Red Hat Data Grid 8.1

Data Grid Security Guide

Enable and configure Data Grid security

Last Updated: 2021-06-09

Red Hat Data Grid 8.1 Data Grid Security Guide

Enable and configure Data Grid security

Legal Notice

Abstract

Protect your Data Grid deployments from network intruders. Restrict data access to authorized users.

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. 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.

1.1. 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.

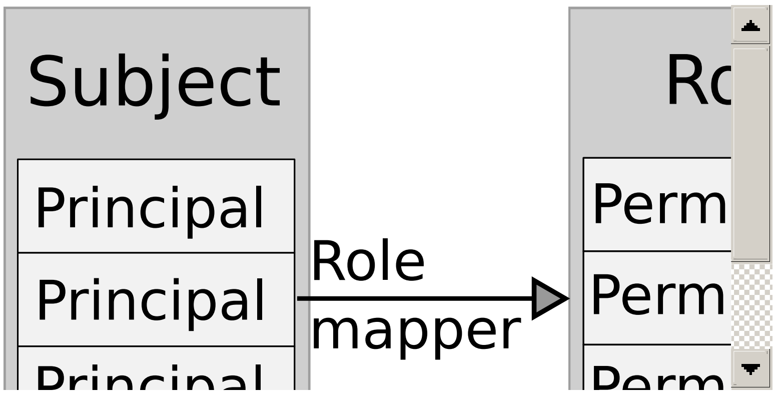
In this scenario, if a user application or client with the **reader** role attempts to write an entry, Data Grid denies the request and throws a security exception. If a user application or client with the **writer** role sends a write request, Data Grid validates authorization and issues a token for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



1.1.1. 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 1.1. Cache Manager Permissions

Permission	Function	Description
CONFIGURATION	**defineConfiguration**	Defines new cache configurations.

Permission	Function	Description
LISTEN	**addListener**	Registers listeners against a Cache Manager.
LIFECYCLE	**stop**	Stops the Cache Manager.
ALL	-	Includes all Cache Manager permissions.

Table 1.2. 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.
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.

Permission	Function	Description
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

1.1.2. 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

1.2. PROGRAMMATICALLY CONFIGURING AUTHORIZATION

When using Data Grid as an embedded library, 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.

```
GlobalConfigurationBuilder global = new GlobalConfigurationBuilder();
global
  .security()
    .authorization().enable() 1
      .principalRoleMapper(new IdentityRoleMapper()) 2
      .role("admin") 3
        .permission(AuthorizationPermission.ALL)
      .role("reader")
        .permission(AuthorizationPermission.READ)
      .role("writer")
        .permission(AuthorizationPermission.WRITE)
      .role("supervisor")
        .permission(AuthorizationPermission.READ)
        .permission(AuthorizationPermission.WRITE)
        .permission(AuthorizationPermission.EXEC);
```

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.

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

```
ConfigurationBuilder config = new ConfigurationBuilder();
config
  .security()
    .authorization()
      .enable(); 1
```

1 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:

```
ConfigurationBuilder config = new ConfigurationBuilder();
config
  .security()
    .authorization()
      .enable()
```

```
        .role("admin") (1)
        .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.

Reference

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

1.3. 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

1.4. CODE EXECUTION WITH SECURE CACHES

When you configure Data Grid authorization and then construct a **DefaultCacheManager**, it returns a **SecureCache** that checks the security context before invoking any operations on the underlying caches. 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:

```
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:

```
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:

```
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. 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.

2.2. 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.
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

2.3. 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 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 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.

3.1.1. 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

3.2. 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

Default Port	Protocol	Description
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.