Red Hat Data Grid 8.3

Data Grid Security Guide

Enable and configure Data Grid security
Enable and configure Data Grid security
Abstract

Protect your Data Grid deployments from network intruders. Restrict data access to authorized users.
# Table of Contents

RED HAT DATA GRID ................................................................. 3
DATA GRID DOCUMENTATION .................................................. 4
DATA GRID DOWNLOADS .......................................................... 5
MAKING OPEN SOURCE MORE INCLUSIVE .................................. 6
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION ..................... 7

## CHAPTER 1. CONFIGURING USER ROLES AND PERMISSIONS ............................. 8
### 1.1. SECURITY AUTHORIZATION .................................................. 8
#### 1.1.1. User roles and permissions ........................................... 8
#### 1.1.2. Permissions ............................................................. 9
##### 1.1.2.1. Cache Manager permissions .................................... 9
##### 1.1.2.2. Cache permissions ............................................... 10
#### 1.1.3. Role mappers .......................................................... 11
##### 1.1.3.1. Cluster role mappers ............................................. 11
##### 1.1.3.2. Identity role mappers ............................................ 11
##### 1.1.3.3. CommonName role mappers .................................... 12
##### 1.1.3.4. Custom role mappers ............................................ 12
### 1.2. ACCESS CONTROL LIST (ACL) CACHE ..................................... 12
### 1.3. CUSTOMIZING ROLES AND PERMISSIONS .................................. 13
Custom roles and permissions configuration ........................................ 13
### 1.4. CONFIGURING CACHES WITH SECURITY AUTHORIZATION ..................... 15
Authorization configuration ....................................................... 15
Custom roles and permissions .................................................... 16
### 1.5. DISABLING SECURITY AUTHORIZATION ...................................... 16
### 1.6. PROGRAMMATICALLY CONFIGURING AUTHORIZATION ....................... 17
### 1.7. CODE EXECUTION WITH SECURITY AUTHORIZATION ....................... 18

## CHAPTER 2. ENCRYPTING CLUSTER TRANSPORT ......................................... 19
### 2.1. SECURING CLUSTER TRANSPORT WITH TLS IDENTITIES ..................... 19
### 2.2. JGROUPS ENCRYPTION PROTOCOLS ........................................ 20
### 2.3. SECURING CLUSTER TRANSPORT WITH ASYMMETRIC ENCRYPTION ............ 21
### 2.4. SECURING CLUSTER TRANSPORT WITH SYMMETRIC ENCRYPTION ............ 22

## CHAPTER 3. DATA GRID PORTS AND PROTOCOLS ...................................... 24
### 3.1. DATA GRID SERVER PORTS AND PROTOCOLS .................................. 24
Single port .............................................................................. 24
#### 3.1.1. Configuring network firewalls for Data Grid traffic ....................... 24
### 3.2. TCP AND UDP PORTS FOR CLUSTER TRAFFIC .............................. 25
Cross-site replication ............................................................ 25
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 8.3 Documentation
- Data Grid 8.3 Component Details
- Supported Configurations for Data Grid 8.3
- 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.
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our technical content and encourage you to tell us what you think. If you’d like to add comments, provide insights, correct a typo, or even ask a question, you can do so directly in the documentation.

NOTE

You must have a Red Hat account and be logged in to the customer portal.

To submit documentation feedback from the customer portal, do the following:

1. Select the Multi-page HTML format.
2. Click the Feedback button at the top-right of the document.
3. Highlight the section of text where you want to provide feedback.
4. Click the Add Feedback dialog next to your highlighted text.
5. Enter your feedback in the text box on the right of the page and then click Submit.

We automatically create a tracking issue each time you submit feedback. Open the link that is displayed after you click Submit and start watching the issue or add more comments.

Thank you for the valuable feedback.
CHAPTER 1. CONFIGURING USER ROLES AND PERMISSIONS

Authorization is a security feature that requires users to have certain permissions before they can access caches or interact with Data Grid resources. You assign roles to users that provide different levels of permissions, from read-only access to full, super user privileges.

1.1. SECURITY AUTHORIZATION

Data Grid authorization secures your deployment by restricting user access.

User applications or clients must belong to a role that is assigned with sufficient permissions before they can perform operations on Cache Managers or caches.

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 `io` 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.

Identities

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.

Identities to roles

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

The following image illustrates how security principals correspond to roles:

1.1.1. User roles and permissions

Data Grid includes a default set of roles that grant users with permissions to access data and interact with Data Grid resources.

`ClusterRoleMapper` is the default mechanism that Data Grid uses to associate security principals to authorization roles.
IMPORTANT

ClusterRoleMapper matches principal names to role names. A user named admin gets admin permissions automatically, a user named deployer gets deployer permissions, and so on.

<table>
<thead>
<tr>
<th>Role</th>
<th>Permissions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>ALL</td>
<td>Superuser with all permissions including control of the Cache Manager lifecycle.</td>
</tr>
<tr>
<td>deployer</td>
<td>ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR, CREATE</td>
<td>Can create and delete Data Grid resources in addition to application permissions.</td>
</tr>
<tr>
<td>application</td>
<td>ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR</td>
<td>Has read and write access to Data Grid resources in addition to observer permissions. Can also listen to events and execute server tasks and scripts.</td>
</tr>
<tr>
<td>observer</td>
<td>ALL_READ, MONITOR</td>
<td>Has read access to Data Grid resources in addition to monitor permissions.</td>
</tr>
<tr>
<td>monitor</td>
<td>MONITOR</td>
<td>Can view statistics via JMX and the metrics endpoint.</td>
</tr>
</tbody>
</table>

Reference

- [org.infinispan.security.AuthorizationPermission Enumeration](#)
- Data Grid configuration schema reference

1.1.2. Permissions

Authorization roles have different permissions with varying levels of access to Data Grid. Permissions let you restrict user access to both Cache Managers and caches.

1.1.2.1. Cache Manager permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIGURATION</td>
<td>defineConfiguration</td>
<td>Defines new cache configurations.</td>
</tr>
<tr>
<td>LISTEN</td>
<td>addListener</td>
<td>Registers listeners against a Cache Manager.</td>
</tr>
<tr>
<td>Permission</td>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LIFECYCLE</td>
<td>stop</td>
<td>Stops the Cache Manager.</td>
</tr>
<tr>
<td>CREATE</td>
<td>createCache, removeCache</td>
<td>Create and remove container resources such as caches, counters, schemas, and scripts.</td>
</tr>
<tr>
<td>MONITOR</td>
<td>getStats</td>
<td>Allows access to JMX statistics and the <strong>metrics</strong> endpoint.</td>
</tr>
<tr>
<td>ALL</td>
<td>-</td>
<td>Includes all Cache Manager permissions.</td>
</tr>
</tbody>
</table>

### 1.1.2.2. Cache permissions

<table>
<thead>
<tr>
<th>Permission</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>get, contains</td>
<td>Retrieves entries from a cache.</td>
</tr>
<tr>
<td>WRITE</td>
<td>put, putIfAbsent, replace, remove, evict</td>
<td>Writes, replaces, removes, evicts data in a cache.</td>
</tr>
<tr>
<td>EXEC</td>
<td>distexec, streams</td>
<td>Allows code execution against a cache.</td>
</tr>
<tr>
<td>LISTEN</td>
<td>addListener</td>
<td>Registers listeners against a cache.</td>
</tr>
<tr>
<td>BULK_READ</td>
<td>keySet, values, entrySet, query</td>
<td>Executes bulk retrieve operations.</td>
</tr>
<tr>
<td>BULK_WRITE</td>
<td>clear, putAll</td>
<td>Executes bulk write operations.</td>
</tr>
<tr>
<td>LIFECYCLE</td>
<td>start, stop</td>
<td>Starts and stops a cache.</td>
</tr>
</tbody>
</table>
### Additional resources

- Data Grid Security API

### 1.1.3. Role mappers

Data Grid includes a PrincipalRoleMapper API that maps security Principals in a Subject to authorization roles that you can assign to users.

#### 1.1.3.1. Cluster role mappers

ClusterRoleMapper uses a persistent replicated cache to dynamically store principal-to-role mappings for the default roles and permissions.

By default uses the Principal name as the role name and implements org.infinispan.security.MutableRoleMapper which exposes methods to change role mappings at runtime.

- Java class: org.infinispan.security.mappers.ClusterRoleMapper
- Declarative configuration: `<cluster-role-mapper />`

#### 1.1.3.2. Identity role mappers
**IdentityRoleMapper** uses the Principal name as the role name.

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

### 1.1.3.3. CommonName role mappers

**CommonNameRoleMapper** uses the Common Name (CN) as the role name if the Principal name is a Distinguished Name (DN).

For example this DN, `cn=managers,ou=people,dc=example,dc=com`, maps to the **managers** role.

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

### 1.1.3.4. Custom role mappers

Custom role mappers are implementations of `org.infinispan.security.PrincipalRoleMapper`.

- Declarative configuration: `<custom-role-mapper class="my.custom.RoleMapper" />

**Additional resources**

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

## 1.2. ACCESS CONTROL LIST (ACL) CACHE

Data Grid caches roles that you grant to users internally for optimal performance. Whenever you grant or deny roles to users, Data Grid flushes the ACL cache to ensure user permissions are applied correctly.

If necessary, you can disable the ACL cache or configure it with the **cache-size** and **cache-timeout** attributes.

**XML**

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

**JSON**

```json
{
  "infinispan": {
    "cache-container": {
      "name": "acl-cache-configuration",
```

Red Hat Data Grid 8.3 Data Grid Security Guide
## 1.3. CUSTOMIZING ROLES AND PERMISSIONS

You can customize authorization settings in your Data Grid configuration to use role mappers with different combinations of roles and permissions.

**Procedure**

1. Declare a role mapper and a set of custom roles and permissions in the Cache Manager configuration.

2. Configure authorization for caches to restrict access based on user roles.

### Custom roles and permissions configuration

**XML**

```xml
<infinispan>
  <cache-container name="custom-authorization">
    <security>
      <authorization>
        <!-- Declare a role mapper that associates a security principal to each role. -->
        <identity-role-mapper />
        <!-- Specify user roles and corresponding permissions. -->
        <role name="admin" permissions="ALL"/>
        <role name="reader" permissions="READ"/>
        <role name="writer" permissions="WRITE"/>
        <role name="supervisor" permissions="READ WRITE EXEC"/>
      </authorization>
    </security>
  </cache-container>
</infinispan>
```

**YAML**

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

### Additional resources

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

```yaml
infinispan:
  cacheContainer:
    name: "custom-authorization"
  security:
    authorization:
      identityRoleMapper: "null"
    roles:
      reader:
        role:
          permissions: "READ"
      admin:
      writer:
        role:
          permissions: "WRITE"
      supervisor:
        role:
          permissions: "READ WRITE EXEC"
```
1.4. CONFIGURING CACHES WITH SECURITY AUTHORIZATION

Use authorization in your cache configuration to restrict user access. Before they can read or write cache entries, or create and delete caches, users must have a role with a sufficient level of permission.

Prerequisites

- Ensure the `authorization` element is included in the `security` section of the `cache-container` configuration.
  
  Data Grid enables security authorization in the Cache Manager by default and provides a global set of roles and permissions for caches.

- If necessary, declare custom roles and permissions in the Cache Manager configuration.

Procedure

1. Open your cache configuration for editing.

2. Add the `authorization` element to caches to restrict user access based on their roles and permissions.

3. Save the changes to your configuration.

Authorization configuration

The following configuration shows how to use implicit authorization configuration with default roles and permissions:

**XML**

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

**JSON**

```json
{
  "distributed-cache": {
    "security": {
      "authorization": {
        "enabled": true
      }
    }
  }
}
1.5. DISABLING SECURITY AUTHORIZATION

In local development environments you can disable authorization so that users do not need roles and permissions. Disabling security authorization means that any user can access data and interact with Data Grid resources.

Procedure

1. Open your Data Grid configuration for editing.
2. Remove any **authorization** elements from the **security** configuration for the Cache Manager.

3. Remove any **authorization** configuration from your caches.

4. Save the changes to your configuration.

### 1.6. PROGRAMMATICALLY CONFIGURING AUTHORIZATION

When using embedded caches, you can configure authorization with the `GlobalSecurityConfigurationBuilder` and `ConfigurationBuilder` classes.

#### Procedure

1. Construct a **GlobalConfigurationBuilder** that enables authorization, specifies a role mapper, and defines a set of roles and permissions.

   ```java
   GlobalConfigurationBuilder global = new GlobalConfigurationBuilder();
   global
       .security()
       .authorization().enable()  // Enables Data Grid authorization for the Cache Manager.
       .principalRoleMapper(new IdentityRoleMapper())  // Specifies an implementation of PrincipalRoleMapper that maps Principals to roles.
       .role("admin")  // Defines roles and their associated permissions.
           .permission(AuthorizationPermission.ALL)
           .role("reader")
           .permission(AuthorizationPermission.READ)
           .role("writer")
           .permission(AuthorizationPermission.WRITE)
           .role("supervisor")
           .permission(AuthorizationPermission.READ)
           .permission(AuthorizationPermission.WRITE)
           .permission(AuthorizationPermission.EXEC);
   ```

2. Enable authorization in the **ConfigurationBuilder** for caches to restrict access based on user roles.

   ```java
   ConfigurationBuilder config = new ConfigurationBuilder();
   config
       .security()
       .authorization()  // Implicitly adds all roles from the global configuration.
           .enable();
   ```

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

```java
ConfigurationBuilder config = new ConfigurationBuilder();
```
The following configuration enables security for a cache:

```java
config
  .security()
  .authorization()
  .enable()
  .role("admin")
  .role("supervisor")
  .role("reader");
```

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.

Additional resources

- `org.infinispan.configuration.global.GlobalSecurityConfigurationBuilder`
- `org.infinispan.configuration.cache.ConfigurationBuilder`

### 1.7. CODE EXECUTION WITH SECURITY AUTHORIZATION

When you configure security authorization for embedded caches and then construct a `DefaultCacheManager`, it returns a `SecureCache` that checks the security context before invoking any operations. A `SecureCache` also ensures that applications cannot retrieve lower-level insecure objects such as `DataContainer`. For this reason, you must execute code with an identity that has the required authorization.

In Java, executing code with a specific identity usually means wrapping the code to be executed within a `PrivilegedAction` as follows:

```java
import org.infinispan.security.Security;

Security.doAs(subject, new PrivilegedExceptionAction<Void>() {
    public Void run() throws Exception {
        cache.put("key", "value");
    }
});
```

With Java 8, you can simplify the preceding call as follows:

```java
Security.doAs(mySubject, PrivilegedAction<String>() -> cache.put("key", "value"));
```

The preceding call uses the `Security.doAs()` method instead of `Subject.doAs()`. You can use either method with Data Grid, however `Security.doAs()` provides better performance.

If you need the current Subject, use the following call to retrieve it from the Data Grid context or from the AccessControlContext:

```java
Security.getSubject();
```
CHAPTER 2. 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.

2.1. SECURING CLUSTER TRANSPORT WITH TLS IDENTITIES

Add SSL/TLS identities to a Data Grid Server security realm and use them to secure cluster transport. Nodes in the Data Grid Server cluster then exchange SSL/TLS certificates to encrypt JGroups messages, including RELAY messages if you configure cross-site replication.

Prerequisites

- Install a Data Grid Server cluster.

Procedure

1. Create a TLS keystore that contains a single certificate to identify Data Grid Server. You can also use a PEM file if it contains a private key in PKCS#1 or PKCS#8 format, a certificate, and has an empty password: `password=""`.

   NOTE
   If the certificate in the keystore is not signed by a public certificate authority (CA) then you must also create a trust store that contains either the signing certificate or the public key.

2. Add the keystore to the `$RHDG_HOME/server/conf` directory.

3. Add the keystore to a new security realm in your Data Grid Server configuration.

   IMPORTANT
   You should create dedicated keystores and security realms so that Data Grid Server endpoints do not use the same security realm as cluster transport.

```xml
<server xmlns="urn:infinispan:server:13.0">
  <security>
    <security-realms>
      <security-realm name="cluster-transport">
        <server-identities>
          <ssl> <!-- Adds a keystore that contains a certificate that provides SSL/TLS identity to encrypt cluster transport. -->
            <keystore path="server.pfx"
              relative-to="infinispan.server.config.path"
              password="secret"
              alias="server"/>
          </ssl>
        </server-identities>
      </security-realm>
    </security-realms>
  </security>
</server>
```
4. Configure cluster transport to use the security realm by specifying the name of the security realm with the `<server:security-realm>` attribute.

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

**Verification**

When you start Data Grid Server, the following log message indicates that the cluster is using the security realm for cluster transport:

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

### 2.2. JGROUPS ENCRYPTION PROTOCOLS

To secure cluster traffic, you can 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.

2.3. SECURING 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:

```xml
<infinispan>
  <jgroups>
    <!-- Creates a secure JGroups stack named "encrypt-tcp" that extends the default TCP stack. -->
    <stack name="encrypt-tcp" extends="tcp">
      <!-- Adds a keystore that nodes use to perform certificate authentication. -->
      <SSL_KEY_EXCHANGE keystore_name="mykeystore.jks" keystore_password="changeit" stack.combine="INSERT_AFTER" stack.position="VERIFY_SUSPECT" />
      <!-- Configures ASYM_ENCRYPT -->
    </stack>
  </jgroups>
</infinispan>
```
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
```

Additional resources

- JGroups 4 Manual
- JGroups 4.2 Schema

### 2.4. SECURING 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.

```xml
<infinispan>
  <jgroups>
    <!-- Creates a secure JGroups stack named "encrypt-tcp" that extends the default TCP stack. -->
    <stack name="encrypt-tcp" extends="tcp"/>
    <!-- Adds a keystore from which nodes obtain secret keys. -->
    <!-- Uses the stack.combine and stack.position attributes to insert SYM_ENCRYPT into the default TCP stack after VERIFY_SUSPECT. -->
    <SYM_ENCRYPT keystore_name="myKeystore.p12"
      keystore_type="PKCS12"
      store_password="changeit"
      key_password="changeit"
      alias="myKey"
      stack_combine="INSERT_AFTER"
      stack_position="VERIFY_SUSPECT"/>
  </stack>
</jgroups>

<cache-container name="default" statistics="true">
  <!-- Configures the cluster to use the JGroups stack. -->
  <transport cluster="${infinispan.cluster.name}" stack="encrypt-tcp" node-name="${infinispan.node.name:}"/>
</cache-container>
</infinispan>
```

**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
```

**Additional resources**

- JGroups 4 Manual
- JGroups 4.2 Schema
CHAPTER 3. DATA GRID PORTS AND PROTOCOLS

As Data Grid distributes data across your network and can establish connections for external client requests, you should be aware of the ports and protocols that Data Grid uses to handle network traffic.

If run Data Grid as a remote server then you might need to allow remote clients through your firewall. Likewise, you should adjust ports that Data Grid nodes use for cluster communication to prevent conflicts or network issues.

3.1. DATA GRID SERVER PORTS AND PROTOCOLS

Data Grid Server provides network endpoints that allow client access with different protocols.

<table>
<thead>
<tr>
<th>Port</th>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11222</td>
<td>TCP</td>
<td>Hot Rod and REST</td>
</tr>
<tr>
<td>11221</td>
<td>TCP</td>
<td>Memcached (disabled by default)</td>
</tr>
</tbody>
</table>

Single port
Data Grid Server exposes multiple protocols through a single TCP port, 11222. Handling multiple protocols with a single port simplifies configuration and reduces management complexity when deploying Data Grid clusters. Using a single port also enhances security by minimizing the attack surface on the network.

Data Grid Server handles HTTP/1.1, HTTP/2, and Hot Rod protocol requests from clients via the single port in different ways.

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 Server. Client applications can then send the Upgrade: protocol header field, where protocol is a server endpoint.

Application-Layer Protocol Negotiation (ALPN)/Transport Layer Security (TLS)
Client requests include Server Name Indication (SNI) mappings for Data Grid Server endpoints to negotiate protocols over a TLS connection.

NOTE
Applications must use a TLS library that supports the ALPN extension. Data Grid uses WildFly OpenSSL bindings for Java.

Automatic Hot Rod detection
Client requests that include Hot Rod headers automatically route to Hot Rod endpoints.

3.1.1. Configuring network firewalls for Data Grid traffic
Adjust firewall rules to allow traffic between Data Grid Server and client applications.

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`

### 3.2. TCP AND UDP PORTS FOR CLUSTER TRAFFIC

Data Grid uses the following ports for cluster transport messages:

<table>
<thead>
<tr>
<th>Default Port</th>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7800</td>
<td>TCP/UDP</td>
<td>JGroups cluster bind port</td>
</tr>
<tr>
<td>46655</td>
<td>UDP</td>
<td>JGroups multicast</td>
</tr>
</tbody>
</table>

**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.