-realm** 属性でセキュリティーレルムを使用するように Data Grid Server エンドポイントを設定します。

9. 変更を設定に保存します。

複数のプロパティーレルム
次の設定は、XML、JSON、または YAML 形式で複数のセキュリティーレルムを設定する方法を示しています。

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="ApplicationRealm">
        <properties-realm groups-attribute="Roles">
          <user-properties path="application-users.properties"/>
          <group-properties path="application-groups.properties"/>
        </properties-realm>
      </security-realm>
      <security-realm name="ManagementRealm">
```

```
        <properties-realm groups-attribute="Roles">
          <user-properties path="management-users.properties"/>
          <group-properties path="management-groups.properties"/>
        </properties-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "ManagementRealm",
        "properties-realm": {
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "ManagementRealm",
            "path": "management-users.properties"
          },
          "group-properties": {
            "path": "management-groups.properties"
          }
        }
      }, {
        "name": "ApplicationRealm",
        "properties-realm": {
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "ApplicationRealm",
            "path": "application-users.properties"
          },
          "group-properties": {
            "path": "application-groups.properties"
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "ManagementRealm"
        propertiesRealm:
          groupsAttribute: "Roles"
          userProperties:
            digestRealmName: "ManagementRealm"
            path: "management-users.properties"
```

```
      groupProperties:
        path: "management-groups.properties"
    - name: "ApplicationRealm"
      propertiesRealm:
        groupsAttribute: "Roles"
        userProperties:
          digestRealmName: "ApplicationRealm"
          path: "application-users.properties"
        groupProperties:
          path: "application-groups.properties"
```

5.2. KERBEROS ID の設定

Data Grid Server 設定のセキュリティーレルムに Kerberos ID を追加して、Kerberos パスワードから派生するサービスプリンシパル名と暗号化されたキーが含まれる **keytab** ファイルを使用します。

前提条件

- Kerberos サービスアカウントプリンシパルがある。

注記

キータブ ファイルには、ユーザーとサービスのアカウントプリンシパルの両方を含めることができます。しかし、Data Grid Server はサービスアカウントプリンシパルのみを使用します。これは、クライアントに ID を提供し、クライアントが Kerberos サーバーで認証できることを意味します。

ほとんどの場合、Hot Rod および REST エンドポイントに固有のプリンシパルを作成します。たとえば、INFINISPAN.ORG ドメインに datagrid サーバーがある場合は、以下のサービスプリンシパルを作成する必要があります。

- **hotrod/datagrid@INFINISPAN.ORG** は Hot Rod サービスを特定します。
- **HTTP/datagrid@INFINISPAN.ORG** は REST サービスを識別します。

手順

1. Hot Rod および REST サービスのキータブファイルを作成します。

   Linux

   ```
   ktutil
   ktutil:  addent -password -p datagrid@INFINISPAN.ORG -k 1 -e aes256-cts
   Password for datagrid@INFINISPAN.ORG: [enter your password]
   ktutil:  wkt http.keytab
   ktutil:  quit
   ```

   Microsoft Windows

   ```
   ktpass -princ HTTP/datagrid@INFINISPAN.ORG -pass * -mapuser
   INFINISPAN\USER_NAME
   ktab -k http.keytab -a HTTP/datagrid@INFINISPAN.ORG
   ```

2. keytab ファイルを Data Grid Server インストールの **server/conf** ディレクトリーにコピーします。
3. Data Grid Server 設定を開いて編集します。
4. **server-identities** 定義を Data Grid サーバーのセキュリティーレルムに追加します。
5. Hot Rod および REST コネクターにサービスプリンシパルを提供するキータブファイルの場所を指定します。
6. Kerberos サービスプリンシパルに名前を付けます。
7. 変更を設定に保存します。

Kerberos ID の設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="kerberos-realm">
        <server-identities>
          <!-- Specifies a keytab file that provides a Kerberos identity. -->
          <!-- Names the Kerberos service principal for the Hot Rod endpoint. -->
          <!-- The required="true" attribute specifies that the keytab file must be present when the server starts. -->
          <kerberos keytab-path="hotrod.keytab"
                    principal="hotrod/datagrid@INFINISPAN.ORG"
                    required="true"/>
          <!-- Specifies a keytab file and names the Kerberos service principal for the REST endpoint. -->
          <kerberos keytab-path="http.keytab"
                    principal="HTTP/localhost@INFINISPAN.ORG"
                    required="true"/>
        </server-identities>
      </security-realm>
    </security-realms>
  </security>
  <endpoints>
    <endpoint socket-binding="default"
              security-realm="KerberosRealm">
      <hotrod-connector>
        <authentication>
          <sasl server-name="datagrid"
                server-principal="hotrod/datagrid@INFINISPAN.ORG"/>
        </authentication>
      </hotrod-connector>
      <rest-connector>
        <authentication server-principal="HTTP/localhost@INFINISPAN.ORG"/>
      </rest-connector>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "KerberosRealm",
        "server-identities": [{
          "kerberos": {
            "principal": "hotrod/datagrid@INFINISPAN.ORG",
            "keytab-path": "hotrod.keytab",
            "required": true
          },
          "kerberos": {
            "principal": "HTTP/localhost@INFINISPAN.ORG",
            "keytab-path": "http.keytab",
            "required": true
          }
        }]
      }]
    },
    "endpoints": {
      "endpoint": {
        "socket-binding": "default",
        "security-realm": "KerberosRealm",
        "hotrod-connector": {
          "authentication": {
            "security-realm": "kerberos-realm",
            "sasl": {
              "server-name": "datagrid",
              "server-principal": "hotrod/datagrid@INFINISPAN.ORG"
            }
          }
        },
        "rest-connector": {
          "authentication": {
            "server-principal": "HTTP/localhost@INFINISPAN.ORG"
          }
        }
      }
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "KerberosRealm"
        serverIdentities:
          - kerberos:
              principal: "hotrod/datagrid@INFINISPAN.ORG"
              keytabPath: "hotrod.keytab"
              required: "true"
          - kerberos:
              principal: "HTTP/localhost@INFINISPAN.ORG"
```

```
            keytabPath: "http.keytab"
            required: "true"
  endpoints:
    endpoint:
      socketBinding: "default"
      securityRealm: "KerberosRealm"
      hotrodConnector:
        authentication:
          sasl:
            serverName: "datagrid"
            serverPrincipal: "hotrod/datagrid@INFINISPAN.ORG"
      restConnector:
        authentication:
          securityRealm: "KerberosRealm"
          serverPrincipal" : "HTTP/localhost@INFINISPAN.ORG"
```

5.3. プロパティーレルム

プロパティーレルムはプロパティーファイルを使用して、ユーザーおよびグループを定義します。

- **users.properties** には Data Grid ユーザーの認証情報が含まれます。パスワードは、 **DIGEST-MD5** および **DIGEST** 認証メカニズムを使用して事前署名できます。
- **groups.properties** は、ユーザーをロールおよびパーミッションに関連付けます。

注記

プロパティーファイルには、Data Grid Server 設定のセキュリティーレルムに関連付けるヘッダーが含まれます。

users.properties

```
myuser=a_password
user2=another_password
```

groups.properties

```
myuser=supervisor,reader,writer
user2=supervisor
```

プロパティーレルム設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <!-- groups-attribute configures the "groups.properties" file to contain security authorization roles.
-->
        <properties-realm groups-attribute="Roles">
          <user-properties path="users.properties"
                           relative-to="infinispan.server.config.path"
```

```
                    plain-text="true"/>
        <group-properties path="groups.properties"
                          relative-to="infinispan.server.config.path"/>
      </properties-realm>
    </security-realm>
  </security-realms>
 </security>
</server>
```

JSON

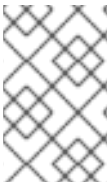
```
{
 "server": {
  "security": {
   "security-realms": [{
    "name": "default",
    "properties-realm": {
     "groups-attribute": "Roles",
     "user-properties": {
      "digest-realm-name": "default",
      "path": "users.properties",
      "relative-to": "infinispan.server.config.path",
      "plain-text": true
     },
     "group-properties": {
      "path": "groups.properties",
      "relative-to": "infinispan.server.config.path"
     }
    }
   }]
  }
 }
}
```

YAML

```
server:
 security:
  securityRealms:
   - name: "default"
     propertiesRealm:
       # groupsAttribute configures the "groups.properties" file
       # to contain security authorization roles.
       groupsAttribute: "Roles"
       userProperties:
         digestRealmName: "default"
         path: "users.properties"
         relative-to: 'infinispan.server.config.path'
         plainText: "true"
       groupProperties:
         path: "groups.properties"
         relative-to: 'infinispan.server.config.path'
```

5.4. LDAP レルム

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。



注記

LDAP サーバーは、サーバーのタイプとデプロイメントに応じて、異なるエントリーレイアウトを持つことができます。考えられるすべての設定の例を提供することは、本書では扱っていません。

重要

LDAP 接続のプリンシパルには、LDAP クエリーを実行し、特定の属性にアクセスするために必要な権限が必要です。

direct-verification 属性を使用してユーザー認証情報を検証する代わりに、**user-password-mapper** 要素を使用してパスワードを検証する LDAP 属性を指定できます。

注記

direct-verification 属性でハッシュ化を実行するエンドポイントの認証メカニズムは使用できません。

Active Directory は **password** 属性を公開しないため、**user-password-mapper** 要素は使用できず、**direct-verification** 属性のみを使用できます。そのため、Active Directory Server と統合するには、REST エンドポイントでは **BASIC** 認証メカニズムを、Hot Rod エンドポイントでは **PLAIN** を使用する必要があります。より安全な代替方法として、**SPNEGO**、**GSSAPI**、および **GS2-KRB5** 認証メカニズムを可能にする Kerberos を使用することができます。

rdn-identifier 属性は、指定された識別子 (通常はユーザー名) をもとにユーザーエントリーを検索する LDAP 属性を指定します (例: **uid** または **sAMAccountName** 属性)。**search-recursive="true"** を設定に追加して、ディレクトリーを再帰的に検索します。デフォルトでは、ユーザーエントリーの検索は **(rdn_identifier={0})** フィルターを使用します。**filter-name** 属性を使用して別のフィルターを指定します。

attribute-mapping 要素は、ユーザーがメンバーであるすべてのグループを取得します。通常、メンバーシップ情報を保存する方法は 2 つあります。

- 通常、**member** 属性にクラス **groupOfNames** を持つグループエントリーの下。この場合は、前述の設定例にあるように、属性フィルターを使用できます。このフィルターは、提供されたフィルターに一致するエントリーを検索します。フィルターは、ユーザーの DN と等しい **member** 属性を持つグループを検索します。次に、フィルターは、**from** で指定されたグループエントリーの CN を抽出し、それをユーザーの **Roles** に追加します。

- **memberOf** 属性のユーザーエントリー。この場合、以下のような属性参照を使用する必要があります。
  <attribute-reference reference="memberOf" from="cn" to="Roles" />

  この参照は、ユーザーエントリーからすべての **memberOf** 属性を取得し、**from** で指定された CN を抽出し、それらをユーザーの **Roles** に追加します。

LDAP レルム設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="LdapRealm">
        <!-- Specifies connection properties. -->
        <ldap-realm url="ldap://my-ldap-server:10389"
                    principal="uid=admin,ou=People,dc=infinispan,dc=org"
                    credential="strongPassword"
                    connection-timeout="3000"
                    read-timeout="30000"
                    connection-pooling="true"
                    referral-mode="ignore"
                    page-size="30"
                    direct-verification="true">
          <!-- Defines how principals are mapped to LDAP entries. -->
          <identity-mapping rdn-identifier="uid"
                            search-dn="ou=People,dc=infinispan,dc=org"
                            search-recursive="false">
            <!-- Retrieves all the groups of which the user is a member. -->
            <attribute-mapping>
              <attribute from="cn" to="Roles"
                         filter="(&amp;(objectClass=groupOfNames)(member={1}))"
                         filter-dn="ou=Roles,dc=infinispan,dc=org"/>
            </attribute-mapping>
          </identity-mapping>
        </ldap-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "LdapRealm",
        "ldap-realm": {
          "url": "ldap://my-ldap-server:10389",
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "credential": "strongPassword",
          "connection-timeout": "3000",
          "read-timeout": "30000",
          "connection-pooling": "true",
          "referral-mode": "ignore",
          "page-size": "30",
          "direct-verification": "true",
          "identity-mapping": {
            "rdn-identifier": "uid",
            "search-dn": "ou=People,dc=infinispan,dc=org",
            "search-recursive": "false",
            "attribute-mapping": [{
```

```
            "from": "cn",
            "to": "Roles",
            "filter": "(&(objectClass=groupOfNames)(member={1}))",
            "filter-dn": "ou=Roles,dc=infinispan,dc=org"
          }]
        }
      }
    }]
  }
 }
}
```

YAML

```
server:
 security:
  securityRealms:
   - name: LdapRealm
     ldapRealm:
       url: 'ldap://my-ldap-server:10389'
       principal: 'uid=admin,ou=People,dc=infinispan,dc=org'
       credential: strongPassword
       connectionTimeout: '3000'
       readTimeout: '30000'
       connectionPooling: true
       referralMode: ignore
       pageSize: '30'
       directVerification: true
       identityMapping:
         rdnIdentifier: uid
         searchDn: 'ou=People,dc=infinispan,dc=org'
         searchRecursive: false
         attributeMapping:
           - filter: '(&(objectClass=groupOfNames)(member={1}))'
             filterDn: 'ou=Roles,dc=infinispan,dc=org'
             from: cn
             to: Roles
```

5.4.1. LDAP レルムプリンシパルの書き換え

GSSAPI、**GS2-KRB5**、**Negotiate** などの SASL 認証メカニズムには、LDAP ディレクトリーの検索に使用する前に クリーンアップ する必要のあるユーザー名が含まれています。

XML

```
<server xmlns="urn:infinispan:server:13.0">
 <security>
  <security-realms>
   <security-realm name="LdapRealm">
    <ldap-realm url="ldap://${org.infinispan.test.host.address}:10389"
                principal="uid=admin,ou=People,dc=infinispan,dc=org"
                credential="strongPassword">
     <name-rewriter>
      <!-- Defines a rewriter that extracts the username from the principal using a regular
```

```
expression. -->
          <regex-principal-transformer name="domain-remover"
                                       pattern="(.*)@INFINISPAN\.ORG"
                                       replacement="$1"/>
        </name-rewriter>
        <identity-mapping rdn-identifier="uid"
                          search-dn="ou=People,dc=infinispan,dc=org">
          <attribute-mapping>
            <attribute from="cn" to="Roles"
                       filter="(&amp;(objectClass=groupOfNames)(member={1}))"
                       filter-dn="ou=Roles,dc=infinispan,dc=org"/>
          </attribute-mapping>
          <user-password-mapper from="userPassword"/>
        </identity-mapping>
      </ldap-realm>
    </security-realm>
  </security-realms>
 </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "LdapRealm",
        "ldap-realm": {
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "url": "ldap://${org.infinispan.test.host.address}:10389",
          "credential": "strongPassword",
          "name-rewriter": {
            "regex-principal-transformer": {
              "pattern": "(.*)@INFINISPAN\\.ORG",
              "replacement": "$1"
            }
          },
          "identity-mapping": {
            "rdn-identifier": "uid",
            "search-dn": "ou=People,dc=infinispan,dc=org",
            "attribute-mapping": {
              "attribute": {
                "filter": "(&(objectClass=groupOfNames)(member={1}))",
                "filter-dn": "ou=Roles,dc=infinispan,dc=org",
                "from": "cn",
                "to": "Roles"
              }
            },
            "user-password-mapper": {
              "from": "userPassword"
            }
          }
        }
      }]
```

```
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "LdapRealm"
        ldapRealm:
          principal: "uid=admin,ou=People,dc=infinispan,dc=org"
          url: "ldap://${org.infinispan.test.host.address}:10389"
          credential: "strongPassword"
          nameRewriter:
            regexPrincipalTransformer:
              pattern: (.*)@INFINISPAN\.ORG
              replacement: "$1"
          identityMapping:
            rdnIdentifier: "uid"
            searchDn: "ou=People,dc=infinispan,dc=org"
            attributeMapping:
              attribute:
                filter: "(&(objectClass=groupOfNames)(member={1}))"
                filterDn: "ou=Roles,dc=infinispan,dc=org"
                from: "cn"
                to: "Roles"
            userPasswordMapper:
              from: "userPassword"
```

5.5. トークンレルム

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。

トークンレルムの設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="TokenRealm">
        <!-- Specifies the URL of the authentication server. -->
        <token-realm name="token"
                     auth-server-url="https://oauth-server/auth/">
          <!-- Specifies the URL of the token introspection endpoint. -->
          <oauth2-introspection introspection-url="https://oauth-
server/auth/realms/infinispan/protocol/openid-connect/token/introspect"
                                client-id="infinispan-server"
                                client-secret="1fdca4ec-c416-47e0-867a-3d471af7050f"/>
        </token-realm>
      </security-realm>
```

```
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "TokenRealm",
        "token-realm": {
          "auth-server-url": "https://oauth-server/auth/",
          "oauth2-introspection": {
            "client-id": "infinispan-server",
            "client-secret": "1fdca4ec-c416-47e0-867a-3d471af7050f",
            "introspection-url": "https://oauth-server/auth/realms/infinispan/protocol/openid-
connect/token/introspect"
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "TokenRealm"
        tokenRealm:
          authServerUrl: 'https://oauth-server/auth/'
          oauth2Introspection:
            clientId: infinispan-server
            clientSecret: '1fdca4ec-c416-47e0-867a-3d471af7050f'
            introspectionUrl: 'https://oauth-server/auth/realms/infinispan/protocol/openid-
connect/token/introspect'
```

5.6. トラストストアレルム

トラストストアレルムは、接続のネゴシエート時に Data Grid Server およびクライアント ID を検証する証明書または証明書チェーンを使用します。

キーストア

Data Grid Server アイデンティティーをクライアントに提供するサーバー証明書が含まれます。サーバー証明書でキーストアを設定する場合、Data Grid Server は業界標準の SSL/TLS プロトコルを使用してトラフィックを暗号化します。

トラストストア

クライアントが Data Grid Server に提示するクライアント証明書または証明書チェーンが含まれます。クライアントのトラストストアはオプションで、Data Grid Server がクライアント証明書認証を実行できるようになっています。

クライアント証明書認証

Data Grid Server でクライアント証明書を検証または認証する場合は、**require-ssl-client-auth="true"** 属性をエンドポイント設定に追加する必要があります。

トラストストアレルムの設定

XML

```xml
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="TrustStoreRealm">
        <server-identities>
          <ssl>
            <!-- Provides an SSL/TLS identity with a keystore that contains server certificates. -->
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      keystore-password="secret"
                      alias="server"/>
            <!-- Configures a trust store that contains client certificates or part of a certificate chain. -->
            <truststore path="trust.p12"
                        relative-to="infinispan.server.config.path"
                        password="secret"/>
          </ssl>
        </server-identities>
        <!-- Authenticates client certificates against the trust store. If you configure this, the trust store must contain the public certificates for all clients. -->
        <truststore-realm/>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```json
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "TrustStoreRealm",
        "server-identities": {
          "ssl": {
            "keystore": {
              "path": "server.p12",
              "relative-to": "infinispan.server.config.path",
              "keystore-password": "secret",
              "alias": "server"
            },
            "truststore": {
              "path": "trust.p12",
              "relative-to": "infinispan.server.config.path",
              "password": "secret"
            }
          }
```

```
        },
        "truststore-realm": {}
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "TrustStoreRealm"
        serverIdentities:
          ssl:
            keystore:
              path: "server.p12"
              relative-to: "infinispan.server.config.path"
              keystore-password: "secret"
              alias: "server"
            truststore:
              path: "trust.p12"
              relative-to: "infinispan.server.config.path"
              password: "secret"
        truststoreRealm: ~
```

5.7. 分散セキュリティーレルム

分散レルムは、複数のタイプのセキュリティーレルムを組み合わせます。ユーザーが Hot Rod または REST エンドポイントにアクセスしようとすると、認証を実行できるものを見つけるまで、Data Grid Server は各セキュリティーレルムを順番に使用します。

分散レルムの設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="DistributedRealm">
        <ldap-realm url="ldap://my-ldap-server:10389"
                    principal="uid=admin,ou=People,dc=infinispan,dc=org"
                    credential="strongPassword">
          <identity-mapping rdn-identifier="uid"
                            search-dn="ou=People,dc=infinispan,dc=org"
                            search-recursive="false">
            <attribute-mapping>
              <attribute from="cn" to="Roles"
                         filter="(&amp;(objectClass=groupOfNames)(member={1}))"
                         filter-dn="ou=Roles,dc=infinispan,dc=org"/>
            </attribute-mapping>
          </identity-mapping>
        </ldap-realm>
```

```
        <properties-realm groups-attribute="Roles">
          <user-properties path="users.properties"
                           relative-to="infinispan.server.config.path"/>
          <group-properties path="groups.properties"
                            relative-to="infinispan.server.config.path"/>
        </properties-realm>
        <distributed-realm/>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "DistributedRealm",
        "ldap-realm": {
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "url": "ldap://my-ldap-server:10389",
          "credential": "strongPassword",
          "identity-mapping": {
            "rdn-identifier": "uid",
            "search-dn": "ou=People,dc=infinispan,dc=org",
            "search-recursive": false,
            "attribute-mapping": {
              "attribute": {
                "filter": "(&(objectClass=groupOfNames)(member={1}))",
                "filter-dn": "ou=Roles,dc=infinispan,dc=org",
                "from": "cn",
                "to": "Roles"
              }
            }
          }
        },
        "properties-realm": {
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "DistributedRealm",
            "path": "users.properties"
          },
          "group-properties": {
            "path": "groups.properties"
          }
        },
        "distributed-realm": {}
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "DistributedRealm"
        ldapRealm:
          principal: "uid=admin,ou=People,dc=infinispan,dc=org"
          url: "ldap://my-ldap-server:10389"
          credential: "strongPassword"
          identityMapping:
            rdnIdentifier: "uid"
            searchDn: "ou=People,dc=infinispan,dc=org"
            searchRecursive: "false"
            attributeMapping:
              attribute:
                filter: "(&(objectClass=groupOfNames)(member={1}))"
                filterDn: "ou=Roles,dc=infinispan,dc=org"
                from: "cn"
                to: "Roles"
        propertiesRealm:
          groupsAttribute: "Roles"
          userProperties:
            digestRealmName: "DistributedRealm"
            path: "users.properties"
          groupProperties:
            path: "groups.properties"
        distributedRealm: ~
```

第6章 TLS/SSL 暗号化の設定

Data Grid の公開鍵と秘密鍵が含まれるキーストアを設定することにより、SSL/TLS 暗号化を使用して Data Grid Server の接続をセキュアにすることができます。相互 TLS が必要な場合、クライアント証明書認証を設定することもできます。

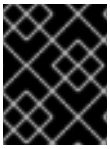
6.1. DATA GRID SERVER キーストアの設定

キーストアを Data Grid Server に追加し、その ID をクライアントに対して検証する SSL/TLS 証明書を提示します。セキュリティーレルムに TLS/SSL アイデンティティーが含まれる場合は、そのセキュリティーレルムを使用する Data Grid Server エンドポイントへの接続を暗号化します。

前提条件

- Data Grid Server の証明書または証明書チェーンが含まれるキーストアを作成します。

Data Grid Server は、JKS、JCEKS、PKCS12/PFX、および PEM のキーストア形式をサポートします。Bouncy Castle ライブラリーが存在する場合は、BKS、BCFKS、および UBER もサポートされます。



重要

実稼働環境では、サーバー証明書は Root または Intermediate CA のいずれかの信頼される認証局によって署名される必要があります。

ヒント

以下のいずれかが含まれる場合には、PEM ファイルをキーストアとして使用できます。

- PKCS#1 または PKCS#8 形式の秘密鍵。
- 1 つ以上の証明書。

PEM ファイルキーストアを空のパスワード (**password=""**) で設定する必要があります。

手順

1. Data Grid Server 設定を開いて編集します。
2. Data Grid Server の SSL/TLS アイデンティティーが含まれるキーストアを **$RHDG_HOME/server/conf** ディレクトリーに追加します。
3. **server-identities** 定義を Data Grid Server セキュリティーレルムに追加します。
4. **path** 属性でキーストアファイル名を指定します。
5. キーストアパスワードと証明書エイリアスに **keystore-password** および **alias** 属性を指定します。
6. 変更を設定に保存します。

次のステップ

クライアントが Data Grid Server の SSL/TLS ID を確認できるように、トラストストアを使用してクライアントを設定します。

キーストアの設定

XML

```xml
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <server-identities>
          <ssl>
            <!-- Adds a keystore that contains server certificates that provide SSL/TLS identities to clients.
-->
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      password="secret"
                      alias="my-server"/>
          </ssl>
        </server-identities>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "server-identities": {
          "ssl": {
            "keystore": {
              "alias": "my-server",
              "path": "server.p12",
              "password": "secret"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        serverIdentities:
          ssl:
            keystore:
```

```
        alias: "my-server"
        path: "server.p12"
        password: "secret"
```

関連情報

- Hot Rod クライアントの暗号化の設定

6.1.1. Data Grid Server キーストアの生成

起動時にキーストアを自動的に生成するように Data Grid Server を設定します。

重要

自動生成されたキーストア:

- 実稼働環境では使用しないでください。
- 必要に応じて生成されます。たとえば、クライアントから最初の接続を取得する際などに生成されます。
- Hot Rod クライアントで直接使用可能な証明書が含まれます。

手順

1. Data Grid Server 設定を開いて編集します。
2. サーバー設定に **keystore** 要素の **generate-self-signed-certificate-host** 属性を含めます。
3. サーバー証明書のホスト名を値として指定します。
4. 変更を設定に保存します。

生成されたキーストアの設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="GeneratedKeystore">
        <server-identities>
          <ssl>
            <!-- Generates a keystore that includes a self-signed certificate with the specified hostname. --
>
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      password="secret"
                      alias="server"
                      generate-self-signed-certificate-host="localhost"/>
          </ssl>
        </server-identities>
      </security-realm>
```

```
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "GeneratedKeystore",
        "server-identities": {
          "ssl": {
            "keystore": {
              "alias": "server",
              "generate-self-signed-certificate-host": "localhost",
              "path": "server.p12",
              "password": "secret"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "GeneratedKeystore"
        serverIdentities:
          ssl:
            keystore:
              alias: "server"
              generateSelfSignedCertificateHost: "localhost"
              path: "server.p12"
              password: "secret"
```

6.1.2. TLS バージョンおよび暗号スイートの設定

SSL/TLS 暗号化を使用してデプロイメントのセキュリティーを保護する場合は、特定のバージョンの TLS プロトコルと、プロトコル内の特定の暗号スイートを使用するように Data Grid Server を設定できます。

手順

1. Data Grid Server 設定を開いて編集します。
2. **engine** 要素を Data Grid Server の SSL 設定に追加します。
3. **enabled-protocols** 属性を持つ 1 つ以上の TLS バージョンを使用するように Data Grid を設定します。

Data Grid Server は、デフォルトで TLS バージョン 1.2 および 1.3 をサポートします。該当する場合は、クライアント接続のセキュリティープロトコルを制限するために、**TLSv1.3** のみを設定できます。Data Grid は、**TLSv1.1** の有効化を推奨していません。これは、サポートが制限された古いプロトコルで、セキュリティー保護が弱いからです。1.1 より古いバージョンの TLS を有効にすることはできません。

警告

Data Grid Server の SSL **engine** 設定を変更する場合は、**enabled-protocols** 属性を使用して TLS バージョンを明示的に設定する必要があります。**enabled-protocols** 属性を省略すると、すべての TLS バージョンが許可されます。

```
<engine enabled-protocols="TLSv1.3 TLSv1.2" />
```

4. **enabled-ciphersuites** 属性 (TLSv1.2 以下) および **enabled-ciphersuites-tls13** 属性 (TLSv1.3) を使用して、1 つまたは複数の暗号スイートを使用するように Data Grid を設定します。
使用する予定のプロトコル機能 (例: **HTTP/2 ALPN**) をサポートする暗号スイートを設定していることを確認する必要があります。

5. 変更を設定に保存します。

SSL エンジンの設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <server-identities>
          <ssl>
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      password="secret"
                      alias="server"/>
            <!-- Configures Data Grid Server to use specific TLS versions and cipher suites. -->
            <engine enabled-protocols="TLSv1.3 TLSv1.2"
                    enabled-ciphersuites="TLS_AES_256_GCM_SHA384,TLS_AES_128_GCM_SHA256"
                    enabled-ciphersuites-tls13="TLS_AES_256_GCM_SHA384"/>
          </ssl>
        </server-identities>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "server-identities": {
          "ssl": {
            "keystore": {
              "alias": "server",
              "path": "server.p12",
              "password": "secret"
            },
            "engine": {
              "enabled-protocols": ["TLSv1.3"],
              "enabled-ciphersuites": "TLS_AES_256_GCM_SHA384,TLS_AES_128_GCM_SHA256",
              "enabled-ciphersuites-tls13": "TLS_AES_256_GCM_SHA384"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        serverIdentities:
          ssl:
            keystore:
              alias: "server"
              path: "server.p12"
              password: "secret"
            engine:
              enabledProtocols:
                - "TLSv1.3"
              enabledCiphersuites: "TLS_AES_256_GCM_SHA384,TLS_AES_128_GCM_SHA256"
              enabledCiphersuitesTls13: "TLS_AES_256_GCM_SHA384"
```

6.2. FIPS 140-2 準拠の暗号を使用するシステムでの DATA GRID SERVER の設定

FIPS (Federal Information Processing Standards) とは、米国連邦政府のコンピューターシステムの標準およびガイドラインです。FIPS は米国連邦政府が使用するために開発されたものですが、民間部門の多くは自発的にこれらの標準を使用しています。

FIPS 140-2 は、暗号モジュールに対するセキュリティー要件を定義しています。代替の JDK セキュリティープロバイダーを使用することで、FIPS 140-2 仕様に準拠する暗号化方式を使用するように Data Grid Server を設定することができます。

関連情報

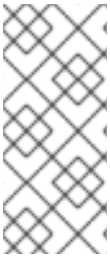
- Java PKCS#11 暗号化プロバイダー
- The Legion of the Bouncy Castle cryptographic provider

6.2.1. PKCS11 暗号プロバイダーの設定

SunPKCS11-NSS-FIPS プロバイダーで PKCS11 キーストアを指定すると、PKCS11 暗号化プロバイダーを設定できます。

前提条件

- FIPS モード用にシステムを設定する。システムが FIPS モードを有効にしているかどうかは、Data Grid のコマンドラインインターフェイス (CLI) で **fips-mode-setup --check** コマンドを発行することで確認できます。
- **certutil** ツールを使用して、システム全体の NSS データベースを初期化します。
- **SunPKCS11** プロバイダーを有効にするように **java.security** ファイルを設定した JDK をインストールします。このプロバイダーは、NSS データベースと SSL プロバイダーを指します。
- NSS データベースに証明書をインストールします。



注記

OpenSSL プロバイダーは秘密鍵を必要としますが、PKCS#11 ストアから秘密鍵を取得することはできません。FIPS では、FIPS 準拠の暗号モジュールから暗号化されていない鍵のエクスポートをブロックしているため、FIPS モードでは TLS 用の OpenSSL プロバイダーを使用することはできません。起動時に **-Dorg.infinispan.openssl=false** 引数で OpenSSL プロバイダーを無効にすることができます。

手順

1. Data Grid Server 設定を開いて編集します。
2. **server-identities** 定義を Data Grid Server セキュリティーレルムに追加します。
3. **SunPKCS11-NSS-FIPS** プロバイダーで PKCS11 キーストアを指定します。
4. 変更を設定に保存します。

キーストアの設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <server-identities>
          <ssl>
            <!-- Adds a keystore that reads certificates from the NSS database. -->
            <keystore provider="SunPKCS11-NSS-FIPS" type="PKCS11"/>
          </ssl>
        </server-identities>
      </security-realm>
```

```
      </security-realms>
    </security>
  </server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "server-identities": {
          "ssl": {
            "keystore": {
              "provider": "SunPKCS11-NSS-FIPS",
              "type": "PKCS11"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        serverIdentities:
          ssl:
            keystore:
              provider: "SunPKCS11-NSS-FIPS"
              type: "PKCS11"
```

6.2.2. Bouncy Castle FIPS 暗号プロバイダーの設定

Bouncy Castle FIPS (Federal Information Processing Standards) 暗号化プロバイダーは、Data Grid サーバーの設定で設定することができます。

前提条件

- FIPS モード用にシステムを設定する。システムが FIPS モードを有効にしているかどうかは、Data Grid のコマンドラインインターフェイス (CLI) で **fips-mode-setup --check** コマンドを発行することで確認できます。
- 証明書を含む BCFKS 形式のキーストアを作成します。

手順

1. Bouncy Castle FIPS JAR ファイルをダウンロードし、Data Grid Server のインストール先の **server/lib** ディレクトリーにファイルを追加してください。

2. Bouncy Castle をインストールするには、**install** コマンドを実行します。

```
[disconnected]> install org.bouncycastle:bc-fips:1.0.2.3
```

3. Data Grid Server 設定を開いて編集します。
4. **server-identities** 定義を Data Grid Server セキュリティーレルムに追加します。
5. **BCFIPS** プロバイダーで BCFKS キーストアを指定します。
6. 変更を設定に保存します。

キーストアの設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="default">
        <server-identities>
          <ssl>
            <!-- Adds a keystore that reads certificates from the BCFKS keystore. -->
            <keystore path="server.bcfks" password="secret" alias="server" provider="BCFIPS"
type="BCFKS"/>
          </ssl>
        </server-identities>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "default",
        "server-identities": {
          "ssl": {
            "keystore": {
              "path": "server.bcfks",
              "password": "secret",
              "alias": "server",
              "provider": "BCFIPS",
              "type": "BCFKS"
            }
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "default"
        serverIdentities:
          ssl:
            keystore:
              path: "server.bcfks"
              password: "secret"
              alias: "server"
              provider: "BCFIPS"
              type: "BCFKS"
```

6.3. クライアント証明書認証の設定

Data Grid Server が相互 TLS を使用してクライアント接続のセキュリティーを保護するように設定します。

トラストストアの証明書からクライアント ID を検証するように Data Grid を設定するには、以下の 2 つの方法があります。

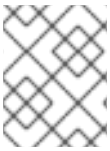
- 通常は認証局 (CA) である署名証明書のみが含まれるトラストストアが必要です。CA によって署名された証明書を提示するクライアントは、Data Grid に接続できます。
- 署名証明書に加えて、すべてのクライアント証明書が含まれるトラストストアが必要です。トラストストアに存在する署名済み証明書を提示するクライアントのみが Data Grid に接続できます。

ヒント

トラストストアを提供する代わりに、共有システム証明書を使用できます。

前提条件

- CA 証明書またはすべての公開証明書のいずれかを含むクライアントトラストストアを作成します。
- Data Grid Server のキーストアを作成し、SSL/TLS アイデンティティーを設定します。



注記

PEM ファイルは、1 つ以上の証明書が含まれるトラストストアとして使用できます。これらのトラクトストアは、空のパスワード **password=""** で設定する必要があります。

手順

1. Data Grid Server 設定を開いて編集します。
2. **require-ssl-client-auth="true"** パラメーターを **endpoints** 設定に追加します。
3. クライアントトラストストアを **$RHDG_HOME/server/conf** ディレクトリーに追加します。

4. Data Grid Server セキュリティーレルム設定で、**truststore** 要素の **path** および **password** 属性を指定します。
5. Data Grid Server で各クライアント証明書を認証する場合は、**<truststore-realm/>** 要素をセキュリティーレルムに追加します。
6. 変更を設定に保存します。

次のステップ

- セキュリティーロールおよびパーミッションでアクセスを制御する場合は、Data Grid Server 設定で、クライアント証明書を使用して承認を設定します。
- クライアントを設定し、Data Grid Server と SSL/TLS 接続をネゴシエートします。

クライアント証明書認証設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="TrustStoreRealm">
        <server-identities>
          <ssl>
            <!-- Provides an SSL/TLS identity with a keystore that
                 contains server certificates. -->
            <keystore path="server.p12"
                      relative-to="infinispan.server.config.path"
                      keystore-password="secret"
                      alias="server"/>
            <!-- Configures a trust store that contains client certificates
                 or part of a certificate chain. -->
            <truststore path="trust.p12"
                        relative-to="infinispan.server.config.path"
                        password="secret"/>
          </ssl>
        </server-identities>
        <!-- Authenticates client certificates against the trust store. If you configure this, the trust store
must contain the public certificates for all clients. -->
        <truststore-realm/>
      </security-realm>
    </security-realms>
  </security>
  <endpoints>
    <endpoint socket-binding="default"
              security-realm="trust-store-realm"
              require-ssl-client-auth="true">
      <hotrod-connector>
        <authentication>
          <sasl mechanisms="EXTERNAL"
                server-name="infinispan"
                qop="auth"/>
        </authentication>
      </hotrod-connector>
```

```
      <rest-connector>
        <authentication mechanisms="CLIENT_CERT"/>
      </rest-connector>
    </endpoint>
  </endpoints>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "TrustStoreRealm",
        "server-identities": {
          "ssl": {
            "keystore": {
              "path": "server.p12",
              "relative-to": "infinispan.server.config.path",
              "keystore-password": "secret",
              "alias": "server"
            },
            "truststore": {
              "path": "trust.p12",
              "relative-to": "infinispan.server.config.path",
              "password": "secret"
            }
          }
        },
        "truststore-realm": {}
      }]
    },
    "endpoints": [{
      "socket-binding": "default",
      "security-realm": "TrustStoreRealm",
      "require-ssl-client-auth": "true",
      "connectors": {
        "hotrod": {
          "hotrod-connector": {
            "authentication": {
              "sasl": {
                "mechanisms": "EXTERNAL",
                "server-name": "infinispan",
                "qop": "auth"
              }
            }
          },
          "rest": {
            "rest-connector": {
              "authentication": {
                "mechanisms": "CLIENT_CERT"
              }
            }
          }
        }
```

```
      }
    }]
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "TrustStoreRealm"
        serverIdentities:
          ssl:
            keystore:
              path: "server.p12"
              relative-to: "infinispan.server.config.path"
              keystore-password: "secret"
              alias: "server"
            truststore:
              path: "trust.p12"
              relative-to: "infinispan.server.config.path"
              password: "secret"
        truststoreRealm: ~
  endpoints:
    socketBinding: "default"
    securityRealm: "trust-store-realm"
    requireSslClientAuth: "true"
    connectors:
      - hotrod:
          hotrodConnector:
            authentication:
              sasl:
                mechanisms: "EXTERNAL"
                serverName: "infinispan"
                qop: "auth"
      - rest:
          restConnector:
            authentication:
              mechanisms: "CLIENT_CERT"
```

関連情報

- Hot Rod クライアントの暗号化の設定
- Using Shared System Certificates (Red Hat Enterprise Linux 7 Security Guide)

6.4. クライアント証明書を使用した承認の設定

クライアント証明書認証を有効にすると、クライアント設定で Data Grid ユーザー認証情報を指定する必要がなくなります。つまり、ロールをクライアント証明書の Common Name (CN) フィールドに関連付ける必要があります。

前提条件

- クライアントに、公開証明書または証明書チェーンの一部 (通常は公開 CA 証明書) のいずれかが含まれる Java キーストアを提供します。
- クライアント証明書認証を実行するように Data Grid Server を設定します。

手順

1. Data Grid Server 設定を開いて編集します。
2. セキュリティー承認設定で **common-name-role-mapper** を有効にします。
3. クライアント証明書から Common Name (**CN**) に、適切な権限を持つロールを割り当てます。
4. 変更を設定に保存します。

クライアント証明書承認設定

XML

```
<infinispan>
  <cache-container name="certificate-authentication" statistics="true">
    <security>
      <authorization>
        <!-- Declare a role mapper that associates the common name (CN) field in client certificate trust stores with authorization roles. -->
        <common-name-role-mapper/>
        <!-- In this example, if a client certificate contains `CN=Client1` then clients with matching certificates get ALL permissions. -->
        <role name="Client1" permissions="ALL"/>
      </authorization>
    </security>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan": {
    "cache-container": {
      "name": "certificate-authentication",
      "security": {
        "authorization": {
          "common-name-role-mapper": null,
          "roles": {
            "Client1": {
              "role": {
                "permissions": "ALL"
              }
            }
          }
        }
      }
```

```
    }
  }
}
```

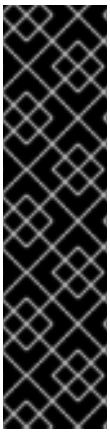
YAML

```
infinispan:
  cacheContainer:
    name: "certificate-authentication"
    security:
      authorization:
        commonNameRoleMapper: ~
        roles:
          Client1:
            role:
              permissions:
                - "ALL"
```

第7章 キーストアへの DATA GRID SERVER 認証情報の保存

外部サービスには、Data Grid Server での認証に認証情報が必要です。パスワードなどの機密なテキスト文字列を保護するには、これらを Data Grid Server 設定ファイルに直接追加するのではなく、認証情報キーストアに追加します。

次に、データベースや LDAP ディレクトリーなどのサービスと接続を確立するためのパスワードを復号化するように、Data Grid Server を設定することができます。



重要

$RHDG_HOME/server/conf のプレーンテキストのパスワードは暗号化されません。ホストファイルシステムへの読み取りアクセス権を持つすべてのユーザーアカウントは、プレーンテキストのパスワードを表示できます。

認証情報キーストアはパスワードで保護されたストア暗号化パスワードですが、ホストファイルシステムへの書き込みアクセス権を持つユーザーアカウントは、キーストア自体を改ざんすることが可能です。

Data Grid Server の認証情報を完全に保護するには、Data Grid Server を設定および実行できるユーザーアカウントにのみ読み書きアクセスを付与する必要があります。

7.1. 認証情報キーストアのセットアップ

Data Grid Server アクセスの認証情報を暗号化するキーストアを作成します。

認証情報キーストアには、暗号化されたパスワードに関連するエイリアスが少なくとも 1 つ含まれます。キーストアの作成後に、データベース接続プールなどの接続設定にエイリアスを指定します。その後、Data Grid Server は、サービスが認証を試行するときに、キーストアからそのエイリアスのパスワードを復号化します。

必要な数のエイリアスを使用して、必要な数の認証情報キーストアを作成できます。

手順

1. **$RHDG_HOME** でターミナルを開きます。
2. キーストアを作成し、**credentials** コマンドを使用して認証情報を追加します。

ヒント

デフォルトでは、キーストアのタイプは PKCS12 です。キーストアのデフォルトの変更に関する詳細は、**help credentials** を実行します。

次の例は、パスワード changeme 用に dbpassword のエイリアスを含むキーストアを作成する方法を示しています。キーストアの作成時に、**-p** 引数を使用してキーストアのパスワードも指定します。

Linux

```
bin/cli.sh credentials add dbpassword -c changeme -p "secret1234!"
```

Microsoft Windows

```
bin\cli.bat credentials add dbpassword -c changeme -p "secret1234!"
```

3. エイリアスがキーストアに追加されていることを確認します。

```
bin/cli.sh credentials ls -p "secret1234!"
dbpassword
```

4. 認証情報キーストアを使用するように Data Grid を設定します。
   a. **credential-stores** 設定の認証情報キーストアの名前と場所を指定します。
   b. **credential-reference** 設定で認証情報キーストアとエイリアスを指定します。

ヒント

credential-reference 設定の属性はオプションです。

- **store** は、複数のキーストアがある場合にのみ必要です。
- **alias** は、キーストアに複数のエイリアスが含まれる場合にのみ必要です。

7.2. 認証情報キーストアの設定

このトピックでは、Data Grid Server 設定の認証情報キーストアの例を説明します。

認証情報キーストア

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <!-- Uses a keystore to manage server credentials. -->
    <credential-stores>
      <!-- Specifies the name and filesystem location of a keystore. -->
      <credential-store name="credentials" path="credentials.pfx">
        <!-- Specifies the password for the credential keystore. -->
        <clear-text-credential clear-text="secret1234!"/>
      </credential-store>
    </credential-stores>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "clear-text-credential": {
          "clear-text": "secret1234!"
```

```
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        clearTextCredential:
          clearText: "secret1234!"
```

データソース接続

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <data-sources>
    <data-source name="postgres"
                 jndi-name="jdbc/postgres">
      <!-- Specifies the database username in the connection factory. -->
      <connection-factory driver="org.postgresql.Driver"
                          username="dbuser"
                          url="${org.infinispan.server.test.postgres.jdbcUrl}">
        <!-- Specifies the credential keystore that contains an encrypted password and the alias for it. --
>
        <credential-reference store="credentials"
                              alias="dbpassword"/>
      </connection-factory>
      <connection-pool max-size="10"
                       min-size="1"
                       background-validation="1000"
                       idle-removal="1"
                       initial-size="1"
                       leak-detection="10000"/>
    </data-source>
  </data-sources>
</server>
```

JSON

```
{
  "server": {
    "data-sources": [{
      "name": "postgres",
      "jndi-name": "jdbc/postgres",
      "connection-factory": {
        "driver": "org.postgresql.Driver",
        "username": "dbuser",
```

```
        "url": "${org.infinispan.server.test.postgres.jdbcUrl}",
        "credential-reference": {
          "store": "credentials",
          "alias": "dbpassword"
        }
      }
    }]
  }
}
```

YAML

```
server:
  dataSources:
    - name: postgres
      jndiName: jdbc/postgres
      connectionFactory:
        driver: org.postgresql.Driver
        username: dbuser
        url: '${org.infinispan.server.test.postgres.jdbcUrl}'
        credentialReference:
          store: credentials
          alias: dbpassword
```

LDAP 接続

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <credential-stores>
      <credential-store name="credentials"
                        path="credentials.pfx">
        <clear-text-credential clear-text="secret1234!"/>
      </credential-store>
    </credential-stores>
    <security-realms>
      <security-realm name="default">
        <!-- Specifies the LDAP principal in the connection factory. -->
        <ldap-realm name="ldap"
                    url="ldap://my-ldap-server:10389"
                    principal="uid=admin,ou=People,dc=infinispan,dc=org">
          <!-- Specifies the credential keystore that contains an encrypted password and the alias for it. -
->
          <credential-reference store="credentials"
                                alias="ldappassword"/>
        </ldap-realm>
      </security-realm>
    </security-realms>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "clear-text-credential": {
          "clear-text": "secret1234!"
        }
      }],
      "security-realms": [{
        "name": "default",
        "ldap-realm": {
          "name": "ldap",
          "url": "ldap://my-ldap-server:10389",
          "principal": "uid=admin,ou=People,dc=infinispan,dc=org",
          "credential-reference": {
            "store": "credentials",
            "alias": "ldappassword"
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        clearTextCredential:
          clearText: "secret1234!"
    securityRealms:
      - name: "default"
        ldapRealm:
          name: ldap
          url: 'ldap://my-ldap-server:10389'
          principal: 'uid=admin,ou=People,dc=infinispan,dc=org'
          credentialReference:
            store: credentials
            alias: ldappassword
```

第8章 ユーザーロールとパーミッションの設定

承認は、ユーザーがキャッシュにアクセスしたり、Data Grid リソースとやり取りしたりする前に、特定の権限を持つ必要があるセキュリティー機能です。読み取り専用アクセスから完全なスーパーユーザー特権まで、さまざまなレベルのパーミッションを提供するロールをユーザーに割り当てます。

8.1. セキュリティー-認証

Data Grid の認証は、ユーザーアクセスを制限することでデプロイメントを保護します。

ユーザーアプリケーションまたはクライアントは、Cache Manager またはキャッシュで操作を実行する前に、十分なパーミッションが割り当てられたロールに属している必要があります。

たとえば、特定のキャッシュインスタンスで承認を設定して、**Cache.get()** を呼び出すには、読み取り権限を持つロールを ID に割り当てる必要があり、**Cache.put()** を呼び出すには書き込み権限を持つロールが必要になるようにします。

このシナリオでは、**io** ロールが割り当てられたユーザーアプリケーションまたはクライアントがエントリーの書き込みを試みると、Data Grid はリクエストを拒否し、セキュリティー例外を出力します。**writer** ロールのあるユーザーアプリケーションまたはクライアントが書き込みリクエストを送信する場合、Data Grid は承認を検証し、後続の操作のためにトークンを発行します。

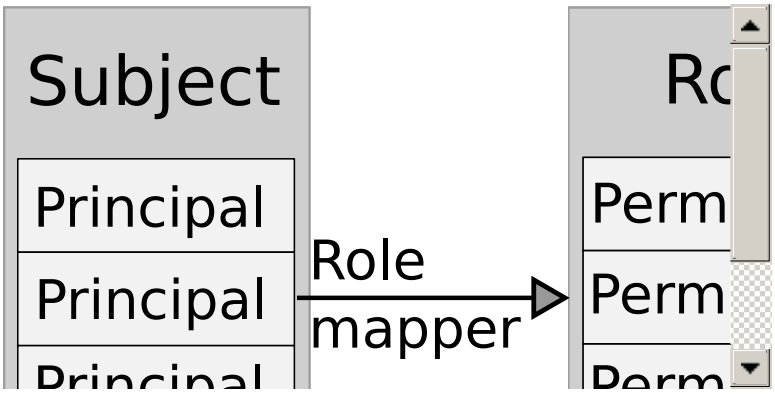
アイデンティティー

アイデンティティーは **java.security.Principal** タイプのセキュリティープリンシパルです。**javax.security.auth.Subject** クラスで実装されたサブジェクトは、セキュリティープリンシパルのグループを表します。つまり、サブジェクトはユーザーとそれが属するすべてのグループを表します。

ロールのアイデンティティー

Data Grid はロールマッパーを使用するため、セキュリティープリンシパルが 1 つ以上のパーミッションを割り当てるロールに対応します。

以下の図は、セキュリティープリンシパルがロールにどのように対応するかを示しています。



8.1.1. ユーザーロールとパーミッション

Data Grid には、データにアクセスして Data Grid リソースと対話するためのパーミッションをユーザーに付与するデフォルトのロールのセットが含まれています。

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。

重要

ClusterRoleMapper は、プリンシパル名をロール名に一致させます。**admin** という名前のユーザーは **admin** パーミッションを自動的に取得し、**deployer** という名前のユーザーは **deployer** パーミッションを取得する、というようになります。

ロール	パーミッション	説明
admin	ALL	Cache Manager ライフサイクルの制御など、すべてのパーミッションを持つスーパーユーザー。
deployer	ALL_READ、ALL_WRITE、LISTEN、EXEC、MONITOR、CREATE	**application** パーミッションに加えて、Data Grid リソースを作成および削除できます。
application	ALL_READ、ALL_WRITE、LISTEN、EXEC、MONITOR	**observer** パーミッションに加え、Data Grid リソースへの読み取りおよび書き込みアクセスがあります。また、イベントをリッスンし、サーバータスクおよびスクリプトを実行することもできます。
observer	ALL_READ、MONITOR	**monitor** パーミッションに加え、Data Grid リソースへの読み取りアクセスがあります。
monitor	MONITOR	JMX および **metrics** エンドポイント経由で統計を表示できます。

参照資料

- org.infinispan.security.AuthorizationPermission Enumeration
- Data Grid 設定スキーマ参照

8.1.2. パーミッション

承認ロールには、Data Grid へのアクセスレベルが異なるさまざまなパーミッションがあります。パーミッションを使用すると、Cache Manager とキャッシュの両方へのユーザーアクセスを制限できます。

8.1.2.1. Cache Manager のパーミッション

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。

パーミッション	機能	説明
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ー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。

8.1.2.2. キャッシュ権限

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。
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。

関連情報

- Data Grid Security API

8.1.3. ロールマッパー

Data Grid には、サブジェクトのセキュリティープリンシパルをユーザーに割り当てる承認ロールにマップする **PrincipalRoleMapper** API が含まれています。

8.1.3.1. クラスターのロールマッパー

ClusterRoleMapper は永続的にレプリケートされたキャッシュを使用して、デフォルトのロールおよびパーミッションのプリンシパルからロールへのマッピングを動的に保存します。

デフォルトでは、プリンシパル名をロール名として使用し、実行時にロールマッピングを変更するメソッドを公開する **org.infinispan.security.MutableRoleMapper** を実装します。

- Java クラス: **org.infinispan.security.mappers.ClusterRoleMapper**

- 宣言型設定: **<cluster-role-mapper />**

8.1.3.2. ID ロールマッパー

IdentityRoleMapper は、プリンシパル名をロール名として使用します。

- Java クラス: **org.infinispan.security.mappers.IdentityRoleMapper**
- 宣言型設定: **<identity-role-mapper />**

8.1.3.3. CommonName ロールマッパー

CommonNameRoleMapper は、プリンシパル名が識別名 (DN) の場合は Common Name (CN) をロール名として使用します。

たとえば、この DN (**cn=managers,ou=people,dc=example,dc=com**) は **managers** ロールにマッピングします。

- Java クラス: **org.infinispan.security.mappers.CommonRoleMapper**
- 宣言型設定: **<common-name-role-mapper />**

8.1.3.4. カスタムロールマッパー

カスタムロールマッパーは **org.infinispan.security.PrincipalRoleMapper** の実装です。

- 宣言型設定: **<custom-role-mapper class="my.custom.RoleMapper" />**

関連情報

- Data Grid Security API
- org.infinispan.security.PrincipalRoleMapper

8.2. アクセス制御リスト (ACL) キャッシュ

Data Grid は、パフォーマンスの最適化のために内部でユーザーに付与するロールをキャッシュします。ロールをユーザーに付与または拒否するたびに、Data Grid は ACL キャッシュをフラッシュして、ユーザーのパーミッションが正しく適用されていることを確認します。

必要に応じて、ACL キャッシュを無効にするか、**cache-size** および **cache-timeout** 属性を使用してこれを設定することができます。

XML

```
<infinispan>
  <cache-container name="acl-cache-configuration">
    <security cache-size="1000"
              cache-timeout="300000">
      <authorization/>
    </security>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "name" : "acl-cache-configuration",
      "security" : {
        "cache-size" : "1000",
        "cache-timeout" : "300000",
        "authorization" : {}
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    name: "acl-cache-configuration"
    security:
      cache-size: "1000"
      cache-timeout: "300000"
      authorization: ~
```

関連情報

- Data Grid 設定スキーマ参照

8.3. ロールおよびパーミッションのカスタマイズ

Data Grid 設定の認証設定をカスタマイズして、異なるロールとパーミッションの組み合わせでロールマッパーを使用できます。

手順

1. Cache Manager 設定でロールマッパーとカスタムロールとパーミッションのセットを宣言します。
2. ユーザーロールに基づいてアクセスを制限するようにキャッシュの承認を設定します。

カスタムロールおよびパーミッションの設定

XML

```
<infinispan>
  <cache-container name="custom-authorization">
    <security>
      <authorization>
        <!-- Declare a role mapper that associates a security principal
             to each role. -->
        <identity-role-mapper />
        <!-- Specify user roles and corresponding permissions. -->
```

```
        <role name="admin" permissions="ALL" />
        <role name="reader" permissions="READ" />
        <role name="writer" permissions="WRITE" />
        <role name="supervisor" permissions="READ WRITE EXEC"/>
      </authorization>
    </security>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "name" : "custom-authorization",
      "security" : {
        "authorization" : {
          "identity-role-mapper" : null,
          "roles" : {
            "reader" : {
              "role" : {
                "permissions" : "READ"
              }
            },
            "admin" : {
              "role" : {
                "permissions" : "ALL"
              }
            },
            "writer" : {
              "role" : {
                "permissions" : "WRITE"
              }
            },
            "supervisor" : {
              "role" : {
                "permissions" : "READ WRITE EXEC"
              }
            }
          }
        }
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    name: "custom-authorization"
    security:
      authorization:
        identityRoleMapper: "null"
```

```
roles:
  reader:
    role:
      permissions:
        - "READ"
  admin:
    role:
      permissions:
        - "ALL"
  writer:
    role:
      permissions:
        - "WRITE"
  supervisor:
    role:
      permissions:
        - "READ"
        - "WRITE"
        - "EXEC"
```

8.4. セキュリティー承認によるキャッシュの設定

キャッシュ設定で承認を使用して、ユーザーアクセスを制限します。キャッシュエントリーの読み取りや書き込み、キャッシュの作成または削除を行う前に、ユーザーは十分なレベルのパーミッションを持つロールを持っている必要があります。

前提条件

- **authorization** 要素が **cache-container** 設定の **security** セクションに含まれていることを確認します。
  Data Grid はデフォルトで Cache Manager でセキュリティー承認を有効にし、キャッシュのグローバルロールおよびパーミッションを提供します。
- 必要な場合は、Cache Manager 設定でカスタムロールとパーミッションを宣言します。

手順

1. キャッシュ設定を開いて編集します。
2. **authorization** 要素をキャッシュに追加し、ロールおよびパーミッションに基づいてユーザーアクセスを制限します。
3. 変更を設定に保存します。

認証設定
以下の設定は、デフォルトのロールおよびパーミッションで暗黙的な認証設定を使用する方法を示しています。

XML

```
<distributed-cache>
  <security>
    <!-- Inherit authorization settings from the cache-container. --> <authorization/>
```

```
  </security>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "security": {
      "authorization": {
        "enabled": true
      }
    }
  }
}
```

YAML

```
distributedCache:
  security:
    authorization:
      enabled: true
```

カスタムロールおよびパーミッション

XML

```
<distributed-cache>
  <security>
    <authorization roles="admin supervisor"/>
  </security>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "security": {
      "authorization": {
        "enabled": true,
        "roles": ["admin","supervisor"]
      }
    }
  }
}
```

YAML

```
distributedCache:
  security:
    authorization:
```

```
enabled: true
roles: ["admin","supervisor"]
```

8.5. セキュリティー承認の無効化

ローカル開発環境では、ユーザーがロールおよびパーミッションを必要としないように、承認を無効にできます。セキュリティー承認を無効にすると、すべてのユーザーがデータにアクセスでき、Data Grid リソースと対話できます。

手順

1. Data Grid 設定を開いて編集します。
2. Cache Manager **security** 設定から **authorization** 要素を削除します。
3. キャッシュから **authorization** 設定を削除します。
4. 変更を設定に保存します。

第9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。

9.1. リモートキャッシュでの統計の有効化

Data Grid Server は、デフォルトのキャッシュマネージャーの統計を自動的に有効にします。ただし、キャッシュの統計を明示的に有効にする必要があります。

手順

1. Data Grid 設定を開いて編集します。
2. **statistics** 属性またはフィールドを追加し、**true** を値として指定します。
3. Data Grid 設定を保存して閉じます。

リモートキャッシュの統計

XML

```
<distributed-cache statistics="true" />
```

JSON

```
{
  "distributed-cache": {
    "statistics": "true"
  }
}
```

YAML

```
distributedCache:
  statistics: true
```

9.2. HOT ROD クライアント統計の有効化

Hot Rod Java クライアントは、リモートキャッシュヒット、ニアキャッシュヒット、ミス、および接続プールの使用状況などの統計を提供できます。

手順

1. Hot Rod Java クライアント設定を開いて編集します。
2. **true** を **statistics** プロパティーの値として指定するか、**statistics().enable()** メソッドを呼び出します。
3. **jmx** および **jmx_domain** プロパティーを使用して Hot Rod クライアントの JMX MBean をエクスポートするか、**jmxEnable()** メソッドおよび **jmxDomain()** メソッドを呼び出します。

4. クライアント設定を保存して閉じます。

Hot Rod Java クライアントの統計

ConfigurationBuilder

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.statistics().enable()
        .jmxEnable()
        .jmxDomain("my.domain.org")
       .addServer()
        .host("127.0.0.1")
        .port(11222);
RemoteCacheManager remoteCacheManager = new RemoteCacheManager(builder.build());
```

hotrod-client.properties

```
infinispan.client.hotrod.statistics = true
infinispan.client.hotrod.jmx = true
infinispan.client.hotrod.jmx_domain = my.domain.org
```

9.3. DATA GRID メトリックの設定

Data Grid は、MicroProfile Metrics API と互換性のあるメトリックを生成します。

- 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。

メトリックの設定

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"
```

検証

Data Grid Server は、**metrics** エンドポイント経由で統計を公開します。Prometheus などの OpenMetrics 形式をサポートするモニタリングツールを使用して、メトリクスを収集できます。

Data Grid メトリックは、**vendor** スコープで提供されます。JVM に関連するメトリックは **base** スコープで提供されます。

以下のように、Data Grid Server からメトリックを取得できます。

```
$ curl -v http://localhost:11222/metrics
```

MicroProfile JSON 形式でメトリックを取得するには、以下を実行します。

```
$ curl --header "Accept: application/json" http://localhost:11222/metrics
```

関連情報

- Eclipse MicroProfile Metrics

9.4. 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

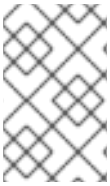
```
{
  "infinispan" : {
    "cache-container" : {
      "statistics" : "true",
      "jmx" : {
        "enabled" : "true",
        "domain" : "example.com"
      }
    }
  }
}
```

YAML

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

9.4.1. 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。



注記

Data Grid サーバーは、単一ポートエンドポイントを使用して JMX をリモートに公開しません。JMX を介して Data Grid サーバーにリモートアクセスする場合は、リモートポートを有効にする必要があります。

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

9.4.2. 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 の詳細なリストおよび説明、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

9.4.3. 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"
```

第10章 管理データソースの DATA GRID SERVER への追加

管理データソースを Data Grid Server 設定に追加して、JDBC 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します。

10.1. 管理対象データソースの設定

Data Grid Server 設定の一部としてマネージドデータソースを作成し、JDBC 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します。その後、キャッシュ内のマネージドデータソースの JDNI 名を指定して、デプロイメントの JDBC 接続設定を一元化できます。

前提条件

- データベースドライバーを、Data Grid Server インストールの **server/lib** ディレクトリーにコピーします。

手順

1. Data Grid Server 設定を開いて編集します。
2. **data-sources** セクションに新しい **data-source** を追加します。
3. **name** 属性またはフィールドでデータソースを一意に識別します。
4. **jndi-name** 属性またはフィールドを使用してデータソースの JNDI 名を指定します。

   ヒント

   JNDI 名を使用して、JDBC キャッシュストア設定でデータソースを指定します。

5. **true** を **statistics** 属性またはフィールドの値に設定し、**/metrics** エンドポイント経由でデータソースの統計を有効にします。
6. **connection-factory** セクションのデータソースへの接続方法を定義する JDBC ドライバーの詳細を提供します。
   a. **driver** 属性またはフィールドを使用して、データベースドライバーの名前を指定します。
   b. **url** 属性またはフィールドを使用して、JDBC 接続 URL を指定します。
   c. **username** および **password** 属性またはフィールドを使用して、認証情報を指定します。
   d. 必要に応じて他の設定を指定します。
7. Data Grid Server ノードが接続をプールして再利用する方法を、**connection-pool** セクションの接続プール調整プロパティーで定義します。
8. 変更を設定に保存します。

検証

以下のように、Data Grid コマンドラインインターフェイス (CLI) を使用して、データソース接続をテストします。

1. CLI セッションを開始します。
   -

```
bin/cli.sh
```

2. すべてのデータソースをリスト表示し、作成したデータソースが利用できることを確認します。

```
server datasource ls
```

3. データソース接続をテストします。

```
server datasource test my-datasource
```

マネージドデータソースの設定

XML

```
<server xmlns="urn:infinispan:server:13.0">
  <data-sources>
    <!-- Defines a unique name for the datasource and JNDI name that you
         reference in JDBC cache store configuration.
         Enables statistics for the datasource, if required. -->
    <data-source name="ds"
                 jndi-name="jdbc/postgres"
                 statistics="true">
      <!-- Specifies the JDBC driver that creates connections. -->
      <connection-factory driver="org.postgresql.Driver"
                          url="jdbc:postgresql://localhost:5432/postgres"
                          username="postgres"
                          password="changeme">
        <!-- Sets optional JDBC driver-specific connection properties. -->
        <connection-property name="name">value</connection-property>
      </connection-factory>
      <!-- Defines connection pool tuning properties. -->
      <connection-pool initial-size="1"
                       max-size="10"
                       min-size="3"
                       background-validation="1000"
                       idle-removal="1"
                       blocking-timeout="1000"
                       leak-detection="10000"/>
    </data-source>
  </data-sources>
</server>
```

JSON

```
{
  "server": {
    "data-sources": [{
      "name": "ds",
      "jndi-name": "jdbc/postgres",
      "statistics": true,
      "connection-factory": {
        "driver": "org.postgresql.Driver",
```

```
        "url": "jdbc:postgresql://localhost:5432/postgres",
        "username": "postgres",
        "password": "changeme",
        "connection-properties": {
          "name": "value"
        }
      },
      "connection-pool": {
        "initial-size": 1,
        "max-size": 10,
        "min-size": 3,
        "background-validation": 1000,
        "idle-removal": 1,
        "blocking-timeout": 1000,
        "leak-detection": 10000
      }
    }]
  }
}
```

YAML

```
server:
  dataSources:
    - name: ds
      jndiName: 'jdbc/postgres'
      statistics: true
      connectionFactory:
        driver: "org.postgresql.Driver"
        url: "jdbc:postgresql://localhost:5432/postgres"
        username: "postgres"
        password: "changeme"
        connectionProperties:
          name: value
      connectionPool:
        initialSize: 1
        maxSize: 10
        minSize: 3
        backgroundValidation: 1000
        idleRemoval: 1
        blockingTimeout: 1000
        leakDetection: 10000
```

10.2. JNDI 名を使用したキャッシュの設定

マネージドデータソースを Data Grid Server に追加するとき、JNDI 名を JDBC ベースのキャッシュストア設定に追加できます。

前提条件

- マネージドデータソースを使用した Data Grid Server の設定

手順

1. キャッシュ設定を開いて編集します。
2. **data-source** 要素またはフィールドを JDBC ベースのキャッシュストア設定に追加します。
3. マネージドデータソースの JNDI 名を **jndi-url** 属性の値として指定します。
4. JDBC ベースのキャッシュストアを適宜設定します。
5. 変更を設定に保存します。

キャッシュ設定の JNDI 名

XML

```
<distributed-cache>
  <persistence>
    <jdbc:string-keyed-jdbc-store>
      <!-- Specifies the JNDI name of a managed datasource on Data Grid Server. -->
      <jdbc:data-source jndi-url="jdbc/postgres"/>
      <jdbc:string-keyed-table drop-on-exit="true" create-on-start="true" prefix="TBL">
        <jdbc:id-column name="ID" type="VARCHAR(255)"/>
        <jdbc:data-column name="DATA" type="BYTEA"/>
        <jdbc:timestamp-column name="TS" type="BIGINT"/>
        <jdbc:segment-column name="S" type="INT"/>
      </jdbc:string-keyed-table>
    </jdbc:string-keyed-jdbc-store>
  </persistence>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "persistence": {
      "string-keyed-jdbc-store": {
        "data-source": {
          "jndi-url": "jdbc/postgres"
          },
        "string-keyed-table": {
          "prefix": "TBL",
          "drop-on-exit": true,
          "create-on-start": true,
          "id-column": {
            "name": "ID",
            "type": "VARCHAR(255)"
          },
          "data-column": {
            "name": "DATA",
            "type": "BYTEA"
          },
          "timestamp-column": {
            "name": "TS",
            "type": "BIGINT"
          },
          "segment-column": {
```

```
            "name": "S",
            "type": "INT"
          }
        }
      }
    }
  }
}
```

YAML

```
distributedCache:
  persistence:
    stringKeyedJdbcStore:
      dataSource:
        jndi-url: "jdbc/postgres"
      stringKeyedTable:
        prefix: "TBL"
        dropOnExit: true
        createOnStart: true
        idColumn:
          name: "ID"
          type: "VARCHAR(255)"
        dataColumn:
          name: "DATA"
          type: "BYTEA"
        timestampColumn:
          name: "TS"
          type: "BIGINT"
        segmentColumn:
          name: "S"
          type: "INT"
```

10.3. 接続プールのチューニングプロパティー

Data Grid Server 設定で、マネージドデータソースの JDBC 接続プールを調整できます。

プロパティー	説明
initial-size	プールが保持する最初の接続数。
max-size	プールの最大接続数。
min-size	プールが保持する必要のある接続の最小数。
blocking-timeout	例外が発生する前に、接続を待機している間にブロックする最大時間 (ミリ秒単位)。新しい接続の作成に非常に長い時間がかかる場合は、これによって例外が出力されることはありません。デフォルトは **0** です。これは、呼び出しが無期限に待機することを意味します。

プロパティー	説明
background-validation	バックグラウンド検証の実行の間隔 (ミリ秒単位)。期間 **0** は、この機能が無効化されていることを意味します。
validate-on-acquisition	ミリ秒単位で指定された、この時間より長いアイドル状態の接続は、取得される前に検証されます (フォアグラウンド検証)。期間 **0** は、この機能が無効化されていることを意味します。
idle-removal	削除される前の接続がアイドル状態でなくてはならない時間 (分単位)。
leak-detection	リーク警告の前に接続を保持しなければならない時間 (ミリ秒単位)。

第11章 DATA GRID クラスト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。

11.1. 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。

この JAR ファイルは **$RHDG_HOME/lib** ディレクトリーにあります。

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

11.2. 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)

11.2.1. 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

11.2.2. 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

11.2.3. 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

11.2.4. 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

11.2.5. 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

```
<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

11.2.6. 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)

11.2.7. 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

1. アーティファクト JAR ファイルとすべての依存関係をダウンロードします。
2. アーティファクト JAR ファイルとすべての依存関係を、Data Grid Server インストールの **$RHDG_HOME/server/lib** ディレクトリーに追加します。
   詳細は、Downloading artifacts for JGroups cloud discover protocols for Data Grid Server (Red Hat ナレッジベースの記事) を参照してください。

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

11.3. 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>
```

- Data Grid Server の起動時に、**cluster-stack** 引数を使用して JGroups スタックファイルを設定します。

```
bin/server.sh --cluster-stack=udp
```

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)

11.4. 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)

11.4.1. 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。

11.5. 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。

```
bin/server.sh -Djgroups.bind.port=1234 -Djgroups.bind.address=192.0.2.0
```

11.5.1. 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	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

システムプロパティー	説明	Default Value	必須/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
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

関連情報

- JGroups system properties
- JGroups protocol list

11.5.2. 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。

11.5.2.1. 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	Default Value	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。	オプション
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。	オプション

11.5.2.2. 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	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

11.5.2.3. 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	Default Value	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

11.5.2.4. 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	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

11.6. 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

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>
```

11.7. 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 **$RHDG_HOME/server/conf** ディレクトリーに追加します。
   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>
```

11.8. 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。

11.8.1. TLS アイデンティティーを使用したクラスタートランスポートのセキュア化

SSL/TLS アイデンティティーを Data Grid Server セキュリティーレルムに追加し、これを使用してクラスタートランスポートをセキュア化します。Data Grid Server クラスターのノードは、SSL/TLS 証明書を交換して、クロスサイトレプリケーションを設定する場合の RELAY メッセージを含む JGroups メッセージを暗号化します。

前提条件

- Data Grid Server クラスターをインストールします。

手順

1. 1 つの証明書が含まれる TLS キーストアを作成し、Data Grid Server を特定します。PKCS#1 または PKCS#8 形式の秘密鍵、証明書、および空のパスワードである **password=""** が含まれている場合は、PEM ファイルを使用することもできます。

注記

キーストアの証明書が公開認証局 (CA) で署名されていない場合は、署名証明書または公開鍵のいずれかが含まれるトラストストアを作成する必要もあります。

2. キーストアを **$RHDG_HOME/server/conf** ディレクトリーに追加します。
3. キーストアを Data Grid Server 設定の新しいセキュリティーレルムに追加します。

重要

Data Grid Server エンドポイントがクラスタートランスポートと同じセキュリティーレルムを使用しないように、専用のキーストアとセキュリティーレルムを作成する必要があります。

```
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="cluster-transport">
```

```
      <server-identities>
        <ssl>
          <!-- Adds a keystore that contains a certificate that provides SSL/TLS identity to
encrypt cluster transport. -->
          <keystore path="server.pfx"
                    relative-to="infinispan.server.config.path"
                    password="secret"
                    alias="server"/>
        </ssl>
      </server-identities>
    </security-realm>
  </security-realms>
 </security>
</server>
```

4. **server:security-realm** 属性でセキュリティーレルムの名前を指定して、セキュリティーレルムを使用するようにクラスタートランスポートを設定します。

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

検証

Data Grid Server を起動すると、以下のログメッセージはクラスターがクラスタートランスポートにセキュリティーレルムを使用していることを示します。

```
[org.infinispan.SERVER] ISPN080060: SSL Transport using realm <security_realm_name>
```

11.8.2. JGroups 暗号化プロトコル

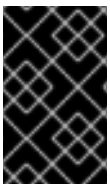
クラスタートラフィックのセキュリティーを保護するには、Data Grid ノードを設定し、シークレットキーで JGroups メッセージペイロードを暗号化します。

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ター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、クラスター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。

11.8.3. 非対称暗号化を使用したクラスター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

11.8.4. 対称暗号化を使用したクラスタ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

11.9. 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
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。

第12章 リモートキャッシュの作成

ランタイム時にリモートキャッシュを作成すると、Data Grid Server はクラスター全体で設定を同期し、全ノードがコピーを持つようにします。このため、常に以下のメカニズムを使用してリモートキャッシュを動的に作成する必要があります。

- Data Grid コンソール
- Data Grid コマンドラインインターフェイス (CLI)
- Hot Rod または HTTP クライアント

12.1. デフォルトの CACHE MANAGER

Data Grid Server は、リモートキャッシュのライフサイクルを制御するデフォルトの Cache Manager を提供します。Data Grid Server を起動すると、Cache Manager が自動的にインスタンス化されるため、リモートキャッシュや Protobuf スキーマなどの他のリソースを作成および削除できます。

Data Grid Server を起動してユーザー認証情報を追加したら、Cache Manager の詳細を表示し、Data Grid コンソールからクラスター情報を取得できます。

- 任意のブラウザーで **127.0.0.1:11222** を開きます。

コマンドラインインターフェイス (CLI) または REST API を使用して Cache Manager に関する情報を取得することもできます。

CLI

デフォルトのコンテナーで **describe** コマンドを使用します。

```
[//containers/default]> describe
```

REST

任意のブラウザーで **127.0.0.1:11222/rest/v2/cache-managers/default/** を開きます。

デフォルトの Cache Manager の設定

XML

```
<infinispan>
  <!-- Creates a Cache Manager named "default" and enables metrics. -->
  <cache-container name="default"
                   statistics="true">
    <!-- Adds cluster transport that uses the default JGroups TCP stack. -->
    <transport cluster="${infinispan.cluster.name:cluster}"
               stack="${infinispan.cluster.stack:tcp}"
               node-name="${infinispan.node.name:}"/>
    <!-- Requires user permission to access caches and perform operations. -->
    <security>
      <authorization/>
    </security>
  </cache-container>
</infinispan>
```

JSON

JSON

```
{
  "infinispan" : {
    "jgroups" : {
      "transport" : "org.infinispan.remoting.transport.jgroups.JGroupsTransport"
    },
    "cache-container" : {
      "name" : "default",
      "statistics" : "true",
      "transport" : {
        "cluster" : "cluster",
        "node-name" : "",
        "stack" : "tcp"
      },
      "security" : {
        "authorization" : {}
      }
    }
  }
}
```

YAML

```
infinispan:
  jgroups:
    transport: "org.infinispan.remoting.transport.jgroups.JGroupsTransport"
  cacheContainer:
    name: "default"
    statistics: "true"
    transport:
      cluster: "cluster"
      nodeName: ""
      stack: "tcp"
    security:
      authorization: ~
```

12.2. DATA GRID コンソールを使用したキャッシュの作成

Data Grid コンソールを使用して、任意の Web ブラウザーから直感的なビジュアルインターフェイスでリモートキャッシュを作成します。

前提条件

- **admin** パーミッションを持つ Data Grid ユーザーを作成します。
- 1 つ以上の Data Grid Server インスタンスを起動します。
- Data Grid キャッシュ設定があります。

手順

1. 任意のブラウザーで **127.0.0.1:11222/console/** を開きます。

2. **Create Cache** を選択し、プロセスを Data Grid コンソールガイドの手順に従ってください。

12.3. DATA GRID CLI を使用したリモートキャッシュの作成

Data Grid コマンドラインインターフェイス (CLI) を使用して、Data Grid Server にリモートキャッシュを追加します。

前提条件

- **admin** パーミッションを持つ Data Grid ユーザーを作成します。
- 1 つ以上の Data Grid Server インスタンスを起動します。
- Data Grid キャッシュ設定があります。

手順

1. CLI を起動し、プロンプトが表示されたら認証情報を入力します。

```
bin/cli.sh
```

2. **create cache** コマンドを使用してリモートキャッシュを作成します。
たとえば、以下のように **mycache.xml** という名前のファイルから"mycache"という名前のキャッシュを作成します。

```
create cache --file=mycache.xml mycache
```

検証

1. **ls** コマンドを使用して、すべてのリモートキャッシュをリスト表示します。

```
ls caches
mycache
```

2. **describe** コマンドでキャッシュ設定を表示します。

```
describe caches/mycache
```

12.4. HOT ROD クライアントからのリモートキャッシュの作成

Data Grid Hot Rod API を使用して、Java、C++、.NET/C#、JS クライアントなどから Data Grid Server にリモートキャッシュを作成します。

この手順では、最初のアクセスでリモートキャッシュを作成する Hot Rod Java クライアントを使用する方法を示します。他の Hot Rod クライアントのコード例は、Data Grid Tutorials を参照してください。

前提条件

- **admin** パーミッションを持つ Data Grid ユーザーを作成します。
- 1 つ以上の Data Grid Server インスタンスを起動します。

- Data Grid キャッシュ設定があります。

手順

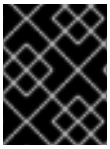
- **ConfigurationBuilder** の一部として **remoteCache()** メソッドを呼び出します。
- クラスパスの **hotrod-client.properties** ファイルで **configuration** または **configuration_uri** プロパティーを設定します。

ConfigurationBuilder

```
File file = new File("path/to/infinispan.xml")
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.remoteCache("another-cache")
      .configuration("<distributed-cache name=\"another-cache\"/>");
builder.remoteCache("my.other.cache")
      .configurationURI(file.toURI());
```

hotrod-client.properties

```
infinispan.client.hotrod.cache.another-cache.configuration=<distributed-cache name=\"another-cache\"/>
infinispan.client.hotrod.cache.[my.other.cache].configuration_uri=file:///path/to/infinispan.xml
```



重要

リモートキャッシュの名前に **.** が含まれる場合は、**hotrod-client.properties** ファイルを使用する場合は角括弧で囲む必要があります。

関連情報

- Hot Rod Client Configuration
- org.infinispan.client.hotrod.configuration.RemoteCacheConfigurationBuilder

12.5. REST API を使用したリモートキャッシュの作成

Data Grid REST API を使用して、適切な HTTP クライアントから Data Grid Server でリモートキャッシュを作成します。

前提条件

- **admin** パーミッションを持つ Data Grid ユーザーを作成します。
- 1 つ以上の Data Grid Server インスタンスを起動します。
- Data Grid キャッシュ設定があります。

手順

- ペイロードにキャッシュ設定を指定して **/rest/v2/caches/<cache_name>** に **POST** 要求を呼び出します。

関連情報

- Creating and Managing Caches with the REST API

第13章 DATA GRID SERVER でのスクリプトおよびタスクの実行

コマンドラインインターフェイス (CLI) および Hot Rod または REST クライアントから、リモート実行用の Data Grid Server デプロイメントにタスクおよびスクリプトを追加します。カスタム Java クラスとしてタスクを実装するか、JavaScript などの言語でスクリプトを定義することができます。

13.1. DATA GRID SERVER デプロイメントへのタスクの追加

カスタムサーバータスククラスを Data Grid Server に追加します。

前提条件

- Data Grid Server が実行している場合は停止します。
  Data Grid Server はカスタムクラスのランタイムデプロイメントをサポートしません。

手順

1. 以下のように、サーバーのタスクの完全修飾名が含まれる **META-INF/services/org.infinispan.tasks.ServerTask** ファイルを追加します。

   ```
   example.HelloTask
   ```

2. JAR ファイルでサーバータスクの実装をパッケージ化します。
3. JAR ファイルを Data Grid Server インストールの **$RHDG_HOME/server/lib** ディレクトリーにコピーします。
4. クラスを Data Grid 設定のデシリアライズ許可リストに追加します。または、システムプロパティーを使用して許可リストを設定します。

参照資料

- Adding Java Classes to Deserialization Allow Lists
- Data Grid 8.3 Configuration Schema

13.1.1. Data Grid Server タスク

Data Grid Server のタスクは、**org.infinispan.tasks.ServerTask** インターフェイスを拡張するクラスで、通常以下のメソッド呼び出しが含まれます。

setTaskContext()

タスクパラメーター、タスクが実行されるキャッシュ参照などを含む実行コンテキスト情報へのアクセスを許可します。ほとんどの場合、実装はこの情報をローカルに保存し、タスクが実際に実行したときに使用します。**SHARED** インスタンス化モードを使用する場合、タスクは同時呼び出しのために **TaskContext** を格納するために **ThreadLocal** を使用する必要があります。

getName()

タスクの一意の名前を返します。クライアントはこれらの名前でタスクを呼び出します。

getExecutionMode()

タスクの実行モードを返します。

- **TaskExecutionMode.ONE_NODE** は、要求を処理するノードのみがスクリプトを実行します。ただし、スクリプトはクラスター化された操作を引き続き呼び出すことができます。こ

れはデフォルトになります。

- **TaskExecutionMode.ALL_NODES** Data Grid は、クラスター化されたエグゼキューターを使用してノード間でスクリプトを実行します。たとえば、ストリーム処理はすべてのノードに分散されるため、ストリーム処理を呼び出したサーバータスクを 1 つのノードで実行する必要があります。

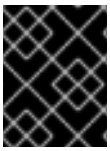
getInstantiationMode()

タスクのインスタンス化モードを返します。

- **TaskInstantiationMode.SHARED** は、同じサーバーでのすべてのタスク実行に再利用される単一のインスタンスを作成します。これはデフォルトになります。
- **TaskInstantiationMode.ISOLATED** は、呼び出しごとに新しいインスタンスを作成します。

call()

結果を計算します。このメソッドは **java.util.concurrent.Callable** インターフェイス内で定義され、サーバータスクにより呼び出されます。



重要

サーバータスクの実装は、サービスローダーパターンの要件に準拠する必要があります。たとえば、実装にはゼロ引数のコンストラクターが必要です。

以下の **HelloTask** クラス実装は、1 つのパラメーターを持つタスクの例を提供します。また、**ThreadLocal** を使用して、同時呼び出し用の **TaskContext** を格納する方法も示しています。

```
package example;

import org.infinispan.tasks.ServerTask;
import org.infinispan.tasks.TaskContext;

public class HelloTask implements ServerTask<String> {

   private static final ThreadLocal<TaskContext> taskContext = new ThreadLocal<>();

   @Override
   public void setTaskContext(TaskContext ctx) {
      taskContext.set(ctx);
   }

   @Override
   public String call() throws Exception {
      TaskContext ctx = taskContext.get();
      String name = (String) ctx.getParameters().get().get("name");
      return "Hello " + name;
   }

   @Override
   public String getName() {
      return "hello-task";
```

```
    }
  }
```

参照

- org.infinispan.tasks.ServerTask
- java.util.concurrent.Callable.call()
- java.util.ServiceLoader

13.2. DATA GRID SERVER デプロイメントへのスクリプトの追加

コマンドラインインターフェイスを使用して、スクリプトを Data Grid Server に追加します。

前提条件

Data Grid Server は、**___script_cache** キャッシュにスクリプトを保存します。キャッシュ承認を有効にする場合、ユーザーは **___script_cache** に追加するための **CREATE** パーミッションを持っている必要があります。

デフォルトの承認設定を使用する場合は、ユーザーに少なくとも **deployer** ロールを割り当てます。

手順

1. 必要に応じてスクリプトを定義します。
   たとえば、単一の Data Grid サーバーで実行し、2 つのパラメーターを持ち、JavaScript を使用して指定の値を掛ける **multiplication.js** という名前のファイルを作成します。

```
// mode=local,language=javascript
multiplicand * multiplier
```

2. Data Grid への CLI 接続を作成します。
3. 以下の例のように、**task** コマンドを使用してスクリプトをアップロードします。

```
task upload --file=multiplication.js multiplication
```

4. スクリプトが利用可能であることを確認します。

```
ls tasks
multiplication
```

13.2.1. Data Grid Server スクリプト

Data Grid Server のスクリプトは **javax.script** API をベースとしており、JVM ベースの ScriptEngine 実装と互換性があります。

Hello world

以下は、1 つの Data Grid Server で実行され、1 つのパラメーターを持ち、JavaScript を使用する簡単な例です。

```
// mode=local,language=javascript,parameters=[greetee]
"Hello " + greetee
```

上記のスクリプトを実行するときに、**greetee** パラメーターの値を渡すと、Data Grid は **"Hello ${value}"** を返します。

13.2.1.1. スクリプトのメタデータ

メタデータは、スクリプトの実行時に Data Grid Server が使用するスクリプトの追加情報を提供します。

スクリプトメタデータは、スクリプトの最初の行でコメントを追加する **property=value** ペアです。以下に例を示します。

```
// name=test, language=javascript
// mode=local, parameters=[a,b,c]
```

- スクリプト言語 (//, **;;**, **#**) に一致するコメントスタイルを使用してください。
- コンマで区切られた **property=value** ペア
- 値は一重引用符 (') または二重引用符 (") で区切ります。

表13.1 メタデータプロパティー

プロパティー	説明
モード	実行モードを定義し、以下の値を取ります。 **local** は、リクエストを処理するノードのみがスクリプトを実行します。ただし、スクリプトはクラスター化された操作を引き続き呼び出すことができます。 **distributed** Data Grid は、クラスター化されたエグゼキューターを使用してノード間でスクリプトを実行します。
言語	スクリプトを実行する ScriptEngine を指定します。
extension	ScriptEngine を設定する代替方法としてファイル名の拡張子を指定します。
role	ユーザーがスクリプトを実行する必要があるロールを指定します。
parameters	このスクリプトの有効なパラメーター名の配列を指定します。このリストに含まれていないパラメーターを指定する呼び出しによって例外が生じます。

プロパティー	説明
datatype	オプションで、データおよびパラメーターおよび戻り値を保存する MediaType(MIME タイプ) を設定します。このプロパティーは、特定のデータフォーマットのみをサポートするリモートクライアントに便利です。 現在、データに String UTF-8 形式を使用するように、**text/plain; charset=utf-8** のみを設定できます。

13.2.1.2. スクリプトバインディング

Data Grid は、内部オブジェクトをスクリプト実行のバインディングとして公開します。

バインディング	説明
cache	スクリプトが実行されるキャッシュを指定します。
marshaller	データをキャッシュにシリアル化するために使用するマーシャラーを指定します。
cacheManager	キャッシュの **cacheManager** を指定します。
scriptingManager	スクリプトを実行するスクリプトマネージャーのインスタンスを指定します。このバインディングを使用して、スクリプトから他のスクリプトを実行することができます。

13.2.1.3. スクリプトのパラメーター

Data Grid を使用すると、スクリプトを実行するためのバインディングとして名前付きパラメーターを渡すことができます。

パラメーターは **name,value** のペアで、**name** は文字列、**value** はマーシャラーが解釈できる任意の値になります。

以下のスクリプトの例では、**multiplicand** と **multiplier** の 2 つのパラメーターがあります。このスクリプトは **multiplicand** の値を取り、その値を **multiplier** の値で乗算します。

```
// mode=local,language=javascript
multiplicand * multiplier
```

上記のスクリプトを実行すると、Data Grid は式の評価の結果で応答します。

13.2.2. プログラムでのスクリプトの作成

以下の例のように、Hot Rod **RemoteCache** インターフェイスを使用してスクリプトを追加します。

```
RemoteCache<String, String> scriptCache = cacheManager.getCache("___script_cache");
scriptCache.put("multiplication.js",
  "// mode=local,language=javascript\n" +
  "multiplicand * multiplier\n");
```

参照資料

org.infinispan.client.hotrod.RemoteCache

13.3. スクリプトおよびタスクの実行

コマンドラインインターフェイスを使用して、Data Grid Server デプロイメントでタスクおよびスクリプトを実行します。また、Hot Rod クライアントからスクリプトとタスクを実行することもできます。

前提条件

- スクリプトまたはタスクを Data Grid Server に追加します。

手順

1. Data Grid への CLI 接続を作成します。
2. 以下の例のように、**task** コマンドを使用して、タスクおよびスクリプトを実行します。
   - **multiplier.js** という名前のスクリプトを実行し、2 つのパラメーターを指定します。

     ```
     task exec multiplier.js -Pmultiplicand=10 -Pmultiplier=20
     200.0
     ```

   - **@@cache@names** という名前のタスクを実行して、利用可能なすべてのキャッシュのリストを取得します。

     ```
     task exec @@cache@names
     ["___protobuf_metadata","mycache","___script_cache"]
     ```

プログラムによる実行

- 以下の例のように、**execute()** を呼び出して、Hot Rod **RemoteCache** インターフェイスを使用してスクリプトを実行します。

スクリプト実行

```
RemoteCache<String, Integer> cache = cacheManager.getCache();
// Create parameters for script execution.
Map<String, Object> params = new HashMap<>();
params.put("multiplicand", 10);
params.put("multiplier", 20);
// Run the script with the parameters.
Object result = cache.execute("multiplication.js", params);
```

タスクの実行

```
// Add configuration for a locally running server.
ConfigurationBuilder builder = new ConfigurationBuilder();
```

```
builder.addServer().host("127.0.0.1").port(11222);

// Connect to the server.
RemoteCacheManager cacheManager = new RemoteCacheManager(builder.build());

// Retrieve the remote cache.
RemoteCache<String, String> cache = cacheManager.getCache();

// Create task parameters.
Map<String, String> parameters = new HashMap<>();
parameters.put("name", "developer");

// Run the server task.
String greet = cache.execute("hello-task", parameters);
System.out.println(greet);
```

関連情報

- org.infinispan.client.hotrod.RemoteCache

第14章 DATA GRID SERVER ロギングの設定

Data Grid Server は Apache Log4j 2 を使用して、トラブルシューティング目的および根本原因分析用に環境およびレコードキャッシュ操作の詳細を把握する設定可能なロギングメカニズムを提供します。

14.1. DATA GRID SERVER のログファイル

Data Grid は、サーバーログを **$RHDG_HOME/server/log** ディレクトリー内の以下のファイルに書き込みます。

server.log

サーバーの起動に関連する起動ログなど、人間が判読できる形式のメッセージ。
Data Grid は、サーバーの起動時にこのファイルを作成します。

server.log.json

Data Grid ログを解析および分析できる JSON 形式のメッセージ。
JSON-FILE アペンダーを有効にすると、Data Grid はこのファイルを作成します。

14.1.1. Data Grid Server ログの設定

Data Grid は Apache Log4j テクノロジーを使用して、サーバーログメッセージを書き込みます。サーバーログを **log4j2.xml** ファイルで設定できます。

手順

1. 任意のテキストエディターで **$RHDG_HOME/server/conf/log4j2.xml** を開きます。
2. 必要に応じてサーバーロギングを変更します。
3. **log4j2.xml** を保存し、閉じます。

関連情報

- Apache Log4j マニュアル

14.1.2. ログレベル

ログレベルは、メッセージの性質と重大度を示します。

ログレベル	説明
TRACE	粒度の細かいデバッグメッセージ。アプリケーションを介して個々のリクエストのフローをキャプチャーします。
DEBUG	個々のリクエストと関連性のない、一般的なデバッグのメッセージ。
INFO	ライフサイクルイベントを含む、アプリケーションの全体的な進捗状況に関するメッセージ。

ログレベル	説明
WARN	エラーやパフォーマンスの低下につながる可能性のあるイベント。
ERROR	操作またはアクティビティーの正常な実行を妨げる可能性がありますが、アプリケーションの実行は妨げないエラー状態。
FATAL	重大なサービス障害やアプリケーションのシャットダウンを引き起こす可能性のあるイベント。

上記の個々のメッセージのレベルに加えて、この設定では、**ALL** (すべてのメッセージを含む) と **OFF** (すべてのメッセージを除外) のさらに 2 つの値を可能にします。

14.1.3. Data Grid のロギングカテゴリー

Data Grid は、機能領域ごとにログを整理する **INFO**、**WARN**、**ERROR**、**FATAL** のレベルのメッセージのカテゴリーを提供します。

org.infinispan.CLUSTER

状態遷移操作、イベントのリバランス、パーティション設定などが含まれる Data Grid クラスタリング固有のメッセージ。

org.infinispan.CONFIG

Data Grid 設定固有のメッセージ

org.infinispan.CONTAINER

有効期限とエビクション操作、キャッシュリスナーの通知、トランザクションなどを含むデータコンテナーに固有のメッセージ。

org.infinispan.PERSISTENCE

キャッシュローダーとストアに固有のメッセージ。

org.infinispan.SECURITY

Data Grid のセキュリティーに固有のメッセージ。

org.infinispan.SERVER

Data Grid Server に固有のメッセージ。

org.infinispan.XSITE

クロスサイトレプリケーション操作に固有のメッセージ。

14.1.4. ログアペンダー

ログアペンダーは、Data Grid Server がログメッセージを記録する方法を定義します。

CONSOLE

ログメッセージをホストの標準出力 (**stdout**) または標準エラー (**stderr**) ストリームに書き込みます。
デフォルトで **org.apache.logging.log4j.core.appender.ConsoleAppender** クラスを使用します。

FILE

ファイルにログメッセージを書き込みます。
デフォルトで **org.apache.logging.log4j.core.appender.RollingFileAppender** クラスを使用します。

JSON-FILE

ログメッセージを JSON 形式でファイルに書き込みます。
デフォルトで **org.apache.logging.log4j.core.appender.RollingFileAppender** クラスを使用します。

14.1.5. ログパターンフォーマッター

CONSOLE および **FILE** アペンダーは、**PatternLayout** を使用して、**pattern** に従ってログメッセージをフォーマットします。

以下は、FILE アペンダーのデフォルトのパターンになります。
%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p (%t) [%c{1}] %m%throwable%n

- **%d{yyyy-MM-dd HH:mm:ss,SSS}** は現在の日時を追加します。
- **%-5p** は、ログレベルを右揃えで指定します。
- **%t** は、現在のスレッドの名前を追加します。
- **%c{1}** はロギングカテゴリーの短縮名を追加します。
- **%m** はログメッセージを追加します。
- **%throwable** は例外スタックトレースを追加します。
- **%n** は改行を追加します。

パターンについては、**PatternLayout** ドキュメント で詳細に説明されています。

14.1.6. JSON ログハンドラーの有効化

Data Grid Server は、JSON 形式でメッセージを書き込むログハンドラーを提供します。

前提条件

- Data Grid Server が実行されている場合は停止します。
ログハンドラーは動的に有効にすることはできません。

手順

1. 任意のテキストエディターで **$RHDG_HOME/server/conf/log4j2.xml** を開きます。
2. **JSON-FILE** アペンダーのコメントを解除して、**FILE** アペンダーをコメントアウトします。

```
<!--<AppenderRef ref="FILE"/>-->
<AppenderRef ref="JSON-FILE"/>
```

3. 必要に応じて、JSON アペンダーと JSON レイアウトを設定します。
4. **log4j2.xml** を保存し、閉じます。

Data Grid を開始すると、各ログメッセージを JSON マップとして **$RHDG_HOME/server/log/server.log.json** ファイルに書き込みます。

関連情報

- RollingFileAppender
- JSONLayout

14.2. アクセスログ

アクセスログは、Hot Rod および REST エンドポイントのすべてのインバウンドクライアント要求を **$RHDG_HOME/server/log** ディレクトリー内のファイルを記録します。

org.infinispan.HOTROD_ACCESS_LOG

Hot Rod アクセスメッセージを **hotrod-access.log** ファイルに書き込むロギングカテゴリー。

org.infinispan.REST_ACCESS_LOG

REST アクセスメッセージを **rest-access.log** ファイルに書き込むロギングカテゴリー。

14.2.1. アクセスログの有効化

Hot Rod および REST エンドポイントアクセスメッセージを記録するには、**log4j2.xml** でロギングカテゴリーを有効にする必要があります。

手順

1. 任意のテキストエディターで **$RHDG_HOME/server/conf/log4j2.xml** を開きます。
2. **org.infinispan.HOTROD_ACCESS_LOG** および **org.infinispan.REST_ACCESS_LOG** のロギングカテゴリーのレベルを **TRACE** に変更します。
3. **log4j2.xml** を保存し、閉じます。

```
<Logger name="org.infinispan.HOTROD_ACCESS_LOG" additivity="false" level="TRACE">
  <AppenderRef ref="HR-ACCESS-FILE"/>
</Logger>
```

14.2.2. アクセスログプロパティー

アクセスログのデフォルト形式は以下のとおりです。

```
%X{address} %X{user} [%d{dd/MMM/yyyy:HH:mm:ss Z}] &amp;quot;%X{method} %m
%X{protocol}&amp;quot; %X{status} %X{requestSize} %X{responseSize} %X{duration}%n
```

前述のフォーマットは、以下のようなログエントリーを作成します。

127.0.0.1 - [DD/MM/YYYY:HH:MM:SS +0000] "PUT /rest/v2/caches/default/key HTTP/1.1" 404 5 77 10

ロギングプロパティーは **%X{name}** 表記を使用し、アクセスログの形式を変更可能にします。以下は、デフォルトのロギングプロパティーです。

プロパティー	説明
address	**X-Forwarded-For** ヘッダーまたはクライアント IP アドレスのいずれか。
user	認証を使用する場合のプリンシパル名。
メソッド	使用されるメソッド。**PUT**、**GET** など。
protocol	使用されるプロトコル**HTTP/1.1**、**HTTP/2**、**HOTROD/2.9** など。
status	REST エンドポイントの HTTP ステータスコード。**OK** または Hot Rod エンドポイントの例外。
requestSize	リクエストのサイズ (バイト単位)。
responseSize	応答のサイズ (バイト単位)。
duration	サーバーによる要求の処理にかかった時間 (ミリ秒数)。

ヒント

h: で始まるヘッダー名を使用して、リクエストに含まれるヘッダーをログに記録します (例: **%X{h:User-Agent}**)。

14.3. 監査ログ

監査ログを使用すると、Data Grid Server デプロイメントへの変更を追跡できるため、変更がいつ発生し、どのユーザーが変更を加えたかを知ることができます。監査ロギングを有効にして、サーバー設定イベントと管理操作を記録するように設定します。

org.infinispan.AUDIT

セキュリティー監査メッセージを **$RHDG_HOME/server/log** ディレクトリーの **audit.log** ファイルに書き込むロギングカテゴリー。

14.3.1. 監査ログの有効化

セキュリティー監査メッセージを記録するには、**log4j2.xml** でロギングカテゴリーを有効にする必要があります。

手順

1. 任意のテキストエディターで **$RHDG_HOME/server/conf/log4j2.xml** を開きます。
2. **org.infinispan.AUDIT** のロギングカテゴリーのレベルを **INFO** に変更します。
3. **log4j2.xml** を保存し、閉じます。

```
<!-- Set to INFO to enable audit logging -->
<Logger name="org.infinispan.AUDIT" additivity="false" level="INFO">
  <AppenderRef ref="AUDIT-FILE"/>
</Logger>
```

14.3.2. 監査ロギングアペンダーの設定

Apache Log4j は、監査メッセージをデフォルトのログファイル以外の宛先に送信するのに使用するさまざまなアペンダーを提供します。たとえば、監査ログを syslog デーモン、JDBC データベース、または Apache Kafka サーバーに送信する場合は、**log4j2.xml** にアペンダーを設定できます。

手順

1. 任意のテキストエディターで **$RHDG_HOME/server/conf/log4j2.xml** を開きます。
2. デフォルトの **AUDIT-FILE** ローリングファイルアペンダーをコメント化するか削除します。

```
<!--RollingFile name="AUDIT-FILE"
  ...
</RollingFile-->
```

3. 監査メッセージに必要なロギングアペンダーを追加します。
   たとえば、以下のように Kafka サーバーのロギングアペンダーを追加できます。

```
<Kafka name="AUDIT-KAFKA" topic="audit">
  <PatternLayout pattern="%date %message"/>
  <Property name="bootstrap.servers">localhost:9092</Property>
</Kafka>
```

4. **log4j2.xml** を保存し、閉じます。

関連情報

- Log4j Appenders

14.3.3. カスタム監査ロギング実装の使用

Log4j アペンダーの設定がニーズを満たさない場合は、**org.infinispan.security.AuditLogger** API のカスタム実装を作成できます。

前提条件

- 必要に応じて **org.infinispan.security.AuditLogger** を実装し、JAR ファイルにパッケージ化します。

手順

1. JAR を Data Grid Server インストールの **server/lib** ディレクトリーに追加します。
2. カスタム監査ロガーの完全修飾クラス名を、キャッシュコンテナーのセキュリティー設定の **authorization** 要素の **audit-logger** 属性の値として指定します。
   たとえば、以下の設定では、ロギング監査メッセージのクラスとして **my.package.CustomAuditLogger** を定義します。

```
<infinispan>
  <cache-container>
    <security>
      <authorization audit-logger="my.package.CustomAuditLogger"/>
    </security>
  </cache-container>
</infinispan>
```

関連情報

- **org.infinispan.security.AuditLogger**

第15章 DATA GRID SERVER クラスターのローリングアップグレードの実行

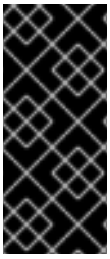
Data Grid クラスターのローリングアップグレードを実行して、ダウンタイムやデータの損失なしにバージョン間で変更し、Hot Rod プロトコルを介してデータを移行します。

15.1. ターゲット DATA GRID クラスターの設定

アップグレードする予定の Data Grid バージョンを使用するクラスターを作成してから、リモートキャッシュストアを使用してソースクラスターをターゲットクラスターに接続します。

前提条件

- ターゲットクラスターに必要なバージョンの Data Grid Server ノードをインストールします。



重要

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

手順

1. ターゲットクラスターがソースクラスターに接続できるリモートキャッシュストア設定を JSON 形式で作成します。
   ターゲットクラスターのリモートキャッシュストアは、Hot Rod プロトコルを使用して、ソースクラスターからデータを取得します。

```
{
  "remote-store": {
    "cache": "myCache",
    "shared": true,
    "raw-values": true,
    "security": {
      "authentication": {
        "digest": {
          "username": "username",
          "password": "changeme",
          "realm": "default"
        }
      }
    },
    "remote-server": [
      {
        "host": "127.0.0.1",
        "port": 12222
      }
    ]
  }
}
```

2. Data Grid コマンドラインインターフェイス (CLI) または REST API を使用して、リモートキャッシュストア設定をターゲットクラスターに追加し、ソースクラスターに接続できるようにします。

   - CLI: ターゲットクラスターで **migrate cluster connect** コマンドを使用します。

     ```
     [//containers/default]> migrate cluster connect -c myCache --file=remote-store.json
     ```

   - REST API: **rolling-upgrade/source-connection** メソッドを使用して、ペイロードにリモートストア設定が含まれる POST リクエストを呼び出します。

     ```
     POST /v2/caches/myCache/rolling-upgrade/source-connection
     ```

3. 移行するキャッシュごとに直前の手順を繰り返します。

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

   a. クライアント設定をターゲットクラスターの場所で更新します。

   b. クライアントを再起動します。

関連情報

- リモートキャッシュストア設定スキーマ

15.2. ターゲットクラスターへのデータの同期

ターゲット Data Grid クラスターを設定してソースクラスターに接続する場合、ターゲットクラスターはリモートキャッシュストアを使用してクライアント要求を処理し、オンデマンドでデータをロードできます。データをターゲットクラスターに完全に移行して、ソースクラスターの使用を停止できるようにするには、データを同期します。この操作はソースクラスターからデータを読み取り、ターゲットクラスターに書き込みます。データは、ターゲットクラスターのすべてのノードに並行して移行され、各ノードはデータのサブセットを受け取ります。ターゲットクラスターに移行する各キャッシュの同期を実行する必要があります。

前提条件

- 適切な Data Grid バージョンでターゲットクラスターを設定している。

手順

1. ターゲットクラスターに移行する各キャッシュと Data Grid コマンドラインインターフェイス (CLI) または REST API との同期を開始します。

   - CLI: **migrate cluster synchronize** コマンドを使用します。

     ```
     migrate cluster synchronize -c myCache
     ```

   - REST API: POST リクエストと共に **?action=sync-data** パラメーターを使用します。

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

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

2. ターゲットクラスター内の各ノードをソースクラスターから切断します。

- CLI: **migrate cluster disconnect** コマンドを使用します。

```
migrate cluster disconnect -c myCache
```

- REST API: DELETE リクエストを呼び出します。

```
DELETE /v2/caches/myCache/rolling-upgrade/source-connection
```

次のステップ

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

第16章 DATA GRID SERVER のデプロイメントのトラブルシューティング

Data Grid Server デプロイメントに関する診断情報を収集し、問題を解決するためのトラブルシューティング手順を実行します。

16.1. DATA GRID SERVER からの診断レポートの取得

Data Grid Server は、サーバーインスタンスおよびホストシステムに関する診断情報が含まれる **tar.gz** アーカイブで集計レポートを提供します。レポートは、設定ファイルやログファイルに加えて、CPU、メモリー、オープンファイル、ネットワークソケットとルーティング、スレッドに関する詳細を提供します。

手順

1. Data Grid Server への CLI 接続を作成します。
2. **server report** コマンドを使用して **tar.gz** アーカイブをダウンロードします。

```
server report
Downloaded report 'infinispan-<hostname>-<timestamp>-report.tar.gz'
```

以下の例のように、コマンドはレポートの名前で応答します。

```
Downloaded report 'infinispan-<hostname>-<timestamp>-report.tar.gz'
```

3. **tar.gz** ファイルをファイルシステムの適切な場所に移動します。
4. 任意のアーカイブツールで **tar.gz** ファイルをデプロイメントします。

16.2. ランタイム時に DATA GRID SERVER のロギング設定の変更

ランタイム時に Data Grid Server のロギング設定を変更し、問題のトラブルシューティングと根本原因分析を実行するためにロギングを一時的に調整します。

CLI を使用したロギング設定の変更は、ランタイム時のみの操作であるため、変更は以下のようになります。

- **log4j2.xml** ファイルに保存されません。サーバーノードまたはクラスター全体を再起動すると、ログ設定が **log4j2.xml** ファイルのデフォルトのプロパティーにリセットされます。
- CLI の呼び出し時に、クラスター内のノードにのみ適用されます。ログ設定を変更した後にクラスターに参加するノードは、デフォルトのプロパティーを使用します。

手順

1. Data Grid Server への CLI 接続を作成します。
2. ロギング を使用して、必要な調整を行います。
   - サーバーで定義されているすべてのアペンダーをリスト表示します。

```
logging list-appenders
```

このコマンドは、以下のような JSON 応答を提供します。

```
{
  "STDOUT" : {
    "name" : "STDOUT"
  },
  "JSON-FILE" : {
    "name" : "JSON-FILE"
  },
  "HR-ACCESS-FILE" : {
    "name" : "HR-ACCESS-FILE"
  },
  "FILE" : {
    "name" : "FILE"
  },
  "REST-ACCESS-FILE" : {
    "name" : "REST-ACCESS-FILE"
  }
}
```

- サーバーで定義されているすべてのロガー設定をリスト表示します。

```
logging list-loggers
```

このコマンドは、以下のような JSON 応答を提供します。

```
[ {
  "name" : "",
  "level" : "INFO",
  "appenders" : [ "STDOUT", "FILE" ]
}, {
  "name" : "org.infinispan.HOTROD_ACCESS_LOG",
  "level" : "INFO",
  "appenders" : [ "HR-ACCESS-FILE" ]
}, {
  "name" : "com.arjuna",
  "level" : "WARN",
  "appenders" : [ ]
}, {
  "name" : "org.infinispan.REST_ACCESS_LOG",
  "level" : "INFO",
  "appenders" : [ "REST-ACCESS-FILE" ]
} ]
```

- **set** サブコマンドを使用して、ロガー設定を追加および変更します。
たとえば、以下のコマンドは、**org.infinispan** パッケージのロギングレベルを **DEBUG** に設定します。

```
logging set --level=DEBUG org.infinispan
```

- **remove** サブコマンドを使用して、既存のロガー設定を削除します。
たとえば、以下のコマンドは **org.infinispan** ロガー設定を削除します。これは、代わりに root 設定が使用されることを意味します。

```
logging remove org.infinispan
```

16.3. CLI からのリソース統計の収集

stats コマンドを使用して、一部の Data Grid Server リソースについてサーバーコレクション化された統計を検査できます。

統計を提供するリソース (コンテナー、キャッシュ) のコンテキストから、またはそのようなリソースへのパスを使用して、**stats** コマンドを使用します。

```
stats
```

```
{
  "statistics_enabled" : true,
  "number_of_entries" : 0,
  "hit_ratio" : 0.0,
  "read_write_ratio" : 0.0,
  "time_since_start" : 0,
  "time_since_reset" : 49,
  "current_number_of_entries" : 0,
  "current_number_of_entries_in_memory" : 0,
  "total_number_of_entries" : 0,
  "off_heap_memory_used" : 0,
  "data_memory_used" : 0,
  "stores" : 0,
  "retrievals" : 0,
  "hits" : 0,
  "misses" : 0,
  "remove_hits" : 0,
  "remove_misses" : 0,
  "evictions" : 0,
  "average_read_time" : 0,
  "average_read_time_nanos" : 0,
  "average_write_time" : 0,
  "average_write_time_nanos" : 0,
  "average_remove_time" : 0,
  "average_remove_time_nanos" : 0,
  "required_minimum_number_of_nodes" : -1
}
```

```
stats /containers/default/caches/mycache
```

```
{
  "time_since_start" : -1,
  "time_since_reset" : -1,
  "current_number_of_entries" : -1,
  "current_number_of_entries_in_memory" : -1,
  "total_number_of_entries" : -1,
  "off_heap_memory_used" : -1,
  "data_memory_used" : -1,
  "stores" : -1,
  "retrievals" : -1,
  "hits" : -1,
```

```
  "misses" : -1,
  "remove_hits" : -1,
  "remove_misses" : -1,
  "evictions" : -1,
  "average_read_time" : -1,
  "average_read_time_nanos" : -1,
  "average_write_time" : -1,
  "average_write_time_nanos" : -1,
  "average_remove_time" : -1,
  "average_remove_time_nanos" : -1,
  "required_minimum_number_of_nodes" : -1
}
```

16.4. REST 経由でのクラスターの正常性へのアクセス

REST API 経由で Data Grid クラスターの正常性を取得します。

手順

- **GET** 要求を呼び出して、クラスターの正常性を取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/health
```

Data Grid は、以下のような **JSON** ドキュメントで応答します。

```
{
   "cluster_health":{
      "cluster_name":"ISPN",
      "health_status":"HEALTHY",
      "number_of_nodes":2,
      "node_names":[
         "NodeA-36229",
         "NodeB-28703"
      ]
   },
   "cache_health":[
      {
         "status":"HEALTHY",
         "cache_name":"___protobuf_metadata"
      },
      {
         "status":"HEALTHY",
         "cache_name":"cache2"
      },
      {
         "status":"HEALTHY",
         "cache_name":"mycache"
      },
      {
         "status":"HEALTHY",
         "cache_name":"cache1"
      }
   ]
}
```

ヒント

以下のように Cache Manager のステータスを取得します。

```
GET /rest/v2/cache-managers/{cacheManagerName}/health/status
```

参照資料

詳細は、**REST v2 (version 2) API**ドキュメントを参照してください。

16.5. JMX 経由でクラスターの正常性へのアクセス

JMX 経由で Data Grid クラスターの正常性統計を取得します。

手順

1. JConsole などの JMX 対応ツールを使用して Data Grid Server に接続し、以下のオブジェクトに移動します。

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

2. 利用可能な MBean を選択し、クラスターの正常性の統計を取得します。

第17章 参照

17.1. DATA GRID SERVER 8.3.1 README

Data Grid Server 13.0.10.Final-redhat-00001 ディストリビューションに関する情報。

17.1.1. 要件

Data Grid Server には、JDK 11 以降が必要です。

17.1.2. サーバーの起動

server スクリプトを使用して Data Grid Server インスタンスを実行します。

Unix / Linux

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

Windows

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

ヒント

コマンド引数を表示するには **--help** または **-h** オプションを追加します。

17.1.3. サーバーの停止

CLI で **shutdown** コマンドを使用して、正常なシャットダウンを実行します。

または、ターミナルから Ctrl-C を入力してサーバープロセスを中断するか、TERM シグナルを介してこれを強制終了します。

17.1.4. 設定

サーバー設定によって、以下のサーバー固有の要素で Data Grid 設定が拡張されます。

cache-container
: キャッシュライフサイクルを管理するためのキャッシュコンテナーを定義します。

endpoints
: クライアントプロトコルのエンドポイントコネクターを有効化および設定します。

security
: エンドポイントセキュリティーレルムを設定します。

socket-bindings
: エンドポイントコネクターをインターフェイスおよびポートにマッピングします。

デフォルトの設定ファイルは **$RHDG_HOME/server/conf/infinispan.xml** です。

クラスタリング機能なしでサーバーを起動する以下の例のように、**-c** 引数を指定してさまざまな設定ファイルを使用します。

Unix / Linux

```
$RHDG_HOME/bin/server.sh -c infinispan-local.xml
```

Windows

```
$RHDG_HOME\bin\server.bat -c infinispan-local.xml
```

17.1.5. バインドアドレス

Data Grid Server は、デフォルトでネットワーク上のループバック IP アドレス **localhost** にバインドします。

すべてのネットワークインターフェイスにバインドする以下の例のように、**-b** 引数を使用して別の IP アドレスを設定します。

Unix / Linux

```
$RHDG_HOME/bin/server.sh -b 0.0.0.0
```

Windows

```
$RHDG_HOME\bin\server.bat -b 0.0.0.0
```

17.1.6. バインドポート

Data Grid Server は、デフォルトでポート **11222** をリッスンします。

-p 引数を使用して別のポートを設定します。

Unix / Linux

```
$RHDG_HOME/bin/server.sh -p 30000
```

Windows

```
$RHDG_HOME\bin\server.bat -p 30000
```

17.1.7. クラスタリングアドレス

Data Grid Server 設定では、クラスタートランスポートが定義されているため、同じネットワーク上の複数のインスタンスが相互に検出し、自動的にクラスターを形成します。

-k 引数を使用して、クラスタートラフィックの IP アドレスを変更します。

Unix / Linux

```
$RHDG_HOME/bin/server.sh -k 192.168.1.100
```

Windows

```
$RHDG_HOME\bin\server.bat -k 192.168.1.100
```

17.1.8. クラスタースタック

JGroups スタックは、クラスタートランスポートのプロトコルを設定します。Data Grid Server は、デフォルトで **tcp** スタックを使用します。

クラスタートランスポートに UDP を使用する以下の例のように、**-j** 引数を指定して代替クラススタックを使用します。

Unix / Linux

```
$RHDG_HOME/bin/server.sh -j udp
```

Windows

```
$RHDG_HOME\bin\server.bat -j udp
```

17.1.9. 認証

Data Grid Server には認証が必要です。

以下のように、CLI を使用してユーザー名およびパスワードを作成します。

Unix / Linux

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

Windows

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

17.1.10. サーバーのホームディレクトリー

Data Grid Server は **infinispan.server.home.path** を使用して、ホストファイルシステム上のサーバーディストリビューションのコンテンツを見つけます。

$RHDG_HOME と呼ばれるサーバーのホームディレクトリーには、以下のフォルダーが含まれます。

```
├── bin
├── boot
├── docs
├── lib
├── server
└── static
```

フォルダー	説明
/bin	サーバーおよび CLI を起動するスクリプトが含まれています。

フォルダー	説明
/boot	サーバーを起動するための **JAR** ファイルが含まれます。
/docs	設定例、スキーマ、コンポーネントライセンス、およびその他のリソースを提供します。
/lib	サーバーが内部で要求する **JAR** ファイルが含まれます。 カスタム **JAR** ファイルはこのフォルダーに配置しないでください。
/server	Data Grid Server インスタンスの root フォルダーを提供します。
/static	Data Grid コンソールの静的リソースが含まれています。

17.1.11. サーバー root ディレクトリー

Data Grid Server は **infinispan.server.root.path** を使用して、Data Grid Server インスタンスの設定ファイルおよびデータを見つけます。

同じディレクトリーまたは別のディレクトリーに複数のサーバー root フォルダーを作成してから、以下の例に示すように **-s** または **--server-root** 引数を使用して場所を指定できます。

Unix / Linux

```
$RHDG_HOME/bin/server.sh -s server2
```

Windows

```
$RHDG_HOME\bin\server.bat -s server2
```

各サーバー roor ディレクトリーには、以下のフォルダーが含まれます。

```
├── server
│   ├── conf
│   ├── data
│   ├── lib
│   └── log
```

フォルダー	説明	システムプロパティーの上書き
/server/conf	サーバー設定ファイルが含まれています。	**infinispan.server.config.path**

フォルダー	説明	システムプロパティーの上書き
/server/data	コンテナー名別に整理されたデータファイルが含まれます。	**infinispan.server.data.path**
/server/lib	サーバー拡張ファイルが含まれます。 このディレクトリーは再帰的にスキャンされ、クラスパスとして使用されます。	**infinispan.server.lib.path** 複数のパスを以下の区切り文字で区切ります: **:** (Unix / Linux) **;** (Windows)
/server/log	サーバーのログファイルが含まれます。	**infinispan.server.log.path**

17.1.12. ロギング

server/conf フォルダーの **log4j2.xml** ファイルを使用して、Data Grid Server のロギングを設定します。

以下のように **--logging-config=<path_to_logfile>** 引数を使用してカスタムパスを使用します。

Unix / Linux

```
$RHDG_HOME/bin/server.sh --logging-config=/path/to/log4j2.xml
```

ヒント

カスタムパスを確実に有効にするには、~ ショートカットを使用しないでください。

Windows

```
$RHDG_HOME\bin\server.bat --logging-config=path\to\log4j2.xml
```