Red Hat Data Grid 8.1

Data Grid Server Guide

Deploy, secure, and manage Data Grid Server

Last Updated: 2023-11-23

Red Hat Data Grid 8.1 Data Grid Server Guide

Deploy, secure, and manage Data Grid Server

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

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.

DATA GRID DOCUMENTATION

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

- Data Grid 8.1 Documentation
- Data Grid 8.1 Component Details
- Supported Configurations for Data Grid 8.1
- Data Grid 8 Feature Support
- Data Grid Deprecated Features and Functionality

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.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

CHAPTER 1. GETTING STARTED WITH DATA GRID SERVER

Quickly set up Data Grid Server and learn the basics.

1.1. DATA GRID SERVER REQUIREMENTS

Data Grid Server requires a Java Virtual Machine. See the Data Grid Supported Configurations for details on supported versions.

1.2. 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 Red Hat Data Grid 8.1 Server from the software downloads section.
3. Run the **md5sum** or **sha256sum** command with the server download archive as the argument, for example:

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

4. 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.

1.3. INSTALLING DATA GRID SERVER

Install the Data Grid Server distribution on a host system.

Prerequisites

Download a Data Grid Server distribution archive.

Procedure

- Use any appropriate tool to extract the Data Grid Server archive to the host filesystem.

```
$ unzip redhat-datagrid-8.1.1-server.zip
```

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

1.4. CREATING AND MODIFYING USERS

Data Grid Server requires users to authenticate against a default property realm. Before you can access Data Grid Server, you must add credentials by creating at least one user and a password. You can also add and modify the security authorization groups to which users belong.

Procedure

1. Open a terminal in **$RHDG_HOME**.
2. Create and modify Data Grid users with the **user** command.

TIP

Run **help user** for more details about using the command.

Creating users and passwords

- Linux

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

- Microsoft Windows

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

Creating users with group membership

- Linux

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

- Microsoft Windows

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

1.5. STARTING DATA GRID SERVER

Run Data Grid Server on your local host.

Prerequisites

- Create at least one Data Grid user.

Procedure

1. Open a terminal in **$RHDG_HOME**.
2. Run Data Grid Server with the **server** script.

   Linux

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

   Microsoft Windows

```
bin\server.bat
```

Data Grid Server is running successfully when it logs the following messages:

```
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
```

Verification

1. Open **127.0.0.1:11222/console/** in any browser.
2. Enter your credentials at the prompt then continue to Data Grid Console.

1.6. VERIFYING CLUSTER VIEWS

Data Grid nodes on the same network automatically discover each other and form clusters.

Complete this procedure to observe cluster discovery with the **MPING** protocol in the default **TCP** stack with locally running Data Grid Server instances. If you want to adjust cluster transport for custom network requirements, see the documentation for setting up Data Grid clusters.

NOTE

This procedure is intended to demonstrate the principle of cluster discovery and is not intended for production environments. Doing things like specifying a port offset on the command line is not a reliable way to configure cluster transport for production.

Prerequisites

Have one instance of Data Grid Server running.

Procedure

1. Open a terminal in **$RHDG_HOME**.
2. Copy the root directory to **server2**.

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

3. Specify a port offset and the **server2** directory.

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

Verification

You can view cluster membership in the console at **127.0.0.1:11222/console/cluster-membership**.

Data Grid also logs the following messages when nodes join clusters:

```
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
```

-

Reference

Setting Up Data Grid Clusters

1.7. SHUTTING DOWN DATA GRID SERVER

Stop individually running servers or bring down clusters gracefully.

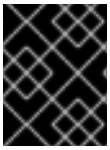
Procedure

1. Create a CLI connection to Data Grid.
2. Shut down Data Grid Server in one of the following ways:
   - Stop all nodes in a cluster with the **shutdown cluster** command, for example:

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

     This command saves cluster state to the **data** folder for each node in the cluster. If you use a cache store, the **shutdown cluster** command also persists all data in the cache.
   - Stop individual server instances with the **shutdown server** command and the server hostname, for example:

     ```
     [//containers/default]> shutdown server <my_server01>
     ```



IMPORTANT

The **shutdown server** command does not wait for rebalancing operations to complete, which can lead to data loss if you specify multiple hostnames at the same time.

TIP

Run **help shutdown** for more details about using the command.

Verification

Data Grid logs the following messages when you shut down servers:

```
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
```

1.7.1. Restarting Data Grid Clusters

When you bring Data Grid clusters back online after shutting them down, you should wait for the cluster to be available before adding or removing nodes or modifying cluster state.

If you shutdown clustered nodes with the **shutdown server** command, you must restart each server in reverse order.
For example, if you shutdown **server1** and then shutdown **server2**, you should first start **server2** and

then start **server1**.

If you shutdown a cluster with the **shutdown cluster** command, clusters become fully operational only after all nodes rejoin.
You can restart nodes in any order but the cluster remains in DEGRADED state until all nodes that were joined before shutdown are running.

1.8. DATA GRID SERVER FILESYSTEM

Data Grid Server uses the following folders on the host filesystem under **$RHDG_HOME**:

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

TIP

See the Data Grid Server README for descriptions of the each folder in your **$RHDG_HOME** directory as well as system properties you can use to customize the filesystem.

1.8.1. Server Root Directory

Apart from resources in the **bin** and **docs** folders, the only folder under **$RHDG_HOME** that you should interact with is the server root directory, which is named **server** by default.

You can create multiple nodes under the same **$RHDG_HOME** directory or in different directories, but each Data Grid Server instance must have its own server root directory. For example, a cluster of 5 nodes could have the following server root directories on the filesystem:

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

Each server root directory should contain the following folders:

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

server/conf
Holds **infinispan.xml** configuration files for a Data Grid Server instance.

Data Grid separates configuration into two layers:

Dynamic

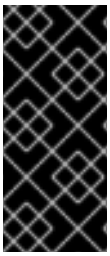
Create mutable cache configurations for data scalability.

Data Grid Server permanently saves the caches you create at runtime along with the cluster state that is distributed across nodes. Each joining node receives a complete cluster state that Data Grid Server synchronizes across all nodes whenever changes occur.

Static

Add configuration to **infinispan.xml** for underlying server mechanisms such as cluster transport, security, and shared datasources.

server/data
Provides internal storage that Data Grid Server uses to maintain cluster state.



IMPORTANT

Never directly delete or modify content in **server/data**.

Modifying files such as **caches.xml** while the server is running can cause corruption. Deleting content can result in an incorrect state, which means clusters cannot restart after shutdown.

server/lib
Contains extension **JAR** files for custom filters, custom event listeners, JDBC drivers, custom **ServerTask** implementations, and so on.

server/log
Holds Data Grid Server log files.

Reference

- Data Grid Server README
- What is stored in the <server>/data directory used by a RHDG server (Red Hat Knowledgebase)

CHAPTER 2. 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.

2.1. SERVER INTERFACES

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

2.1.1. 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.

2.1.2. 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>
```

2.1.3. Non-Loopback Strategy

Selects a non-loopback address.

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

2.1.4. Network Address Strategy

Selects networks based on IP address.

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

2.1.5. 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>
```

2.1.6. 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>
```

2.1.7. 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>
```

2.1.8. 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>
```

2.1.9. 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>
```

2.1.10. 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>
```

2.1.11. 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>
```

2.1.12. 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
```

2.2. 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>
```

2.2.1. 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
```

2.3. 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.

2.3.1. 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 3. CONFIGURING DATA GRID SERVER ENDPOINTS

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

3.1. 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.

3.1.1. 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.

3.1.2. 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.

3.1.3. Protocol Comparison

Table 3.1. 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

3.2. 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.

3.2.1. 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.

3.2.2. 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.

3.3. DATA GRID SERVER PORTS AND PROTOCOLS

Data Grid Server exposes endpoints on your network for remote client access.

Port	Protocol	Description
11222	TCP	Hot Rod and REST endpoint
11221	TCP	Memcached endpoint, which is disabled by default.

3.3.1. Configuring Network Firewalls for Remote Connections

Adjust any firewall rules to allow traffic between the server and external clients.

Procedure

On Red Hat Enterprise Linux (RHEL) workstations, for example, you can allow traffic to port **11222** with firewalld as follows:

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

To configure firewall rules that apply across a network, you can use the nftables utility.

Reference

- Using and configuring firewalld
- Getting started with nftables

CHAPTER 4. SECURING ACCESS TO DATA GRID SERVERS

Configure authentication and encryption mechanisms to secure access to Data Grid servers and protect your data.

4.1. DEFINING DATA GRID SERVER SECURITY REALMS

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

4.1.1. 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.

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

Property realm configuration

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:11.0 https://infinispan.org/schemas/infinispan-server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <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**

4.1.1.1. Creating and Modifying Users

Data Grid Server requires users to authenticate against a default property realm. Before you can access Data Grid Server, you must add credentials by creating at least one user and a password. You can also add and modify the security authorization groups to which users belong.

Procedure

1. Open a terminal in **$RHDG_HOME**.
2. Create and modify Data Grid users with the **user** command.

TIP

Run **help user** for more details about using the command.

Creating users and passwords

- Linux

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

- Microsoft Windows

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

Creating users with group membership

- Linux

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

- Microsoft Windows

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

4.1.2. 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:11.0 https://infinispan.org/schemas/infinispan-server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <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

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**

4.1.2.1. 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:11.0 https://infinispan.org/schemas/infinispan-
server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <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.

4.1.3. 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:11.0 https://infinispan.org/schemas/infinispan-server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <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**

4.1.4. 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:11.0 https://infinispan.org/schemas/infinispan-server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <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**

4.2. CREATING DATA GRID SERVER IDENTITIES

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

4.2.1. 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.

4.2.1.1. 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:11.0 https://infinispan.org/schemas/infinispan-server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <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.

4.2.1.2. 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.

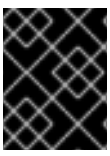
SSL server identity with a generated keystore

```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:11.0 https://infinispan.org/schemas/infinispan-
server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <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**

4.2.1.3. 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

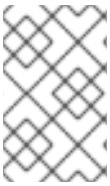
```
<security xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:infinispan:server:11.0
        https://infinispan.org/schemas/infinispan-server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <security-realms>
    <security-realm name="default">
      <server-identities>
        <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.

4.2.2. 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.

4.2.2.1. 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:11.0 https://infinispan.org/schemas/infinispan-
server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0">
  <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.

4.3. CONFIGURING ENDPOINT AUTHENTICATION MECHANISMS

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

Data Grid servers require user authentication to access the command line interface (CLI) and console as well as the Hot Rod and REST endpoints. Data Grid servers also automatically configure authentication mechanisms based on the security realms that you define.

4.3.1. Data Grid Server Authentication

Data Grid servers automatically configure authentication mechanisms based on the security realm that you assign to endpoints.

SASL Authentication Mechanisms

The following SASL authentication mechanisms apply to Hot Rod endpoints:

Security Realm	SASL Authentication Mechanism
Property Realms and LDAP Realms	SCRAM-*, DIGEST-*, CRAM-MD5
Token Realms	OAUTHBEARER
Trust Realms	EXTERNAL
Kerberos Identities	GSSAPI, GS2-KRB5
SSL/TLS Identities	PLAIN

HTTP Authentication Mechanisms

The following HTTP authentication mechanisms apply to REST endpoints:

Security Realm	HTTP Authentication Mechanism
Property Realms and LDAP Realms	DIGEST
Token Realms	BEARER_TOKEN
Trust Realms	CLIENT_CERT
Kerberos Identities	SPNEGO

Security Realm	HTTP Authentication Mechanism
SSL/TLS Identities	BASIC

Default Configuration

Data Grid servers provide a security realm named "default" that uses a property realm with plain text credentials defined in **$RHDG_HOME/server/ conf/users.properties**, as shown in the following snippet:

```
<security-realm name="default">
  <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>
```

The **endpoints** configuration assigns the "default" security realm to the Hot Rod and REST connectors, as follows:

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

As a result of the preceding configuration, Data Grid servers require authentication with a mechanism that the property realm supports.

4.3.2. Manually Configuring Hot Rod Authentication

Explicitly configure Hot Rod connector authentication to override the default SASL authentication mechanisms that Data Grid servers use for security realms.

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.

4.3.2.1. 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:11.0
           https://infinispan.org/schemas/infinispan-server-11.0.xsd"
           xmlns="urn:infinispan:server:11.0"
```

```
        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:11.0 https://infinispan.org/schemas/infinispan-
server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0"
        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.

4.3.2.2. Hot Rod Endpoint Authentication Mechanisms

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

Authentication mechanism	Description	Related details

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.
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.

4.3.2.3. 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.

4.3.2.4. 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
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.

4.3.3. Manually Configuring REST Authentication

Explicitly configure REST connector authentication to override the default HTTP authentication mechanisms that Data Grid servers use for security realms.

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 authentication mechanisms for the REST endpoint to use.

4.3.3.1. 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:11.0 https://infinispan.org/schemas/infinispan-server-11.0.xsd"
          xmlns="urn:infinispan:server:11.0"
          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:11.0 https://infinispan.org/schemas/infinispan-server-11.0.xsd"
        xmlns="urn:infinispan:server:11.0"
        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.

4.3.3.2. 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.
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.

4.4. DISABLING DATA GRID SERVER AUTHENTICATION

In local development environments or on isolated networks you can configure Data Grid servers to allow unauthenticated client requests.

Procedure

1. Remove any **security-realm** attributes from the **endpoints** configuration.
2. Ensure that the Hot Rod and REST connectors do not include any **authentication** definitions.

For example, the following configuration allows unauthenticated access to Data Grid:

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

4.5. CONFIGURING DATA GRID AUTHORIZATION

Authorization restricts the ability to perform operations with Data Grid and access data. You assign users with roles that have different permission levels.

4.5.1. Data Grid Authorization

Data Grid lets you configure authorization to secure Cache Managers and cache instances. When user applications or clients attempt to perform an operation on secured Cached Managers and caches, they must provide an identity with a role that has sufficient permissions to perform that operation.

For example, you configure authorization on a specific cache instance so that invoking **Cache.get()** requires an identity to be assigned a role with read permission while **Cache.put()** requires a role with write permission.

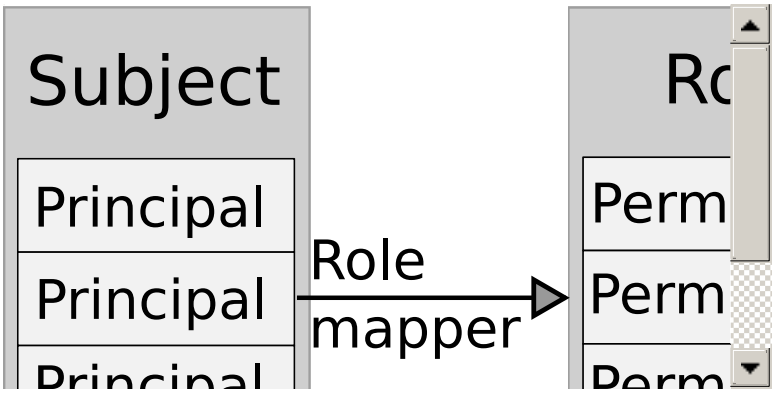
In this scenario, if a user application or client with the **reader** role attempts to write an entry, Data Grid denies the request and throws a security exception. If a user application or client with the **writer** role sends a write request, Data Grid validates authorization and issues a token for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

Identity to Role Mapping

Identities are security Principals of type **java.security.Principal**. Subjects, implemented with the **javax.security.auth.Subject** class, represent a group of security Principals. In other words, a Subject represents a user and all groups to which it belongs.

Data Grid uses role mappers so that security principals correspond to roles, which represent one or more permissions.

The following image illustrates how security principals map to roles:



4.5.1.1. Permissions

Permissions control access to Cache Managers and caches by restricting the actions that you can perform. Permissions can also apply to specific entities such as named caches.

Table 4.1. Cache Manager Permissions

Permission	Function	Description
CONFIGURATION	**defineConfiguration**	Defines new cache configurations.
LISTEN	**addListener**	Registers listeners against a Cache Manager.
LIFECYCLE	**stop**	Stops the Cache Manager.
ALL	-	Includes all Cache Manager permissions.

Table 4.2. Cache Permissions

Permission	Function	Description
READ	**get**, **contains**	Retrieves entries from a cache.
WRITE	**put**, **putIfAbsent**, **replace**, **remove**, **evict**	Writes, replaces, removes, evicts data in a cache.
EXEC	**distexec**, **streams**	Allows code execution against a cache.
LISTEN	**addListener**	Registers listeners against a cache.
BULK_READ	**keySet**, **values**, **entrySet**, **query**	Executes bulk retrieve operations.
BULK_WRITE	**clear**, **putAll**	Executes bulk write operations.

Permission	Function	Description
LIFECYCLE	**start**, **stop**	Starts and stops a cache.
ADMIN	**getVersion**, **addInterceptor***, **removeInterceptor**, **getInterceptorChain**, **getEvictionManager**, **getComponentRegistry**, **getDistributionManager**, **getAuthorizationManager**, **evict**, **getRpcManager**, **getCacheConfiguration**, **getCacheManager**, **getInvocationContextContainer**, **setAvailability**, **getDataContainer**, **getStats**, **getXAResource**	Allows access to underlying components and internal structures.
ALL	-	Includes all cache permissions.
ALL_READ	-	Combines the READ and BULK_READ permissions.
ALL_WRITE	-	Combines the WRITE and BULK_WRITE permissions.

Combining permissions

You might need to combine permissions so that they are useful. For example, to allow "supervisors" to run stream operations but restrict "standard" users to puts and gets only, you can define the following mappings:

```
<role name="standard" permission="READ WRITE" />
<role name="supervisors" permission="READ WRITE EXEC BULK"/>
```

Reference

- Data Grid Security API

4.5.1.2. Role Mappers

Data Grid includes a **PrincipalRoleMapper** API that maps security Principals in a Subject to authorization roles. There are two role mappers available by default:

IdentityRoleMapper

Uses the Principal name as the role name.

- Java class: **org.infinispan.security.mappers.IdentityRoleMapper**
- Declarative configuration: **<identity-role-mapper />**

CommonNameRoleMapper

Uses the Common Name (CN) as the role name if the Principal name is a Distinguished Name (DN). For example the **cn=managers,ou=people,dc=example,dc=com** DN maps to the **managers** role.

- Java class: **org.infinispan.security.mappers.CommonRoleMapper**
- Declarative configuration: **<common-name-role-mapper />**

You can also use custom role mappers that implement the **org.infinispan.security.PrincipalRoleMapper** interface. To configure custom role mappers declaratively, use: **<custom-role-mapper class="my.custom.RoleMapper" />**

Reference

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

4.5.2. Declaratively Configuring Authorization

Configure authorization in your **infinispan.xml** file.

Procedure

1. Configure the global authorization settings in the **cache-container** that specify a role mapper, and define a set of roles and permissions.
2. Configure authorization for caches to restrict access based on user roles.

```
<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/> 4
      </security>
    </local-cache>
  </cache-container>
</infinispan>
```

1. Enables Data Grid authorization for the Cache Manager.
2. Specifies an implementation of **PrincipalRoleMapper** that maps Principals to roles.
3. Defines roles and their associated permissions.

4 Implicitly adds all roles from the global configuration.

If you do not want to apply all roles to a cache, explicitly define the roles that are authorized for caches as follows:

```
<infinispan>
  <cache-container default-cache="secured" name="secured">
    <security>
      <authorization>
        <identity-role-mapper />
        <role name="admin" permissions="ALL" />
        <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 supervisor reader"/> 1
      </security>
    </local-cache>
  </cache-container>

</infinispan>
```

1 Defines authorized roles for the cache. In this example, users who have the **writer** role only are not authorized for the "secured" cache. Data Grid denies any access requests from those users.

Reference

- Data Grid Configuration Schema Reference

CHAPTER 5. SETTING UP DATA GRID CLUSTERS

Data Grid requires a transport layer so nodes can automatically join and leave clusters. The transport layer also enables Data Grid nodes to replicate or distribute data across the network and perform operations such as re-balancing and state transfer.

5.1. GETTING STARTED WITH DEFAULT STACKS

Data Grid uses JGroups protocol stacks so nodes can send each other messages on dedicated cluster channels.

Data Grid provides preconfigured JGroups stacks for **UDP** and **TCP** protocols. You can use these default stacks as a starting point for building custom cluster transport configuration that is optimized for your network requirements.

Procedure

1. Locate the default JGroups stacks, **default-jgroups-*.xml**, in the **default-configs** directory inside the **infinispan-core-11.0.9.Final-redhat-00001.jar** file.
   The **jar** file is in the **$RHDG_HOME/lib** directory.

2. Do one of the following:

   - Use the **stack** attribute in your **infinispan.xml** file.

```
<infinispan>
  <cache-container default-cache="replicatedCache">
    <transport cluster="${infinispan.cluster.name}"
               stack="udp" 1
               node-name="${infinispan.node.name:}"/>
  </cache-container>
</infinispan>
```

   1 Uses **default-jgroups-udp.xml** for cluster transport.

   - Use the **cluster-stack** argument when you start the server:

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

Data Grid logs the following message to indicate which stack it uses:

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

Reference

- JGroups cluster transport configuration for Data Grid 8.x (Red Hat knowledgebase article)

5.1.1. Default JGroups Stacks

Learn about default JGroups stacks that configure cluster transport.

File name	Stack name	Description
default-jgroups-udp.xml	**udp**	Uses UDP for transport and UDP multicast for discovery. Suitable for larger clusters (over 100 nodes) or if you are using replicated caches or invalidation mode. Minimizes the number of open sockets.
default-jgroups-tcp.xml	**tcp**	Uses TCP for transport and the **MPING** protocol for discovery, which uses **UDP** multicast. Suitable for smaller clusters (under 100 nodes) *only if* you are using distributed caches because TCP is more efficient than UDP as a point-to-point protocol.
default-jgroups-ec2.xml	**ec2**	Uses TCP for transport and **S3_PING** for discovery. Suitable for Amazon EC2 nodes where UDP multicast is not available.
default-jgroups-kubernetes.xml	**kubernetes**	Uses TCP for transport and **DNS_PING** for discovery. Suitable for Kubernetes and Red Hat OpenShift nodes where UDP multicast is not always available.
default-jgroups-google.xml	**google**	Uses TCP for transport and **GOOGLE_PING2** for discovery. Suitable for Google Cloud Platform nodes where UDP multicast is not available.
default-jgroups-azure.xml	**azure**	Uses TCP for transport and **AZURE_PING** for discovery. Suitable for Microsoft Azure nodes where UDP multicast is not available.

Reference

- JGroups Protocols

5.1.2. TCP and UDP Ports for Cluster Traffic

Data Grid uses the following ports for cluster transport messages:

Default Port	Protocol	Description
7800	TCP/UDP	JGroups cluster bind port
46655	UDP	JGroups multicast

Cross-Site Replication
Data Grid uses the following ports for the JGroups RELAY2 protocol:

7900

For Data Grid clusters running on OpenShift.

7800

If using UDP for traffic between nodes and TCP for traffic between clusters.

7801

If using TCP for traffic between nodes and TCP for traffic between clusters.

5.2. CUSTOMIZING JGROUPS STACKS

Adjust and tune properties to create a cluster transport configuration that works for your network requirements.

Data Grid provides attributes that let you extend the default JGroups stacks for easier configuration. You can inherit properties from the default stacks while combining, removing, and replacing other properties.

Procedure

1. Create a new JGroups stack declaration in your **infinispan.xml** file.

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

1 Creates a custom JGroups stack named "my-stack".

2. Add the **extends** attribute and specify a JGroups stack to inherit properties from.

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

1 Inherits from the default TCP stack.

3. Use the **stack.combine** attribute to modify properties for protocols configured in the inherited stack.

4. Use the **stack.position** attribute to define the location for your custom stack.
For example, you might evaluate using a Gossip router and symmetric encryption with the default TCP stack as follows:

```
<jgroups>
  <stack name="my-stack" extends="tcp">
    <TCPGOSSIP initial_hosts="${jgroups.tunnel.gossip_router_hosts:localhost[12001]}"
          stack.combine="REPLACE"
          stack.position="MPING" /> 1
```

```
    <FD_SOCK stack.combine="REMOVE"/> 2
    <VERIFY_SUSPECT timeout="2000"/> 3
    <SYM_ENCRYPT sym_algorithm="AES"
                 keystore_name="mykeystore.p12"
                 keystore_type="PKCS12"
                 store_password="changeit"
                 key_password="changeit"
                 alias="myKey"
                 stack.combine="INSERT_AFTER"
                 stack.position="VERIFY_SUSPECT" /> 4
  </stack>
</jgroups>
```

1. Uses the **TCPGOSSIP** protocol as the discovery mechanism instead of **MPING**.
2. Removes the **FD_SOCK** protocol from the stack.
3. Modifies the timeout value for the **VERIFY_SUSPECT** protocol.
4. Adds the **SYM_ENCRYPT** protocol to the stack after the **VERIFY_SUSPECT** protocol.

5. Specify the stack name as the value for the **stack** attribute in the **transport** configuration.

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

1. Configures Data Grid to use "my-stack" for cluster transport.

6. Check Data Grid logs to ensure it uses the stack.

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

Reference

- JGroups cluster transport configuration for Data Grid 8.x (Red Hat knowledgebase article)

5.2.1. Inheritance Attributes

When you extend a JGroups stack, inheritance attributes let you adjust protocols and properties in the stack you are extending.

- **stack.position** specifies protocols to modify.

- **stack.combine** uses the following values to extend JGroups stacks:

Value	Description
COMBINE	Overrides protocol properties.
REPLACE	Replaces protocols.
INSERT_AFTER	Adds a protocol into the stack after another protocol. Does not affect the protocol that you specify as the insertion point. Protocols in JGroups stacks affect each other based on their location in the stack. For example, you should put a protocol such as **NAKACK2** after the **SYM_ENCRYPT** or **ASYM_ENCRYPT** protocol so that **NAKACK2** is secured.
REMOVE	Removes protocols from the stack.

5.3. USING JGROUPS SYSTEM PROPERTIES

Pass system properties to Data Grid at startup to tune cluster transport.

Procedure

- Use **-D<property-name>=<property-value>** arguments to set JGroups system properties as required.

For example, set a custom bind port and IP address as follows:

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

5.3.1. System Properties for JGroups Stacks

Set system properties that configure JGroups cluster transport stacks.

System Property	Description	Default Value	Required/ Optional
jgroups.bind.address	Bind address for cluster transport.	**SITE_LOCAL**	Optional
jgroups.bind.port	Bind port for the socket.	**7800**	Optional
jgroups.mcast_addr	IP address for multicast, both discovery and inter-cluster communication. The IP address must be a valid "class D" address that is suitable for IP multicast.	**228.6.7.8**	Optional

System Property	Description	Default Value	Required/ Optional
jgroups.mcast_port	Port for the multicast socket.	**46655**	Optional
jgroups.ip_ttl	Time-to-live (TTL) for IP multicast packets. The value defines the number of network hops a packet can make before it is dropped.	2	Optional
jgroups.thread_pool.min_threads	Minimum number of threads for the thread pool.	0	Optional
jgroups.thread_pool.max_threads	Maximum number of threads for the thread pool.	200	Optional
jgroups.join_timeout	Maximum number of milliseconds to wait for join requests to succeed.	2000	Optional
jgroups.thread_dumps_threshold	Number of times a thread pool needs to be full before a thread dump is logged.	10000	Optional

Amazon EC3

The following system properties apply only to **default-jgroups-ec2.xml**:

System Property	Description	Default Value	Required/ Optional
jgroups.s3.access_key	Amazon S3 access key for an S3 bucket.	No default value.	Optional
jgroups.s3.secret_access_key	Amazon S3 secret key used for an S3 bucket.	No default value.	Optional
jgroups.s3.bucket	Name of the Amazon S3 bucket. The name must exist and be unique.	No default value.	Optional

Kubernetes

The following system properties apply only to **default-jgroups-kubernetes.xml**:

System Property	Description	Default Value	Required/ Optional
jgroups.dns.query	Sets the DNS record that returns cluster members.	No default value.	Required

Google Cloud Platform

The following system properties apply only to **default-jgroups-google.xml**:

System Property	Description	Default Value	Required/ Optional
jgroups.google.bucket_name	Name of the Google Compute Engine bucket. The name must exist and be unique.	No default value.	Required

Reference

- JGroups System Properties
- JGroups Protocol List

5.4. USING INLINE JGROUPS STACKS

You can insert complete JGroups stack definitions into **infinispan.xml** files.

Procedure

- Embed a custom JGroups stack declaration in your **infinispan.xml** file.

```
<infinispan>
  <jgroups> 1
    <stack name="prod"> 2
      <TCP bind_port="7800" port_range="30" recv_buf_size="20000000"
send_buf_size="640000"/>
      <MPING bind_addr="127.0.0.1" break_on_coord_rsp="true"
           mcast_addr="${jgroups.mping.mcast_addr:228.2.4.6}"
           mcast_port="${jgroups.mping.mcast_port:43366}"
           num_discovery_runs="3"
           ip_ttl="${jgroups.udp.ip_ttl:2}"/>
      <MERGE3 />
      <FD_SOCK />
      <FD_ALL timeout="3000" interval="1000" timeout_check_interval="1000" />
      <VERIFY_SUSPECT timeout="1000" />
      <pbcast.NAKACK2 use_mcast_xmit="false" xmit_interval="100"
xmit_table_num_rows="50"
                      xmit_table_msgs_per_row="1024" xmit_table_max_compaction_time="30000"
/>
```

```
      <UNICAST3 xmit_interval="100" xmit_table_num_rows="50"
xmit_table_msgs_per_row="1024"
                xmit_table_max_compaction_time="30000" />
      <pbcast.STABLE stability_delay="200" desired_avg_gossip="2000" max_bytes="1M" />
      <pbcast.GMS print_local_addr="false" join_timeout="${jgroups.join_timeout:2000}" />
      <UFC max_credits="4m" min_threshold="0.40" />
      <MFC max_credits="4m" min_threshold="0.40" />
      <FRAG3 />
    </stack>
  </jgroups>
  <cache-container default-cache="replicatedCache">
    <transport stack="prod" /> 3
    ...
  </cache-container>
</infinispan>
```

1. Contains one or more JGroups stack definitions.
2. Defines a custom JGroups stack named "prod".
3. Configures Data Grid to use "prod" for cluster transport.

5.5. USING EXTERNAL JGROUPS STACKS

Reference external files that define custom JGroups stacks in **infinispan.xml** files.

Procedure

1. Add custom JGroups stack files to the **$RHDG_HOME/server/conf** directory.
   Alternatively you can specify an absolute path when you declare the external stack file.
2. Reference the external stack file with the **stack-file** element.

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

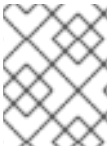
1. Creates a stack named "prod-tcp" that uses the "prod-jgroups-tcp.xml" definition.
2. Configures Data Grid to use "prod-tcp" for cluster transport.

5.6. CLUSTER DISCOVERY PROTOCOLS

Data Grid supports different protocols that allow nodes to automatically find each other on the network and form clusters.

There are two types of discovery mechanisms that Data Grid can use:

- Generic discovery protocols that work on most networks and do not rely on external services.
- Discovery protocols that rely on external services to store and retrieve topology information for Data Grid clusters.
  For instance the DNS_PING protocol performs discovery through DNS server records.



NOTE

Running Data Grid on hosted platforms requires using discovery mechanisms that are adapted to network constraints that individual cloud providers impose.

Reference

- JGroups Discovery Protocols
- JGroups cluster transport configuration for Data Grid 8.x (Red Hat knowledgebase article)

5.6.1. PING

PING, or UDPPING is a generic JGroups discovery mechanism that uses dynamic multicasting with the UDP protocol.

When joining, nodes send PING requests to an IP multicast address to discover other nodes already in the Data Grid cluster. Each node responds to the PING request with a packet that contains the address of the coordinator node and its own address. C=coordinator's address and A=own address. If no nodes respond to the PING request, the joining node becomes the coordinator node in a new cluster.

PING configuration example

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

Reference

- JGroups PING

5.6.2. TCPPING

TCPPING is a generic JGroups discovery mechanism that uses a list of static addresses for cluster members.

With TCPPING, you manually specify the IP address or hostname of each node in the Data Grid cluster as part of the JGroups stack, rather than letting nodes discover each other dynamically.

TCPPING configuration example

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

```
        port_range="0" 1
        num_initial_members="3"/>
  ...
</config>
```

1 For reliable discovery, Red Hat recommends **port-range=0**.

Reference

- JGroups TCPPING

5.6.3. MPING

MPING uses IP multicast to discover the initial membership of Data Grid clusters.

You can use MPING to replace TCPPING discovery with TCP stacks and use multicasing for discovery instead of static lists of initial hosts. However, you can also use MPING with UDP stacks.

MPING configuration example

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

Reference

- JGroups MPING

5.6.4. TCPGOSSIP

Gossip routers provide a centralized location on the network from which your Data Grid cluster can retrieve addresses of other nodes.

You inject the address (**IP:PORT**) of the Gossip router into Data Grid nodes as follows:

1. Pass the address as a system property to the JVM; for example, **-DGossipRouterAddress="10.10.2.4[12001]"**.

2. Reference that system property in the JGroups configuration file.

Gossip router configuration example

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

Reference

- JGroups Gossip Router

5.6.5. JDBC_PING

JDBC_PING uses shared databases to store information about Data Grid clusters. This protocol supports any database that can use a JDBC connection.

Nodes write their IP addresses to the shared database so joining nodes can find the Data Grid cluster on the network. When nodes leave Data Grid clusters, they delete their IP addresses from the shared database.

JDBC_PING configuration example

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

IMPORTANT

Add the appropriate JDBC driver to the classpath so Data Grid can use JDBC_PING.

Reference

- JDBC_PING
- JDBC_PING Wiki

5.6.6. DNS_PING

JGroups DNS_PING queries DNS servers to discover Data Grid cluster members in Kubernetes environments such as OKD and Red Hat OpenShift.

DNS_PING configuration example

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

Reference

- JGroups DNS_PING
- DNS for Services and Pods (Kubernetes documentation for adding DNS entries)

5.7. ENCRYPTING CLUSTER TRANSPORT

Secure cluster transport so that nodes communicate with encrypted messages. You can also configure Data Grid clusters to perform certificate authentication so that only nodes with valid identities can join.

5.7.1. Data Grid Cluster Security

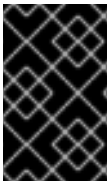
To secure cluster traffic, you configure Data Grid nodes to encrypt JGroups message payloads with secret keys.

Data Grid nodes can obtain secret keys from either:

- The coordinator node (asymmetric encryption).
- A shared keystore (symmetric encryption).

Retrieving secret keys from coordinator nodes

You configure asymmetric encryption by adding the **ASYM_ENCRYPT** protocol to a JGroups stack in your Data Grid configuration. This allows Data Grid clusters to generate and distribute secret keys.



IMPORTANT

When using asymmetric encryption, you should also provide keystores so that nodes can perform certificate authentication and securely exchange secret keys. This protects your cluster from man-in-the-middle (MitM) attacks.

Asymmetric encryption secures cluster traffic as follows:

1. The first node in the Data Grid cluster, the coordinator node, generates a secret key.
2. A joining node performs certificate authentication with the coordinator to mutually verify identity.
3. The joining node requests the secret key from the coordinator node. That request includes the public key for the joining node.
4. The coordinator node encrypts the secret key with the public key and returns it to the joining node.
5. The joining node decrypts and installs the secret key.
6. The node joins the cluster, encrypting and decrypting messages with the secret key.

Retrieving secret keys from shared keystores

You configure symmetric encryption by adding the **SYM_ENCRYPT** protocol to a JGroups stack in your Data Grid configuration. This allows Data Grid clusters to obtain secret keys from keystores that you provide.

1. Nodes install the secret key from a keystore on the Data Grid classpath at startup.
2. Node join clusters, encrypting and decrypting messages with the secret key.

Comparison of asymmetric and symmetric encryption

ASYM_ENCRYPT with certificate authentication provides an additional layer of encryption in comparison with **SYM_ENCRYPT**. You provide keystores that encrypt the requests to coordinator nodes for the secret key. Data Grid automatically generates that secret key and handles cluster traffic,

while letting you specify when to generate secret keys. For example, you can configure clusters to generate new secret keys when nodes leave. This ensures that nodes cannot bypass certificate authentication and join with old keys.

SYM_ENCRYPT, on the other hand, is faster than **ASYM_ENCRYPT** because nodes do not need to exchange keys with the cluster coordinator. A potential drawback to **SYM_ENCRYPT** is that there is no configuration to automatically generate new secret keys when cluster membership changes. Users are responsible for generating and distributing the secret keys that nodes use to encrypt cluster traffic.

5.7.2. Configuring Cluster Transport with Asymmetric Encryption

Configure Data Grid clusters to generate and distribute secret keys that encrypt JGroups messages.

Procedure

1. Create a keystore with certificate chains that enables Data Grid to verify node identity.
2. Place the keystore on the classpath for each node in the cluster.
   For Data Grid Server, you put the keystore in the $RHDG_HOME directory.
3. Add the **SSL_KEY_EXCHANGE** and **ASYM_ENCRYPT** protocols to a JGroups stack in your Data Grid configuration, as in the following example:

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

1. Creates a secure JGroups stack named "encrypt-tcp" that extends the default TCP stack for Data Grid.
2. Names the keystore that nodes use to perform certificate authentication.
3. Specifies the keystore password.
4. Uses the **stack.combine** and **stack.position** attributes to insert **SSL_KEY_EXCHANGE** into the default TCP stack after the **VERIFY_SUSPECT** protocol.

into the default TCP stack after the **VERIFY_SUSPECT** protocol.

5. Specifies the length of the secret key that the coordinator node generates. The default value is **2048**.
6. Specifies the cipher engine the coordinator node uses to generate secret keys. The default value is **RSA**.
7. Configures Data Grid to generate and distribute a new secret key when the coordinator node changes.
8. Configures Data Grid to generate and distribute a new secret key when nodes leave.
9. Configures Data Grid nodes to use the **SSL_KEY_EXCHANGE** protocol for certificate authentication.
10. Uses the **stack.combine** and **stack.position** attributes to insert **ASYM_ENCRYPT** into the default TCP stack after the **SSL_KEY_EXCHANGE** protocol.
11. Configures the Data Grid cluster to use the secure JGroups stack.

Verification

When you start your Data Grid cluster, the following log message indicates that the cluster is using the secure JGroups stack:

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

Data Grid nodes can join the cluster only if they use **ASYM_ENCRYPT** and can obtain the secret key from the coordinator node. Otherwise the following message is written to Data Grid logs:

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

Reference

The example **ASYM_ENCRYPT** configuration in this procedure shows commonly used parameters. Refer to JGroups documentation for the full set of available parameters.

- JGroups 4 Manual
- JGroups 4.2 Schema

5.7.3. Configuring Cluster Transport with Symmetric Encryption

Configure Data Grid clusters to encrypt JGroups messages with secret keys from keystores that you provide.

Procedure

1. Create a keystore that contains a secret key.
2. Place the keystore on the classpath for each node in the cluster.
   For Data Grid Server, you put the keystore in the $RHDG_HOME directory.

3. Add the **SYM_ENCRYPT** protocol to a JGroups stack in your Data Grid configuration, as in the following example:

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

(1) Creates a secure JGroups stack named "encrypt-tcp" that extends the default TCP stack for Data Grid.

(2) Names the keystore from which nodes obtain secret keys.

(3) Specifies the keystore type. JGroups uses JCEKS by default.

(4) Specifies the keystore password.

(5) Specifies the secret key password.

(6) Specifies the secret key alias.

(7) Uses the **stack.combine** and **stack.position** attributes to insert **SYM_ENCRYPT** into the default TCP stack after the **VERIFY_SUSPECT** protocol.

(8) Configures the Data Grid cluster to use the secure JGroups stack.

Verification

When you start your Data Grid cluster, the following log message indicates that the cluster is using the secure JGroups stack:

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

Data Grid nodes can join the cluster only if they use **SYM_ENCRYPT** and can obtain the secret key from the shared keystore. Otherwise the following message is written to Data Grid logs:

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

Reference

The example **SYM_ENCRYPT** configuration in this procedure shows commonly used parameters. Refer to JGroups documentation for the full set of available parameters.

- JGroups 4 Manual
- JGroups 4.2 Schema

CHAPTER 6. REMOTELY CREATING DATA GRID CACHES

Add caches to Data Grid Server so you can store data.

6.1. CACHE CONFIGURATION WITH DATA GRID SERVER

Caches configure the data container on Data Grid Server.

You create caches at run-time by adding definitions based on **org.infinispan** templates or Data Grid configuration through the console, the Command Line Interface (CLI), the Hot Rod endpoint, or the REST endpoint.

IMPORTANT

When you create caches at run-time, Data Grid Server replicates your cache definitions across the cluster.

Configuration that you declare directly in **infinispan.xml** is not automatically synchronized across Data Grid clusters. In this case you should use configuration management tooling, such as Ansible or Chef, to ensure that configuration is propagated to all nodes in your cluster.

6.2. DEFAULT CACHE MANAGER

Data Grid Server provides a default Cache Manager configuration. When you start Data Grid Server, it instantiates the Cache Manager so you can remotely create caches at run-time.

Default Cache Manager

```
<cache-container name="default" 1
                 statistics="true"> 2
  <transport cluster="${infinispan.cluster.name:cluster}" 3
             stack="${infinispan.cluster.stack:tcp}" 4
             node-name="${infinispan.node.name:}"/> 5
</cache-container>
```

1. Creates a Cache Manager named "default".
2. Exports Cache Manager statistics through the **metrics** endpoint.
3. Adds a JGroups cluster transport that allows Data Grid servers to automatically discover each other and form clusters.
4. Uses the default TCP stack for cluster traffic.
5. Individual name for the node, must be unique across the cluster. Uses a unified hostname by default.

Examining the Cache Manager

After you start Data Grid Server and add user credentials, you can access the default Cache Manager through the Command Line Interface (CLI) or REST endpoint as follows:

- CLI: Use the **describe** command in the default container.

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

- REST: Navigate to **<server_hostname>:11222/rest/v2/cache-managers/default/** in any browser.

6.3. CREATING CACHES WITH THE DATA GRID CONSOLE

Dynamically add caches from templates or configuration files through the Data Grid console.

Prerequisites

Create a user and start at least one Data Grid server instance.

Procedure

1. Navigate to **<server_hostname>:11222/console/** in any browser.
2. Log in to the console.
3. Open the **Data Container** view.
4. Select **Create Cache** and then add a cache from a template or with Data Grid configuration in XML or JSON format.
5. Return to the **Data Container** view and verify your Data Grid cache.

6.4. CREATING CACHES WITH THE DATA GRID COMMAND LINE INTERFACE (CLI)

Use the Data Grid CLI to add caches from templates or configuration files in XML or JSON format.

Prerequisites

Create a user and start at least one Data Grid server instance.

Procedure

1. Create a CLI connection to Data Grid.
2. Add cache definitions with the **create cache** command.
   - Add a cache definition from an XML or JSON file with the **--file** option.

```
[//containers/default]> create cache --file=configuration.xml mycache
```

   - Add a cache definition from a template with the **--template** option.

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

TIP

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

3. Verify the cache exists with the **ls** command.

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

4. Retrieve the cache configuration with the **describe** command.

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

Reference

- Creating Data Grid CLI Connections
- Performing Cache Operations with the Data Grid CLI

6.5. CREATING CACHES WITH HOT ROD CLIENTS

Programmatically create caches on Data Grid Server through the **RemoteCacheManager** API.

NOTE

The following procedure demonstrates programmatic cache creation with the Hot Rod Java client. However Hot Rod clients are available in different languages such as Javascript or C++.

Prerequisites

- Create a user and start at least one Data Grid server instance.
- Get the Hot Rod Java client.

Procedure

1. Configure your client with the **ConfigurationBuilder** class.

```
import org.infinispan.client.hotrod.RemoteCacheManager;
import org.infinispan.client.hotrod.DefaultTemplate;
import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
import org.infinispan.commons.configuration.XMLStringConfiguration;
...

ConfigurationBuilder builder = new ConfigurationBuilder();
builder.addServer()
        .host("127.0.0.1")
        .port(11222)
      .security().authentication()
        .enable()
        .username("username")
        .password("password")
        .realm("default")
        .saslMechanism("DIGEST-MD5");

manager = new RemoteCacheManager(builder.build());
```

2. Use the **XMLStringConfiguration** class to add cache definitions in XML format.

3. Call the **getOrCreateCache()** method to add the cache if it already exists or create it if not.

```
private void createCacheWithXMLConfiguration() {
   String cacheName = "CacheWithXMLConfiguration";
   String xml = String.format("<infinispan>" +
                              "<cache-container>" +
                              "<distributed-cache name=\"%s\" mode=\"SYNC\"
                              statistics=\"true\">" +
                                "<locking isolation=\"READ_COMMITTED\"/>" +
                                "<transaction mode=\"NON_XA\"/>" +
                                "<expiration lifespan=\"60000\" interval=\"20000\"/>" +
                              "</distributed-cache>" +
                              "</cache-container>" +
                            "</infinispan>"
                            , cacheName);
   manager.administration().getOrCreateCache(cacheName, new
XMLStringConfiguration(xml));
   System.out.println("Cache created or already exists.");
}
```

4. Create caches with **org.infinispan** templates as in the following example with the **createCache()** invocation:

```
private void createCacheWithTemplate() {
   manager.administration().createCache("myCache", "org.infinispan.DIST_SYNC");
   System.out.println("Cache created.");
}
```

Next Steps

Try some working code examples that show you how to create remote caches with the Hot Rod Java client. Visit the Data Grid Tutorials.

Reference

- **RemoteCacheManager** Javadoc
- Getting the Hot Rod Java Client

6.6. CREATING DATA GRID CACHES WITH HTTP CLIENTS

Add cache definitions to Data Grid servers through the REST endpoint with any suitable HTTP client.

Prerequisites

Create a user and start at least one Data Grid server instance.

Procedure

- Create caches with **POST** requests to **/rest/v2/caches/$cacheName**.

Use XML or JSON configuration by including it in the request payload.

```
POST /rest/v2/caches/mycache
```

Use the **?template=** parameter to create caches from **org.infinispan** templates.

```
POST /rest/v2/caches/mycache?template=org.infinispan.DIST_SYNC
```

Reference

- Creating and Managing Caches with the REST API

6.7. DATA GRID CONFIGURATION

Data Grid configuration in XML and JSON format.

6.7.1. XML Configuration

Data Grid configuration in XML format must conform to the schema and include:

- **<infinispan>** root element.
- **<cache-container>** definition.

Example XML Configuration

```
<infinispan>
   <cache-container>
      <distributed-cache name="myCache" mode="SYNC">
        <encoding media-type="application/x-protostream"/>
        <memory max-count="1000000" when-full="REMOVE"/>
      </distributed-cache>
   </cache-container>
</infinispan>
```

6.7.2. JSON Configuration

Data Grid configuration in JSON format:

- Requires the cache definition only.
- Must follow the structure of an XML configuration.
  - XML elements become JSON objects.
  - XML attributes become JSON fields.

Example JSON Configuration

```
{
  "distributed-cache": {
    "name": "myCache",
    "mode": "SYNC",
    "encoding": {
      "media-type": "application/x-protostream"
```

```
    },
    "memory": {
      "max-count": 1000000,
      "when-full": "REMOVE"
    }
  }
}
```

CHAPTER 7. CONFIGURING DATA GRID SERVER DATASOURCES

Create managed datasources to optimize connection pooling and performance for database connections.

You can specify database connection properties as part of a JDBC cache store configuration. However you must do this for each cache definition, which duplicates configuration and wastes resources by creating multiple distinct connection pools.

By using shared, managed datasources, you centralize connection configuration and pooling for more efficient usage.

7.1. DATASOURCE CONFIGURATION FOR JDBC CACHE STORES

Data Grid server configuration for datasources is composed of two sections:

- A **connection factory** that defines how to connect to the database.
- A **connection pool** that defines how to pool and reuse connections.

```
<data-sources>
  <data-source name="ds" jndi-name="jdbc/datasource" statistics="true"> 1
    <connection-factory driver="org.database.Driver" 2
                        username="db_user" 3
                        password="secret" 4
                        url="jdbc:db://database-host:10000/dbname" 5
                        new-connection-sql="SELECT 1" 6
                        transaction-isolation="READ_COMMITTED"> 7
      <connection-property name="name">value</connection-property> 8
    </connection-factory>
    <connection-pool
      initial-size="1" 9
      max-size="10" 10
      min-size="3" 11
      background-validation="1000" 12
      idle-removal="1" 13
      blocking-timeout="1000" 14
      leak-detection="10000"/> 15
  </data-source>
</data-sources>
```

1. Defines a datasource name, JNDI name, and whether to enable statistics collection.
2. Specifies the JDBC driver that creates connections. Place driver JARs in the **server/lib** directory.
3. Specifies a username for the connection.
4. Specifies a corresponding password for the connection.
5. Specifies the JDBC URL specific to the driver in use.

6 Adds a query that verifies new connections.

7 Configures one of the transaction isolation levels for the connection: **NONE**, **READ_UNCOMMITTED**, **READ_COMMITTED**, **REPEATABLE_READ**, **SERIALIZABLE**.

8 Sets optional JDBC driver-specific connection properties.

9 Defines the initial number of connections the pool contains.

10 Sets the maximum number of connections in the pool.

11 Sets the minimum number of connections the pool should contain.

12 Specifies the time, in milliseconds, between background validation runs.

13 Specifies the time, in minutes, a connections can remain idle before it is removed.

14 Specifies the amount of time, in milliseconds, to block while waiting for a connection, after which an exception is thrown.

15 Specifies the time, in milliseconds, a connection can be held before a leak warning occurs.

7.2. USING DATASOURCES IN JDBC CACHE STORES

Use a shared, managed datasource in your JDBC cache store configuration instead of specifying individual connection properties for each cache definition.

Prerequisites

Create a managed datasource for JDBC cache stores in your Data Grid server configuration.

Procedure

- Reference the JNDI name of the datasource in the JDBC cache store configuration of your cache configuration, as in the following example:

```
<distributed-cache-configuration name="persistent-cache"
xmlns:jdbc="urn:infinispan:config:store:jdbc:11.0">
   <persistence>
      <jdbc:string-keyed-jdbc-store>
         <jdbc:data-source jndi-url="jdbc/postgres"/> 1
         <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-configuration>
```

1 Specifies the JNDI name that you provided for the datasource connection in your Data Grid server configuration.

CHAPTER 8. 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.

8.1. CREATING SERVER TASKS

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

8.1.1. 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**

8.1.2. 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 8.1 Configuration Schema

8.2. CREATING SERVER SCRIPTS

Create custom scripts and add them to Data Grid servers.

8.2.1. 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}"**.

8.2.1.1. 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 8.1. 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.

8.2.1.2. 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.

8.2.1.3. 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.

8.2.2. Adding Scripts to Data Grid Servers

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

Prerequisites

Data Grid Server stores scripts in the **___script_cache** cache. If you enable cache authorization, users require 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. Create a CLI connection to Data Grid.
3. Use the **task** command to upload scripts, as in the following example:

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

4. Verify that your scripts are available.

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

8.2.3. 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

8.3. RUNNING SERVER-SIDE TASKS AND SCRIPTS

Execute tasks and custom scripts on Data Grid servers.

8.3.1. Running Tasks and Scripts

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

Procedure

1. Create a CLI connection to Data Grid.
2. Use the **task** command to run tasks and scripts, 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"]
```

8.3.2. 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

8.3.3. 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 9. MONITORING DATA GRID SERVERS

9.1. 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.

9.1.1. 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.

9.1.2. 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**.

9.1.2.1. 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.

9.1.2.2. 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.

9.1.2.3. 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.

9.1.2.4. 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 .

9.1.2.5. 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

9.1.3. 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.

9.1.3.1. 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="TRACE">
  <AppenderRef ref="HR-ACCESS-FILE"/>
</Logger>
```

9.1.3.2. Access Log Properties

The default format for access logs is as follows:

```
%X{address} %X{user} [%d{dd/MMM/yyyy:HH:mm:ss Z}] &amp;quot;%X{method} %m
%X{protocol}&amp;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}**.

9.2. 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.

9.2.1. Enabling Data Grid Statistics

Data Grid lets you enable statistics for Cache Managers and caches. However, enabling statistics for a Cache Manager does not enable statistics for the caches that it controls. You must explicitly enable statistics for your caches.

NOTE

Data Grid server enables statistics for Cache Managers by default.

Procedure

- Enable statistics declaratively or programmatically.

Declaratively

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

1 Enables statistics for the Cache Manager.

2 Enables statistics for the named cache.

Programmatically

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

 ...

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

1 Enables statistics for the Cache Manager.

2 Enables statistics for the named cache.

9.2.2. Enabling Data Grid Metrics

Configure Data Grid to export gauges and histograms.

Procedure

- Configure metrics declaratively or programmatically.

Declaratively

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

1 Computes and collects statistics about the Cache Manager.

2 Exports collected statistics as gauge and histogram metrics.

Programmatically

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

1 Computes and collects statistics about the Cache Manager.

2 Exports collected statistics as gauge and histogram metrics.

9.2.3. 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

9.2.4. Configuring Data Grid to Register JMX MBeans

Data Grid can register JMX MBeans that you can use to collect statistics and perform administrative operations. However, you must enable statistics separately to JMX otherwise Data Grid provides **0** values for all statistic attributes.

Procedure

- Enable JMX declaratively or programmatically.

Declaratively

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

1 Registers Data Grid JMX MBeans.

Programmatically

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

1 Registers Data Grid JMX MBeans.

9.2.4.1. 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

9.3. 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

9.3.1. 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.

9.3.2. 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 10. 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.

10.1. 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.

10.1.1. 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:11.0"
                cache="myDistCache" 2
                protocol-version="2.5" 3
                hotrod-wrapping="true" 4
                raw-values="true" 5
                segmented="false"> 6
    <remote-server host="127.0.0.1" port="11222"/> 7
  </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. Disables segmentation for the remote cache store. You should enable segmentation for remote cache stores only if the number of segments in the target cluster matches the number of segements for the cache in the source cluster.
7. Points to the location of the source cluster.

Reference

- Remote cache store configuration schema
- RemoteStore
- RemoteStoreConfigurationBuilder

10.2. 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 **POST** 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:

   ```
   POST /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 **POST** request:

```
POST /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 11. TROUBLESHOOTING DATA GRID SERVERS

Gather diagnostic information about Data Grid server deployments and perform troubleshooting steps to resolve issues.

11.1. GETTING DIAGNOSTIC REPORTS FOR DATA GRID SERVERS

Data Grid servers provide aggregated reports in **tar.gz** archives that contain diagnostic information about both the Data Grid server and the host. The report provides details about CPU, memory, open files, network sockets and routing, threads, in addition to configuration and log files.

Procedure

1. Create a CLI connection to Data Grid.
2. Use the **server report** command to download a **tar.gz** archive:

```
[//containers/default]> server report
Downloaded report 'infinispan-<hostname>-<timestamp>-report.tar.gz'
```

3. Move the **tar.gz** file to a suitable location on your filesystem.
4. Extract the **tar.gz** file with any archiving tool.

11.2. CHANGING DATA GRID SERVER LOGGING CONFIGURATION AT RUNTIME

Modify the logging configuration for Data Grid servers at runtime to temporarily adjust logging to troubleshoot issues and perform root cause analysis.

Modifying the logging configuration through the CLI is a runtime-only operation, which means that changes:

- Are not saved to the **log4j2.xml** file. Restarting server nodes or the entire cluster resets the logging configuration to the default properties in the **log4j2.xml** file.
- Apply only to the nodes in the cluster when you invoke the CLI. Nodes that join the cluster after you change the logging configuration use the default properties.

Procedure

1. Create a CLI connection to Data Grid.
2. Use the **logging** to make the required adjustments.
   - List all appenders defined on the server:

```
[//containers/default]> logging list-appenders
```

The preceding command returns:

```
{
  "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"
  }
}
```

- List all logger configurations defined on the server:

```
[//containers/default]> logging list-loggers
```

The preceding command returns:

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

- Add and modify logger configurations with the **set** subcommand

For example, the following command sets the logging level for the **org.infinispan** package to **DEBUG**:

```
[//containers/default]> logging set --level=DEBUG org.infinispan
```

- Remove existing logger configurations with the **remove** subcommand.

For example, the following command removes the **org.infinispan** logger configuration, which means the root configuration is used instead:

```
[//containers/default]> logging remove org.infinispan
```

11.3. RESOURCE STATISTICS

You can inspect server-collected statistics for some of the resources within a Data Grid server using the **stats** command.

Use the **stats** command either from the context of a resource which collects statistics (containers, caches) or with a path to such a resource:

```
[//containers/default]> 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
}
```

```
[//containers/default]> 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
}
```

CHAPTER 12. REFERENCE

12.1. DATA GRID SERVER 8.1.1 README

Information about the Data Grid Server 11.0.9.Final-redhat-00001 distribution.

12.1.1. Requirements

Data Grid Server requires JDK 11 or later.

12.1.2. Starting servers

Use the **server** script to run Data Grid Server instances.

Unix / Linux

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

Windows

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

TIP

Include the **--help** or **-h** option to view command arguments.

12.1.3. Stopping servers

Use the **shutdown** command with the CLI to perform a graceful shutdown.

Alternatively, enter Ctrl-C from the terminal to interrupt the server process or kill it via the TERM signal.

12.1.4. Configuration

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

cache-container
: Defines cache containers for managing cache lifecycles.

endpoints
: Enables and configures endpoint connectors for client protocols.

security
: Configures endpoint security realms.

socket-bindings
: Maps endpoint connectors to interfaces and ports.

The default configuration file is **$RHDG_HOME/server/conf/infinispan.xml**.

Use different configuration files with the **-c** argument, as in the following example that starts a server without clustering capabilities:

Unix / Linux

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

Windows

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

12.1.5. Bind address

Data Grid Server binds to the loopback IP address **localhost** on your network by default.

Use the **-b** argument to set a different IP address, as in the following example that binds to all network interfaces:

Unix / Linux

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

Windows

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

12.1.6. Bind port

Data Grid Server listens on port **11222** by default.

Use the **-p** argument to set an alternative port:

Unix / Linux

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

Windows

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

12.1.7. Clustering address

Data Grid Server configuration defines cluster transport so multiple instances on the same network discover each other and automatically form clusters.

Use the **-k** argument to change the IP address for cluster traffic:

Unix / Linux

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

Windows

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

12.1.8. Cluster stacks

JGroups stacks configure the protocols for cluster transport. Data Grid Server uses the **tcp** stack by default.

Use alternative cluster stacks with the **-j** argument, as in the following example that uses UDP for cluster transport:

Unix / Linux

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

Windows

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

12.1.9. Authentication

Data Grid Server requires authentication.

Create a username and password with the CLI as follows:

Unix / Linux

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

Windows

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

12.1.10. Server home directory

Data Grid Server uses **infinispan.server.home.path** to locate the contents of the server distribution on the host filesystem.

The server home directory, referred to as **$RHDG_HOME**, contains the following folders:

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

Folder	Description
/bin	Contains scripts to start servers and CLI.

Folder	Description
/boot	Contains **JAR** files to boot servers.
/docs	Provides configuration examples, schemas, component licenses, and other resources.
/lib	Contains **JAR** files that servers require internally. Do not place custom **JAR** files in this folder.
/server	Provides a root folder for Data Grid Server instances.
/static	Contains static resources for Data Grid Console.

12.1.11. Server root directory

Data Grid Server uses **infinispan.server.root.path** to locate configuration files and data for Data Grid Server instances.

You can create multiple server root folders in the same directory or in different directories and then specify the locations with the **-s** or **--server-root** argument, as in the following example:

Unix / Linux

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

Windows

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

Each server root directory contains the following folders:

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

Folder	Description	System property override
/server/conf	Contains server configuration files.	**infinispan.server.config.path**
/server/data	Contains data files organized by container name.	**infinispan.server.data.path**

Folder	Description	System property override
/server/lib	Contains server extension files. This directory is scanned recursively and used as a classpath.	**infinispan.server.lib.path** Separate multiple paths with the following delimiters: **:** on Unix / Linux **;** on Windows
/server/log	Contains server log files.	**infinispan.server.log.path**

12.1.12. Logging

Configure Data Grid Server logging with the **log4j2.xml** file in the **server/conf** folder.

Use the **--logging-config=<path_to_logfile>** argument to use custom paths, as follows:

Unix / Linux

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

TIP

To ensure custom paths take effect, do not use the ~ shortcut.

Windows

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