Red Hat Data Grid 8.4

Data Grid Server 指南

部署、保护和管理 Data Grid 服务器部署

Last Updated: 2024-05-02

Red Hat Data Grid 8.4 Data Grid Server 指南

部署、保护和管理 Data Grid 服务器部署

法律通告

摘要

安装和配置数据网格服务器部署。

目录

RED HAT DATA GRID

数据网格是高性能分布式内存数据存储。

Schemaless 数据结构

灵活性以将不同对象存储为键值对。

基于网格的数据存储

旨在在集群中分发和复制数据。

弹性扩展

动态调整节点数量，以在不中断服务的情况下满足需求。

数据互操作性

从不同端点在网格中存储、检索和查询数据。

DATA GRID 文档

红帽客户门户网站中提供了数据网格的文档。

- Data Grid 8.4 文档
- Data Grid 8.4 组件详情
- Data Grid 8.4 支持的配置
- Data Grid 8 功能支持
- Data Grid 已弃用功能和功能

DATA GRID 下载

访问红帽客户门户网站中的 Data Grid 软件下载。

注意

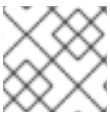
您必须有一个红帽帐户才能访问和下载 Data Grid 软件。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

对红帽文档提供反馈

我们非常感谢您对我们的技术内容提供反馈，并鼓励您告诉我们您的想法。如果您想添加评论，提供见解、纠正拼写错误甚至询问问题，您可以在文档中直接这样做。



注意

您必须有一个红帽帐户并登录到客户门户网站。

要从客户门户网站提交文档反馈，请执行以下操作：

1. 选择 **Multi-page HTML** 格式。
2. 点文档右上角的 **反馈** 按钮。
3. 突出显示您要提供反馈的文本部分。
4. 点高亮文本旁的**添加反馈**对话框。
5. 在页面右侧的文本框中输入您的反馈，然后单击 **Submit**。

每次提交反馈时，我们都会自动创建跟踪问题。打开在点 **Submit** 后显示的链接，并开始监视问题或添加更多注释。

感谢您的宝贵反馈。

第 1 章 DATA GRID SERVER 入门

安装服务器分发，创建一个用户，并启动您的第一个数据网格集群。

Ansible 集合

使用我们的 Ansible 集合自动安装 Data Grid 集群，该集合包括 Keycloak 缓存和跨站点复制配置。Ansible 集合还可让您在安装过程中将 Data Grid 缓存注入每个服务器实例的静态配置中。

Red Hat **Automation Hub** 提供了 Data Grid 的 Ansible 集合 。

1.1. DATA GRID SERVER 要求

数据网格服务器需要 Java 虚拟机。有关支持的版本的详情，请查看 Data Grid 支持的配置。

1.2. 下载 DATA GRID 服务器发行版本

Data Grid Server 发行版是 Java 库(**JAR** 文件)和配置文件的存档。

流程

1. 访问红帽客户门户。
2. 从 软件下载一节 下载 Red Hat Data Grid 8.4 服务器。
3. 使用服务器下载存档作为参数运行 **md5sum** 或 **sha256sum** 命令，例如：

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

4. 与 Data Grid **Software Details** 页面中的 **MD5** 或 **SHA-256** checksum 值进行比较。

参考

- Data Grid Server README 描述服务器分发的内容。

1.3. 安装 DATA GRID SERVER

在主机系统上安装 Data Grid 服务器分发。

先决条件

- 下载数据网格服务器分发存档。

流程

- 使用任何适当的工具将 Data Grid 服务器存档提取到主机文件系统。

```
unzip redhat-datagrid-8.4.1-server.zip
```

生成的目录是您 **$RHDG_HOME**。

1.4. 启动 DATA GRID SERVER

在任何受支持的主机上的 Java 虚拟机(JVM)中运行数据网格服务器实例。

先决条件

- 下载并安装服务器分发。

流程

1. 在 **$RHDG_HOME** 中打开一个终端。
2. 使用 **服务器** 脚本启动 Data Grid 服务器实例。

   Linux

   ```
   bin/server.sh
   ```

   Microsoft Windows

   ```
   bin\server.bat
   ```

Data Grid Server 在记录以下信息时成功运行：

```
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
```

验证

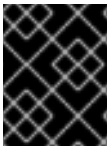
1. 在任意浏览器中打开 **127.0.0.1:11222/console/**。
2. 在提示符处输入您的凭证并继续到 Data Grid Console。

1.5. 在启动时传递 DATA GRID SERVER 配置

启动 Data Grid Server 时指定自定义配置。

数据网格服务器可以使用 **--server-config** 参数在启动时解析您覆盖的多个配置文件。您可以根据需要，以任何顺序使用任意数量的配置覆盖文件。配置覆盖文件：

- 必须是有效的 Data Grid 配置，并包含根服务器 **元素** 或字段。
- 只要您的覆盖文件的组合会导致完整的配置，则无需进行完整配置。



重要

数据网格服务器不会检测覆盖文件之间的冲突配置。每个覆盖文件覆盖上述配置中任何冲突的配置。

注意

如果在启动时将缓存配置传递给 Data Grid Server，则它不会在集群中动态创建这些缓存。您必须手动将缓存传播到每个节点。

另外，在启动时传递给 Data Grid Server 的缓存配置必须包含 **infinispan** 和 **cache-container** 元素。

先决条件

- 下载并安装服务器分发。
- 将自定义服务器配置添加到 Data Grid Server 安装的 **server/conf** 目录中。

流程

1. 在 **$RHDG_HOME** 中打开一个终端。
2. 使用 **--server-config=** 或 **-c** 参数指定一个或多个配置文件，例如：

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

1.6. 创建 DATA GRID 用户

添加凭证以通过 Hot Rod 和 REST 端点与数据网格服务器部署进行身份验证。在访问数据网格控制台或执行缓存操作前，您必须使用 Data Grid 命令行界面(CLI)创建至少一个用户。

提示

数据网格通过基于角色的访问控制(RBAC)强制实施安全授权。在第一次添加凭证时创建一个 **管理员用户**，为您的 Data Grid 部署获取完整的 **ADMIN** 权限。

先决条件

- 下载并安装 Data Grid 服务器。

流程

1. 在 **$RHDG_HOME** 中打开一个终端。
2. 使用 **user create** 命令创建 **admin** 用户。

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

提示

从 CLI 会话运行 **help 用户** 以获取完整的命令详情。

验证

打开 **user.properties** 并确认用户存在。

```
cat server/conf/users.properties

admin=scram-sha-1\:BYGcIAwvf6b...
```

注意

使用 CLI 将凭证添加到 properties realm 中，只能在您连接的服务器实例上创建用户。您必须手动将 properties 域中的凭证同步到集群中的每个节点。

1.6.1. 授予用户角色

为用户分配角色，并授予他们执行缓存操作并与 Data Grid 资源交互的权限。

提示

如果要为多个用户分配同一角色并集中维护其权限，将角色授予组而不是用户。

先决条件

- 为 Data Grid 具有 **ADMIN** 权限。
- 创建 Data Grid 用户。

流程

1. 创建与 Data Grid 的 CLI 连接。
2. 使用 **user roles grant** 命令为用户分配角色，例如：

```
user roles grant --roles=deployer katie
```

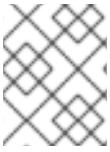
验证

使用用户角色 **ls 命令列出您授予用户的角色**。

```
user roles ls katie
["deployer"]
```

1.6.2. 将用户添加到组中

组允许您更改多个用户的权限。您可以为组分配角色，然后将用户添加到该组中。用户从组角色继承权限。



注意

您可以在 Data Grid 服务器配置中将组用作属性域的一部分。每个组都是特殊的用户，还需要用户名和密码。

先决条件

- 为 Data Grid 具有 **ADMIN** 权限。
- 创建 Data Grid 用户。

流程

1. 创建与 Data Grid 的 CLI 连接。
2. 使用 **user create** 命令创建组。
   a. 使用 **--groups** 参数指定组名称。
   b. 为组设置用户名和密码。

```
user create --groups=developers developers -p changeme
```

3. 列出组。

```
user ls --groups
```

4. 为组授予角色。

```
user roles grant --roles=application developers
```

5. 列出组的角色。

```
user roles ls developers
```

6. 一次将用户添加到组中。

```
user groups john --groups=developers
```

验证

打开 **groups.properties** 并确认组存在。

```
cat server/conf/groups.properties
```

1.6.3. Data Grid 用户角色和权限

数据网格包括多个角色，为用户提供访问缓存和数据网格资源的权限。

角色	权限	Description
admin	ALL	具有所有权限的超级用户，包括缓存管理器生命周期的控制。
deployer	ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR, CREATE	除了 **应用程序** 权限外，还可创建和删除数据网格资源。
application	ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR	除 **观察者** 权限之外，还具有对 Data Grid 资源的读写访问权限。还可以侦听事件并执行服务器任务和脚本。

角色	权限	Description
observer	ALL_READ, MONITOR	除了监控权限外，还具有对数据网格 **资源** 的读取访问权限。
monitor	MONITOR	可以通过 JMX 和 **指标端点** 查看统计信息。

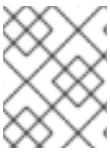
其他资源

- org.infinispan.security.AuthorizationPermission Enum
- Data Grid 配置模式参考

1.7. 验证集群视图

同一网络上的 Data Grid Server 实例可自动发现彼此的群集和表单集群。

完成这个步骤，通过本地运行 Data Grid Server 实例，使用默认 **TCP** 堆栈中的 **MPING** 协议观察集群发现。如果要针对自定义网络要求调整集群传输，请参阅有关设置 Data Grid 集群的文档。



注意

此流程旨在展示集群发现的原则，它不用于生产环境。在命令行中指定端口偏移的操作不是为生产配置集群传输的可靠方法。

先决条件

拥有一个运行数据网格服务器的实例。

流程

1. 在 **$RHDG_HOME** 中打开一个终端。
2. 将根目录复制到 **server2**。

```
cp -r server server2
```

3. 指定端口偏移和 **server2** 目录。

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

验证

您可以在控制台中的群集成员身份为 **127.0.0.1:11222/console/cluster-membership**。

数据网格还在节点加入集群时记录以下信息：

```
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. 关闭 DATA GRID 服务器

停止单独运行的服务器或正常关闭集群。

流程

1. 创建与 Data Grid 的 CLI 连接。
2. 通过以下方法关闭 Data Grid Server:
   - 使用关闭集群命令 **停止集群中的所有节点**，例如：

     ```
     shutdown cluster
     ```

     此命令将集群状态保存到集群中每个节点的 **data** 文件夹。如果使用缓存存储，则 **关闭的集群** 命令也会保留缓存中的所有数据。

   - 使用 **shutdown server** 命令和服务器主机名停止各个服务器实例，例如：

     ```
     shutdown server <my_server01>
     ```

重要

shutdown server 命令不会等待重新平衡操作完成，如果您同时指定多个主机名，则可能会导致数据丢失。

提示

运行 **help shutdown** 以了解有关使用 命令的更多详细信息。

验证

Data Grid 在关闭服务器时记录以下信息：

```
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 集群重启

当您在关闭数据网格集群后重新上线时，您应该在添加或删除节点或修改集群状态前等待集群可用。

如果使用 **关闭 server** 命令关闭集群节点，您必须按顺序重启每个服务器。
例如，如果您关闭 **server1**，然后关闭 **server2**，您应首先启动 **server2**，然后启动 **server1**。

如果使用关闭集群命令关闭集群，**集群** 只有在所有节点重新加入后才能够全面运行。
您可以按任何顺序重启节点，但集群会处于 DEGRADED 状态，直到所有在关闭前加入的节点都处于运行状态。

1.9. DATA GRID SERVER 安装目录结构

Data Grid Server 在主机文件系统上使用以下文件夹 **$RHDG_HOME** ：

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

提示

如需了解 **$RHDG_HOME** 目录中每个文件夹的描述以及可用于自定义文件系统的系统属性，请参阅 Data Grid Server README。

1.9.1. 服务器根目录

除了 **bin** 和 **docs** 文件夹中的资源外，您应与 的 **$RHDG_HOME** 下的唯一文件夹是服务器根目录，默认命名为 **server**。

您可以在同一 **$RHDG_HOME** 目录或不同的目录中创建多个节点，但每个 Data Grid Server 实例都必须有自己的服务器根目录。例如，文件系统中的 5 个节点集群可能有以下服务器根目录：

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

每个服务器根目录都应该包含以下文件夹：

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

server/conf
存放 Data Grid Server 实例的 **infinispan.xml** 配置文件。

Data Grid 将配置分成两层：

动态

为数据可扩展性创建可变的缓存配置。
Data Grid Server 永久保存您在运行时创建的缓存，以及跨节点分布的集群状态。每个加入节点都会获得一个完整的集群状态，Data Grid Server 在每次更改时在所有节点中同步。

Static

为底层服务器机制（如集群传输、安全性和共享数据源）添加配置 **infinispan.xml**。

server/data
提供 Data Grid 服务器用来维护群集状态的内部存储。

重要

切勿直接删除或修改 **服务器/数据** 中的内容。

在服务器运行时修改 **cache.xml** 等文件可能会导致崩溃。删除内容可能会导致状态不正确，这意味着集群在关闭后无法重启。

server/lib
包含用于自定义过滤器、自定义事件监听程序、JDBC 驱动程序、自定义 **服务器Task** 实施等的扩展 **JAR** 文件。

server/log
保管 Data Grid Server 日志文件。

其他资源

- Data Grid Server README
- RHDG 服务器使用的 <server>/data 目录中 （红帽知识库）

第 2 章 网络接口和套接字绑定

通过将 Data Grid Server 绑定到 IP 地址，通过网络接口公开 Data Grid Server。然后，您可以将端点配置为使用接口，以便 Data Grid Server 可以处理来自远程客户端应用程序的请求。

2.1. 网络接口

Data Grid Server 将多个端点用于单个 TCP/IP 端口，并自动检测入站客户端请求的协议。您可以配置数据网格服务器如何绑定到网络接口以侦听客户端请求。

互联网协议(IP)地址

XML

```xml
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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"
```

环回地址

XML

```xml
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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: ~
```

非环回地址

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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: ~
```

任何地址

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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: ~
```

链**接本地**

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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: ~
```

站点本地

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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. 匹配和回退策略

数据网格服务器可以枚举主机系统上的所有网络接口，并绑定到与值匹配的接口、主机或 IP 地址，这可能包含用于额外灵活性的正则表达式。

匹配主机

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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"
```

匹配接口

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <!--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"
```

匹配地址

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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\\..*"
```

回退

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <!-- 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. 套接字绑定

套接字绑定将端点连接器映射到网络接口和端口。默认情况下，Data Grid 服务器包含一个套接字绑定配置，侦听 localhost 接口 **127.0.0.1**，用于 REST 和 Hot Rod 端点的端口 **11222**。如果启用 Memcached 端点，则默认套接字绑定将 Data Grid Server 配置为绑定到端口 **11221**。

默认套接字绑定

```
<server xmlns="urn:infinispan:server:14.0">
  <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>
```

配置元素或属性	Description
socket-bindings	包含所有网络接口和端口的 root 元素，数据网格服务器端点可以绑定和侦听客户端连接。
default-interface	声明 Data Grid 服务器默认侦听的网络接口。
port-offset	指定 Data Grid Server 适用于套接字绑定的端口声明的偏移。
socket-binding	配置 Data Grid Server，以绑定到网络接口上的端口。

自定义套接字绑定声明

以下示例配置会将名为"private" **的接口** 声明添加名为"private"和将 Data Grid Server 绑定到私有 IP 地址的 **socket-binding** 声明：

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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. 更改 DATA GRID 服务器的绑定地址

数据网格服务器绑定到网络 IP 地址，以侦听 Hot Rod 和 REST 端点上的入站客户端连接。您可以在 Data Grid Server 配置中直接指定该 IP 地址，或者在启动服务器实例时直接指定该 IP 地址。

先决条件

- 至少具有一个数据网格服务器安装。

流程

使用以下方法之一指定 Data Grid Server bind 的 IP 地址：

- 打开 Data Grid Server 配置并设置 **inet-address** 元素的值，例如：

```
<server xmlns="urn:infinispan:server:14.0">
  <interfaces>
    <interface name="custom">
      <inet-address value="${infinispan.bind.address:192.0.2.0}"/>
    </interface>
  </interfaces>
</server>
```

- 使用 **-b** 选项或 **infinispan.bind.address** 系统属性。

  Linux

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

  Windows

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

2.3.1. 侦听所有地址

如果您将 **0.0.0.0** meta-address 或 **INADDR_ANY** 指定为 Data Grid 服务器配置中的绑定地址，它将侦听所有可用网络接口上的传入客户端连接。

客户端智能

将 Data Grid 配置为侦听其对所有地址的影响，影响其提供带有集群拓扑的 Hot Rod 客户端。如果有多个接口，Data Grid Server bind，它会为每个接口发送 IP 地址列表。

例如，每个服务器节点绑定到的集群：

- **10.0.0.0/8** 子网
- **192.168.0.0/16** subnet
- **127.0.0.1** 环回

热环客户端接收属于客户端连接接口的服务器节点的 IP 地址。例如，如果客户端连接到 **192.168.0.0**，它不会接收侦听 **10.0.0.0** 的节点的任何集群拓扑详情。

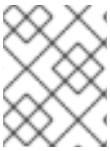
子网掩码覆盖

Kubernetes 以及其他一些环境，将 IP 地址空间划分为子网，并将这些子网用作单一网络。例如：**10.129.2.100/23** 和 **10.129.4.100/23** 位于不同的子网中，但属于 **10.0.0.0/8** 网络。

因此，Data Grid Server 覆盖了主机系统提供的子网掩码，子网掩码为私有和保留网络的 IANA 约定：

- IPv4: **10.0.0.0/8**,**100.64.0.0/10**, 192.168.0.0/16', **172.16.0.0**/ 12,**169.254.0.0/16** 和 **240.0.0.0/4**
- IPv6： **fc00::/7** 和 **fe80::/10**

请参阅 **RFC 1112**,**RFC 1918**,**RFC 3927**,**RFC 6598** 用于 IPv4 或 **RFC 4193**，对于 IPv6**RFC 3513**。



注意

您可以选择将 Hot Rod 连接器配置为使用主机系统为带有 Data Grid 服务器配置中的 **network-prefix-override** 属性提供的子网掩码。

其他资源

- Data Grid Server schema 参考
- RFC 1112
- RFC 1918
- RFC 3513
- RFC 3927
- RFC 4193
- RFC 6598

2.4. DATA GRID SERVER 端口和协议

Data Grid Server 提供网络端点，允许客户端访问不同的协议。

端口	协议	Description
11222	TCP	热环和 REST
11221	TCP	memcached （默认为禁用）

单个端口

数据网格服务器通过单个 TCP 端口 **11222** 公开多个协议。使用单一端口处理多个协议简化了配置，降低部署数据网格集群时的管理复杂性。使用单个端口还通过最小化网络上的攻击面来增强安全性。

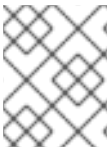
数据网格服务器以不同的方式通过单一端口处理来自客户端的 HTTP/1.1、HTTP/2 和 Hot Rod 协议请求。

HTTP/1.1 升级标头

客户端请求可以包含 **HTTP/1.1 upgrade** 标头字段来发起与 Data Grid Server 的 HTTP/1.1 连接。然后，客户端应用程序可以发送 **Upgrade: protocol** 标头字段，其中 **protocol** 是服务器端点。

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

客户端请求包括 Data Grid Server 端点的 Server Name Indication (SNI)映射，以便在 TLS 连接中协商协议。



注意

应用程序必须使用支持 ALPN 扩展的 TLS 库。数据网格将 WildFly OpenSSL 绑定用于 Java。

自动 Hot Rod 检测

包含 Hot Rod 标头的客户端请求会自动路由到 Hot Rod 端点。

2.4.1. 为 Data Grid 流量配置网络防火墙

调整防火墙规则，以允许 Data Grid 服务器和客户端应用程序之间的流量。

流程

例如，在 Red Hat Enterprise Linux (RHEL)工作站中，您可以使用 firewalld 来允许到端口 **11222** 的流量，如下所示：

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

要配置网络适用的防火墙规则，您可以使用 nftables 实用程序。

参考

- 使用和配置 firewalld
- nftables 入门

2.5. 指定端口偏移

在同一主机上为多个数据网格服务器实例配置端口偏移。默认端口偏移是 **0**。

流程

使用 **-o** 切换与 Data Grid CLI 或 **infinispan.socket.binding.port-offset** 系统属性设置端口偏移。

例如，启动偏移 **100** 的服务器实例，如下所示：使用默认配置，这将导致数据网格服务器侦听端口 **11322**。

Linux

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

Windows

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

第 3 章 DATA GRID SERVER 端点

Data Grid Server 端点通过 Hot Rod 和 REST 协议提供对缓存管理器的客户端访问。

3.1. DATA GRID SERVER 端点

3.1.1. hot Rod

hot Rod 是一个二进制 TCP 客户端服务器协议，与基于文本的协议相比，可以提供更快的数据访问并提高性能。

数据网格提供 Java、C++、C#、Node.js 和其他编程语言中的 Hot Rod 客户端库。

拓扑状态传输

数据网格使用拓扑缓存来为客户端提供集群视图。拓扑缓存包含将内部 JGroups 传输地址映射到公开的 Hot Rod 端点的条目。

当客户端发送请求时，Data Grid 服务器将请求标头中的拓扑 ID 与缓存中的拓扑 ID 进行比较。如果客户端有旧的拓扑 ID，Data Grid 服务器发送新的拓扑视图。

集群拓扑视图允许 Hot Rod 客户端立即检测节点何时加入和离开节点，这将启用动态负载平衡和故障转移。

在分布式缓存模式中，一致的哈希算法也可以将 Hot Rod 客户端请求直接路由到主所有者。

3.1.2. REST

数据网格提供了 RESTful 接口，允许 HTTP 客户端访问数据、监控和维护集群，以及执行管理操作。

您可以使用标准 HTTP 负载均衡器为客户端提供负载平衡和故障转移功能。但是，HTTP 负载均衡器维护静态集群视图，需要在集群拓扑更改时手动更新。

3.1.3. Memcached

数据网格为远程客户端访问提供 Memcached 文本协议的实现。

重要

Memcached 端点已弃用，计划在以后的发行版本中删除。

Data Grid Memcached 端点支持使用复制和分布式缓存模式的集群。

有一些 Memcached 客户端实现（如 Cache::Memcached Perl 客户端）可以使用 Data Grid 服务器地址静态列表提供负载平衡和故障转移检测功能，在进行集群拓扑更改时需要手动更新。

3.1.4. 端点协议的比较

	hot Rod	HTTP / REST
topology-aware	Y	N

	hot Rod	HTTP / REST
hash-aware	Y	N
Encryption	Y	Y
身份验证	Y	Y
条件 ops	Y	Y
批量运营	Y	N
Transactions	Y	N
监听器	Y	N
查询	Y	Y
执行	Y	N
跨站点故障切换	Y	N

3.1.5. 热 Rod 客户端与 Data Grid 服务器兼容

数据网格服务器允许您将 Hot Rod 客户端与不同的版本连接。对于在迁移或升级到数据网格集群期间的实例，Hot Rod 客户端版本可能比 Data Grid Server 更低。

提示

数据网格建议使用最新的 Hot Rod 客户端版本来受益于最新功能和安全增强。

Data Grid 8 及更新的版本

hot Rod 协议版本 3.x 自动协商与 Data Grid Server 的客户端可能的最高版本。

数据网格 7.3 及更早版本

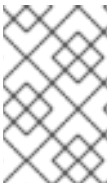
使用比 Data Grid Server 版本更大的 Hot Rod 协议版本的客户端必须设置 **infinispan.client.hotrod.protocol_version** 属性。

其他资源

- 热 Rod 协议参考
- 将 Hot Rod 客户端连接到具有不同版本的服务器 （红帽知识库）

3.2. 配置 DATA GRID SERVER 端点

控制 Hot Rod 和 REST 端点如何绑定到套接字并使用安全域配置。您还可以配置多个端点并禁用管理功能。



注意

每个唯一端点配置都必须包含 Hot Rod 连接器和 REST 连接器。数据网格服务器隐式地在 **端点配置** 中包括 **热 式连接器 和其余的连接器** 元素，或字段。您应该只将这些元素添加到自定义配置中，以指定端点的验证机制。

先决条件

- 在您的 Data Grid 服务器配置中添加套接字绑定和安全域。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 将多个 **端点配置** 嵌套到 **端点** 元素。
3. 指定端点与 **socket-binding** 属性使用的套接字绑定。
4. 指定端点与 **security-realm** 属性使用的安全域。
5. 如果需要，禁用 **admin="false"** 属性的管理员访问权限。
   使用这个配置用户无法通过端点访问 Data Grid Console 或 Command Line Interface (CLI)。
6. 保存对您的配置的更改。

多个端点配置

以下 Data Grid Server 配置会在单独的套接字绑定中与专用安全域创建端点：

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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
```

其他资源

- 网络接口和套接字绑定

3.3. 端点连接器

连接器配置 Hot Rod 和 REST 端点，以使用套接字绑定和安全域。

默认端点配置

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

配置元素或属性	Description
端点	将端点连接器配置嵌套。
端点	声明一个数据网格服务器端点，该端点配置 Hot Rod 和 REST 连接器以使用套接字绑定和安全域。
hotrod-connector	在端点配置中包含 Hot Rod **端点**。
rest-connector	在端点配置中包含 Hot Rod **端点**。
memcached-connector	配置 Memcached 端点，默认为禁用。

其他资源

- Data Grid 架构参考

3.4. 端点 IP 地址过滤规则

数据网格服务器端点可以使用过滤规则来控制客户端是否可以根据其 IP 地址连接。数据网格服务器按顺序应用过滤规则，直到找到客户端 IP 地址匹配为止。

CIDR 块是 IP 地址及其关联的网络掩码的紧凑表示。CIDR 表示法指定 IP 地址、斜杠('/')字符和十进制数。十进制数是网络掩码中前导 1 位的计数。数字也可看作网络前缀的宽度（以位为单位）。CIDR 标记中的 IP 地址始终根据 IPv4 或 IPv6 的标准表示。

该地址可以表示特定的接口地址，包括主机标识符，如 **10.0.0.1/8**，或者可以是整个网络接口范围的开头地址，使用主机标识符 0，如 **10.0.0.0/8** 或 **10/8**。

例如：

- **192.168.100.14/24** 代表 IPv4 地址 **192.168.100.14** 及其关联的网络前缀 **192.168.100.0**，或相当于其子网掩码 **255.255.255.0**，其子网掩码为 24 个前 1 位。
- IPv4 块 **192.168.100.0/22** 代表来自 **192.168.100.0** 到 **192.168.103.255** 的 1024 IPv4 地址。
- IPv6 块 **2001:db8::/48** 表示从 **2001:db8:0:0:0:0:0:0** 到 **2001:db8:0:ffff:ffff:ffff:ffff:**ffff:ffff:ffff 的 IPv6 地址块。
- **::1/128** 代表 IPv6 环回地址。其前缀长度为 128，这是地址中的位数。

IP 地址过滤器配置

在以下配置中，Data Grid Server 仅接受来自 **192.168.0.0/16** 和 **10.0.0.0/8** CIDR 块中的地址的连接。数据网格服务器拒绝所有其他连接。

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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

YAML

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

3.5. 检查和修改用于过滤 IP 地址的规则

在 Data Grid Server 端点上配置 IP 地址过滤规则，以根据客户端地址接受或拒绝连接。

先决条件

- 安装 Data Grid 命令行界面(CLI)。

流程

1. 创建与 Data Grid 服务器的 CLI 连接。
2. 根据需要检查并修改 IP 过滤规则 **服务器连接器 ipfilter** 命令。
   a. 列出集群中连接器上活跃的所有 IP 过滤规则：

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

   b. 为集群设置 IP 过滤规则。

   注意

   此命令替换任何现有的规则。

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

   c. 删除集群中连接器上的所有 IP 过滤规则。

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

第 4 章 端点验证机制

Data Grid 服务器可以为 Hot Rod 和 REST 端点使用自定义 SASL 和 HTTP 验证机制。

4.1. DATA GRID 服务器身份验证

身份验证限制了用户对端点的访问以及数据网格控制台和命令行界面(CLI)。

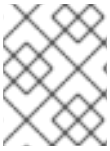
Data Grid Server 包括可强制执行用户身份验证的"默认"安全域。默认身份验证使用属性 realm，其用户凭据存储在 **server/conf/users.properties** 文件中。Data Grid Server 还默认启用安全授权，因此您必须为用户授予存储在 **server/conf/groups.properties** 文件中的权限。

提示

使用用户 **create** 命令及命令行界面(CLI)来添加用户并分配权限。如需示例和更多信息，运行 **user create --help**。

4.2. 配置 DATA GRID 服务器验证机制

您可以明确配置 Hot Rod 和 REST 端点，以使用特定的身份验证机制。只有在需要显式覆盖安全域的默认机制时才需要配置验证机制。



注意

配置中的 **每个端点** 部分都必须包含 **热连接器和其它 连接器** 元素或字段。例如，如果您明确声明一个 **热式连接器**，还必须声明其他 **连接器**，即使它没有配置验证机制。

先决条件

- 根据需要在 Data Grid Server 配置中添加安全域。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 添加 **端点** 元素或字段，并指定它与 **security-realm** 属性使用的安全域。
3. 添加 **hotrod-connector** 元素或字段来配置 Hot Rod 端点。
    a. 添加 **身份验证** 元素或字段。
    b. 指定用于 **sasl mechanisms** 属性的 Hot Rod 端点的 SASL 验证机制。
    c. 如果适用，使用 **qop** 属性指定 SASL 质量保护设置。
    d. 如果需要，使用 **server-name** 属性指定 Data Grid Server 身份。
4. 添加 **rest-connector** 元素或字段来配置 REST 端点。
    a. 添加 **身份验证** 元素或字段。
    b. 指定与 mechanisms 属性一起使用的 REST 端点的 HTTP **验证机制**。
5. 保存对您的配置的更改。

身份验证机制配置

以下配置为用于身份验证的 Hot Rod 端点指定 SASL 机制：

XML

```xml
<server xmlns="urn:infinispan:server:14.0">
  <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. 禁用身份验证

在本地开发环境或隔离的网络上，您可以将 Data Grid 配置为允许未经身份验证的客户端请求。当您禁用用户身份验证时，您也应该在 Data Grid 安全配置中禁用授权。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 从 **端点** 元素或字段移除 **security-realm** 属性。
3. 从 **cache-container** 和每个缓存配置 **的安全** 配置中删除任何 **授权** 元素。
4. 保存对您的配置的更改。

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <endpoints socket-binding="default"/>
</server>
```

JSON

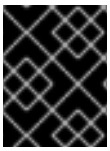
```
{
  "server": {
    "endpoints": {
      "endpoint": {
        "socket-binding": "default"
      }
    }
  }
}
```

YAML

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

4.3. DATA GRID 服务器验证机制

数据网格服务器自动配置端点，并提供与您的安全域配置匹配的身份验证机制。例如，如果您添加了 Kerberos 安全域，Data Grid Server 为 Hot Rod 端点启用 **GSSAPI** 和 **GS2-KRB5** 验证机制。



重要

目前，您不能使用带有 **DIGEST** 或 **SCRAM** 身份验证机制的轻量级目录访问协议(LDAP)协议，因为这些机制需要访问特定的哈希密码。

热环**端点**

当配置包含对应的安全域时，Data Grid 服务器为 Hot Rod 端点启用以下 SASL 身份验证机制：

安全域	SASL 身份验证机制
属性域和 LDAP 域	SCRAM-*, SCRAM-*, **SCRAM-***
令牌域	OAUTHBEARER
信任域	EXTERNAL
Kerberos 身份	GSSAPI, GS2-KRB5
SSL/TLS 身份	PLAIN

REST 端点

当配置包含对应的安全域时，Data Grid 服务器为 REST 端点启用以下 HTTP 身份验证机制：

安全域	HTTP 身份验证机制
属性域和 LDAP 域	DIGEST
令牌域	BEARER_TOKEN
信任域	CLIENT_CERT
Kerberos 身份	SPNEGO
SSL/TLS 身份	BASIC

4.3.1. SASL 验证机制

数据网格服务器支持以下 SASL 身份验证机制及 Hot Rod 端点：

身份验证机制	Description	Security realm 类型	相关详情
PLAIN	以纯文本格式使用凭证。您应该只在加密连接中使用 **PLAIN** 身份验证。	属性域和 LDAP 域	与 **BASIC** HTTP 机制类似。
DIGEST-*	使用哈希算法和非ce 值。热环连接器支持 **DIGEST-MD** 5、**DIGEST-SHA - 256**、**DIGEST-SHA-384**、DIGEST-SHA-384 和 **DIGEST-SHA-512** 哈希算法。	属性域和 LDAP 域	与 **Digest** HTTP 机制类似。
SCRAM-*	除了哈希算法和非值外，还使用 *salt* 值。热环连接器支持 **SCRAM-SHA**、**SCRAM-SHA-256**、**SCRAM-SHA-384** 和 **SCRAM-SHA-512** 哈希算法，按强度排列。	属性域和 LDAP 域	与 **Digest** HTTP 机制类似。

身份验证机制	Description	Security realm 类型	相关详情
GSSAPI	使用 Kerberos 票据并需要一个 Kerberos 域控制器。您必须在域配置中添加对应的 **kerberos** 服务器身份。在大多数情况下，您还要指定一个 **ldap-realm** 以提供用户成员资格信息。	Kerberos realms	与 **SPNEGO** HTTP 机制类似。
GS2-KRB5	使用 Kerberos 票据并需要一个 Kerberos 域控制器。您必须在域配置中添加对应的 **kerberos** 服务器身份。在大多数情况下，您还要指定一个 **ldap-realm** 以提供用户成员资格信息。	Kerberos realms	与 **SPNEGO** HTTP 机制类似。
EXTERNAL	使用客户端证书。	信任存储域	与 **CLIENT_CERT** HTTP 机制类似。
OAUTHBEARER	使用 OAuth 令牌并且需要 **token-realm** 配置。	令牌域	与 **BEARER_TOKEN** HTTP 机制类似。

4.3.2. SASL 质量保护(QoP)

如果 SASL 机制支持完整性和隐私保护(QoP)设置，您可以使用 **qop** 属性将它们添加到 Hot Rod 端点配置中。

QoP 设置	Description
auth	仅身份验证.
auth-int	具备完整性保护的身份验证。
auth-conf	通过完整性和隐私保护进行身份验证.

4.3.3. SASL 策略

SASL 策略提供对 Hot Rod 验证机制的精细控制。

提示

Data Grid 缓存授权限制根据角色和权限对缓存的访问。配置缓存授权，然后设置 & **lt;no-anonymous value=false /&** gt; 以允许匿名登录并将访问逻辑委派给缓存授权。

策略	Description	默认值
forward-secrecy	仅使用支持会话间转发保密的 SASL 机制。这意味着，进入一个会话不会自动提供用于破坏未来会话的信息。	false
pass-credentials	仅使用需要客户端证书的 SASL 机制。	false
no-plain-text	请勿使用 SASL 机制，这些机制对于简单的纯被动攻击是不可避免的。	false
no-active	请勿使用 SASL 机制，这些机制容易激活、非字典、攻击。	false
no-dictionary	不要使用受被动字典攻击的 SASL 机制。	false
no-anonymous	不要使用接受匿名登录的 SASL 机制。	true

SASL 策略配置
在以下配置中，Hot Rod 端点使用 **GSSAPI** 机制进行身份验证，因为它是唯一符合所有 SASL 策略的机制：

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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 验证机制

Data Grid Server 使用 REST 端点支持以下 HTTP 身份验证机制：

身份验证机制	Description	Security realm 类型	相关详情
BASIC	以纯文本格式使用凭证。您应该只使用带有加密连接的 **BASIC** 身份验证。	属性域和 LDAP 域	对应于 **基本** HTTP 身份验证方案，并类似于 **PLAIN** SASL 机制。

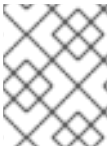
身份验证机制	Description	Security realm 类型	相关详情
摘要	使用哈希算法和非ce值。REST 连接器支持 **SHA-512**、**SHA-256** 和 **MD5** 哈希算法。	属性域和 LDAP 域	对应于 **Digest** HTTP 验证方案，它类似于 **DIGEST-*** SASL 机制。
SPNEGO	使用 Kerberos 票据并需要一个 Kerberos 域控制器。您必须在域配置中添加对应的 **kerberos** 服务器身份。在大多数情况下，您还要指定一个 **ldap-realm** 以提供用户成员资格信息。	Kerberos realms	对应于 **Negotiate** HTTP 身份验证方案，并类似于 **GSSAPI** 和 **GS2-KRB5** SASL 机制。
BEARER_TOKEN	使用 OAuth 令牌并且需要 **token-realm** 配置。	令牌域	对应于 **Bearer** HTTP 身份验证方案，并类似于 **OAUTHBEARER** SASL 机制。
CLIENT_CERT	使用客户端证书。	信任存储域	与 **EXTERNAL** SASL 机制类似。

第 5 章 安全域

Security realms 将 Data Grid Server 部署与控制访问权限的环境中的网络协议和基础架构集成，并验证用户身份。

5.1. 创建安全域

将安全域添加到 Data Grid 服务器配置，以控制对部署的访问。您可以在配置中添加一个或多个安全域。



注意

在配置中添加安全域时，Data Grid 服务器会自动为 Hot Rod 和 REST 端点启用匹配的身份验证机制。

先决条件

- 根据需要在 Data Grid Server 配置中添加套接字绑定。
- 创建密钥存储或 PEM 文件，以使用 TLS/SSL 加密配置安全域。
  数据网格服务器也可以在启动时生成密钥存储。
- 调配安全域配置所依赖的资源或服务。
  例如，如果添加令牌域，则需要置备 OAuth 服务。

此流程演示了如何配置多个属性域。开始之前，您需要创建添加用户的属性文件并使用命令行接口(CLI)分配权限。使用用户 **创建命令**，如下所示：

```
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
```

提示

如需示例和更多信息，运行 **user create --help**。

注意

使用 CLI 将凭证添加到 properties realm 中，只能在您连接的服务器实例上创建用户。您必须手动将 properties 域中的凭证同步到集群中的每个节点。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 使用安全配置中的 **security -realms** 元素来包含多个安全域。
3. 添加具有 **security-realm** 元素的安全域，并为它指定具有 **name** 属性的唯一名称。
   要遵循该示例，请创建一个名为 **application-realm** 的安全域，以及另一个名为 **management-realm** 的安全域：

4. 提供 Data Grid Server 的 TLS/SSL，**可根据需要配置服务器** 识别元素并配置密钥存储。
5. 通过添加以下元素或字段来指定安全域类型：
   - **properties-realm**
   - **ldap-realm**
   - **token-realm**
   - **truststore-realm**
6. 根据情况指定您要配置的安全域类型的属性。
   要跟踪示例，请使用 **user-properties** 和 **group-properties** 元素或字段上的 **path** 属性指定通过 CLI 创建的 ***.properties** 文件。
7. 如果您在配置中添加多种不同类型的安全域，请包括 **distributed-realm** 元素或字段，以便 Data Grid 服务器将 realm 与彼此结合使用。
8. 配置 Data Grid Server 端点，将安全域与 **security-realm** 属性搭配使用。
9. 保存对您的配置的更改。

多个属性域

XML

```xml
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="application-realm">
        <properties-realm groups-attribute="Roles">
          <user-properties path="application-users.properties"/>
          <group-properties path="application-groups.properties"/>
        </properties-realm>
      </security-realm>
      <security-realm name="management-realm">
        <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": "management-realm",
        "properties-realm": {
          "groups-attribute": "Roles",
```

```
          "user-properties": {
            "digest-realm-name": "management-realm",
            "path": "management-users.properties"
          },
          "group-properties": {
            "path": "management-groups.properties"
          }
        }
      }, {
        "name": "application-realm",
        "properties-realm": {
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "application-realm",
            "path": "application-users.properties"
          },
          "group-properties": {
            "path": "application-groups.properties"
          }
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "management-realm"
        propertiesRealm:
          groupsAttribute: "Roles"
          userProperties:
            digestRealmName: "management-realm"
            path: "management-users.properties"
          groupProperties:
            path: "management-groups.properties"
      - name: "application-realm"
        propertiesRealm:
          groupsAttribute: "Roles"
          userProperties:
            digestRealmName: "application-realm"
            path: "application-users.properties"
          groupProperties:
            path: "application-groups.properties"
```

5.2. 设置 KERBEROS 身份

在 Data Grid Server 配置中的安全域中添加 Kerberos 身份，以使用包含服务主体名称和加密密钥（派生自 Kerberos 密码）的 *keytab* 文件。

先决条件

- 具有 Kerberos 服务帐户主体。

注意

keytab 文件可以包含用户和服务帐户主体。但是，Data Grid 服务器只使用服务帐户主体，这意味着它可以向客户端提供身份并允许客户端与 Kerberos 服务器进行身份验证。

在大多数情况下，您可以为 Hot Rod 和 REST 端点创建唯一的主体。例如，如果您在 "INFINISPAN.ORG" 域中有"datagrid"服务器，则应创建以下服务主体：

- **HotRod/datagrid@INFINISPAN.ORG** 标识 Hot Rod 服务。
- **HTTP/datagrid@INFINISPAN.ORG** 标识 REST 服务。

流程

1. 为 Hot Rod 和 REST 服务创建 keytab 文件。

   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. 将 keytab 文件复制到 Data Grid Server 安装的 **server/conf** 目录中。
3. 打开 Data Grid Server 配置进行编辑。
4. 在 Data **Grid 服务器** 安全域中添加服务器事件定义。
5. 指定向 Hot Rod 和 REST 连接器提供服务主体的 keytab 文件的位置。
6. 将 Kerberos 服务主体命名为 Kerberos。
7. 保存对您的配置的更改。

Kerberos 身份配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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="kerberos-realm">
   <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": "kerberos-realm",
        "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": "kerberos-realm",
        "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"
          }
        }
      }
    }
  }
}
```

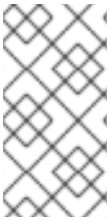
YAML

```
server:
  security:
    securityRealms:
      - name: "kerberos-realm"
        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: "kerberos-realm"
      hotrodConnector:
        authentication:
          sasl:
            serverName: "datagrid"
            serverPrincipal: "hotrod/datagrid@INFINISPAN.ORG"
      restConnector:
        authentication:
          securityRealm: "kerberos-realm"
          serverPrincipal" : "HTTP/localhost@INFINISPAN.ORG"
```

5.3. 属性域

属性域使用属性文件来定义用户和组。

- **users.properties** 包含 Data Grid 用户凭据。密码可以使用 **DIGEST-MD5** 和 **DIGEST** 验证机制进行预先说明。
- **groups.properties** 将用户与角色和权限相关联。



注意

您可以使用 Data Grid CLI 为文件输入正确的安全域名称来避免与属性文件相关的身份验证问题。您可以通过打开 **infinispan.xml** 文件并导航到 <security **-realm name> 属性来查找 Data Grid Server 的正确安全域名称**。当您将属性文件从一个 Data Grid Server 复制到另一个 Data Grid Server 时，请确保安全域名称符合目标端点的正确身份验证机制。

users.properties

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

groups.properties

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

属性域配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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": {
```

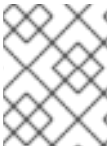
```
          "groups-attribute": "Roles",
          "user-properties": {
            "digest-realm-name": "default",
            "path": "users.properties",
            "relative-to": "infinispan.server.config.path",
            "plain-text": true
          },
          "group-properties": {
      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

LDAP 域连接到 LDAP 服务器（如 OpenLDAP、Red Hat Directory Server、Apache Directory Server 或 Microsoft Active Directory）以验证用户身份并获取成员资格信息。



注意

LDAP 服务器可以有不同的条目布局，具体取决于服务器和部署的类型。此文档超出了本文档的范围，提供了有关所有可能的配置的示例。

重要

LDAP 连接主体必须具有必要的权限才能执行 LDAP 查询和访问特定属性。

作为使用 **direct-verification** 属性验证用户凭据的替代选择，您可以指定一个 LDAP 属性来验证密码的 **user-password-mapper** 元素。

注意

您不能使用通过 **直接验证属性执行哈希的端点** 身份验证机制。

由于 Active Directory 没有公开 **密码** 属性，所以只能使用 **direct-verification** 属性，而不是 **user-password-mapper** 元素。因此，您必须将 **BASIC** 身份验证机制与 REST 端点搭配使用，PL **AIN** 与 Hot Rod 端点与 Active Directory 服务器集成。更为安全的替代方案是使用 Kerberos，它允许 **SPNEGO**、**GSSAPI** 和 **GS2-KRB5** 身份验证机制。

rdn-identifier 属性指定 LDAP 属性，它根据提供的标识符查找用户条目，后者通常是用户名；例如， **uid** 或 **sAMAccountName** 属性。将 **search-recursive="true"** 添加到配置中，以递归搜索目录。默认情况下，搜索用户条目使用 **(rdn_identifier={0})** 过滤器。使用 **filter-name** 属性指定不同的过滤器。

The `attribute-mapping` element retrieves all the groups of which the user is a member. There are typically two ways in which membership information is stored:

- 在 **member** 属性中通常具有类 **groupOfNames** 的组条目下。在这种情况下，您可以使用如上例配置中的属性过滤器。此过滤器搜索与提供的过滤器匹配的条目，该过滤器查找带有与用户 DN 相等的 **member** 属性的组。然后，过滤器提取组条目的 CN (**从** 指定)，并将其添加到用户的 **Roles** 中。
- 在 **memberOf** 属性中的用户条目中。在这种情况下，您应该使用属性引用，如下所示： **<attribute-reference reference="memberOf" from="cn" to="Roles" />**

  此引用从用户的条目获取所有 **memberOf** 属性，按照 **从** 提取 CN，并将它们添加到用户的 **Roles** 中。

LDAP 域配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="ldap-realm">
        <!-- 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": "ldap-realm",
        "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: ldap-realm
        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 域主体重新编写

SASL 验证机制（如 **GSSAPI**、**GS2-KRB5** 和 **Negotiate** ）包括需要 *清理* 的用户名，然后才能使用它来搜索 LDAP 目录。

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="ldap-realm">
        <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": "ldap-realm",
        "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: "ldap-realm"
        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. 令牌域

令牌域使用外部服务来验证令牌并需要兼容 RFC-7662 (OAuth2 Token Introspection)的提供程序，如 Red Hat SSO。

令牌域配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="token-realm">
        <!-- 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": "token-realm",
        "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: token-realm
        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. 信任存储域

信任存储域使用证书或证书链，验证数据网格服务器和客户端身份。

密钥存储

包含向客户端提供 Data Grid Server 身份的服务器证书。如果您使用服务器证书配置密钥存储，Data Grid 服务器使用行业标准 SSL/TLS 协议对流量进行加密。

信任存储

包含客户端提供给 Data Grid Server 的客户端证书或证书链。客户端信任存储是可选的，并允许数据网格服务器执行客户端证书身份验证。

客户端证书验证

如果您希望 Data Grid Server 来验证或验证客户端证书，您必须将 **require-ssl-client-auth="true"** 属性添加到端点配置。

信任存储域配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="trust-store-realm">
        <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

```
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "trust-store-realm",
        "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: "trust-store-realm"
        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. 分布式安全域

分布式域组合了多个不同类型的安全域。当用户尝试访问 Hot Rod 或 REST 端点时，Data Grid Server 会依次使用每个安全域，直到它找到可以执行身份验证的用户。

分布式域配置

XML

```xml
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="distributed-realm">
        <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

```json
{
  "server": {
    "security": {
      "security-realms": [{
        "name": "distributed-realm",
        "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": "distributed-realm",
        "path": "users.properties"
      },
      "group-properties": {
        "path": "groups.properties"
      }
    },
    "distributed-realm": {}
  }]
 }
}
}
```

YAML

```
server:
  security:
    securityRealms:
      - name: "distributed-realm"
        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: "distributed-realm"
            path: "users.properties"
          groupProperties:
            path: "groups.properties"
        distributedRealm: ~
```

第 6 章 配置 TLS/SSL 加密

您可以通过配置包含 Data Grid 的公钥和私钥的密钥存储来保护数据网格服务器连接的安全性。如果需要 mutual TLS，您还可以配置客户端证书身份验证。

6.1. 配置 DATA GRID SERVER 密钥存储

将密钥存储添加到 Data Grid Server，并将它配置为提供 SSL/TLS 证书，以将其身份验证给客户端。如果安全域包含 TLS/SSL 身份，它将加密所有使用该安全域的 Data Grid Server 端点的连接。

先决条件

- 创建含有 Data Grid Server 证书或证书链的密钥存储。

数据网格服务器支持以下密钥存储格式：JKS、JCEKS、PKCS12/PFX 和 PEM。如果存在 Bouncy Castle 库，还支持 BKS、BCFKS 和 UBER。

重要

在生产环境中，服务器证书应该由可信证书颁发机构（根）或 Intermediate CA 签名。

提示

如果它们同时包含以下内容，您可以使用 PEM 文件作为密钥存储：

- PKCS#1 或 PKCS#8 格式的私钥。
- 个或多个证书。

您还应该使用空密码(**密码=""**)配置 PEM 文件密钥存储。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 将含有 Data Grid Server 的 SSL/TLS 身份的密钥存储添加到 **$RHDG_HOME/server/conf** 目录中。
3. 在 Data **Grid 服务器** 安全域中添加服务器事件定义。
4. 使用 **path** 属性指定密钥存储文件名。
5. 为密钥存储密码和证书别名 **提供密钥存储密码** 和 **别名** 属性。
6. 保存对您的配置的更改。

后续步骤

使用信任存储配置客户端，以便他们可以验证用于 Data Grid 服务器的 SSL/TLS 身份。

密钥存储配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
```

```
  <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"
```

其他资源

- 配置 Hot Rod 客户端加密

6.1.1. 生成 Data Grid Server 密钥存储

配置 Data Grid Server，使其在启动时自动生成密钥存储。

重要

自动生成的密钥存储：

- 不应在生产环境中使用。
- 在需要时生成；例如，从客户端获取第一个连接。
- 包含您可以在 Hot Rod 客户端中使用的证书。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 在服务器配置中，包含 **keystore** 元素的 **generate-self-signed-certificate-host** 属性。
3. 将服务器证书的主机名指定为值。
4. 保存对您的配置的更改。

生成的密钥存储配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
      <security-realm name="generated-keystore">
        <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": "generated-keystore",
      "server-identities": {
        "ssl": {
          "keystore": {
            "alias": "server",
            "generate-self-signed-certificate-host": "localhost",
            "path": "server.p12",
            "password": "secret"
          }
        }
      }
    }]
  }
 }
}
```

YAML

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

6.1.2. 配置 TLS 版本和密码套件

当使用 SSL/TLS 加密来保护部署时，您可以将 Data Grid 服务器配置为使用特定的 TLS 协议版本以及协议中的特定密码套件。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 将 **engine** 元素添加到 Data Grid Server 的 SSL 配置中。
3. 将 Data Grid 配置为使用具有 **enabled-protocols** 属性的一个或多个 TLS 版本。
   Data Grid Server 默认支持 TLS 版本 1.2 和 1.3。如果正确设置了 **TLSv1.3**，则只为了限制客户端连接的安全协议。数据网格不建议启用 **TLSv1.1**， 因为它是支持有限的旧协议，并提供弱的安全性。您应该不会启用任何比 1.1 旧的 TLS 版本。

警告

如果您修改 Data Grid Server 的 SSL **引擎** 配置，则必须使用 **enabled-protocols** 属性显式配置 TLS 版本。省略 **enabled-protocols** 属性允许任何 TLS 版本。

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

4. 将 Data Grid 配置为使用 **enabled-ciphersuites** 属性（用于 TLSv1.2 及以下）以及 **enabled-ciphersuites-tls13** 属性（用于 TLSv1.3）的一个或多个加密套件。
   您必须确保设置支持计划使用的任何协议功能的密码套件，例如 **HTTP/2 ALPN**。

5. 保存对您的配置的更改。

SSL 引擎配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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. 在具有 FIPS 140-2 兼容加密的系统上配置 DATA GRID SERVER

FIPS (Federal Information 处理标准)是美国联邦计算机系统的标准和指南。虽然 FIPS 由美国联邦政府开发使用，但很多在私营部门中都使用了这些标准。

FIPS 140-2 定义了加密模块的安全要求。您可以使用其他 JDK 安全供应商将 Data Grid 服务器配置为使用遵循 FIPS 140-2 规范的加密密码。

其他资源

- Java PKCS#11 加密供应商
- Bouncy Castle 加密供应商的 Legion

6.2.1. 配置 PKCS11 加密供应商

您可以使用 **SunPKCS11-NSS-FIPS** 11-NSS-FIPS provider 指定 PKCS11 密钥存储来配置 PKCS11 加密策略。

先决条件

- 为 FIPS 模式配置您的系统。您可以通过在 Data Grid 命令行界面(CLI)中发出 **fips-mode-setup --check** 命令来检查您的系统是否启用了 FIPS 模式。
- 使用 **certutil** 工具初始化系统范围 NSS 数据库。
- 使用配置为启用 **SunPKCS11** 提供程序的 **java.security** 文件安装 JDK。此提供程序指向 NSS 数据库和 SSL 提供程序。
- 在 NSS 数据库中安装证书。

注意

OpenSSL 供应商需要一个私钥，但您无法从 PKCS#11 存储检索私钥。FIPS 阻止从 FIPS 兼容的加密模块中导出未加密密钥，因此在 FIPS 模式中无法使用 OpenSSL 供应商。您可以使用 **-Dorg.infinispan.openssl=false** 参数在启动时禁用 OpenSSL 供应商。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 在 Data **Grid 服务器** 安全域中添加服务器事件定义。
3. 使用 **SunPKCS11-NSS-FIPS** 供应商指定 PKCS11 密钥存储。
4. 保存对您的配置的更改。

密钥存储配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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. 配置 Bouncy Castle FIPS 加密供应商

您可以在 Data Grid 服务器配置中配置 Bouncy Castle FIPS (Federal Information Processing Standards) 加密供应商。

先决条件

- 为 FIPS 模式配置您的系统。您可以通过在 Data Grid 命令行界面(CLI)中发出 **fips-mode-setup --check** 命令来检查您的系统是否启用了 FIPS 模式。
- 以 BCFKS 格式创建包含证书的密钥存储。

流程

1. 下载 Bouncy Castle FIPS JAR 文件，并将文件添加到数据网格服务器安装的 **server/lib** 目录中。
2. 要安装 Bouncy Castle，请发出 **安装** 命令：

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

3. 打开 Data Grid Server 配置进行编辑。
4. 在 Data **Grid 服务器** 安全域中添加服务器事件定义。
5. 使用 **BCFIPS** 提供商指定 BCFKS 密钥存储。
6. 保存对您的配置的更改。

密钥存储配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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. 配置客户端证书身份验证

配置 Data Grid 服务器以使用 mutual TLS 来保护客户端连接。

您可以通过两种方式配置 Data Grid，以两种方式验证来自证书的客户端身份：

- 需要仅包含签名证书的信任存储，通常是证书颁发机构(CA)。显示 CA 签名的证书的任何客户端都可以连接到 Data Grid。
- 需要包含签名证书外的所有客户端证书的信任存储。只有存在信任存储中签名的证书的客户端才能连接到 Data Grid。

提示

另外，要提供信任存储，您可以使用共享系统证书。

先决条件

- 创建包含 CA 证书或所有公共证书的客户端信任存储。
- 为 Data Grid Server 创建密钥存储并配置 SSL/TLS 身份。

注意

PEM 文件可用作其包含一个或多个证书的信任存储。这些信任存储应配置为使用空密码：**password=""**。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 将 **require-ssl-client-auth="true"** 参数添加到您的 **端点** 配置中。
3. 将客户端信任存储添加到 **$RHDG_HOME/server/conf** 目录中。
4. 在 Data Grid Server security realm 配置中指定 **truststore** 元素 **的路径 和密码** 属性。
5. 如果您希望 Data Grid Server 验证每个 **客户端证书，请将 <truststore-realm** /> 元素添加到安全域中。
6. 保存对您的配置的更改。

后续步骤

- 如果您使用安全角色和权限控制访问，请在 Data Grid Server 配置中使用客户端证书设置授权。
- 将客户端配置为与 Data Grid Server 协商 SSL/TLS 连接。

客户端证书验证配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <security-realms>
```

```
      <security-realm name="trust-store-realm">
        <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": "trust-store-realm",
        "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": "trust-store-realm",
      "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: "trust-store-realm"
        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"
```

其他资源

- 配置 Hot Rod 客户端加密
- 使用共享系统证书 (Red Hat Enterprise Linux 7 安全指南)

6.4. 使用客户端证书配置授权

启用客户端证书身份验证意味着您无需在客户端配置中指定 Data Grid 用户凭证，这意味着您必须将角色与客户端证书中的通用名称(CN)字段关联。

先决条件

- 为客户端提供包含其公共证书或证书链的一部分（通常是公共 CA 证书）的 Java 密钥存储。
- 配置 Data Grid Server 以执行客户端证书身份验证。

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 在安全授权配置中启用 **common-name-role-mapper**。
3. 为客户端证书中的通用名称(**CN**)分配一个具有适当权限的角色。
4. 保存对您的配置的更改。

客户端证书授权配置

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"
              }
            }
          }
        }
      }
    }
  }
}
```

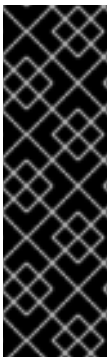
YAML

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

第 7 章 在密钥存储中存储 DATA GRID SERVER 凭证

外部服务需要凭证来使用 Data Grid 服务器进行身份验证。为保护敏感文本字符串，如密码，将它们添加到凭据密钥存储中，而不是直接在 Data Grid Server 配置文件中。

然后，您可以配置 Data Grid Server 以解密密码以便与数据库或 LDAP 目录等服务建立连接。



重要

$RHDG_HOME/server/conf 中的纯文本密码未加密。任何对主机文件系统具有读取访问权限的用户帐户都可以查看纯文本密码。

虽然凭据密钥存储是受密码保护的存储加密密码，对主机文件系统具有写入访问权限的任何用户帐户都可以篡改。

要完全安全的 Data Grid 服务器凭证，您应该只向可以配置和运行 Data Grid Server 的用户帐户授予读写访问权限。

7.1. 设置凭证密钥存储

创建用于加密 Data Grid 服务器访问的凭据的密钥存储。

凭据密钥存储至少包含一个与加密密码关联的别名。创建密钥存储后，您可以在连接配置（如数据库连接池）中指定别名。然后，在服务尝试身份验证时，Data Grid 服务器从密钥存储解密该别名的密码。

您可以根据需要，创建多个凭据密钥存储，使其具有许多别名。

注意

作为安全最佳实践，密钥存储应只能由运行 Data Grid Server 进程的用户读取。

流程

1. 在 **$RHDG_HOME** 中打开一个终端。
2. 创建密钥存储并使用 credentials 命令向其添加 **凭据**。

提示

默认情况下，密钥存储类型为 PKCS12。运行 **帮助凭据**，以获取有关更改密钥存储的默认值的详细信息。

以下示例演示了如何为密码"changeme"创建包含"dbpassword"别名的密钥存储。在创建密钥存储时，您还要通过 **-p** 参数指定用于访问密钥存储的密码。

Linux

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

Microsoft Windows

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

3. 检查别名是否已添加到密钥存储中。

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

4. 打开 Data Grid Server 配置进行编辑。

5. 配置 Data Grid 以使用凭据密钥存储。

   a. 在 **安全配置** 中添加 **credential-stores** 部分。

   b. 指定凭据密钥存储的名称和位置。

   c. 指定密码，以使用 **明文-credential** 配置访问凭据密钥存储。

注意

除了在 Data Grid Server 配置中为凭证密钥存储添加明文密码外，您可以使用外部命令或屏蔽密码来提高安全性。

在一个凭证存储中，您还可以使用密码作为另一个凭据存储的主密码。

6. 在 Data Grid Server 用来连接外部系统（如数据源或 LDAP 服务器）的配置中，引用凭据密钥存储。

   a. 添加 **credential-reference** 部分。

   b. 使用 **store** 属性指定凭据密钥存储的名称。

   c. 使用 **alias** 属性指定密码别名。

提示

credential-reference 配置中的属性是可选的。

- 只有在有多个密钥存储时才需要 **存储**。
- 只有在密钥存储包含多个密码别名时才需要别名。

7. **保存对您的配置的更改。**

7.2. 保护凭证密钥存储的密码

Data Grid 服务器需要密码才能访问凭据密钥存储。您可以在 Data Grid Server 配置中以明文形式添加该密码，或者作为增加了安全层，您可以使用外部命令作为密码，或者您可以屏蔽密码。

先决条件

- 为 Data Grid 服务器设置凭据密钥存储。

流程

执行以下操作之一：

- 使用 凭证掩码 命令使密码混淆，例如：

```
bin/cli.sh credentials mask -i 100 -s pepper99 "secret1234!"
```

屏蔽的密码使用基于密码的加密(PBE)，且必须在您的数据网格配置中使用以下格式：<MASKED_VALUE;SALT;ITERATION>。

- 使用作为标准输出提供密码的外部命令。

外部命令可以是使用 java.lang. Runtime#exec (java.lang.lang.String) 的任何可执行文件，如 shell 脚本或二进制代码。
如果该命令需要参数，以空格分隔的字符串列表形式提供。

7.3. 凭证密钥存储配置

您可以将凭据密钥存储添加到 Data Grid Server 配置中，并使用明文密码、屏蔽的密码或提供密码的外部命令。

凭证密钥存储（带有明文密码）

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <credential-stores>
      <credential-store name="credentials" path="credentials.pfx">
        <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!"
```

带有屏蔽密码的凭证密钥存储

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <credential-stores>
```

```
      <credential-store name="credentials"
                        path="credentials.pfx">
        <masked-credential masked="1oTMDZ5JQj6DVepJviXMnX;pepper99;100"/>
      </credential-store>
    </credential-stores>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "masked-credential": {
          "masked": "1oTMDZ5JQj6DVepJviXMnX;pepper99;100"
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        maskedCredential:
          masked: "1oTMDZ5JQj6DVepJviXMnX;pepper99;100"
```

外部命令密码

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <credential-stores>
      <credential-store name="credentials"
                        path="credentials.pfx">
        <command-credential command="/path/to/executable.sh arg1 arg2"/>
      </credential-store>
    </credential-stores>
  </security>
</server>
```

JSON

```
{
  "server": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "command-credential": {
          "command": "/path/to/executable.sh arg1 arg2"
        }
      }]
    }
  }
}
```

YAML

```
server:
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        commandCredential:
          command: "/path/to/executable.sh arg1 arg2"
```

7.4. 凭证密钥存储引用

将凭据密钥存储添加到 Data Grid 服务器后，您可以在连接配置中引用它们。

数据源连接

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <security>
    <credential-stores>
      <credential-store name="credentials"
                        path="credentials.pfx">
        <clear-text-credential clear-text="secret1234!"/>
      </credential-store>
    </credential-stores>
  </security>
  <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": {
    "security": {
      "credential-stores": [{
        "name": "credentials",
        "path": "credentials.pfx",
        "clear-text-credential": {
          "clear-text": "secret1234!"
        }
      }],
      "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:
  security:
    credentialStores:
      - name: credentials
        path: credentials.pfx
        clearTextCredential:
          clearText: "secret1234!"
  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 连接

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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 具有基于角色的访问控制的安全授权

基于角色的访问控制(RBAC)功能使用不同的权限级别来限制用户与 Data Grid 的交互。

注意

有关创建用户并配置特定于远程或嵌入式缓存的授权的详情，请参考：

- 使用 Data Grid Server 配置用户角色和权限
- 以编程方式配置用户角色和权限

8.1. DATA GRID 用户角色和权限

数据网格包括多个角色，为用户提供访问缓存和数据网格资源的权限。

角色	权限	Description
admin	ALL	具有所有权限的超级用户，包括缓存管理器生命周期的控制。
deployer	ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR, CREATE	除了 **应用程序** 权限外，还可创建和删除数据网格资源。
application	ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR	除 **观察者** 权限之外，还具有对 Data Grid 资源的读写访问权限。还可以侦听事件并执行服务器任务和脚本。
observer	ALL_READ, MONITOR	除了监控权限外，还具有对数据网格 **资源** 的读取访问权限。
monitor	MONITOR	可以通过 JMX 和 **指标端点** 查看统计信息。

其他资源

- org.infinispan.security.AuthorizationPermission Enum

- Data Grid 配置模式参考

8.1.1. 权限

用户角色是具有不同访问级别的权限集合。

表 8.1. 缓存管理器权限

权限	功能	Description
配置	**defineConfiguration**	定义新的缓存配置。
LISTEN	**addListener**	针对缓存管理器注册监听程序。
生命周期	**stop**	停止缓存管理器。
创建	**createCache,removeCache**	创建和删除容器资源，如缓存、计数器、架构和脚本。
MONITOR	**getStats**	允许访问 JMX 统计数据和 **指标端点**。
ALL	-	包括所有缓存管理器权限。

表 8.2. 缓存权限

权限	功能	Description
READ	**get，包含**	从缓存检索条目。
写	**put,putIfAbsent,replace,remove,evict**	在缓存中写入、替换、删除、驱除数据。
EXEC	**distexec,流**	允许对缓存执行代码。
LISTEN	**addListener**	根据缓存注册监听程序。
BULK_READ	**keySet,值,entrySet,query**	执行批量检索操作。
BULK_WRITE	**清除，放置All**	执行批量写入操作。
生命周期	**启动，停止**	启动和停止缓存。

ADMIN	**getVersion**,**addInterceptor***, **removeInterceptor Chain** ,**getEvictionManager**,**getComponentRegistry**,**getDistributionManager**,**getAuthorizationManager**,**evict**,**getRpcManager**, getCacheConfiguration ,**getCacheConfiguration**, **getCacheManager**,**getInvocationContextContainer**,**setAvailability**,**getDataContainer**,**getStats**,**getXAResource**	允许访问底层组件和内部结构。
MONITOR	**getStats**	允许访问 JMX 统计数据和 **指标端点**。
ALL	-	包括所有缓存权限。
ALL_READ	-	组合 READ 和 BULK_READ 权限。
ALL_WRITE	-	组合 WRITE 和 BULK_WRITE 权限。

其他资源

- Data Grid Security API

8.1.2. 角色和权限映射器

数据网格用户通过 **javax.security.auth.Subject** 类实施，并代表一组类型为 **java.security.Principal** 的安全主体。

Data Grid 包含 PrincipalRoleMapper API，将安全主体与角色以及 RolePermissionMapper API 与权限集关联。数据网格还提供以下角色和权限映射程序实施：

集群角色映射器

在集群 registry 中存储到角色映射的主体。

集群权限映射程序

将角色存储到集群 registry 中的权限映射，并允许您动态修改用户角色和权限。

身份角色映射器

使用主体名称作为角色名称。主体名称的类型或格式取决于源。例如，在 LDAP 目录中，主体名称可以是可辨识的名称(DN)。

通用名称角色映射程序

使用通用名称(CN)作为角色名称。您可以将此角色 mapper 与包含可辨uished Names (DN)的 LDAP 目录一起使用；例如 cn=managers,ou= people,dc=example,dc=com 映射到 managers 角色。

其他资源

- Data Grid Security API
- org.infinispan.security.PrincipalRoleMapper
- org.infinispan.security.RolePermissionMapper
- org.infinispan.security.mappers.IdentityRoleMapper
- org.infinispan.security.mappers.CommonNameRoleMapper

8.1.3. 配置角色映射器

Data Grid 默认启用集群角色映射程序和集群权限映射程序。如果要使用身份角色映射程序、通用名称(CN)角色映射程序或自定义实施，您应该配置角色映射程序。例如，如果您的部署与 LDAP 目录集成，且您想要使用可辨识的名称(DN)作为安全主体，您可以将 Data Grid 配置为使用通用名称(CN)角色映射程序。

流程

1. 打开 Data Grid 配置进行编辑。
2. 在 Cache Manager 配置中声明 role mapper 作为安全授权的一部分。
3. 保存对您的配置的更改。

角色映射程序配置

XML

```
<cache-container>
  <security>
    <authorization>
      <common-name-role-mapper />
    </authorization>
  </security>
</cache-container>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "security" : {
        "authorization" : {
          "common-name-role-mapper": {}
        }
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
    security:
      authorization:
        commonNameRoleMapper: ~
```

其他资源

- Data Grid 配置模式参考

8.2. 使用安全授权配置缓存

为缓存添加安全授权，以强制基于角色的访问控制(RBAC)。这需要 Data Grid 用户拥有足够级别的权限来执行缓存操作。

先决条件

- 创建 Data Grid 用户，并为他们授予角色，或将其分配到组。

流程

1. 打开 Data Grid 配置进行编辑。
2. 在配置中添加 security 部分。
3. 指定用户必须使用 授权 元素执行缓存操作的角色。

   您可以隐式添加 Cache Manager 中定义的所有角色，或者显式定义角色子集。
4. 保存对您的配置的更改。

隐式角色配置

以下配置隐式添加 Cache Manager 中定义的每个角色：

XML

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

–

JSON

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

YAML

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

显式角色配置

以下配置明确添加缓存管理器中定义的角色子集。在这种情况下，Data Grid 拒绝任何没有配置的角色的用户缓存操作。

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"]
```

第 9 章 启用和配置数据网格统计信息和 JMX 监控

数据网格可以提供缓存管理器和缓存统计信息，以及导出 JMX MBean。

9.1. 在远程缓存中启用统计

Data Grid Server 会自动启用默认缓存管理器的统计信息。但是，您必须明确为缓存启用统计。

流程

1. 打开 Data Grid 配置进行编辑。
2. 添加 statistics 属性或字段，并将 true 指定为值。
3. 保存并关闭您的数据网格配置。

远程缓存统计

XML

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

JSON

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

YAML

```
distributedCache:
  statistics: true
```

9.2. 启用 HOT ROD 客户端统计信息

热 Rod Java 客户端可以提供包括远程缓存和近缓存命中率和未命中以及连接池使用量的统计信息。

流程

1. 打开 Hot Rod Java 客户端配置进行编辑。
2. 将 true 设置为 statistics 属性的值，或调用 statistics （）.enable （） 方法。
3. 使用 jmx 和 jmx_domain 属性或调用 jmx Enable （） 和 jmxDomain （） 方法的 Hot Rod 客户端导出 JMX MBeans。
4. 保存并关闭您的客户端配置。

热 Rod Java 客户端统计信息

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. 配置 DATA GRID 指标

数据网格会生成与任何监控系统兼容的指标。

- 量表提供值，如用于写操作或 JVM 运行时间的平均纳秒数。
- histograms 提供有关读取、写入和删除时间等操作执行时间的详细信息。

默认情况下，Data Grid 在启用统计数据时会生成量表，但您也可以将其配置为生成直方图。

注意

数据网格指标在 供应商 范围内提供。与 JVM 相关的指标在 基础 范围内提供。

流程

1. 打开 Data Grid 配置进行编辑。
2. 将 metrics 元素或对象添加到缓存容器。
3. 通过量表属性或字段启用或禁用量表。

4. **使用直 方属性或字段启用或禁用直方图。**

5. **保存并关闭您的客户端配置。**

指标配置

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"
```

验证

Data Grid Server 通过 metrics 端点公开统计数据，端点可通过 Prometheus 等监控工具来收集。要验证是否将统计导出到 指标端点，您可以执行以下操作：

Prometheus 格式

```
curl -v http://localhost:11222/metrics \
--digest -u username:password
```

OpenMetrics format

```
curl -v http://localhost:11222/metrics \
--digest -u username:password \
-H "Accept: application/openmetrics-text"
```

注意

Data Grid 不再以 MicroProfile JSON 格式提供指标。

其他资源

- Micrometer Prometheus

9.4. 注册 JMX MBEANS

数据网格可以注册 JMX MBeans，用于收集统计信息和执行管理操作。您还必须为 JMX MBeans 中的所有统计属性提供 0 值。

流程

1. 打开 Data Grid 配置进行编辑。
2. 将 jmx 元素或对象添加到缓存容器，并将 true 指定为 enabled 属性或字段的值。
3. 添加 domain 属性或字段，并根据需要指定公开 JMX MBeans 的域。
4. 保存并关闭您的客户端配置。

JMX 配置

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"
      }
```

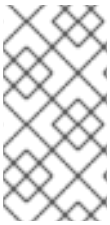
```
    }
  }
}
```

YAML

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

9.4.1. 启用 JMX 远程端口

提供唯一的远程 JMX 端口，以通过 JMXServiceURL 格式的连接公开数据网格 MBeans。



注意

数据网格服务器不通过单一端口端点远程公开 JMX。如果要通过 JMX 远程访问数据网格服务器，您必须启用远程端口。

您可以使用以下方法之一启用远程 JMX 端口：

- 启用需要身份验证到其中一个数据网格服务器安全域的远程 JMX 端口。
- 使用标准的 Java 管理配置选项手动启用远程 JMX 端口。

先决条件

- 对于带有身份验证的远程 JMX，使用默认安全域定义 JMX 特定的用户角色。用户必须具有读写访问权限的 controlRole 或具有只读访问权限的 monitorRole 才能访问任何 JMX 资源。

流程

通过以下方法之一启用远程 JMX 端口启动 Data Grid Server：

- **通过端口 9999 启用远程 JMX。**

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

警告

禁用了 SSL 的远程 JMX 不用于生产环境。

- **在启动时将以下系统属性传递给 Data Grid 服务器：**

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

警告

在无需身份验证或 SSL 的情况下启用远程 JMX 并不安全，不建议在任何环境中使用。禁用身份验证和 SSL 可让未授权用户连接到您的服务器并访问其中托管的数据。

其他资源

- 创建安全域

9.4.2. Data Grid MBeans

数据网格公开了代表可管理资源的 JMX MBeans。

org.infinispan:type=Cache

用于缓存实例的属性和操作。

org.infinispan:type=CacheManager

用于缓存管理器的属性和操作，包括数据网格缓存和集群健康统计。

有关可用 JMX MBeans 以及描述以及可用操作和属性的完整列表，请参阅 *数据网格 JMX 组件* 文档。

其他资源

- 数据网格 JMX 组件

9.4.3. 在自定义 MBean 服务器中注册 MBeans

数据网格包含一个 MBeanServerLookup 接口，可用于在自定义 MBeanServer 实例中注册 MBeans。

先决条件

- 创建 MBeanServerLookup 的实施，使 getMBeanServer （） 方法返回自定义 MBeanServer 实例。
- 配置数据网格以注册 JMX MBeans.

流程

1. 打开 Data Grid 配置进行编辑。
2. 将 mbean-server-lookup 属性或字段添加到 Cache Manager 的 JMX 配置中。
3. 指定 MBeanServerLookup 实施的完全限定域名(FQN)。

4. 保存并关闭您的客户端配置。

JMX MBean 服务器查找配置

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"
```

9.5. 在状态传输操作过程中导出指标

您可以为 Data Grid 在节点间重新分发的集群缓存导出时间指标。

当集群缓存拓扑更改时，会进行状态传输操作，如节点加入或离开集群。在状态传输操作期间，Data Grid 从每个缓存中导出指标，以便您可以确定缓存的状态。状态传输以属性形式公开属性，这样数据网格可以从每个缓存中导出指标。

注意

您不能以validation 模式执行状态传输操作。

数据网格会生成与 REST API 和 JMX API 兼容的时间指标。

先决条件

- 配置数据网格指标。
- 为您的缓存类型启用指标，如嵌入缓存或远程缓存。
- 通过更改集群缓存拓扑来发起状态传输操作。

流程

- 选择以下任一方法：
  - 配置 Data Grid 以使用 REST API 来收集指标。

- 配置数据网格以使用 JMX API 来收集指标。

其他资源

- 启用和配置数据网格统计信息和 JMX 监控（数据网格缓存）
- StateTransferManager (Data Grid 14.0 API)

第 10 章 将受管数据源添加到 DATA GRID 服务器

通过在 Data Grid 服务器配置中添加受管数据源，优化 JDBC 数据库连接的连接池和性能。

10.1. 配置受管数据源

作为 Data Grid 服务器配置的一部分创建受管 datasources，以优化 JDBC 数据库连接的连接池和性能。然后，您可以在缓存中指定受管数据源的 JDNI 名称，这会集中部署的 JDBC 连接配置。

先决条件

- 将数据库驱动程序复制到 Data Grid Server 安装中的 server/lib 目录中。

  提示

  使用 install 命令和 Data Grid 命令行界面(CLI)将所需的驱动程序下载到 server/lib 目录，例如：

  ```
  install org.postgresql:postgresql:42.1.3
  ```

流程

1. 打开 Data Grid Server 配置进行编辑。
2. 将新的 数据源添加到 data- sources 部分。
3. 通过 name 属性或字段唯一标识数据源。
4. 使用 jndi-name 属性或字段为数据源指定 JNDI 名称。

   提示

   您可以使用 JNDI 名称在 JDBC 缓存存储配置中指定数据源。

5. 将 true 设置为 statistics 属性或字段的值，以通过 /metrics 端点为数据源启用统计。

6. 提供 JDBC 驱动程序详细信息，以定义如何在 connection-factory 部分中定义数据源连接。

   a. 使用 driver 属性或字段指定数据库驱动程序的名称。

   b. 使用 url 属性或字段指定 JDBC 连接 url。

   c. 使用 用户名和密码 属性或字段指定凭证。

   d. 根据需要提供任何其他配置。

7. 在 connection-pool 部分中，定义 Data Grid Server 节点池并重复使用与连接池调整属性的连接。

8. 保存对您的配置的更改。

验证

使用 Data Grid 命令行界面(CLI)测试数据源连接，如下所示：

1. 启动 CLI 会话。

```
bin/cli.sh
```

2. 列出所有数据源，并确认您创建的数据源可用。

```
server datasource ls
```

3. 测试数据源连接。

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

管理的数据源配置

XML

```
<server xmlns="urn:infinispan:server:14.0">
  <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. 使用 JNDI 名称配置缓存

将受管数据源添加到 Data Grid 服务器时，您可以将 JNDI 名称添加到基于 JDBC 的缓存存储配置中。

先决条件

- 使用受管理数据源配置 Data Grid 服务器。

流程

1. 打开缓存配置进行编辑。
2. 将 data-source 元素或字段添加到基于 JDBC 的缓存存储配置中。
3. 将受管数据源的 JNDI 名称指定为 jndi-url 属性的值。
4. 根据情况配置基于 JDBC 的缓存存储。
5. 保存对您的配置的更改。

缓存配置中的 JNDI 名称

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. 连接池调整属性

您可以在 Data Grid Server 配置中为 managed datasources 调优 JDBC 连接池。

属性	Description
initial-size	池应有的初始连接数。
max-size	池中的最大连接数。
min-size	池应拥有的最低连接数。
blocking-timeout	在引发异常前，在等待连接时以毫秒为单位进行阻止的最大时间（毫秒）。如果创建新连接需要较长的时间内，这不会抛出异常。默认值为 **0，** 表示调用将无限期等待。
background-validation	后台验证运行间隔的时间（毫秒）。**0** 持续时间表示禁用这个功能。
validate-on-acquisition	在获取前，连接闲置时间超过这个值（以毫秒为单位指定）在被验证前进行验证。**0** 持续时间表示禁用这个功能。
idle-removal	在删除连接前，连接必须闲置时间（以分钟为单位）。

属性	Description
leak-detection	在泄漏警告前，连接必须保留以毫秒为单位。

第 11 章 设置 DATA GRID 集群传输

数据网格需要一个传输层，以便节点可以自动加入并离开集群。传输层还使 Data Grid 节点可以在网络间复制或分发数据，并执行负载平衡和状态传输等操作。

11.1. 默认 JGROUPS 堆栈

Data Grid 在 infinispan-core-14.0.6Final-redhat-00001.jar 文件的 default-configs 目录中提供默认的 JGroups 堆栈文件 default-jgroups-*. xml。

您可以在 $RHDG_HOME/lib 目录中找到此 JAR 文件。

文件名	堆栈名称	Description
default-jgroups-udp.xml	**udp**	使用 UDP 进行传输和 UDP 多播进行发现。适用于较大的集群（超过 100 个节点），或者如果您使用复制缓存或无效模式。最小化开放插槽的数量。
default-jgroups-tcp.xml	**tcp**	使用 TCP 进行传输，使用 **MPING** 协议进行发现，它使用 **UDP** 多播。*仅适用于使用分布式缓存时*（在 100 个节点下），因为 TCP 的效率比 UDP 作为点对点协议效率更高。
default-jgroups-kubernetes.xml	**kubernetes**	使用 TCP 进行传输和 **DNS_PING** 进行发现。适用于不总是可用的 UDP 多播的 Kubernetes 和 Red Hat OpenShift 节点。
default-jgroups-ec2.xml	**ec2**	使用 TCP 进行传输，并使用 **aws.S3_PING** 进行发现。适用于 UDP 多播不可用的 Amazon EC2 节点。需要额外的依赖项。
default-jgroups-google.xml	**google**	使用 TCP 进行传输和 **GOOGLE_PING2** 进行发现。适用于 UDP 多播不可用的 Google Cloud Platform 节点。需要额外的依赖项。
default-jgroups-azure.xml	**azure**	使用 TCP 进行传输和 **AZURE_PING** 进行发现。适用于不可用的 UDP 多播的 Microsoft Azure 节点。需要额外的依赖项。

其他资源

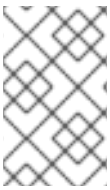
- JGroups 协议

11.2. 集群发现协议

数据网格支持不同的协议，允许节点在网络和集群中自动查找彼此。

Data Grid 可以使用两种发现机制：

- 用于处理大多数网络的通用发现协议，不依赖于外部服务。
- 依赖于外部服务的发现协议为 Data Grid 集群存储和检索拓扑信息。
例如，DNS_PING 协议通过 DNS 服务器记录执行发现。



注意

在托管平台上运行数据网格需要使用发现机制来适应各个云供应商所实施的网络限制。

其他资源

- JGroups 发现协议
- JGroup 集群传输配置 Data Grid 8.x （红帽知识库文章）

11.2.1. PING

PING 或 UDPPING 是一种通用 JGroups 发现机制，通过 UDP 协议使用动态多播。

加入后，节点会将 PING 请求发送到 IP 多播地址，以发现已位于 Data Grid 集群中的其他节点。每个节点使用包含协调器节点地址和其自身地址的数据包响应 PING 请求。c=coordinator 地址和 A=own 地址.如果没有节点响应 PING 请求，加入节点就成为新集群中的协调节点。

PING 配置示例

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

其他资源

- JGroups PING

11.2.2. TCPPING

TCPPING 是一个通用 JGroups 发现机制，为集群成员使用静态地址列表。

使用 TCPPING 时，您将把 Data Grid 集群中每个节点的 IP 地址或主机名手动指定为 JGroups 堆栈的一部分，而不是让节点能够动态发现节点。

TCPPING 配置示例

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

其他资源

- JGroups TCPPING

11.2.3. MPING

MPING 使用 IP 多播来发现数据网格集群的初始成员资格。

您可以使用 MPING 将 TCPPING 发现替换为 TCP 堆栈，并使用多caing 进行发现，而不是初始主机的静态列表。但是，您还可以将 MPING 与 UDP 堆栈搭配使用。

MPING 配置示例

```
<MPING mcast_addr="${jgroups.mcast_addr:239.6.7.8}"
      mcast_port="${jgroups.mcast_port:46655}"
      num_discovery_runs="3"
      ip_ttl="${jgroups.udp.ip_ttl:2}"/>
```

其他资源

- JGroups MPING

11.2.4. TCPGOSSIP

Gossip 路由器在网络上提供一个中央位置，您的 Data Grid 集群可以检索其他节点的地址。

将 Gosssip 路由器的地址(IP:PORT)注入 Data Grid 节点，如下所示：

1. **将地址作为系统属性传递给 JVM；例如，-DGossipRouterAddress="10.10.2.4[12001]".**
2. **在 JGroups 配置文件中引用该系统属性。**

Gossip 路由器配置示例

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

其他资源

- JGroups Gossip Router

11.2.5. JDBC_PING

JDBC_PING 使用共享数据库存储数据网格集群的信息。此协议支持任何可以使用 JDBC 连接的数据库。

节点将其 IP 地址写入共享数据库，以便加入节点可以在网络上找到 Data Grid 集群。当节点离开 Data Grid 集群时，它们会从共享数据库中删除其 IP 地址。

JDBC_PING 配置示例

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

重要

将适当的 JDBC 驱动程序添加到类路径，以便数据网格可以使用 JDBC_PING。

其他资源

- JDBC_PING
- JDBC_PING Wiki

11.2.6. DNS_PING

JGroups DNS_PING 查询 DNS 服务器，以在 Kubernetes 环境中发现数据网格群集成员，如 OKD 和红帽 OpenShift。

DNS_PING 配置示例

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

其他资源

- JGroups DNS_PING
- Service 和 Pod 的 DNS （用于添加 DNS 条目的 Kubernetes 文档）

11.2.7. 云发现协议

数据网格包括默认的 JGroups 堆栈，它使用特定于云提供商的发现协议实施。

发现协议	默认堆栈文件	工件	版本
aws.S3_PING	default-jgroups-ec2.xml	org.jgroups.aws:jgroups-aws	2.0.1.Final
GOOGLE_PING2	default-jgroups-google.xml	org.jgroups.google:jgroups-google	1.0.0.Final
azure.AZURE_PING	default-jgroups-azure.xml	org.jgroups.azure:jgroups-azure	2.0.0.Final

为云发现协议提供依赖项

要使用 aws.S3_PING，GOOGLE_PING2 或 azure.AZURE_PING 云发现协议，您需要为数据网格提供依赖的库。

流程

1. 下载工件 JAR 文件及所有依赖项。
2. 将工件 JAR 文件和所有依赖项添加到 Data Grid Server 安装的 $RHDG_HOME/server/lib 目录中。

如需了解更多详细信息，请参阅 JGroups 云发现数据网格服务器协议 （红帽知识库文章）

然后，您可以将云发现协议配置为 JGroups 堆栈文件或系统属性的一部分。

其他资源

- JGroups aws.S3_PING
- JGroups GOOGLE_PING2
- JGroups azure.AZURE_PING

11.3. 使用默认 JGROUPS 堆栈

数据网格使用 JGroups 协议堆栈，以便节点可以在专用集群频道上互相发送其他消息。

数据网格为 UDP 和 TCP 协议提供预配置的 JGroups 堆栈。您可以使用这些默认堆栈作为构建自定义集群传输配置的起点，该配置根据您的网络要求进行了优化。

流程

之一使用默认 JGroups 堆栈之一：

- 使用 infinispan.xml 文件中的 stack 属性。

```
<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>
```

- 在 Data Grid Server 启动时，使用 cluster-stack 参数设置 JGroups 堆栈文件：

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

验证

Data Grid 记录以下信息来指示其使用的堆栈：

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

其他资源

- JGroup 集群传输配置 Data Grid 8.x （红帽知识库文章）

11.4. 自定义 JGROUPS 堆栈

调整和调优属性，以创建适用于您的网络要求的集群传输配置。

数据网格提供属性，使您能够扩展默认的 JGroups 堆栈以简化配置。您可以在组合、删除和替换其他属性的同时从默认堆栈中继承属性。

流程

1. 在 infinispan.xml 文件中创建一个新的 JGroups 堆栈声明。
2. 添加 extend 属性，并指定 JGroups 堆栈来继承属性。
3. 使用 stack.combine 属性修改继承堆栈中配置的协议属性。
4. 使用 stack.position 属性定义自定义堆栈的位置。
5. 将堆栈名称指定为 传输 配置中 stack 属性的值。

例如，您可以使用默认 TCP 堆栈使用 Gossip 路由器和对称加密进行评估，如下所示：

```
<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="${jgroups.tunnel.gossip_router_hosts:localhost[12001]}"
          stack.combine="REPLACE"
          stack.position="MPING" />
    <!-- Removes the FD_SOCK2 protocol from the stack. -->
    <FD_SOCK2 stack.combine="REMOVE"/>
    <!-- Modifies the timeout value for the VERIFY_SUSPECT2 protocol. -->
    <VERIFY_SUSPECT2 timeout="2000"/>
    <!-- Adds SYM_ENCRYPT to the stack after VERIFY_SUSPECT2. -->
    <SYM_ENCRYPT sym_algorithm="AES"
                 keystore_name="mykeystore.p12"
                 keystore_type="PKCS12"
                 store_password="changeit"
                 key_password="changeit"
                 alias="myKey"
                 stack.combine="INSERT_AFTER"
                 stack.position="VERIFY_SUSPECT2" />
   </stack>
   <cache-container name="default" statistics="true">
    <!-- Uses "my-stack" for cluster transport. -->
    <transport cluster="${infinispan.cluster.name}"
               stack="my-stack"
               node-name="${infinispan.node.name:}"/>
   </cache-container>
  </jgroups>
</infinispan>
```

6. 检查 **Data Grid** 日志，以确保其使用堆栈。

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

参考

- **JGroup 集群传输配置 Data Grid 8.x （红帽知识库文章）**

11.4.1. 继承属性

当您扩展 JGroups 堆栈时，继承属性可让您调整您要扩展的堆栈中的协议和属性。

- **stack.position 指定要修改的协议。**
- **stack.combine 使用以下值来扩展 JGroups 堆栈：**

值	Description
组合	覆盖协议属性。
替换	替换协议。
INSERT_AFTER	在另一个协议后向堆栈中添加协议。不会影响您指定为插入点的协议。 JGroups 堆栈中的协议根据它们在堆栈中的位置相互影响。例如，您应该在 **SYM_ENCRYPT** 或 **ASYM_ENCRYPT** 协议后面放置等协议，以便 **NAKACK2** 已安全。
INSERT_BEFORE	在其他协议前，将协议插入到堆栈中。影响您指定为插入点的协议。
删除	从堆栈中删除协议。

11.5. 使用 JGROUPS 系统属性

在启动时将系统属性传递给 Data Grid，以调优集群传输。

流程

- **使用 -D<property-name>=<property-value > 参数，根据需要设置 JGroups 系统属性。**

例如，设置自定义绑定端口和 IP 地址，如下所示：

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

11.5.1. 集群传输属性

使用下列属性自定义 JGroups 集群传输：

系统属性	Description	默认值	必填/选填
jgroups.bind.address	集群传输的绑定地址。	**SITE_LOCAL**	选填

系统属性	Description	默认值	必填/选填
jgroups.bind.port	绑定套接字的端口。	**7800**	选填
jgroups.mcast_addr	用于多播的 IP 地址，发现和集群间通信。IP 地址必须是适合 IP 多播的有效"class D"地址。	**239.6.7.8**	选填
jgroups.mcast_port	多播套接字的端口。	**46655**	选填
jgroups.ip_ttl	IP 多播数据包的生存时间(TTL)该值定义了数据包在丢弃之前可以进行的网络跃点数。	2	选填
jgroups.thread_pool.min_threads	线程池的最小线程数量。	0	选填
jgroups.thread_pool.max_threads	线程池的最大线程数。	200	选填
jgroups.join_timeout	等待加入请求成功的最大毫秒数。	2000	选填
jgroups.thread_dumps_threshold	在记录线程转储前，线程池需要满的次数。	10000	选填
jgroups.fd.port-offset	来自 **jgroups.bind.port** 端口的偏移量 FD（失败检测协议）套接字。	**50000** （端口 **57800** ）	选填
jgroups.frag_size	消息中的最大字节数。大于碎片的消息。	60000	选填
jgroups.diag.enabled	启用 JGroups 诊断。	false	选填

其他资源

- JGroups 系统属性

- JGroups 协议列表

11.5.2. 云发现协议的系统属性

使用下列属性为托管平台配置 JGroups 发现协议。

11.5.2.1. Amazon EC2

用于配置 aws.S3_PING 的系统属性。

系统属性	Description	默认值	必填/选填
jgroups.s3.region_name	Amazon S3 区域的名称。	没有默认值。	选填
jgroups.s3.bucket_name	Amazon S3 存储桶的名称。名称必须存在，且是唯一的。	没有默认值。	选填

11.5.2.2. Google Cloud Platform

用于配置 GOOGLE_PING2 的系统属性。

系统属性	Description	默认值	必填/选填
jgroups.google.bucket_name	Google Compute Engine 存储桶的名称。名称必须存在，且是唯一的。	没有默认值。	必需

11.5.2.3. Azure

azure.AZURE_PING' 的系统属性。

系统属性	Description	默认值	必填/选填
jboss.jgroups.azure_ping.storage_account_name	Azure 存储帐户的名称。名称必须存在，且是唯一的。	没有默认值。	必需
jboss.jgroups.azure_ping.storage_access_key	Azure 存储访问密钥的名称。	没有默认值。	必需
jboss.jgroups.azure_ping.container	存储 ping 信息的容器的有效 DNS 名称。	没有默认值。	必需

11.5.2.4. OpenShift

DNS_PING 的系统属性.

系统属性	Description	默认值	必填/选填
jgroups.dns.query	设置返回群集成员的 DNS 记录。	没有默认值。	必需

11.6. 使用内联 JGROUPS 堆栈

您可以将完整的 JGroups 堆栈定义插入到 infinispan.xml 文件中。

流程

- **在您的 infinispan.xml 文件中嵌入自定义 JGroups 堆栈声明。**

```
<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"/>
      <RED/>
      <MPING break_on_coord_rsp="true"
```

```
            mcast_addr="${jgroups.mping.mcast_addr:239.2.4.6}"
            mcast_port="${jgroups.mping.mcast_port:43366}"
            num_discovery_runs="3"
            ip_ttl="${jgroups.udp.ip_ttl:2}"/>
        <MERGE3 />
        <FD_SOCK2 />
        <FD_ALL3 timeout="3000" interval="1000" timeout_check_interval="1000" />
        <VERIFY_SUSPECT2 timeout="1000" />
        <pbcast.NAKACK2 use_mcast_xmit="false" xmit_interval="200"
xmit_table_num_rows="50"
                        xmit_table_msgs_per_row="1024" xmit_table_max_compaction_time="30000"
/>
        <UNICAST3 conn_close_timeout="5000" xmit_interval="200" xmit_table_num_rows="50"
                xmit_table_msgs_per_row="1024" xmit_table_max_compaction_time="30000" />
        <pbcast.STABLE desired_avg_gossip="2000" max_bytes="1M" />
        <pbcast.GMS print_local_addr="false" join_timeout="${jgroups.join_timeout:2000}" />
        <UFC max_credits="4m" min_threshold="0.40" />
        <MFC max_credits="4m" min_threshold="0.40" />
        <FRAG4 />
      </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. 使用外部 JGROUPS 堆栈

在 infinispan.xml 文件中引用定义自定义 JGroups 堆栈的外部文件。

流程

1. **将自定义 JGroups 堆栈文件添加到 $RHDG_HOME/server/conf 目录。**

   另外，您还可以在声明外部堆栈文件时指定绝对路径。

2. **使用 stack-file 元素引用外部堆栈文件。**

```
<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. 加密集群传输

保护集群传输，以便节点与加密消息通信。您还可以配置 Data Grid 集群来执行证书验证，以便只有具有有效身份的节点可以加入。

11.8.1. 使用 TLS 身份保护集群传输

将 SSL/TLS 身份添加到 Data Grid 服务器安全域，并使用它们来保护集群传输。然后，Data Grid Server 集群中的节点交换 SSL/TLS 证书来加密 JGroups 消息，包括 RELAY 消息（如果配置了跨站点复制）。

先决条件

- 安装 Data Grid Server 集群。

流程

1. 创建一个 TLS 密钥存储，其中包含用于识别数据网格服务器的单一证书。

   如果它包含 PKCS#1 或 PKCS#8 格式的私钥，以及证书，以及一个空密码：password=""，您可以使用 PEM 文件。

   注意

   如果密钥存储中的证书不是由公共证书颁发机构(CA)签名，则还必须创建包含签名证书或公钥的信任存储。

2. 将密钥存储添加到 $RHDG_HOME/server/conf 目录中。

3. 将密钥存储添加到 Data Grid Server 配置中的新安全域。

重要

您应该创建专用的密钥存储和安全域，以便 Data Grid Server 端点不像集群传输一样使用相同的安全域。

```
<server xmlns="urn:infinispan:server:14.0">
  <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. 使用 **server:security-realm** 属性指定安全域，将集群传输配置为使用安全域。

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

验证

当您启动 Data Grid Server 时，以下日志消息表示集群正在将安全域用于集群传输：

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

11.8.2. JGroups 加密协议

为保护集群流量，您可以配置 Data Grid 节点，以使用机密密钥加密 JGroups 消息有效负载。

Data Grid 节点可以从以下任一位置获取 secret 密钥：

- 协调器节点（基本指标加密）。
- 共享密钥存储(symmetric 加密)。

从协调器节点检索 secret 密钥

您可以通过在 Data Grid 配置中将 ASYM_ENCRYPT 协议添加到 JGroups 堆栈来配置非对称加密。这允许 Data Grid 集群生成并分发 secret 密钥。

重要

在使用非对称加密时，您还应提供密钥存储，以便节点能够执行证书身份验证和安全地交换密钥。这会防止集群不受中间人(MitM)攻击。

非对称加密可以保护集群流量，如下所示：

1. Data Grid 集群中的第一个节点（协调器节点）会生成 secret 密钥。
2. 加入的节点使用协调器执行证书验证，以相互验证身份。
3. 加入的节点从协调器节点请求 secret 密钥。该请求包括加入节点的公钥。
4. 协调器节点使用公钥加密 secret 密钥，并将其返回到加入的节点。
5. 加入的节点解密并安装 secret 密钥。
6. 节点加入集群，使用 secret 密钥加密和解密信息。

从共享密钥存储检索 secret 密钥

您可以通过在 Data Grid 配置中将 SYM_ENCRYPT 协议添加到 JGroups 堆栈来配置对称加密。这允许 Data Grid 集群从您提供的密钥存储中获取 secret 密钥。

1. 节点在启动时从 Data Grid classpath 上的密钥存储安装 secret 密钥。

2. 节点加入集群，使用 secret 密钥加密和解密信息。

非对称和对称加密的比较

带有证书身份验证的 ASYM_ENCRYPT 提供了额外的加密层，与 SYM_ENCRYPT 进行比较。您提供对请求进行加密的密钥存储，以协调机密密钥的节点。Data Grid 会自动生成该 secret 密钥并处理集群流量，同时让您指定何时生成 secret 密钥。例如，您可以配置集群以在节点离开时生成新的 secret 密钥。这样可确保节点无法绕过证书身份验证，并使用旧密钥加入。

另一方面，SY M_ENCRYPT 比 ASYM_ENCRYPT 快，因为节点不需要与集群协调器交换密钥。SYM_ENCRYPT 的一个潜在的缺点是，当集群成员资格更改时，没有配置可自动生成新的 secret 密钥。用户负责生成和分发节点用于加密集群流量的 secret 密钥。

11.8.3. 使用非对称加密保护集群传输

配置 Data Grid 集群，以生成和分发加密 JGroups 消息的 secret 密钥。

流程

1. 创建具有证书链的密钥存储，使 Data Grid 能够验证节点身份。

2. 将密钥存储放在集群中的每个节点的类路径上。

   对于 Data Grid Server，您可以将密钥存储放在 $RHDG_HOME 目录中。

3. 将 SSL_KEY_EXCHANGE 和 ASYM_ENCRYPT 协议添加到数据网格配置中的 JGroups 堆栈，如下例所示：

```
<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_SUSPECT2. -->
     <SSL_KEY_EXCHANGE keystore_name="mykeystore.jks"
```

```
                keystore_password="changeit"
                stack.combine="INSERT_AFTER"
                stack.position="VERIFY_SUSPECT2"/>
    <!-- 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>
```

验证

当您启动 Data Grid 集群时，以下日志消息表示集群使用 secure JGroups 堆栈：

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

只有使用 ASYM_ENCRYPT，并且可以从协调器节点获取 secret 密钥时，才可以加入集群。否则，以下消息被写入 Data Grid 日志：

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

其他资源

- JGroups 4 手册
- JGroups 4.2 Schema

11.8.4. 使用对称加密保护集群传输

配置 Data Grid 集群，以使用您提供的密钥存储的机密密钥加密 JGroups 消息。

流程

1. 创建包含机密密钥的密钥存储。
2. 将密钥存储放在集群中的每个节点的类路径上。

   对于 Data Grid Server，您可以将密钥存储放在 $RHDG_HOME 目录中。
3. 将 SYM_ENCRYPT 协议添加到数据网格配置中的 JGroups 堆栈。

```
<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_SUSPECT2. -->
      <SYM_ENCRYPT keystore_name="myKeystore.p12"
                   keystore_type="PKCS12"
                   store_password="changeit"
                   key_password="changeit"
                   alias="myKey"
                   stack.combine="INSERT_AFTER"
                   stack.position="VERIFY_SUSPECT2"/>
    </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>
```

验证

当您启动 Data Grid 集群时，以下日志消息表示集群使用 secure JGroups 堆栈：

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

只有使用 SYM_ENCRYPT 并且可以从共享密钥存储获取 secret 密钥时，才可以加入集群。否则，以下消息被写入 Data Grid 日志：

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

其他资源

- JGroups 4 手册
- JGroups 4.2 Schema

11.9. 集群流量的 TCP 和 UDP 端口

Data Grid 将以下端口用于集群传输消息：

默认端口	协议	Description
7800	TCP/UDP	JGroups 集群绑定端口
46655	UDP	JGroups 多播

跨站点复制

数据网格为 JGroups RELAY2 协议使用以下端口：

7900

对于在 OpenShift 上运行的 Data Grid 集群。

7800

如果将 UDP 用于节点和 TCP 间的流量，用于集群间的流量。

7801

如果将 TCP 用于节点和 TCP 间的流量，用于集群间的流量。

第 12 章 创建远程缓存

当您在运行时创建远程缓存时，Data Grid 服务器会在集群间同步您的配置，以便所有节点都具有副本。因此，您应该总是使用以下机制动态创建远程缓存：

- Data Grid Console
- Data Grid 命令行界面(CLI)
- 热环或 HTTP 客户端

12.1. 默认缓存管理器

Data Grid Server 提供了一个默认缓存管理器，用于控制远程缓存的生命周期。启动 Data Grid Server 会自动实例化 Cache Manager，以便您可以创建和删除远程缓存和其他资源，如 Protobuf 模式。

启动 Data Grid Server 并添加用户凭证后，您可以查看 Cache Manager 的详情，并从 Data Grid Console 获取集群信息。

- 在任意浏览器中打开 127.0.0.1:11222。

您还可以通过命令行界面(CLI)或 REST API 获取有关缓存管理器的信息：

CLI

在默认容器中运行 describe 命令。

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

REST

在任意浏览器中打开 127.0.0.1:11222/rest/v2/cache-managers/default/。

默认缓存管理器配置

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

```
{
  "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. 使用 DATA GRID 控制台创建缓存

使用数据网格控制台在任何 Web 浏览器的直观直观界面中创建远程缓存。

先决条件

- 创建具有管理权限的 Data Grid 用户。
- 至少启动一个数据网格服务器实例。
- 具有数据网格缓存配置。

流程

1. 在任意浏览器中打开 127.0.0.1:11222/console/。
2. 选择 Create Cache 并按照 Data Grid Console 指南的步骤进行操作。

12.3. 使用 DATA GRID CLI 创建远程缓存

使用 Data Grid 命令行界面(CLI)在 Data Grid Server 中添加远程缓存。

先决条件

- 创建具有管理权限的 Data Grid 用户。
- 至少启动一个数据网格服务器实例。
- 具有数据网格缓存配置。

流程

1. 启动 CLI。

```
bin/cli.sh
```

2. 运行 connect 命令，并在系统提示时输入您的用户名和密码。
3. 使用 create cache 命令创建远程缓存。

   例如，从名为 mycache.xml 的文件创建一个名为 "mycache" 的缓存，如下所示：

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

验证

1. 使用 ls 命令列出所有远程缓存。

```
ls caches
mycache
```

2. 使用 describe 命令查看缓存配置。

```
describe caches/mycache
```

12.4. 从 HOT ROD 客户端创建远程缓存

使用 Data Grid Hot Rod API 从 Java、C++、.NET/C{hash}、JS 客户端等数据网格服务器上创建远程缓存。

此流程演示了如何使用 Hot Rod Java 客户端在第一次引导时创建远程缓存。您可以在 Data Grid Tutorials 中找到其他 Hot Rod 客户端的代码示例。

先决条件

- 创建具有管理权限的 Data Grid 用户。
- 至少启动一个数据网格服务器实例。
- 具有数据网格缓存配置。

流程

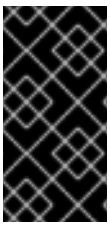
- 调用 remoteCache （） 方法，作为 ConfigurationBuilder 的一部分。
- 在类路径上的 hotrod-client.properties 文件中设置 configuration 或 configuration_uri 属性。

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

如果远程缓存的名称包含 . 字符，在使用 hotrod-client.properties 文件时，必须将其括在方括号中。

其他资源

- 热 Rod 客户端配置
- org.infinispan.client.hotrod.configuration.RemoteCacheConfigurationBuilder

12.5. 使用 REST API 创建远程缓存

使用 Data Grid REST API 在 Data Grid Server 上创建来自任何合适的 HTTP 客户端的远程缓存。

先决条件

- 创建具有管理权限的 Data Grid 用户。
- 至少启动一个数据网格服务器实例。
- 具有数据网格缓存配置。

流程

- 使用有效负载中的缓存配置调用 POST 请求到 /rest/v2/caches/<cache_name >。

其他资源

- 使用 REST API 创建和管理缓存

第 13 章 在 DATA GRID SERVER 上运行脚本和任务

通过命令行界面(CLI)和 Hot Rod 或 REST 客户端，将任务和脚本添加到 Data Grid 服务器部署中。您可以将任务实施为自定义 Java 类，也可以使用 JavaScript 等语言定义脚本。

13.1. 在 DATA GRID 服务器部署中添加任务

将您的自定义服务器任务类添加到 Data Grid Server。

先决条件

- 如果数据网格服务器正在运行，停止该服务器。

  Data Grid Server 不支持自定义类的运行时部署。

流程

1. 添加包含服务器任务的完全限定域名的 META-INF/services/org.infinispan.tasks.ServerTask 文件，例如：

```
example.HelloTask
```

2. 将服务器任务打包在 JAR 文件中。
3. 将 JAR 文件复制到 Data Grid Server 安装的 $RHDG_HOME/server/lib 目录中。
4. 将类添加到 Data Grid 配置中的反序列化 allow 列表中。另外，也可使用系统属性设置 allow 列表。

参考

- 将 Java 类添加到序列化 Allow 列表
- Data Grid Configuration Schema

13.1.1. Data Grid Server 任务

Data Grid Server 任务是扩展 org.infinispan.tasks.ServerTask 接口的类，通常包括以下方法调用：

setTaskContext()

允许访问执行上下文信息，包括任务参数、执行任务的缓存引用等。在大多数情况下，实施会在本地存储此信息，并在实际执行任务时使用它。使用 SHARED 实例化模式时，任务应使用 ThreadLocal 来存储 TaskContext 进行并发调用。

getName()

为任务返回唯一名称。客户端使用这些名称调用任务。

getExecutionMode()

返回任务的执行模式。

- TaskExecutionMode.ONE_NODE 仅处理请求执行脚本的节点。虽然脚本仍然可以调用集群操作。这是默认值。
- TaskExecutionMode.ALL_NODES 数据网格使用集群执行器在节点间运行脚本。例如，调用流处理的服务器任务需要在单个节点上执行，因为流处理被分发到所有节点。

getInstantiationMode()

返回任务的实例化模式。

- TaskInstantiationMode.SHARED 创建一个实例，它被重复使用，用于在同一服务器上执行每个任务。这是默认值。
- TaskInstantiationMode.ISOLATED 会为每个调用创建新实例。

call()

计算结果。此方法在 java.util.concurrent.Callable 接口中定义，并通过 server 任务调用。

重要

服务器任务实施必须遵循服务加载模式要求。例如，实现必须有一个零参数构造器。

以下 HelloTask 类实施提供了一个示例任务，它有一个参数。它还演示了使用 ThreadLocal 存储 TaskContext 进行并发调用。

```
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";
  }

}
```

参考

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

13.2. 在 DATA GRID 服务器部署中添加脚本

使用 命令行界面向 Data Grid 服务器添加脚本。

先决条件

数据网格服务器在 ___script_cache 缓存中保存脚本。如果启用缓存授权，用户必须具有 CREATE 权限才能添加到 ___script_cache。

如果您使用默认的授权设置，则至少为用户分配 deployer 角色。

流程

1. 根据需要定义脚本。

   例如，创建一个名为 multiplication.js 的文件，该文件在单个 Data Grid 服务器上运行，具有两个参数，并使用 JavaScript 来多地给定值：

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

2. 创建与 Data Grid 的 CLI 连接。

3. 使用 task 命令上传脚本，如下例所示：

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

4. 验证您的脚本是否可用。

```
ls tasks
multiplication
```

13.2.1. Data Grid Server 脚本

数据网格服务器脚本基于 javax.script API，兼容任何基于 JVM 的 ScriptEngine 实施。

您好世界

以下是在单一数据网格服务器上运行的简单示例，有一个参数，并使用 JavaScript：

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

运行上述脚本时，您将传递一个 greetee 参数的值，Data Grid 则返回 "Hello ${value}"。

13.2.1.1. 脚本元数据

元数据提供有关 Data Grid 服务器在运行时使用的脚本的附加信息。

脚本元数据是 attribute=value 对，您添加到脚本第一行中的注释中，例如：

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

- 使用符合脚本语言的评论样式(//、 ；；、#)。
- 使用逗号分隔 attribute=value 对。
- 使用单个(')或双引号字符分隔值。

表 13.1. 元数据属性

属性	Description
模式	定义执行模式，并包含以下值： 仅限 **本地** 处理请求的节点执行该脚本。虽然脚本仍然可以调用集群操作。 **分布式** 数据网格使用集群执行程序在节点间运行脚本。
语言	指定执行脚本的脚本引擎。
extension	指定文件名扩展，作为设置 ScriptEngine 的替代方法。
role	指定用户必须执行脚本的角色。

属性	Description
parameters	指定此脚本的有效参数名称的数组。指定此列表中不包含的参数的调用会导致异常。
datatype	（可选）设置用于存储数据的 MediaType (MIME 类型)以及参数和返回值。此属性可用于仅用于支持特定数据格式的远程客户端。 目前，您只能设置 **text/plain; charset=utf-8** 以使用 String UTF-8 格式数据。

13.2.1.2. 脚本绑定

数据网格将内部对象公开为脚本执行的绑定。

绑定	Description
缓存	指定运行该脚本的缓存。
marshaller	指定用于将数据序列化到缓存的 marshaller。
cacheManager	指定缓存的 **cacheManager**。
scriptingManager	指定运行该脚本的脚本管理器的实例。您可以使用此绑定从脚本运行其他脚本。

13.2.1.3. 脚本参数

数据网格允许您将指定参数作为运行脚本的绑定传递。

参数是 name、value 对，其中 name 是一个字符串，value 是 marshaller 可以解释的任何值。

以下示例脚本有两个参数，分别是 multiplicand 和 multiplier。该脚本取乘 的值并 乘以倍数乘以 倍数。

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

运行前面的脚本时，Data Grid 会响应表达式评估的结果。

13.2.2. 以编程方式创建脚本

使用 Hot Rod RemoteCache 接口添加脚本，如下例所示：

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

参考

org.infinispan.client.hotrod.RemoteCache

13.3. 运行脚本和任务

使用命令行界面在 Data Grid Server 部署上运行任务和脚本。或者，您也可以从 Hot Rod 客户端执行脚本和任务。

先决条件

- 在 Data Grid Server 中添加脚本或任务。

流程

1. 创建与 Data Grid 的 CLI 连接。
2. 使用 task 命令运行任务和脚本，如下例所示：
   - 执行名为 multiplier.js 的脚本，并指定两个参数：

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

   - 执行名为 @@cache@names 的任务，以检索所有可用缓存的列表：

     -

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

程序执行

- **使用 Hot Rod RemoteCache 接口调用 execute （） 方法来运行脚本，如下例所示：**

脚本执行

```
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);
```

任务执行

```
// 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);
```

其他资源

- org.infinispan.client.hotrod.RemoteCache

第 14 章 配置 DATA GRID 服务器日志记录

数据网格服务器使用 Apache Log4j 2 提供可配置的日志记录机制，用于捕获环境详情并记录缓存操作以进行故障排除和根本原因分析。

14.1. DATA GRID SERVER 日志文件

数据网格将服务器日志写入 $RHDG_HOME/server/log 目录中的以下文件：

server.log

人类可读的格式的消息，包括与服务器启动相关的引导日志。Data Grid 在您启动服务器时会创建此文件。

server.log.json

JSON 格式的消息，可让您解析和分析数据网格日志。当启用 JSON-FILE 附加器时，Data Grid 会创建此文件。

14.1.1. 配置 Data Grid 服务器日志

数据网格使用 Apache Log4j 技术编写服务器日志消息。您可以在 log4j2.xml 文件中配置服务器日志。

流程

1. 使用任何文本编辑器打开 $RHDG_HOME/server/conf/log4j2.xml。
2. 根据需要更改服务器日志记录。
3. 保存并关闭 log4j2.xml。

其他资源

- Apache Log4j 手册

14.1.2. 日志级别

日志级别表示消息的性质和严重性。

日志级别	Description
TRACE	精细调试消息，通过应用程序捕获单个请求流。
DEBUG	用于常规调试的消息，与单个请求无关。
INFO	有关应用程序总体进度的消息，包括生命周期事件。
WARN	导致错误或降级性能的事件。
ERROR	防止操作或活动无法成功但不会阻止应用程序运行的错误条件。
FATAL	可能导致关键服务故障和应用程序关闭的事件。

除了上述各个消息级别外，该配置还允许两个值更多： ALL 可以包括所有消息，使用 OFF 来排除所有消息。

14.1.3. 数据网格日志记录类别

数据网格为 INFO、WARN、ERROR、FATAL 级消息提供按功能区域组织日志的类别。

org.infinispan.CLUSTER

特定于数据网格集群的消息，包括状态转移操作、重新平衡事件、分区等等。

org.infinispan.CONFIG

特定于数据网格配置的消息。

org.infinispan.CONTAINER

特定于数据容器的消息，包括过期和驱除操作、缓存侦听器通知、事务等。

org.infinispan.PERSISTENCE

特定于缓存加载器和存储的消息。

org.infinispan.SECURITY

特定于数据网格安全性的消息.

org.infinispan.SERVER

特定于数据网格服务器的消息.

org.infinispan.XSITE

特定于跨站点复制操作的消息。

14.1.4. 日志附加器

Log appenders 定义 Data Grid Server 如何记录日志消息。

控制台

将日志消息写入主机标准输出(stdout)或标准错误(stderr)流。
默认使用 org.apache.logging.log4j.core.appender.ConsoleAppender 类。

FILE

将日志消息写入文件。
默认使用 org.apache.logging.log4j.core.appender.RollingFileAppender 类。

JSON-FILE

以 JSON 格式将日志消息写入文件。
默认使用 org.apache.logging.log4j.core.appender.RollingFileAppender 类。

14.1.5. 日志模式格式

CONSOLE 和 FILE appenders 使用 PatternLayout 来根据 模式 格式化日志消息。

示例是 FILE 附加器中的默认模式：
%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p (%t)[%c114] %m%throwable%n

- %d{yyyy-MM-dd HH:mm:ss,SSS} 增加了当前的时间和日期。
- %-5p 指定日志级别，与右侧一致。

- % T 添加当前线程的名称。
- %c{1} 添加日志记录类别的短名称。
- % M 添加日志消息。
- %rowable 添加异常堆栈跟踪。
- %n 添加新行。

PatternLayout 文档 中已完全描述模式。

14.1.6. 启用 JSON 日志处理程序

Data Grid Server 提供了一个日志处理程序，用于以 JSON 格式编写消息。

先决条件

- 如果数据网格服务器正在运行，停止该服务器。
您无法动态启用日志处理程序。

流程

1. 使用任何文本编辑器打开 $RHDG_HOME/server/conf/log4j2.xml。
2. 取消注释 JSON-FILE 附加程序，并注释掉 FILE appender：

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

3. （可选）根据需要配置 JSON 附加程序和 JSON 布局。

4. 保存并关闭 log4j2.xml。

当您启动 Data Grid 时，它会将每个日志消息作为 JSON 映射写入以下文件中的 JSON 映射：$114G_HOME/server/log/server.log.json

其他资源

- RollingFileAppender
- JSONLayout

14.2. 访问日志

访问日志记录 Hot Rod 及 $RHDG_HOME/server/log 目录中文件的 REST 端点的所有入站客户端请求。

org.infinispan.HOTROD_ACCESS_LOG

将 Hot Rod 访问消息写入 hotrod-access.log 文件中的日志记录类别。

org.infinispan.REST_ACCESS_LOG

将 REST 访问消息写入 rest-access.log 文件的日志记录类别。

14.2.1. 启用访问日志

要记录 Hot Rod 和 REST 端点访问消息，您需要在 log4j2.xml 中启用日志记录类别。

流程

1. 使用任何文本编辑器打开 $RHDG_HOME/server/conf/log4j2.xml。
2. 将 org.infinispan.HOTROD_ACCESS_LOG 和 org.infinispan.REST_ACCESS_LOG 日志记录类别的级别更改为 TRACE。

3. 保存并关闭 **log4j2.xml**。

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

14.2.2. 访问日志属性

访问日志的默认格式如下：

```
%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
```

以上格式创建了日志条目，如下所示：

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

日志记录属性使用 **%X{name}** 表示法，并允许您修改访问日志的格式。以下是默认的日志属性：

属性	Description
address	**X-Forwarded-For** 标头或客户端 IP 地址。
user	主体名称（如果使用身份验证）
方法	使用的方法。**PUT**、**GET** 等。
protocol	使用的协议。**HTTP/1.1**,**HTTP/2**,**HOTROD/2.9** 等等。
status	REST 端点的 HTTP 状态代码。Hot Rod 端点 **确定** 或异常.
requestSize	请求的大小（以字节为单位）。
responseSize	响应的大小（以字节为单位）。
duration	服务器处理请求所需的毫秒数。

提示

使用前缀为 h: 的标头名称来记录请求中包含的标头，例如 %X{h:User-Agent}。

14.3. 审计日志

审计日志可让您跟踪数据网格服务器部署的更改，以便您了解何时更改以及哪些用户对其进行更改。启用并配置审计日志记录服务器配置事件和管理操作。

org.infinispan.AUDIT

将安全审核消息写入 $RHDG_HOME/server/log 目录中的 audit.log 文件中的日志记录类别。

14.3.1. 启用审计日志记录

要记录安全审计消息，您需要在 log4j2.xml 中启用日志记录类别。

流程

1. 使用任何文本编辑器打开 $RHDG_HOME/server/conf/log4j2.xml。
2. 将 org.infinispan.AUDIT logging 类别的级别更改为 INFO。
3. 保存并关闭 log4j2.xml。

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

14.3.2. 配置审计日志的附加器

Apache Log4j 提供不同的附加器，可用于将审计消息发送到默认日志文件以外的目的地。例如，如果要将审计日志发送到 syslog 守护进程、JDBC 数据库或 Apache Kafka 服务器，您可以在 log4j2.xml 中配置附加程序。

流程

1. 使用任何文本编辑器打开 $RHDG_HOME/server/conf/log4j2.xml。

2. 注释或删除默认 AUDIT-FILE 滚动文件 appender。

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

3. 为审计消息添加所需的日志记录附加程序。

   例如，您可以为 Kafka 服务器添加日志附加程序，如下所示：

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

4. 保存并关闭 log4j2.xml。

其他资源

- Log4j Appenders

14.3.3. 使用自定义审计日志记录实现

如果配置 Log4j 附加器没有满足您的需要，您可以创建 org.infinispan.security.AuditLogger API 的自定义实现。

先决条件

- 根据需要实施 org.infinispan.security.AuditLogger，并将它打包在 JAR 文件中。

流程

1. 将您的 JAR 添加到 Data Grid Server 安装中的 server/lib 目录中。

2.

指定自定义审计日志记录器的完全限定类名称，作为 cache 容器安全配置中 authorization 元素的值。

例如，以下配置定义了 my.package.CustomAuditLogger 作为日志记录审核信息的类：

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

其他资源

- org.infinispan.security.AuditLogger

第 15 章 为 DATA GRID SERVER 集群执行滚动升级

执行数据网格集群的滚动升级，以在版本之间更改，而无需停机或数据丢失，并通过 Hot Rod 协议迁移数据。

15.1. 设置目标 DATA GRID 集群

创建使用您计划升级的 Data Grid 版本的集群，然后使用远程缓存存储将源集群连接到目标集群。

先决条件

- 使用目标集群所需的版本安装 Data Grid Server 节点。

重要

确保目标集群的网络属性与源集群的重叠。您应该在 JGroups 传输配置中为目标集群指定唯一名称。根据您的环境，您还可以使用不同的网络接口和端口偏移来分隔目标和源集群。

流程

1. 创建一个远程缓存存储配置，采用 JSON 格式，允许目标集群连接到源集群。

   目标集群上的远程缓存存储使用 Hot Rod 协议从源集群检索数据。

```
{
 "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. 使用 Data Grid 命令行界面(CLI)或 REST API 将远程缓存存储配置添加到目标集群，以便它可以连接到源集群。

- CLI：在目标集群中使用 迁移的集群连接 命令。

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

- REST API：使用 rolling-upgrade/source-connection 方法在有效负载中包含远程存储配置的 POST 请求。

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

3. 对您要迁移的每个缓存重复前面的步骤。

4. 将客户端切换到目标集群，以便它开始处理所有请求。

a. 使用目标集群的位置更新客户端配置。

b. 重新启动客户端。

其他资源

- 远程缓存存储配置模式

15.2. 将数据同步到目标集群

当您设置目标 Data Grid 集群并将其连接到源集群时，目标集群可以使用远程缓存存储和按需加载数据来处理客户端请求。要完全将数据迁移到目标集群，因此您可以弃用源集群，您可以同步数据。此操作会从源集群读取数据并将其写入目标集群。数据会并行迁移到目标集群中的所有节点，每个节点都接收数据子集。您必须对您要迁移到目标集群的每个缓存执行同步。

先决条件

- 使用适当的 Data Grid 版本设置目标集群。

流程

1. 开始使用 Data Grid 命令行界面(CLI)或 REST API 将您要迁移到目标集群的每个缓存进行同步。

   - CLI：使用 migrate cluster synchronize 命令。

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

   - REST API：使用带有 POST 请求的 ?action=sync-data 参数。

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

   当操作完成后，Data Grid 按照复制到目标集群的条目总数进行响应。

2. 断开目标集群中的每个节点与源集群的连接。

   - CLI：使用 migrate cluster disconnect 命令。

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

   - REST API：调用 DELETE 请求。

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

后续步骤

同步源集群中的所有数据后，滚动升级过程已完成。现在，您可以弃用源集群。

第 16 章 DATA GRID SERVER 部署故障排除

收集数据网格服务器部署的诊断信息，并执行故障排除步骤来解决问题。

16.1. 从 DATA GRID SERVER 获取诊断报告

数据网格服务器在 tar.gz 存档中提供汇总报告，其中包含有关服务器实例和主机系统的诊断信息。除了配置和日志文件外，报告还提供有关 CPU、内存、打开文件、网络套接字和路由、线程的详细信息。

流程

1. 创建与 Data Grid 服务器的 CLI 连接。
2. 使用 服务器 report 命令下载 tar.gz 归档：

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

该命令使用报告名称进行响应，如下例所示：

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

3. 将 tar.gz 文件移动到文件系统上的适当位置。
4. 使用任何存档工具提取 tar.gz 文件。

16.2. 在运行时更改数据网格服务器日志记录配置

在运行时修改 Data Grid Server 的日志记录配置，以临时调整日志记录来排除问题并执行根本原因分析。

通过 CLI 修改日志记录配置是仅运行时的操作，这意味着更改：

- 不会保存到 log4j2.xml 文件。重启服务器节点或整个集群会将日志记录配置重置为

log4j2.xml 文件中的默认属性。

- 仅在调用 CLI 时应用到集群中的节点。更改日志记录配置后加入集群的节点使用默认属性。

流程

1. 创建与 Data Grid 服务器的 CLI 连接。
2. 使用 日志记录 进行必要的调整。

- 列出服务器中定义的所有附加器：

```
logging list-appenders
```

命令提供 JSON 响应，如下所示：

```
{
 "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"
 }
}
```

- 列出服务器中定义的所有日志记录器配置：

```
logging list-loggers
```

命令提供 JSON 响应，如下所示：

```
[ {
  "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" ]
} ]
```

- **使用 set 子命令添加和修改日志记录器配置**

  例如，以下命令将 org.infinispan 软件包的日志级别设置为 DEBUG ：

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

- **使用 remove 子命令删除现有日志记录器配置。**

  例如，以下命令删除 org.infinispan logger 配置，这意味着改为使用 root 配置：

```
logging remove org.infinispan
```

16.3. 通过 CLI 收集资源统计

您可以使用 stats 命令检查一些 Data Grid Server 资源相关的 server-collected 统计信息。

使用提供统计（容器、缓存）的资源上下文中的 stats 命令，或使用这类资源的路径：

```
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. 通过 REST 访问集群健康状况

通过 REST API 获取 Data Grid 集群的健康状态。

流程

- **调用 GET 请求以检索集群健康状况。**

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

Data Grid 使用 JSON 文档响应，如下所示：

```
{
  "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"
    }
  ]
}
```

提示

获取缓存管理器状态，如下所示：

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

参考

如需更多信息，请参阅 *REST v2 （版本 2）API* 文档。

16.5. 通过 JMX 访问集群健康状况

通过 JMX 检索数据网格集群运行状况统计数据。

流程

1. 使用 JConsole 等任何 JMX 功能工具连接到 Data Grid 服务器，然后导航到以下对象：

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

2. 选择可用的 MBeans 来检索集群健康统计。

第 17 章 参考

17.1. DATA GRID SERVER 8.4.1 README

有关 Data Grid Server 14.0.6Final-redhat-00001 分发的信息。

17.1.1. 要求

数据网格服务器需要 JDK 11 或更高版本。

17.1.2. 启动服务器

使用 服务器 脚本运行 Data Grid 服务器实例。

UNIX/ Linux

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

Windows

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

提示

包含 --help 或 -h 选项以查看命令参数。

17.1.3. 停止服务器

使用带有 CLI 的 shutdown 命令执行安全关闭。

或者，从终端输入 Ctrl-C 以中断服务器进程或通过 TERM 信号将它终止。

17.1.4. 配置

服务器配置通过以下特定于服务器的元素来扩展 Data Grid 配置：

cache-container

定义用于管理缓存生命周期的缓存容器。

端点

为客户端协议启用和配置端点连接器。

安全

配置端点安全域。

socket-bindings

将端点连接器映射到接口和端口。

默认配置文件为 **$RHDG_HOME/server/conf/infinispan.xml**。

使用带有 **-c** 参数的不同配置文件，如下例所示，启动没有集群功能的服务器：

UNIX/ Linux

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

Windows

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

17.1.5. 绑定地址

默认情况下，Data Grid Server 绑定到您网络中的环回 IP 地址 localhost。

使用 -b 参数设置不同的 IP 地址，如下例所示，绑定到所有网络接口：

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. 绑定端口

数据网格服务器默认侦听端口 11222。

使用 -p 参数设置备选端口：

UNIX/ Linux

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

Windows

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

17.1.7. 集群地址

数据网格服务器配置定义了群集传输，因此同一网络上的多个实例可以互相发现，并自动组成集群。

使用 **-k** 参数更改集群流量的 **IP** 地址：

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. 集群堆栈

JGroups 堆栈配置群集传输协议。**Data Grid** 服务器默认使用 **tcp** 堆栈。

使用带有 -j 参数的替代集群堆栈，如下例所示，使用 UDP 进行集群传输：

UNIX/ Linux

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

Windows

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

17.1.9. 身份验证

数据网格服务器需要身份验证。

使用 CLI 创建用户名和密码，如下所示：

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. 服务器主目录

Data Grid Server 使用 infinispan.server.home.path 来查找主机文件系统上的服务器分发的内容。

服务器主目录称为 $RHDG_HOME，包含以下文件夹：

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

目录	Description
/bin	包含启动服务器和 CLI 的脚本。
/boot	包含用于启动服务器的 JAR 文件。
/docs	提供配置示例、架构、组件许可证和其他资源。
/lib	包含服务器内部需要的 JAR 文件。 不要将自定义 JAR 文件放在此文件夹中。
/server	为 Data Grid Server 实例提供根文件夹。
/static	包含数据网格控制台的静态资源。

17.1.11. 服务器根目录

数据网格服务器使用 infinispan.server.root.path 来查找 Data Grid Server 实例的配置文件和数据。

您可以在同一目录或不同目录中创建多个服务器根文件夹，然后使用 -s 或 --server-root 参数指定位置，如下例所示：

UNIX/ Linux

```
$RHDG_HOME/bin/server.sh -s server2
```

Windows

```
$RHDG_HOME\bin\server.bat -s server2
```

每个服务器根目录都包含以下文件夹：

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

目录	Description	系统属性覆盖
/server/conf	包含服务器配置文件。	**infinispan.server.config.path**
/server/data	包含按容器名称组织的数据文件。	**infinispan.server.data.path**
/server/lib	包含服务器扩展文件。 该目录以递归方式扫描，并用作类路径。	**Infinispan.server.lib.path** 使用以下分隔符分隔多个路径 **：** 在 Windows 上的 Unix / Linux **；**中
/server/log	包含服务器日志文件。	**infinispan.server.log.path**

17.1.12. 日志记录

使用 server/conf 文件夹中的 log4j2.xml 文件配置 Data Grid Server 日志。

使用 --logging-config=<path_to_logfile > 参数来使用自定义路径，如下所示：

UNIX/ Linux

```
$RHDG_HOME/bin/server.sh --logging-config=/path/to/log4j2.xml
```

提示

为**确保自定义路径生效，请不要使用 ~ 快捷方式。**

Windows

```
$RHDG_HOME\bin\server.bat --logging-config=path\to\log4j2.xml
```