Red Hat Data Grid 8.0

Data Grid Server Guide

Data Grid Documentation

Last Updated: 2022-05-10

Red Hat Data Grid 8.0 Data Grid Server Guide

Data Grid Documentation

Legal Notice

Abstract

Configure, run, and monitor Data Grid servers and access your data from remote client applications.

Table of Contents

CHAPTER 1. RED HAT DATA GRID 5
1.1. DATA GRID DOCUMENTATION 5
1.2. DATA GRID DOWNLOADS 5

CHAPTER 2. GETTING STARTED WITH DATA GRID SERVER 6
2.1. DATA GRID SERVER REQUIREMENTS 6
2.2. DOWNLOADING SERVER DISTRIBUTIONS 6
2.3. INSTALLING DATA GRID SERVER 6
2.4. RUNNING DATA GRID SERVERS 6
2.4.1. Starting Data Grid Servers 7
2.4.2. Verifying Data Grid Cluster Discovery 7
2.5. PERFORMING OPERATIONS WITH THE DATA GRID CLI 8
2.5.1. Starting the Data Grid CLI 8
2.5.2. Connecting to Data Grid Servers 8
2.5.3. Creating Caches from Templates 8
2.5.4. Adding Cache Entries 9
2.5.5. Shutting Down Data Grid Servers 10

CHAPTER 3. CONFIGURING DATA GRID SERVER NETWORKING 11
3.1. SERVER INTERFACES 11
3.1.1. Address Strategy 11
3.1.2. Loopback Strategy 11
3.1.3. Non-Loopback Strategy 11
3.1.4. Network Address Strategy 12
3.1.5. Any Address Strategy 12
3.1.6. Link Local Strategy 12
3.1.7. Site Local Strategy 12
3.1.8. Match Host Strategy 12
3.1.9. Match Interface Strategy 13
3.1.10. Match Address Strategy 13
3.1.11. Fallback Strategy 13
3.1.12. Changing the Default Bind Address for Data Grid Servers 14
3.2. SOCKET BINDINGS 14
3.2.1. Specifying Port Offsets 15
3.3. DATA GRID PROTOCOL HANDLING 15
3.3.1. Configuring Clients for ALPN 15

CHAPTER 4. CONFIGURING DATA GRID SERVER ENDPOINTS 17
4.1. DATA GRID ENDPOINTS 17
4.1.1. Hot Rod 17
4.1.2. REST 17
4.1.3. Protocol Comparison 17
4.2. ENDPOINT CONNECTORS 18
4.2.1. Hot Rod Connectors 18
4.2.2. REST Connectors 19

CHAPTER 5. MONITORING DATA GRID SERVERS 20
5.1. WORKING WITH DATA GRID SERVER LOGS 20
5.1.1. Data Grid Log Files 20
5.1.2. Configuring Data Grid Log Properties 20
5.1.2.1. Log Levels 20
5.1.2.2. Data Grid Log Categories 21

5.1.2.3. Log Appenders 21
5.1.2.4. Log Patterns 22
5.1.2.5. Enabling and Configuring the JSON Log Handler 22
5.1.3. Access Logs 22
5.1.3.1. Enabling Access Logs 23
5.1.3.2. Access Log Properties 23
5.2. CONFIGURING STATISTICS, METRICS, AND JMX 24
5.2.1. Enabling Data Grid Statistics 24
5.2.2. Enabling Data Grid Metrics 24
5.2.3. Collecting Data Grid Metrics 25
5.2.4. Enabling and Configuring JMX 26
5.2.4.1. Naming Multiple Cache Managers 27
5.2.4.2. Registering MBeans In Custom MBean Servers 27
5.2.4.3. Data Grid MBeans 28
5.3. RETRIEVING SERVER HEALTH STATISTICS 28
5.3.1. Accessing the Health API via JMX 28
5.3.2. Accessing the Health API via REST 28

CHAPTER 6. SECURING DATA GRID SERVERS 30
6.1. CACHE AUTHORIZATION 30
6.1.1. Cache Authorization Configuration 30
6.2. DEFINING DATA GRID SERVER SECURITY REALMS 31
6.2.1. Property Realms 31
6.2.1.1. Adding Users to Property Realms 32
6.2.2. LDAP Realms 32
6.2.2.1. LDAP Realm Principal Rewriting 34
6.2.3. Trust Store Realms 34
6.2.4. Token Realms 35
6.3. CREATING DATA GRID SERVER IDENTITIES 36
6.3.1. Setting Up SSL Identities 36
6.3.1.1. SSL Identity Configuration 37
6.3.1.2. Automatically Generating Keystores 37
6.3.1.3. Tuning SSL Protocols and Cipher Suites 38
6.3.2. Setting Up Kerberos Identities 39
6.3.2.1. Kerberos Identity Configuration 40
6.4. CONFIGURING ENDPOINT AUTHENTICATION MECHANISMS 40
6.4.1. Setting Up Hot Rod Authentication 40
6.4.1.1. Hot Rod Authentication Configuration 41
6.4.1.2. Hot Rod Endpoint Authentication Mechanisms 42
6.4.1.3. SASL Quality of Protection (QoP) 43
6.4.1.4. SASL Policies 43
6.4.2. Setting Up REST Authentication 44
6.4.2.1. REST Authentication Configuration 45
6.4.2.2. REST Endpoint Authentication Mechanisms 45

CHAPTER 7. REMOTELY EXECUTING SERVER-SIDE TASKS 47
7.1. CREATING SERVER TASKS 47
7.1.1. Server Tasks 47
7.1.2. Deploying Server Tasks to Data Grid Servers 48
7.2. CREATING SERVER SCRIPTS 49
7.2.1. Server Scripts 49
7.2.1.1. Script Metadata 49
7.2.1.2. Script Bindings 50

7.2.1.3. Script Parameters 50
7.2.2. Adding Scripts to Data Grid Servers 50
7.2.3. Programmatically Creating Scripts 51
7.3. RUNNING SERVER-SIDE TASKS AND SCRIPTS 51
7.3.1. Running Tasks and Scripts 51
7.3.2. Programmatically Running Scripts 52
7.3.3. Programmatically Running Tasks 52

CHAPTER 8. PERFORMING ROLLING UPGRADES FOR DATA GRID SERVERS 54
8.1. SETTING UP TARGET CLUSTERS 54
8.1.1. Remote Cache Stores for Rolling Upgrades 54
8.2. SYNCHRONIZING DATA TO TARGET CLUSTERS 55

CHAPTER 9. DATA GRID SERVER REFERENCE 57
9.1. DATA GRID SERVER 57
9.1.1. Directory Structure 57
9.1.2. Server Paths 57
9.1.3. Command Arguments 58
9.1.4. Logging Configuration 58
9.1.5. Configuration 58
9.1.6. Additional Details 59

CHAPTER 1. RED HAT DATA GRID

Data Grid is a high-performance, distributed in-memory data store.

Schemaless data structure

Flexibility to store different objects as key-value pairs.

Grid-based data storage

Designed to distribute and replicate data across clusters.

Elastic scaling

Dynamically adjust the number of nodes to meet demand without service disruption.

Data interoperability

Store, retrieve, and query data in the grid from different endpoints.

1.1. DATA GRID DOCUMENTATION

Documentation for Data Grid is available on the Red Hat customer portal.

- Data Grid 8.0 Documentation
- Data Grid 8.0 Component Details
- Supported Configurations for Data Grid 8.0

1.2. DATA GRID DOWNLOADS

Access the Data Grid Software Downloads on the Red Hat customer portal.

NOTE

You must have a Red Hat account to access and download Data Grid software.

CHAPTER 2. GETTING STARTED WITH DATA GRID SERVER

Quickly set up Data Grid server and learn the basics.

2.1. DATA GRID SERVER REQUIREMENTS

Check host system requirements for the Data Grid server.

See the Red Hat Data Grid Supported Configurations .

2.2. DOWNLOADING SERVER DISTRIBUTIONS

The Data Grid server distribution is an archive of Java libraries (**JAR** files), configuration files, and a **data** directory.

Procedure

1. Access the Red Hat customer portal.
2. Download the Red Hat Data Grid 8.0 server from the software downloads section.

Verification

Use the checksum to verify the integrity of your download.

1. Run the **md5sum** or **sha256sum** command with the server download archive as the argument, for example:

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

2. Compare with the **MD5** or **SHA-256** checksum value on the Data Grid **Software Details** page.

Reference

Data Grid Server README describes the contents of the server distribution.

2.3. INSTALLING DATA GRID SERVER

Extract the Data Grid server archive to any directory on your host.

Procedure

Use any extraction tool with the server archive, for example:

```
$ unzip infinispan-server-${version}.zip
```

The resulting directory is your **$RHDG_HOME**.

2.4. RUNNING DATA GRID SERVERS

Spin up Data Grid server instances that automatically form clusters. Learn how to create cache definitions to store your data.

2.4.1. Starting Data Grid Servers

Launch Data Grid server with the startup script.

Procedure

1. Open a terminal in **$RHDG_HOME**.
2. Run the **server** script.

   Linux

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

   Microsoft Windows

   ```
   bin\server.bat
   ```

   Server logs display the following messages:

   ```
   ISPN080004: Protocol SINGLE_PORT listening on 127.0.0.1:11222

   ISPN080001: Data Grid Server <$version> started in <mm>ms
   ```

Hello Data Grid!

- Open **127.0.0.1:11222** in any browser to access the Data Grid console.

Reference

Data Grid Server README describes command line arguments for the **server** script.

2.4.2. Verifying Data Grid Cluster Discovery

By default Data Grid servers running on the same network discover each other with the **MPING** protocol.

This procedure shows you how to start two Data Grid servers on the same host and verify that the cluster view.

Prerequisites

Start a Data Grid server instance on your host.

Procedure

1. Install and run a new Data Grid server instance.
   a. Open a terminal in **$RHDG_HOME**.
   b. Copy the root directory to **server2**.

      ```
      $ cp -r server server2
      ```

2. Specify a port offset and the location of the **server2** root directory.

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

Verification

Check the server logs for the following messages:

```
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
```

2.5. PERFORMING OPERATIONS WITH THE DATA GRID CLI

Connect to servers with the Data Grid command line interface (CLI) to access data and perform administrative functions.

2.5.1. Starting the Data Grid CLI

Start the Data Grid CLI as follows:

1. Open a terminal in **$ISPN_HOME**.
2. Run the CLI.

```
$ bin/cli.sh
[disconnected]>
```

2.5.2. Connecting to Data Grid Servers

Do one of the following:

- Run the **connect** command to connect to a Data Grid server on the default port of **11222**:

```
[disconnected]> connect
[hostname1@cluster//containers/default]>
```

- Specify the location of a Data Grid server. For example, connect to a local server that has a port offset of 100:

```
[disconnected]> connect 127.0.0.1:11322
[hostname2@cluster//containers/default]>
```

TIP

Press the tab key to display available commands and options. Use the **-h** option to display help text.

2.5.3. Creating Caches from Templates

Use Data Grid cache templates to add caches with recommended default settings.

Procedure

1. Create a distributed, synchronous cache from a template and name it "mycache".

```
[//containers/default]> create cache --template=org.infinispan.DIST_SYNC mycache
```

TIP

Press the tab key after the **--template=** argument to list available cache templates.

2. Retrieve the cache configuration.

```
[//containers/default]> describe caches/mycache
{
  "distributed-cache" : {
    "mode" : "SYNC",
    "remote-timeout" : 17500,
    "state-transfer" : {
      "timeout" : 60000
    },
    "transaction" : {
      "mode" : "NONE"
    },
    "locking" : {
      "concurrency-level" : 1000,
      "acquire-timeout" : 15000,
      "striping" : false
    },
    "statistics" : true
  }
}
```

2.5.4. Adding Cache Entries

Add data to caches with the Data Grid CLI.

Prerequisites

- Create a cache named "mycache" and **cd** into it.

```
[//containers/default]> cd caches/mycache
```

Procedure

1. Put an entry into "mycache".

```
[//containers/default/caches/mycache]> put hello world
```

TIP

If not in the context of a cache, use the **--cache=** parameter. For example:

```
[//containers/default]> put --cache=mycache hello world
```

2. Get the entry to verify it.

```
[//containers/default/caches/mycache]> get hello
world
```

2.5.5. Shutting Down Data Grid Servers

Use the CLI to gracefully shutdown running servers. This ensures that Data Grid passivates all entries to disk and persists state.

- Use the **shutdown server** command to stop individual servers.

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

- Use the **shutdown cluster** command to stop all servers joined to the cluster.

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

Verification

Check the server logs for the following messages:

```
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
```

CHAPTER 3. CONFIGURING DATA GRID SERVER NETWORKING

Data Grid servers let you configure interfaces and ports to make endpoints available across your network.

By default, Data Grid servers multiplex endpoints to a single TCP/IP port and automatically detect protocols of inbound client requests.

3.1. SERVER INTERFACES

Data Grid servers can use different strategies for binding to IP addresses.

3.1.1. Address Strategy

Uses an **inet-address** strategy that maps a single **public** interface to the IPv4 loopback address (**127.0.0.1**).

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

TIP

You can use the CLI **-b** argument or the **infinispan.bind.address** property to select a specific address from the command-line. See Changing the Default Bind Address.

3.1.2. Loopback Strategy

Selects a loopback address.

- **IPv4** the address block **127.0.0.0/8** is reserved for loopback addresses.
- **IPv6** the address block **::1** is the only loopback address.

```
<interfaces>
   <interface name="public">
      <loopback/>
   </interface>
</interfaces>
```

3.1.3. Non-Loopback Strategy

Selects a non-loopback address.

```
<interfaces>
   <interface name="public">
      <non-loopback/>
   </interface>
</interfaces>
```

3.1.4. Network Address Strategy

Selects networks based on IP address.

```
<interfaces>
   <interface name="public">
      <inet-address value="10.1.2.3"/>
   </interface>
</interfaces>
```

3.1.5. Any Address Strategy

Selects the **INADDR_ANY** wildcard address. As a result Data Grid servers listen on all interfaces.

```
<interfaces>
   <interface name="public">
      <any-address/>
   </interface>
</interfaces>
```

3.1.6. Link Local Strategy

Selects a *link-local* IP address.

- **IPv4** the address block **169.254.0.0/16** (**169.254.0.0 – 169.254.255.255**) is reserved for link-local addressing.
- **IPv6** the address block **fe80::/10** is reserved for link-local unicast addressing.

```
<interfaces>
   <interface name="public">
      <inet-address value="10.1.2.3"/>
   </interface>
</interfaces>
```

3.1.7. Site Local Strategy

Selects a *site-local* (private) IP address.

- **IPv4** the address blocks **10.0.0.0/8**, **172.16.0.0/12**, and **192.168.0.0/16** are reserved for site-local addressing.
- **IPv6** the address block **fc00::/7** is reserved for site-local unicast addressing.

```
<interfaces>
   <interface name="public">
      <inet-address value="10.1.2.3"/>
   </interface>
</interfaces>
```

3.1.8. Match Host Strategy

Resolves the host name and selects one of the IP addresses that is assigned to any network interface.

Data Grid servers enumerate all available operating system interfaces to locate IP addresses resolved from the host name in your configuration.

```
<interfaces>
   <interface name="public">
      <match-host value="my_host_name"/>
   </interface>
</interfaces>
```

3.1.9. Match Interface Strategy

Selects an IP address assigned to a network interface that matches a regular expression.

Data Grid servers enumerate all available operating system interfaces to locate the interface name in your configuration.

TIP

Use regular expressions with this strategy for additional flexibility.

```
<interfaces>
   <interface name="public">
      <match-interface value="eth0"/>
   </interface>
</interfaces>
```

3.1.10. Match Address Strategy

Similar to **inet-address** but selects an IP address using a regular expression.

Data Grid servers enumerate all available operating system interfaces to locate the IP address in your configuration.

TIP

Use regular expressions with this strategy for additional flexibility.

```
<interfaces>
   <interface name="public">
      <match-address value="132\..*"/>
   </interface>
</interfaces>
```

3.1.11. Fallback Strategy

Interface configurations can include multiple strategies. Data Grid servers try each strategy in the declared order.

For example, with the following configuration, Data Grid servers first attempt to match a host, then an IP address, and then fall back to the **INADDR_ANY** wildcard address:

```
<interfaces>
   <interface name="public">
      <match-host value="my_host_name"/>
      <match-address value="132\..*"/>
      <any-address/>
   </interface>
</interfaces>
```

3.1.12. Changing the Default Bind Address for Data Grid Servers

You can use the server **-b** switch or the **infinispan.bind.address** system property to bind to a different address.

For example, bind the **public** interface to **127.0.0.2** as follows:

Linux

```
$ bin/server.sh -b 127.0.0.2
```

Windows

```
bin\server.bat -b 127.0.0.2
```

3.2. SOCKET BINDINGS

Socket bindings map endpoint connectors to server interfaces and ports.

By default, Data Grid servers provide the following socket bindings:

```
<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>
```

- **socket-bindings** declares the default interface and port offset.
- **default** binds to hotrod and rest connectors to the default port **11222**.
- **memcached** binds the memcached connector to port **11221**.

NOTE

The memcached endpoint is disabled by default.

To override the default interface for **socket-binding** declarations, specify the **interface** attribute.

For example, you add an **interface** declaration named "private":

```
<interfaces>
 ...
 <interface name="private">
```

```
    <inet-address value="10.1.2.3"/>
  </interface>
</interfaces>
```

You can then specify **interface="private"** in a **socket-binding** declaration to bind to the private IP address, as follows:

```
<socket-bindings default-interface="public" port-offset="${infinispan.socket.binding.port-offset:0}">
  ...
  <socket-binding name="private_binding" interface="private" port="1234"/>
</socket-bindings>
```

3.2.1. Specifying Port Offsets

Configure port offsets with Data Grid servers when running multiple instances on the same host. The default port offset is **0**.

Use the **-o** switch with the Data Grid CLI or the **infinispan.socket.binding.port-offset** system property to set port offsets.

For example, start a server instance with an offset of **100** as follows. With the default configuration, this results in the Data Grid server listening on port **11322**.

Linux

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

Windows

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

3.3. DATA GRID PROTOCOL HANDLING

Data Grid servers use a router connector to expose multiple protocols over the same TCP port, **11222**. Using a single port for multiple protocols simplifies configuration and management and increases security by reducing the attack surface for unauthorized users.

Data Grid servers handle HTTP/1.1, HTTP/2, and Hot Rod protocol requests via port **11222** as follows:

HTTP/1.1 upgrade headers

Client requests can include the **HTTP/1.1 upgrade** header field to initiate HTTP/1.1 connections with Data Grid servers. Client applications can then send the **Upgrade: protocol** header field, where **protocol** is a Data Grid server endpoint.

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

Client applications specify Server Name Indication (SNI) mappings for Data Grid server endpoints to negotiate protocols in a secure manner.

Automatic Hot Rod detection

Client requests that include Hot Rod headers automatically route to Hot Rod endpoints if the single port router configuration includes Hot Rod.

3.3.1. Configuring Clients for ALPN

Configure clients to provide ALPN messages for protocol negotiation during TLS handshakes with Data Grid servers.

Prerequisites

- Enable Data Grid server endpoints with encryption.

Procedure

1. Provide your client application with the appropriate libraries to handle ALPN/TLS exchanges with Data Grid servers.

NOTE

Data Grid uses Wildfly OpenSSL bindings for Java.

2. Configure clients with trust stores as appropriate.

Programmatically

```
ConfigurationBuilder builder = new ConfigurationBuilder()
    .addServers("127.0.0.1:11222");

builder.security().ssl().enable()
    .trustStoreFileName("truststore.pkcs12")
    .trustStorePassword(DEFAULT_TRUSTSTORE_PASSWORD.toCharArray());

RemoteCacheManager remoteCacheManager = new RemoteCacheManager(builder.build());
RemoteCache<String, String> cache = remoteCacheManager.getCache("default"");
```

Hot Rod client properties

```
infinispan.client.hotrod.server_list = 127.0.0.1:11222
infinispan.client.hotrod.use_ssl = true
infinispan.client.hotrod.trust_store_file_name = truststore.pkcs12
infinispan.client.hotrod.trust_store_password = trust_store_password
```

Reference

- Data Grid Endpoint Connectors
- Wildfly OpenSSL
- SslConfigurationBuilder
- Hot Rod client configuration properties

CHAPTER 4. CONFIGURING DATA GRID SERVER ENDPOINTS

Data Grid servers provide listener endpoints that handle requests from remote client applications.

4.1. DATA GRID ENDPOINTS

Data Grid endpoints expose the **CacheManager** interface over different connector protocols so you can remotely access data and perform operations to manage and maintain Data Grid clusters.

You can define multiple endpoint connectors on different socket bindings.

4.1.1. Hot Rod

Hot Rod is a binary TCP client-server protocol designed to provide faster data access and improved performance in comparison to text-based protocols.

Data Grid provides Hot Rod client libraries in Java, C++, C#, Node.js and other programming languages.

Topology state transfer

Data Grid uses topology caches to provide clients with cluster views. Topology caches contain entries that map internal JGroups transport addresses to exposed Hot Rod endpoints.

When client send requests, Data Grid servers compare the topology ID in request headers with the topology ID from the cache. Data Grid servers send new topology views if client have older topology IDs.

Cluster topology views allow Hot Rod clients to immediately detect when nodes join and leave, which enables dynamic load balancing and failover.

In distributed cache modes, the consistent hashing algorithm also makes it possible to route Hot Rod client requests directly to primary owners.

4.1.2. REST

Reference

Data Grid exposes a RESTful interface that allows HTTP clients to access data, monitor and maintain clusters, and perform administrative operations.

You can use standard HTTP load balancers to provide clients with load balancing and failover capabilities. However, HTTP load balancers maintain static cluster views and require manual updates when cluster topology changes occur.

4.1.3. Protocol Comparison

Table 4.1. Reference

	Hot Rod	HTTP / REST
Topology-aware	Y	N
Hash-aware	Y	N

	Hot Rod	HTTP / REST
Encryption	Y	Y
Authentication	Y	Y
Conditional ops	Y	Y
Bulk ops	Y	N
Transactions	Y	N
Listeners	Y	N
Query	Y	Y
Execution	Y	N
Cross-site failover	Y	N

4.2. ENDPOINT CONNECTORS

You configure Data Grid server endpoints with connector declarations that specify socket bindings, authentication mechanisms, and encryption configuration.

The default endpoint connector configuration is as follows:

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

- **endpoints** contains endpoint connector declarations and defines global configuration for endpoints such as default socket bindings, security realms, and whether clients must present valid TLS certificates.
- **<hotrod-connector name="hotrod"/>** declares a Hot Rod connector.
- **<rest-connector name="rest"/>** declares a Hot Rod connector.
- **<memcached-connector socket-binding="memcached"/>** declares a Memcached connector that uses the memcached socket binding.

Reference

urn:infinispan:server schema provides all available endpoint configuration.

4.2.1. Hot Rod Connectors

Hot Rod connector declarations enable Hot Rod servers.

```
<hotrod-connector name="hotrod">
  <topology-state-transfer />
  <authentication>
    ...
  </authentication>
  <encryption>
    ...
  </encryption>
</hotrod-connector>
```

- **name="hotrod"** logically names the Hot Rod connector.
- **topology-state-transfer** tunes the state transfer operations that provide Hot Rod clients with cluster topology.
- **authentication** configures SASL authentication mechanisms.
- **encryption** configures TLS settings for client connections.

Reference

urn:infinispan:server schema provides all available Hot Rod connector configuration.

4.2.2. REST Connectors

REST connector declarations enable REST servers.

```
<rest-connector name="rest">
  <authentication>
    ...
  </authentication>
  <cors-rules>
    ...
  </cors-rules>
  <encryption>
    ...
  </encryption>
</rest-connector>
```

- **name="rest"** logically names the REST connector.
- **authentication** configures authentication mechanisms.
- **cors-rules** specifies CORS (Cross Origin Resource Sharing) rules for cross-domain requests.
- **encryption** configures TLS settings for client connections.

Reference

urn:infinispan:server schema provides all available REST connector configuration.

CHAPTER 5. MONITORING DATA GRID SERVERS

5.1. WORKING WITH DATA GRID SERVER LOGS

Data Grid uses Apache Log4j 2 to provide configurable logging mechanisms that capture details about the environment and record cache operations for troubleshooting purposes and root cause analysis.

5.1.1. Data Grid Log Files

Data Grid writes log messages to the following directory:
$RHDG_HOME/${infinispan.server.root}/log

server.log

Messages in human readable format, including boot logs that relate to the server startup. Data Grid creates this file by default when you launch servers.

server.log.json

Messages in JSON format that let you parse and analyze Data Grid logs. Data Grid creates this file when you enable the **JSON-FILE** appender.

5.1.2. Configuring Data Grid Log Properties

You configure Data Grid logs with **log4j2.xml**, which is described in the Log4j 2 manual.

Procedure

1. Open **$RHDG_HOME/${infinispan.server.root}/conf/log4j2.xml** with any text editor.
2. Change logging configuration as appropriate.
3. Save and close **log4j2.xml**.

5.1.2.1. Log Levels

Log levels indicate the nature and severity of messages.

Log level	Description
TRACE	Fine-grained debug messages, capturing the flow of individual requests through the application.
DEBUG	Messages for general debugging, not related to an individual request.
INFO	Messages about the overall progress of applications, including lifecycle events.
WARN	Events that can lead to error or degrade performance.

Log level	Description
ERROR	Error conditions that might prevent operations or activites from being successful but do not prevent applications from running.
FATAL	Events that could cause critical service failure and application shutdown.

In addition to the levels of individual messages presented above, the configuration allows two more values: **ALL** to include all messages, and **OFF** to exclude all messages.

5.1.2.2. Data Grid Log Categories

Data Grid provides categories for **INFO**, **WARN**, **ERROR**, **FATAL** level messages that organize logs by functional area.

org.infinispan.CLUSTER

Messages specific to Data Grid clustering that include state transfer operations, rebalancing events, partitioning, and so on.

org.infinispan.CONFIG

Messages specific to Data Grid configuration.

org.infinispan.CONTAINER

Messages specific to the data container that include expiration and eviction operations, cache listener notifications, transactions, and so on.

org.infinispan.PERSISTENCE

Messages specific to cache loaders and stores.

org.infinispan.SECURITY

Messages specific to Data Grid security.

org.infinispan.SERVER

Messages specific to Data Grid servers.

org.infinispan.XSITE

Messages specific to cross-site replication operations.

5.1.2.3. Log Appenders

Log appenders define how Data Grid records log messages.

CONSOLE

Write log messages to the host standard out (**stdout**) or standard error (**stderr**) stream.
Uses the **org.apache.logging.log4j.core.appender.ConsoleAppender** class by default.

FILE

Write log messages to a file.
Uses the **org.apache.logging.log4j.core.appender.RollingFileAppender** class by default.

JSON-FILE

Write log messages to a file in JSON format.
Uses the **org.apache.logging.log4j.core.appender.RollingFileAppender** class by default.

5.1.2.4. Log Patterns

The **CONSOLE** and **FILE** appenders use a **PatternLayout** to format the log messages according to a **pattern**.

An example is the default pattern in the FILE appender:
%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p (%t) [%c{1}] %m%throwable%n

- **%d{yyyy-MM-dd HH:mm:ss,SSS}** adds the current time and date.
- **%-5p** specifies the log level, aligned to the right.
- **%t** adds the name of the current thread.
- **%c{1}** adds the short name of the logging category.
- **%m** adds the log message.
- **%throwable** adds the exception stack trace.
- **%n** adds a new line.

Patterns are fully described in the **PatternLayout** documentation .

5.1.2.5. Enabling and Configuring the JSON Log Handler

Data Grid provides a JSON log handler to write messages in JSON format.

Prerequisites

Ensure that Data Grid is not running. You cannot dynamically enable log handlers.

Procedure

1. Open **$RHDG_HOME/${infinispan.server.root}/conf/log4j2.xml** with any text editor.
2. Uncomment the **JSON-FILE** appender and comment out the **FILE** appender:

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

3. Optionally configure the JSON appender and layout.
4. Save and close **logging.properties**.

When you start Data Grid, it writes each log message as a JSON map in the following file:
$RHDG_HOME/${infinispan.server.root}/log/server.log.json

5.1.3. Access Logs

Hot Rod and REST endpoints can record all inbound client requests as log entries with the following categories:

- **org.infinispan.HOTROD_ACCESS_LOG** logging category for the Hot Rod endpoint.
- **org.infinispan.REST_ACCESS_LOG** logging category for the REST endpoint.

5.1.3.1. Enabling Access Logs

Access logs for Hot Rod and REST endpoints are disabled by default. To enable either logging category, set the level to **TRACE** in the Data Grid logging configuration, as in the following example:

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

5.1.3.2. Access Log Properties

The default format for access logs is as follows:

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

The preceding format creates log entries such as the following:

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

Logging properties use the **%X{name}** notation and let you modify the format of access logs. The following are the default logging properties:

Property	Description
address	Either the **X-Forwarded-For** header or the client IP address.
user	Principal name, if using authentication.
method	Method used. **PUT**, **GET**, and so on.
protocol	Protocol used. **HTTP/1.1**, **HTTP/2**, **HOTROD/2.9**, and so on.
status	An HTTP status code for the REST endpoint. **OK** or an exception for the Hot Rod endpoint.
requestSize	Size, in bytes, of the request.
responseSize	Size, in bytes, of the response.
duration	Number of milliseconds that the server took to handle the request.

TIP

Use the header name prefixed with **h:** to log headers that were included in requests; for example, **%X{h:User-Agent}**.

5.2. CONFIGURING STATISTICS, METRICS, AND JMX

Enable statistics that Data Grid exports to a MicroProfile Metrics endpoint or via JMX MBeans. You can also register JMX MBeans to perform management operations.

5.2.1. Enabling Data Grid Statistics

Data Grid lets you enable statistics for Cache Managers and specific cache instances.

NOTE

Data Grid servers provide default **infinispan.xml** configuration files that enable statistics for Cache Managers. If you use the default Data Grid server configuration, you only need to enable statistics when you configure caches.

Procedure

- Enable statistics declaratively or programmatically.

Declaratively

```
<cache-container statistics="true"> 1
  <local-cache name="mycache" statistics="true"/> 2
</cache-container>
```

1 collects statistics about the Cache Manager.

2 collects statistics about the named cache.

Programmatically

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()
  .cacheContainer().statistics(true) 1
  .build();

 ...

Configuration config = new ConfigurationBuilder()
  .statistics().enable() 2
  .build();
```

1 collects statistics about the Cache Manager.

2 collects statistics about the named cache.

NOTE

Enabling statistics for Cache Managers does not globally enable statistics for caches. This configuration only collects statistics for the Cache Manager.

5.2.2. Enabling Data Grid Metrics

Data Grid is compatible with the Eclipse MicroProfile Metrics API and can generate gauge and histogram metrics.

- Data Grid metrics are provided at the **vendor** scope. Metrics related to the JVM are provided in the **base** scope for Data Grid server.
- Gauges provide values such as the average number of nanoseconds for write operations or JVM uptime. Gauges are enabled by default. If you enable statistics, Data Grid automatically generates gauges.
- Histograms provide details about operation execution times such as read, write, and remove times. Data Grid does not enable histograms by default because they require additional computation.

Procedure

- Configure metrics declaratively or programmatically.

Declaratively

```
<cache-container statistics="true"> 1
  <metrics gauges="true" histograms="true" /> 2
</cache-container>
```

Programmatically

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()
  .statistics().enable() 1
  .metrics().gauges(true).histograms(true) 2
  .build();
```

1 1 computes and collects statistics about the Cache Manager.

2 2 exports collected statistics as gauge and histogram metrics.

Reference

- Eclipse MicroProfile Metrics
- Enabling Data Grid Statistics

5.2.3. Collecting Data Grid Metrics

Collect Data Grid metrics with monitoring tools such as Prometheus.

Prerequisites

- Enable statistics. If you do not enable statistics, Data Grid provides **0** and **-1** values for metrics.
- Optionally enable histograms. By default Data Grid generates gauges but not histograms.

Procedure

- Get metrics in Prometheus (OpenMetrics) format:

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

- Get metrics in MicroProfile JSON format:

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

Next steps

Configure monitoring applications to collect Data Grid metrics. For example, add the following to **prometheus.yml**:

```
static_configs:
  - targets: ['localhost:11222']
```

Reference

- Prometheus Configuration
- Enabling Data Grid Statistics

5.2.4. Enabling and Configuring JMX

You can enable JMX with Data Grid servers to collect statistics and perform administrative operations.

NOTE

You can enable JMX without enabling statistics if you want to use management operations only. In this case, Data Grid provides **0** values for all statistics.

Procedure

- Enable JMX declaratively or programmatically.

Declaratively

```
<cache-container>
  <jmx enabled="true" /> 1
</cache-container>
```

Programmatically

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()
  .jmx().enable() 1
  .build();
```

1 1 registers Data Grid JMX MBeans.

Reference

- Enabling Data Grid Statistics

5.2.4.1. Naming Multiple Cache Managers

In cases where multiple Data Grid Cache Managers run on the same JVM, you should uniquely identify each Cache Manager to prevent conflicts.

Procedure

- Uniquely identify each cache manager in your environment.

For example, the following examples specify "Hibernate2LC" as the cache manager name, which results in a JMX MBean named **org.infinispan:type=CacheManager,name="Hibernate2LC"**.

Declaratively

```
<cache-container name="Hibernate2LC">
  <jmx enabled="true" />
  ...
</cache-container>
```

Programmatically

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()
  .cacheManagerName("Hibernate2LC")
  .jmx().enable()
  .build();
```

Reference

- GlobalConfigurationBuilder
- Data Grid Configuration Schema

5.2.4.2. Registering MBeans In Custom MBean Servers

Data Grid includes an **MBeanServerLookup** interface that you can use to register MBeans in custom MBeanServer instances.

Procedure

1. Create an implementation of **MBeanServerLookup** so that the **getMBeanServer()** method returns the custom MBeanServer instance.
2. Configure Data Grid with the fully qualified name of your class, as in the following example:

Declaratively

```
<cache-container>
   <jmx enabled="true" mbean-server-lookup="com.acme.MyMBeanServerLookup" />
</cache-container>
```

Programmatically

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()
  .jmx().enable().mBeanServerLookup(new com.acme.MyMBeanServerLookup())
  .build();
```

Reference

- Data Grid Configuration Schema
- MBeanServerLookup

5.2.4.3. Data Grid MBeans

Data Grid exposes JMX MBeans that represent manageable resources.

org.infinispan:type=Cache

Attributes and operations available for cache instances.

org.infinispan:type=CacheManager

Attributes and operations available for cache managers, including Data Grid cache and cluster health statistics.

For a complete list of available JMX MBeans along with descriptions and available operations and attributes, see the *Data Grid JMX Components* documentation.

Reference

Data Grid JMX Components

5.3. RETRIEVING SERVER HEALTH STATISTICS

Monitor the health of your Data Grid clusters in the following ways:

- Programmatically with **embeddedCacheManager.getHealth()** method calls.
- JMX MBeans
- Data Grid REST Server

5.3.1. Accessing the Health API via JMX

Retrieve Data Grid cluster health statistics via JMX.

Procedure

1. Connect to Data Grid server using any JMX capable tool such as JConsole and navigate to the following object:

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

2. Select available MBeans to retrieve cluster health statistics.

5.3.2. Accessing the Health API via REST

Get Data Grid cluster health via the REST API.

Procedure

- Invoke a **GET** request to retrieve cluster health.

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

Data Grid responds with a **JSON** document such as the following:


```
{
   "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"
      }
   ]

}
```


TIP

Get cache manager status as follows:

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

Reference

See the *REST v2 (version 2) API* documentation for more information.

CHAPTER 6. SECURING DATA GRID SERVERS

Protect Data Grid servers against network attacks and unauthorized access.

6.1. CACHE AUTHORIZATION

Data Grid can restrict access to data by authorizing requests to perform cache operations.

Data Grid maps identities, or Principals of type **java.security.Principal**, to security roles in your configuration. For example, a Principal named **reader** maps to a security role named **reader**.

Data Grid lets you assign permissions to the various roles to authorize cache operations. For example, **Cache.get()** requires read permission while **Cache.put()** requires write permission.

In this case, iff a client with the **reader** role attempts to write an entry, Data Grid denies the request and throws a security exception. However, if a client with the **writer** role sends a write request, Data Grid validates authorization and issues the client with a token for subsequent operations.

6.1.1. Cache Authorization Configuration

Data Grid configuration for cache authorization is as follows:

```
<infinispan>
  <cache-container default-cache="secured" name="secured">
    <security>
      <authorization> 1
        <identity-role-mapper /> 2
        <role name="admin" permissions="ALL" /> 3
        <role name="reader" permissions="READ" />
        <role name="writer" permissions="WRITE" />
        <role name="supervisor" permissions="READ WRITE EXEC"/>
      </authorization>
    </security>
    <local-cache name="secured">
      <security>
        <authorization roles="admin reader writer supervisor" /> 4
      </security>
    </local-cache>
  </cache-container>

</infinispan>
```

1. configures cache authorization for the cache container.
2. specifies an implementation of **PrincipalRoleMapper** that converts Principal names to roles.
3. names roles and assigns permissions that control access to data.
4. defines the authorized roles for the cache.

Reference

- Data Grid Configuration

- org.infinispan.security.PrincipalRoleMapper

6.2. DEFINING DATA GRID SERVER SECURITY REALMS

Security realms provide identity, encryption, authentication, and authorization information to Data Grid server endpoints.

6.2.1. Property Realms

Property realms use property files to define users and groups.

users.properties maps usernames to passwords in plain-text format. Passwords can also be pre-digested if you use the **DIGEST-MD5** SASL mechanism or **Digest** HTTP mechanism.

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

groups.properties maps users to roles.

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

Property realm configuration

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <properties-realm groups-attribute="Roles"> 1
        <user-properties path="users.properties" 2
                         relative-to="infinispan.server.config.path" 3
                         plain-text="true"/> 4
        <group-properties path="groups.properties" 5
                          relative-to="infinispan.server.config.path"/>
      </properties-realm>
    </security-realm>
  </security-realms>
</security>
```

1. Defines groups as roles for Data Grid server authorization.
2. Specifies the **users.properties** file.
3. Specifies that the file is relative to the **$ISPN_HOME/server/conf** directory.
4. Specifies that the passwords in **users.properties** are in plain-text format.
5. Specifies the **groups.properties** file.

Supported authentication mechanisms

Property realms support the following authentication mechanisms:

- SASL: **PLAIN**, **DIGEST-***, and **SCRAM-***
- HTTP (REST): **Basic** and **Digest**

6.2.1.1. Adding Users to Property Realms

Data Grid server provides a **user-tool** script that lets you easily add new user/role mappings to properties files.

Procedure

1. Navigate to your **$ISPN_HOME** directory.
2. Run the **user-tool** script in the **bin** folder.

For example, create a new user named "myuser" with a password of "qwer1234!" that belongs to the "supervisor", "reader", and "writer" groups:

Linux

```
$ bin/user-tool.sh -u myuser -p "qwer1234!" -g supervisor,reader,writer
```

Microsoft Windows

```
$ bin\user-tool.bat -u myuser -p "qwer1234!" -g supervisor,reader,writer
```

6.2.2. LDAP Realms

LDAP realms connect to LDAP servers, such as OpenLDAP, Red Hat Directory Server, Apache Directory Server, or Microsoft Active Directory, to authenticate users and obtain membership information.

NOTE

LDAP servers can have different entry layouts, depending on the type of server and deployment. For this reason, LDAP realm configuration is complex. It is beyond the scope of this document to provide examples for all possibile configurations.

LDAP realm configuration

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-
server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <ldap-realm name="ldap" 1
                  url="ldap://my-ldap-server:10389" 2
                  principal="uid=admin,ou=People,dc=infinispan,dc=org" 3
                  credential="strongPassword"
```

```
            connection-timeout="3000" read-timeout="30000" 4
            connection-pooling="true" referral-mode="ignore"
            page-size="30"
            direct-verification="true"> 5
          <identity-mapping rdn-identifier="uid" 6
                    search-dn="ou=People,dc=infinispan,dc=org"> 7
            <attribute-mapping> 8
              <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>
```

1. Names the LDAP realm.
2. Specifies the LDAP server connection URL.
3. Specifies a principal and credentials to connect to the LDAP server.

IMPORTANT

The principal for LDAP connections must have necessary privileges to perform LDAP queries and access specific attributes.

4. Optionally tunes LDAP server connections by specifying connection timeouts and so on.
5. Verifies user credentials. Data Grid attempts to connect to the LDAP server using the configured credentials. Alternatively, you can use the **user-password-mapper** element that specifies a password.
6. Maps LDAP entries to identities. The **rdn-identifier** specifies an LDAP attribute that finds the user entry based on a provided identifier, which is typically a username; for example, the **uid** or **sAMAccountName** attribute.
7. Defines a starting context that limits searches to the LDAP subtree that contains the user entries.
8. Retrieves all the groups of which the user is a member. There are typically two ways in which membership information is stored:

   - Under group entries that usually have class **groupOfNames** in the **member** attribute. In this case, you can use an attribute filter as in the preceding example configuration. This filter searches for entries that match the supplied filter, which locates groups with a **member** attribute equal to the user's DN. The filter then extracts the group entry's CN as specified by **from**, and adds it to the user's **Roles**.
   - In the user entry in the **memberOf** attribute. In this case you should use an attribute reference such as the following:
     <attribute-reference reference="memberOf" from="cn" to="Roles" />

This reference gets all **memberOf** attributes from the user's entry, extracts the CN as specified by **from**, and adds them to the user's **Roles**.

Supported authentication mechanisms

LDAP realms support the following authentication mechanisms directly:

- SASL: **PLAIN**, **DIGEST-***, and **SCRAM-***
- HTTP (REST): **Basic** and **Digest**

6.2.2.1. LDAP Realm Principal Rewriting

Some SASL authentication mechanisms, such as **GSSAPI**, **GS2-KRB5** and **Negotiate**, supply a username that needs to be *cleaned up* before you can use it to search LDAP servers.

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-
server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <ldap-realm name="ldap"
                  url="ldap://${org.infinispan.test.host.address}:10389"
                  principal="uid=admin,ou=People,dc=infinispan,dc=org"
                  credential="strongPassword">
        <name-rewriter> (1)
          <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>
```

(1) Defines a rewriter that extracts the username from the principal using a regular expression.

6.2.3. Trust Store Realms

Trust store realms use keystores that contain the public certificates of all clients that are allowed to connect to Data Grid server.

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-
server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <server-identities>
        <ssl>
          <keystore path="server.p12" 1
                    relative-to="infinispan.server.config.path" 2
                    keystore-password="secret" 3
                    alias="server"/> 4
        </ssl>
      </server-identities>
      <truststore-realm path="trust.p12" 5
                        relative-to="infinispan.server.config.path"
                        keystore-password="secret"/>
    </security-realm>
  </security-realms>
</security>
```

1. Provides an SSL server identity with a keystore that contains server certificates.
2. Specifies that the file is relative to the **$ISPN_HOME/server/conf** directory.
3. Specifies a keystore password.
4. Specifies a keystore alias.
5. Provides a keystore that contains public certificates of all clients.

Supported authentication mechanisms

Trust store realms work with client-certificate authentication mechanisms:

- SASL: **EXTERNAL**
- HTTP (REST): **CLIENT_CERT**

6.2.4. Token Realms

Token realms use external services to validate tokens and require providers that are compatible with RFC-7662 (OAuth2 Token Introspection), such as Red Hat SSO.

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-
server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <token-realm name="token"
                   auth-server-url="https://oauth-server/auth/"> 1
        <oauth2-introspection
              introspection-url="https://oauth-server/auth/realms/infinispan/protocol/openid-
connect/token/introspect" 2
```

```
            client-id="infinispan-server" 3
            client-secret="1fdca4ec-c416-47e0-867a-3d471af7050f"/> 4
        </token-realm>
      </security-realm>
    </security-realms>
  </security>
```

1. Specifies the URL of the authentication server.
2. Specifies the URL of the token introspection endpoint.
3. Names the client identifier for Data Grid server.
4. Specifies the client secret for Data Grid server.

Supported authentication mechanisms

Token realms support the following authentication mechanisms:

- SASL: **OAUTHBEARER**
- HTTP (REST): **Bearer**

6.3. CREATING DATA GRID SERVER IDENTITIES

Server identities are defined within security realms and enable Data Grid servers to prove their identity to clients.

6.3.1. Setting Up SSL Identities

SSL identities use keystores that contain either a certificate or chain of certificates.

NOTE

If security realms contain SSL identities, Data Grid servers automatically enable encryption for the endpoints that use those security realms.

Procedure

1. Create a keystore for Data Grid server.

IMPORTANT

Data Grid server supports the following keystore formats: JKS, JCEKS, PKCS12, BKS, BCFKS and UBER.

In production environments, server certificates should be signed by a trusted Certificate Authority, either Root or Intermediate CA.

2. Add the keystore to the **$ISPN_HOME/server/conf** directory.
3. Add a **server-identities** definition to the Data Grid server security realm.
4. Specify the name of the keystore along with the password and alias.

6.3.1.1. SSL Identity Configuration

The following example configures an SSL identity for Data Grid server:

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <server-identities> 1
        <ssl> 2
          <keystore path="server.p12" 3
                  relative-to="infinispan.server.config.path" 4
                  keystore-password="secret" 5
                  alias="server"/> 6
        </ssl>
      </server-identities>
    </security-realm>
  </security-realms>
</security>
```

1. Defines identities for Data Grid server.
2. Configures an SSL identity for Data Grid server.
3. Names a keystore that contains Data Grid server SSL certificates.
4. Specifies that the keystore is relative to the **server/conf** directory in **$ISPN_HOME**.
5. Specifies a keystore password.
6. Specifies a keystore alias.

6.3.1.2. Automatically Generating Keystores

Configure Data Grid servers to automatically generate keystores at startup.

IMPORTANT

Automatically generated keystores:

- Should not be used in production environments.
- Are generated whenever necessary; for example, while obtaining the first connection from a client.
- Contain certificates that you can use directly in Hot Rod clients.

Procedure

1. Include the **generate-self-signed-certificate-host** attribute for the **keystore** element in the server configuration.

2. Specify a hostname for the server certificate as the value.

SSL server identity with a generated keystore

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <server-identities>
        <ssl>
          <keystore path="server.p12"
                    relative-to="infinispan.server.config.path"
                    keystore-password="secret"
                    alias="server"
                    generate-self-signed-certificate-host="localhost"/> 1
        </ssl>
      </server-identities>
    </security-realm>
  </security-realms>
</security>
```

1 generates a keystore using **localhost**

6.3.1.3. Tuning SSL Protocols and Cipher Suites

You can configure the SSL engine, via the Data Grid server SSL identity, to use specific protocols and ciphers.

IMPORTANT

You must ensure that you set the correct ciphers for the protocol features you want to use; for example HTTP/2 ALPN.

Procedure

1. Add the **engine** element to your Data Grid server SSL identity.
2. Configure the SSL engine with the **enabled-protocols** and **enabled-ciphersuites** attributes.

SSL engine configuration

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1
        https://infinispan.org/schemas/infinispan-server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <server-identities>
        <ssl>
          <keystore path="server.p12"
                    relative-to="infinispan.server.config.path"
```

```
                keystore-password="secret" alias="server"/>
            <engine enabled-protocols="TLSv1.2 TLSv1.1" 1
                enabled-ciphersuites="SSL_RSA_WITH_AES_128_GCM_SHA256 2
                SSL_RSA_WITH_AES_128_CBC_SHA256"/>
        </ssl>
    </server-identities>
  </security-realm>
 </security-realms>
</security>
```

1 Configures the SSL engine to use TLS v1 and v2 protocols.

2 Configures the SSL engine to use the specified cipher suites.

6.3.2. Setting Up Kerberos Identities

Kerberos identities use *keytab* files that contain service principal names and encrypted keys, derived from Kerberos passwords.

NOTE

keytab files can contain both user and service account principals. However, Data Grid servers use service account principals only. As a result, Data Grid servers can provide identity to clients and allow clients to authenticate with Kerberos servers.

In most cases, you create unique principals for the Hot Rod and REST connectors. For example, you have a "datagrid" server in the "INFINISPAN.ORG" domain. In this case you should create the following service principals:

- **hotrod/datagrid@INFINISPAN.ORG** identifies the Hot Rod service.
- **HTTP/datagrid@INFINISPAN.ORG** identifies the REST service.

Procedure

1. Create keytab files for the Hot Rod and REST services.

   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. Copy the keytab files to the **$ISPN_HOME/server/conf** directory.

3. Add a **server-identities** definition to the Data Grid server security realm.
4. Specify the location of keytab files that provide service principals to Hot Rod and REST connectors.
5. Name the Kerberos service principals.

6.3.2.1. Kerberos Identity Configuration

The following example configures Kerberos identities for Data Grid server:

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1">
  <security-realms>
    <security-realm name="default">
      <server-identities> 1
        <kerberos keytab-path="hotrod.keytab" 2
                principal="hotrod/datagrid@INFINISPAN.ORG" 3
                required="true"/> 4
        <kerberos keytab-path="http.keytab" 5
                principal="HTTP/localhost@INFINISPAN.ORG" 6
                required="true"/>
      </server-identities>
    </security-realm>
  </security-realms>
</security>
```

1. Defines identities for Data Grid server.
2. Specifies a keytab file that provides a Kerberos identity for the Hot Rod connector.
3. Names the Kerberos service principal for the Hot Rod connector.
4. Specifies that the keytab file must exist when Data Grid server starts.
5. Specifies a keytab file that provides a Kerberos identity for the REST connector.
6. Names the Kerberos service principal for the REST connector.

6.4. CONFIGURING ENDPOINT AUTHENTICATION MECHANISMS

Configure Hot Rod and REST connectors with SASL authentication mechanisms to authenticate with clients.

6.4.1. Setting Up Hot Rod Authentication

Configure Hot Rod connectors to authenticate with clients to Data Grid server security realms using specific SASL authentication mechanisms .

Procedure

1. Add an **authentication** definition to the Hot Rod connector configuration.
2. Specify which Data Grid security realm the Hot Rod connector uses for authentication.
3. Specify the SASL authentication mechanisms for the Hot Rod endpoint to use.
4. Configure SASL authentication properties as appropriate.

6.4.1.1. Hot Rod Authentication Configuration

Hot Rod connector with SCRAM, DIGEST, and PLAIN authentication

```
<endpoints xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1
        https://infinispan.org/schemas/infinispan-server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1"
        socket-binding="default" security-realm="default"> 1
  <hotrod-connector name="hotrod">
    <authentication>
      <sasl mechanisms="SCRAM-SHA-512 SCRAM-SHA-384 SCRAM-SHA-256 2
                  SCRAM-SHA-1 DIGEST-SHA-512 DIGEST-SHA-384
                  DIGEST-SHA-256 DIGEST-SHA DIGEST-MD5 PLAIN"
          server-name="infinispan" 3
          qop="auth"/> 4
    </authentication>
  </hotrod-connector>
</endpoints>
```

1. enables authentication against the security realm named "default".
2. specifies SASL mechanisms to use for authentication.
3. defines the name that Data Grid servers declare to clients. The server name should match the client configuration.
4. enables **auth** QoP.

Hot Rod connector with Kerberos authentication

```
<endpoints xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-
server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1"
        socket-binding="default" security-realm="default">
  <hotrod-connector name="hotrod">
    <authentication>
      <sasl mechanisms="GSSAPI GS2-KRB5" 1
          server-name="datagrid" 2
          server-principal="hotrod/datagrid@INFINISPAN.ORG"/> 3
    </authentication>
  </hotrod-connector>
</endpoints>
```

1. Enables the **GSSAPI** and **GS2-KRB5** mechanisms for Kerberos authentication.
2. Defines the Data Grid server name, which is equivalent to the Kerberos service name.
3. Specifies the Kerberos identity for the server.

6.4.1.2. Hot Rod Endpoint Authentication Mechanisms

Data Grid supports the following SASL authentications mechanisms with the Hot Rod connector:

Authentication mechanism	Description	Related details
PLAIN	Uses credentials in plain-text format. You should use **PLAIN** authentication with encrypted connections only.	Similar to the **Basic** HTTP mechanism.
DIGEST-*	Uses hashing algorithms and nonce values. Hot Rod connectors support **DIGEST-MD5**, **DIGEST-SHA**, **DIGEST-SHA-256**, **DIGEST-SHA-384**, and **DIGEST-SHA-512** hashing algorithms, in order of strength.	Similar to the **Digest** HTTP mechanism.
SCRAM-*	Uses *salt* values in addition to hashing algorithms and nonce values. Hot Rod connectors support **SCRAM-SHA**, **SCRAM-SHA-256**, **SCRAM-SHA-384**, and **SCRAM-SHA-512** hashing algorithms, in order of strength.	Similar to the **Digest** HTTP mechanism.
GSSAPI	Uses Kerberos tickets and requires a Kerberos Domain Controller. You must add a corresponding **kerberos** server identity in the realm configuration. In most cases, you also specify an **ldap-realm** to provide user membership information.	Similar to the **SPNEGO** HTTP mechanism.
GS2-KRB5	Uses Kerberos tickets and requires a Kerberos Domain Controller. You must add a corresponding **kerberos** server identity in the realm configuration. In most cases, you also specify an **ldap-realm** to provide user membership information.	Similar to the **SPNEGO** HTTP mechanism.

Authentication mechanism	Description	Related details
EXTERNAL	Uses client certificates.	Similar to the **CLIENT_CERT** HTTP mechanism.
OAUTHBEARER	Uses OAuth tokens and requires a **token-realm** configuration.	Similar to the **BEARER_TOKEN** HTTP mechanism.

6.4.1.3. SASL Quality of Protection (QoP)

If SASL mechanisms support integrity and privacy protection settings, you can add them to your Hot Rod connector configuration with the **qop** attribute.

QoP setting	Description
auth	Authentication only.
auth-int	Authentication with integrity protection.
auth-conf	Authentication with integrity and privacy protection.

6.4.1.4. SASL Policies

SASL policies let you control which authentication mechanisms Hot Rod connectors can use.

Policy	Description	Default value
forward-secrecy	Use only SASL mechanisms that support forward secrecy between sessions. This means that breaking into one session does not automatically provide information for breaking into future sessions.	false
pass-credentials	Use only SASL mechanisms that require client credentials.	false
no-plain-text	Do not use SASL mechanisms that are susceptible to simple plain passive attacks.	false
no-active	Do not use SASL mechanisms that are susceptible to active, non-dictionary, attacks.	false

Policy	Description	Default value
no-dictionary	Do not use SASL mechanisms that are susceptible to passive dictionary attacks.	false
no-anonymous	Do not use SASL mechanisms that accept anonymous logins.	true

TIP

Data Grid cache authorization restricts access to caches based on roles and permissions. If you configure cache authorization, you can then set **<no-anonymous value=false />** to allow anonymous login and delegate access logic to cache authorization.

Hot Rod connector with SASL policy configuration

```
<hotrod-connector socket-binding="hotrod" cache-container="default">
  <authentication security-realm="ApplicationRealm">
    <sasl server-name="myhotrodserver"
          mechanisms="PLAIN DIGEST-MD5 GSSAPI EXTERNAL" 1
          qop="auth">
        <policy> 2
          <no-active value="true" />
          <no-anonymous value="true" />
          <no-plain-text value="true" />
        </policy>
    </sasl>
  </authentication>
</hotrod-connector>
```

1 Specifies multiple SASL authentication mechanisms for the Hot Rod connector.

2 Defines policies for SASL mechanisms.

As a result of the preceding configuration, the Hot Rod connector uses the **GSSAPI** mechanism because it is the only mechanism that complies with all policies.

6.4.2. Setting Up REST Authentication

Configure REST connectors to authenticate with clients to Data Grid server security realms using specific SASL authentication mechanisms .

Procedure

1. Add an **authentication** definition to the REST connector configuration.
2. Specify which Data Grid security realm the REST connector uses for authentication.
3. Specify the SASL authentication mechanisms for the REST endpoint to use.

4. Configure SASL authentication properties as appropriate.

6.4.2.1. REST Authentication Configuration

REST connector with BASIC and DIGEST authentication

```
<endpoints xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1"
        socket-binding="default" security-realm="default"> (1)
  <rest-connector name="rest">
    <authentication mechanisms="DIGEST BASIC"/> (2)
  </rest-connector>
</endpoints>
```

1. Enables authentication against the security realm named "default".
2. Specifies SASL mechanisms to use for authentication

REST connector with Kerberos authentication

```
<endpoints xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-server-10.1.xsd"
        xmlns="urn:infinispan:server:10.1"
        socket-binding="default" security-realm="default">
  <rest-connector name="rest">
    <authentication mechanisms="SPNEGO" (1)
                server-principal="HTTP/localhost@INFINISPAN.ORG"/> (2)
  </rest-connector>
</endpoints>
```

1. Enables the **SPENGO** mechanism for Kerberos authentication.
2. Specifies the Kerberos identity for the server.

6.4.2.2. REST Endpoint Authentication Mechanisms

Data Grid supports the following authentications mechanisms with the REST connector:

Authentication mechanism	Description	Related details
BASIC	Uses credentials in plain-text format. You should use **BASIC** authentication with encrypted connections only.	Corresponds to the **Basic** HTTP authentication scheme and is similar to the **PLAIN** SASL mechanism.

Authentication mechanism	Description	Related details
DIGEST	Uses hashing algorithms and nonce values. REST connectors support **SHA-512**, **SHA-256** and **MD5** hashing algorithms.	Corresponds to the **Digest** HTTP authentication scheme and is similar to **DIGEST-*** SASL mechanisms.
SPNEGO	Uses Kerberos tickets and requires a Kerberos Domain Controller. You must add a corresponding **kerberos** server identity in the realm configuration. In most cases, you also specify an **ldap-realm** to provide user membership information.	Corresponds to the **Negotiate** HTTP authentication scheme and is similar to the **GSSAPI** and **GS2-KRB5** SASL mechanisms.
BEARER_TOKEN	Uses OAuth tokens and requires a **token-realm** configuration.	Corresponds to the **Bearer** HTTP authentication scheme and is similar to **OAUTHBEARER** SASL mechanism.
CLIENT_CERT	Uses client certificates.	Similar to the **EXTERNAL** SASL mechanism.

CHAPTER 7. REMOTELY EXECUTING SERVER-SIDE TASKS

Define and add tasks to Data Grid servers that you can invoke from the Data Grid command line interface, REST API, or from Hot Rod clients.

You can implement tasks as custom Java classes or define scripts in languages such as JavaScript.

7.1. CREATING SERVER TASKS

Create custom task implementations and add them to Data Grid servers.

7.1.1. Server Tasks

Data Grid server tasks are classes that extend the **org.infinispan.tasks.ServerTask** interface and generally include the following method calls:

setTaskContext()

Allows access to execution context information including task parameters, cache references on which tasks are executed, and so on. In most cases, implementations store this information locally and use it when tasks are actually executed.

getName()

Returns unique names for tasks. Clients invoke tasks with these names.

getExecutionMode()

Returns the execution mode for tasks.

- **TaskExecutionMode.ONE_NODE** only the node that handles the request executes the script. Although scripts can still invoke clustered operations.
- **TaskExecutionMode.ALL_NODES** Data Grid uses clustered executors to run scripts across nodes. For example, server tasks that invoke stream processing need to be executed on a single node because stream processing is distributed to all nodes.

call()

Computes a result. This method is defined in the **java.util.concurrent.Callable** interface and is invoked with server tasks.

IMPORTANT

Server task implementations must adhere to service loader pattern requirements. For example, implementations must have a zero-argument constructors.

The following **HelloTask** class implementation provides an example task that has one parameter:

```
package example;

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

public class HelloTask implements ServerTask<String> {

   private TaskContext ctx;
```

```
  @Override
  public void setTaskContext(TaskContext ctx) {
    this.ctx = ctx;
  }

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

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

}
```

Reference

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

7.1.2. Deploying Server Tasks to Data Grid Servers

Add your custom server task classes to Data Grid servers.

Prerequisites

Stop any running Data Grid servers. Data Grid does not support runtime deployment of custom classes.

Procedure

1. Package your server task implementation in a JAR file.
2. Add a **META-INF/services/org.infinispan.tasks.ServerTask** file that contains the fully qualified names of server tasks, for example:

```
example.HelloTask
```

3. Copy the JAR file to the **$RHDG_HOME/server/lib** directory of your Data Grid server.
4. Add your classes to the deserialization whitelist in your Data Grid configuration. Alternatively set the whitelist using system properties.

Reference

- Adding Java Classes to Deserialization White Lists
- Data Grid 8.0 Configuration Schema

7.2. CREATING SERVER SCRIPTS

Create custom scripts and add them to Data Grid servers.

7.2.1. Server Scripts

Data Grid server scripting is based on the **javax.script** API and is compatible with any JVM-based ScriptEngine implementation.

Hello World Script Example

The following is a simple example that runs on a single Data Grid server, has one parameter, and uses JavaScript:

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

When you run the preceding script, you pass a value for the **greetee** parameter and Data Grid returns **"Hello ${value}"**.

7.2.1.1. Script Metadata

Metadata provides additional information about scripts that Data Grid servers use when running scripts.

Script metadata are **property=value** pairs that you add to comments in the first lines of scripts, such as the following example:

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

- Use comment styles that match the scripting language (//, **;;**, **#**).
- Separate **property=value** pairs with commas.
- Separate values with single (') or double (") quote characters.

Table 7.1. Metadata Properties

Property	Description
mode	Defines the exection mode and has the following values: **local** only the node that handles the request executes the script. Although scripts can still invoke clustered operations. **distributed** Data Grid uses clustered executors to run scripts across nodes.
language	Specifies the ScriptEngine that executes the script.
extension	Specifies filename extensions as an alternative method to set the ScriptEngine.

Property	Description
role	Specifies roles that users must have to execute scripts.
parameters	Specifies an array of valid parameter names for this script. Invocations which specify parameters not included in this list cause exceptions.
datatype	Optionally sets the MediaType (MIME type) for storing data as well as parameter and return values. This property is useful for remote clients that support particular data formats only. Currently you can set only **text/plain; charset=utf-8** to use the String UTF-8 format for data.

7.2.1.2. Script Bindings

Data Grid exposes internal objects as bindings for script execution.

Binding	Description
cache	Specifies the cache against which the script is run.
marshaller	Specifies the marshaller to use for serializing data to the cache.
cacheManager	Specifies the **cacheManager** for the cache.
scriptingManager	Specifies the instance of the script manager that runs the script. You can use this binding to run other scripts from a script.

7.2.1.3. Script Parameters

Data Grid lets you pass named parameters as bindings for running scripts.

Parameters are **name,value** pairs, where **name** is a string and **value** is any value that the marshaller can interpret.

The following example script has two parameters, **multiplicand** and **multiplier**. The script takes the value of **multiplicand** and multiplies it with the value of **multiplier**.

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

When you run the preceding script, Data Grid responds with the result of the expression evaluation.

7.2.2. Adding Scripts to Data Grid Servers

Use the command line interface to add scripts to Data Grid servers.

Prerequisites

Data Grid servers store scripts in the **___script_cache** cache. If you enable cache authorization, users must have the **___script_manager** role to access **___script_cache**.

Procedure

1. Define scripts as required.
   For example, create a file named **multiplication.js** that runs on a single Data Grid server, has two parameters, and uses JavaScript to multiply a given value:

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

2. Open a CLI connection to Data Grid and use the **task** command to upload your scripts as in the following example:

```
[//containers/default]> task upload --file=multiplication.js multiplication
```

3. Verify that your scripts are available.

```
[//containers/default]> ls tasks
multiplication
```

7.2.3. Programmatically Creating Scripts

Add scripts with the Hot Rod **RemoteCache** interface as in the following example:

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

Reference

org.infinispan.client.hotrod.RemoteCache

7.3. RUNNING SERVER-SIDE TASKS AND SCRIPTS

Execute tasks and custom scripts on Data Grid servers.

7.3.1. Running Tasks and Scripts

Use the command line interface to run tasks and scripts on Data Grid servers.

Prerequisites

- Open a CLI connection to Data Grid.

Procedure

Use the **task** command to run tasks and scripts on Data Grid servers, as in the following examples:

- Execute a script named **multipler.js** and specify two parameters:

```
[//containers/default]> task exec multipler.js -Pmultiplicand=10 -Pmultiplier=20
200.0
```

- Execute a task named **@@cache@names** to retrieve a list of all available caches:

```
//containers/default]> task exec @@cache@names
["___protobuf_metadata","mycache","___script_cache"]
```

7.3.2. Programmatically Running Scripts

Call the **execute()** method to run scripts with the Hot Rod **RemoteCache** interface, as in the following example:

```
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);
```

Reference

org.infinispan.client.hotrod.RemoteCache

7.3.3. Programmatically Running Tasks

Call the **execute()** method to run tasks with the Hot Rod **RemoteCache** interface, as in the following example:

```
// 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);
```

Reference

org.infinispan.client.hotrod.RemoteCache

CHAPTER 8. PERFORMING ROLLING UPGRADES FOR DATA GRID SERVERS

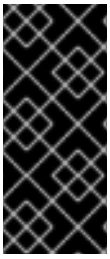
Perform rolling upgrades of your Data Grid clusters to change between versions without downtime or data loss. Rolling upgrades migrate both your Data Grid servers and your data to the target version over Hot Rod.

8.1. SETTING UP TARGET CLUSTERS

Create a cluster that runs the target Data Grid version and uses a remote cache store to load data from the source cluster.

Prerequisites

- Install a Data Grid cluster with the target upgrade version.



IMPORTANT

Ensure the network properties for the target cluster do not overlap with those for the source cluster. You should specify unique names for the target and source clusters in the JGroups transport configuration. Depending on your environment you can also use different network interfaces and specify port offsets to keep the target and source clusters separate.

Procedure

1. Add a **RemoteCacheStore** on the target cluster for each cache you want to migrate from the source cluster.
   Remote cache stores use the Hot Rod protocol to retrieve data from remote Data Grid clusters. When you add the remote cache store to the target cluster, it can lazily load data from the source cluster to handle client requests.

2. Switch clients over to the target cluster so it starts handling all requests.

   a. Update client configuration with the location of the target cluster.

   b. Restart clients.

8.1.1. Remote Cache Stores for Rolling Upgrades

You must use specific remote cache store configuration to perform rolling upgrades, as follows:

```
<persistence passivation="false"> 1
  <remote-store xmlns="urn:infinispan:config:store:remote:10.1"
                cache="myDistCache" 2
                protocol-version="2.5" 3
                hotrod-wrapping="true" 4
                raw-values="true"> 5
    <remote-server host="127.0.0.1" port="11222"/> 6
  </remote-store>
</persistence>
```

1. Disables passivation. Remote cache stores for rolling upgrades must disable passivation.
2. Matches the name of a cache in the source cluster. Target clusters load data from this cache using the remote cache store.
3. Matches the Hot Rod protocol version of the source cluster. **2.5** is the minimum version and is suitable for any upgrade paths. You do not need to set another Hot Rod version.
4. Ensures that entries are wrapped in a suitable format for the Hot Rod protocol.
5. Stores data in the remote cache store in raw format. This ensures that clients can use data directly from the remote cache store.
6. Points to the location of the source cluster.

Reference

- Remote cache store configuration schema
- RemoteStore
- RemoteStoreConfigurationBuilder

8.2. SYNCHRONIZING DATA TO TARGET CLUSTERS

When your target cluster is running and handling client requests using a remote cache store to load data on demand, you can synchronize data from the source cluster to the target cluster.

This operation reads data from the source cluster and writes it to the target cluster. Data migrates to all nodes in the target cluster in parallel, with each node receiving a subset of the data. You must perform the synchronization for each cache in your Data Grid configuration.

Procedure

1. Start the synchronization operation for each cache in your Data Grid configuration that you want to migrate to the target cluster.
   Use the Data Grid REST API and invoke **GET** requests with the **?action=sync- data** parameter. For example, to synchronize data in a cache named "myCache" from a source cluster to a target cluster, do the following:

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

   When the operation completes, Data Grid responds with the total number of entries copied to the target cluster.

   Alternatively, you can use JMX by invoking **synchronizeData(migratorName=hotrod)** on the **RollingUpgradeManager** MBean.

2. Disconnect each node in the target cluster from the source cluster.
   For example, to disconnect the "myCache" cache from the source cluster, invoke the following **GET** request:

   ```
   GET /v2/caches/myCache?action=disconnect-source
   ```

To use JMX, invoke **disconnectSource(migratorName=hotrod)** on the **RollingUpgradeManager** MBean.

Next steps

After you synchronize all data from the source cluster, the rolling upgrade process is complete. You can now decommission the source cluster.

CHAPTER 9. DATA GRID SERVER REFERENCE

9.1. DATA GRID SERVER

Data Grid server provides fast, reliable, and secure access to your data. This release of the server improves on previous versions and offers:

- Small code-base with minimal duplication of existing functionality (e.g. configuration).
- Embeddable: the server should allow for easy testability in both single and clustered configurations.
- RESTful administration capabilities.
- Logging with Apache Log4j 2.
- Security with WildFly Elytron SPIs.

9.1.1. Directory Structure

Server distributions have the following directory structure:

- /**bin**: scripts to start the server and create/modify users.
- /**boot**: **JAR** files to boot the server.
- /**docs**: examples and component licenses.
- /**lib**: server **JAR** files.
- /**server**: default server instance folder.
  - /**server/conf**: configuration files.
  - /**server/data**: data files organized by container name.
  - /**server/lib**: extension **JAR** files for custom filters, listeners, and so on.
  - /**server/log**: server log files.

9.1.2. Server Paths

Servers use the following *paths*:

- **infinispan.server.home** defaults to the directory that contains the server distribution.
- **infinispan.server.root** contains configuration and data for Data Grid servers and defaults to the **server** directory under **infinispan.server.home**.
  Multiple roots can exist in the same or different directories. To start multiple server instances you can specify the path to **infinispan.server.root** with a command line argument.

NOTE

When the server starts it locks **infinispan.server.root** to avoid concurrent use by multiple server instances.

- **infinispan.server.configuration** defaults to **infinispan.xml**, which resides in the **conf** folder under **infinispan.server.root**.

9.1.3. Command Arguments

The **server.sh|bat** startup script includes arguments that you can use to get information and configure the server runtime environment.

To view the available command arguments, include the **--help** or **-h** option as follows:

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

9.1.4. Logging Configuration

You configure logging with **log4j2.xml** in the **server/conf** directory.

9.1.5. Configuration

The server configuration extends the standard Data Grid configuration with the following server-specific elements:

- **security** configures the available security realms for the endpoints.
- **cache-container** multiple containers may be configured, distinguished by name.
- **endpoints** lists the enabled endpoint connectors (hotrod, rest, and so on).
- **socket-bindings** lists the socket bindings.

The following is an example server configuration:

```
<infinispan
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:infinispan:config:10.1 https://infinispan.org/schemas/infinispan-config-10.1.xsd
                          urn:infinispan:server:10.1 https://infinispan.org/schemas/infinispan-server-10.1.xsd"
      xmlns="urn:infinispan:config:10.1"
      xmlns:server="urn:infinispan:server:10.1">

  <cache-container default-cache="default" statistics="false">
    <transport cluster="${infinispan.cluster.name}" stack="udp"/>
    <global-state/>
    <distributed-cache name="default"/>
  </cache-container>

  <server xmlns="urn:infinispan:server:10.1">
    <interfaces>
      <interface name="public">
        <loopback/>
      </interface>
      <interface name="admin">
        <loopback/>
      </interface>
    </interfaces>
```

```
    <socket-bindings default-interface="public" port-offset="${infinispan.socket.binding.port-offset:0}">
      <socket-binding name="hotrod" port="11222"/>
      <socket-binding name="rest" port="8080"/>
      <socket-binding name="memcached" port="11221"/>
      <socket-binding name="admin" port="9990" interface="admin"/>
    </socket-bindings>

    <security>
      <security-realms>
        <security-realm name="ManagementRealm">
          <properties-realm groups-attribute="Roles">
            <user-properties path="mgmt-users.properties" relative-to="infinispan.server.config.dir"
plain-text="true"/>
            <group-properties path="mgmt-groups.properties" relative-to="infinispan.server.config.dir"
/>
          </properties-realm>
        </security-realm>
        <security-realm name="PublicRealm">
          <properties-realm groups-attribute="Roles">
            <user-properties path="public-users.properties" relative-to="infinispan.server.config.dir"
plain-text="true"/>
            <group-properties path="public-groups.properties" relative-to="infinispan.server.config.dir"
/>
          </properties-realm>
          <server-identities>
            <ssl>
              <keystore path="application.keystore" relative-to="infinispan.server.config.dir"
                  keystore-password="password" alias="server" key-password="password"
                  generate-self-signed-certificate-host="localhost"/>
            </ssl>
          </server-identities>
        </security-realm>
      </security-realms>
    </security>

    <endpoints>
      <hotrod-connector socket-binding="hotrod"/>
      <memcached-connector socket-binding="memcached"/>
      <rest-connector socket-binding="rest"/>
    </endpoints>
  </server>
</infinispan>
```

9.1.6. Additional Details

The following is a list of additional details about the server, in no particular order:

- All containers handled by the same server share the same thread pools and transport.