Red Hat Data Grid 8.1

Data Grid Library Mode

Run Data Grid as an embedded library

Last Updated: 2021-06-09

Red Hat Data Grid 8.1 Data Grid Library Mode

Run Data Grid as an embedded library

Legal Notice

Abstract

Set up dependencies and run Data Grid as an embedded library in custom 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. CONFIGURING THE DATA GRID MAVEN REPOSITORY

Data Grid Java distributions are available from Maven.

You can download the Data Grid Maven repository from the customer portal or pull Data Grid dependencies from the public Red Hat Enterprise Maven repository.

1.1. DOWNLOADING THE DATA GRID MAVEN REPOSITORY

Download and install the Data Grid Maven repository to a local file system, Apache HTTP server, or Maven repository manager if you do not want to use the public Red Hat Enterprise Maven repository.

Procedure

1. Log in to the Red Hat customer portal.
2. Navigate to the Software Downloads for Data Grid.
3. Download the Red Hat Data Grid 8.1 Maven Repository.
4. Extract the archived Maven repository to your local file system.
5. Open the **README.md** file and follow the appropriate installation instructions.

1.2. ADDING RED HAT MAVEN REPOSITORIES

Include the Red Hat GA repository in your Maven build environment to get Data Grid artifacts and dependencies.

Procedure

- Add the Red Hat GA repository to your Maven settings file, typically ~/**.m2**/**settings.xml**, or directly in the **pom.xml** file of your project.

```
<repositories>
  <repository>
    <id>redhat-ga-repository</id>
    <name>Red Hat GA Repository</name>
    <url>https://maven.repository.redhat.com/ga/</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <id>redhat-ga-repository</id>
    <name>Red Hat GA Repository</name>
    <url>https://maven.repository.redhat.com/ga/</url>
  </pluginRepository>
</pluginRepositories>
```

Reference

- Red Hat Enterprise Maven Repository

1.3. CONFIGURING YOUR DATA GRID POM

Maven uses configuration files called Project Object Model (POM) files to define projects and manage builds. POM files are in XML format and describe the module and component dependencies, build order, and targets for the resulting project packaging and output.

Procedure

1. Open your project **pom.xml** for editing.
2. Define the **version.infinispan** property with the correct Data Grid version.
3. Include the **infinispan-bom** in a **dependencyManagement** section.
   The Bill Of Materials (BOM) controls dependency versions, which avoids version conflicts and means you do not need to set the version for each Data Grid artifact you add as a dependency to your project.
4. Save and close **pom.xml**.

The following example shows the Data Grid version and BOM:

```
<properties>
  <version.infinispan>11.0.9.Final-redhat-00001</version.infinispan>
</properties>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-bom</artifactId>
      <version>${version.infinispan}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Next Steps

Add Data Grid artifacts as dependencies to your **pom.xml** as required.

CHAPTER 2. INSTALLING DATA GRID IN LIBRARY MODE

Add Data Grid as an embedded library in your project.

Procedure

- Add the **infinispan-core** artifact as a dependency in your **pom.xml** as follows:

```
<dependencies>
  <dependency>
    <groupId>org.infinispan</groupId>
    <artifactId>infinispan-core</artifactId>
  </dependency>
</dependencies>
```

CHAPTER 3. RUNNING DATA GRID AS AN EMBEDDED LIBRARY

Learn how to run Data Grid as an embedded data store in your project.

Procedure

- Initialize the default Cache Manager and add a cache definition as follows:

```
GlobalConfigurationBuilder global = GlobalConfigurationBuilder.defaultClusteredBuilder();
DefaultCacheManager cacheManager = new DefaultCacheManager(global.build());
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.clustering().cacheMode(CacheMode.DIST_SYNC);
cacheManager.administration().withFlags(CacheContainerAdmin.AdminFlag.VOLATILE).getOrCreateC
ache("myCache", builder.build());
```

The preceding code initializes a default, clustered Cache Manager. Cache Managers contain your cache definitions and control cache lifecycles.

Data Grid does not provide default cache definitions so after initializing the default Cache Manager, you need to add at least one cache instance. This example uses the **ConfigurationBuilder** class to create a cache definition that uses the distributed, synchronous cache mode. You then call the **getOrCreateCache()** method that either creates a cache named "myCache" on all nodes in the cluster or returns it if it already exists.

Next steps

Now that you have a running Cache Manager with a cache created, you can add some more cache definitions, put some data into the cache, or configure Data Grid as needed.

Reference

- Configuring Data Grid Programmatically
- org.infinispan.configuration.global.GlobalConfigurationBuilder
- org.infinispan.manager.EmbeddedCacheManager
- org.infinispan.Cache

CHAPTER 4. 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.

4.1. 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.
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 **addProperty()** method to set the JGroups stack file:

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder().transport()
        .defaultTransport()
        .clusterName("qa-cluster")
        .addProperty("configurationFile", "default-jgroups-udp.xml") 1
        .build();
```

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

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)

4.1.1. 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

4.1.2. 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.

4.2. 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)

4.2.1. 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.

4.3. 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:

```
$ java -cp ... -Djgroups.bind.port=1234 -Djgroups.bind.address=192.0.2.0
```

NOTE

When you embed Data Grid clusters in clustered Red Hat JBoss EAP applications, JGroups system properties can clash or override each other.

For example, you do not set a unique bind address for either your Data Grid cluster or your Red Hat JBoss EAP application. In this case both Data Grid and your Red Hat JBoss EAP application use the JGroups default property and attempt to form clusters using the same bind address.

4.3.1. 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
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

4.4. 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.

4.5. USING EXTERNAL JGROUPS STACKS

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

Procedure

1. Put custom JGroups stack files on the application classpath.
   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>
```

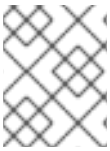
1. Creates a stack named "prod-tcp" that uses the "prod-jgroups-tcp.xml" definition.
2. Configures Data Grid to use "prod-tcp" for cluster transport.

4.6. 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)

4.6.1. 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

4.6.2. 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

4.6.3. 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

4.6.4. 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

4.6.5. 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

4.6.6. 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)

4.7. USING CUSTOM JCHANNELS

Construct custom JGroups JChannels as in the following example:

```
GlobalConfigurationBuilder global = new GlobalConfigurationBuilder();
JChannel jchannel = new JChannel();
// Configure the jchannel as needed.
JGroupsTransport transport = new JGroupsTransport(jchannel);
global.transport().transport(transport);
new DefaultCacheManager(global.build());
```

NOTE

Data Grid cannot use custom JChannels that are already connected.

Reference

JGroups JChannel

4.8. 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.

4.8.1. Data Grid Cluster Security

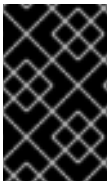
To secure cluster traffic, you configure Data Grid nodes to encrypt JGroups message payloads with secret keys.

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.

4.8.2. 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

4.8.3. 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