Embedding Infinispan 13.0

Table of Contents

Chapter 1. Configuring the Infinispan Maven repository

Infinispan Java distributions are available from Maven.

Infinispan artifacts are available from Maven central. See the org.infinispan group for available Infinispan artifacts.

1.1. Configuring your Infinispan 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 Infinispan 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 Infinispan artifact you add as a dependency to your project.

4. Save and close `pom.xml`.

The following example shows the Infinispan version and BOM:

```
<properties>
  <version.infinispan>13.0.0.CR2</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 Infinispan artifacts as dependencies to your `pom.xml` as required.

Chapter 2. Creating embedded caches

Infinispan provides an `EmbeddedCacheManager` API that lets you control both the Cache Manager and embedded cache lifecycles programmatically.

2.1. Adding Infinispan to your project

Add Infinispan to your project to create embedded caches in your applications.

Prerequisites

- Configure your project to get Infinispan artifacts from the Maven repository.

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>
```

2.2. Configuring embedded caches

Infinispan provides a `GlobalConfigurationBuilder` API that controls the Cache Manager and a `ConfigurationBuilder` API that configures embedded caches.

Prerequisites

- Add the `infinispan-core` artifact as a dependency in your `pom.xml`.

Procedure

1. Initialize the default Cache Manager so you can add embedded caches.
2. Add at least one embedded cache with the `ConfigurationBuilder` API.
3. Invoke the `getOrCreateCache()` method that either creates embedded caches on all nodes in the cluster or returns caches that already exist.

```
// Set up a clustered Cache Manager.
GlobalConfigurationBuilder global = GlobalConfigurationBuilder.
defaultClusteredBuilder();
// Initialize the default Cache Manager.
DefaultCacheManager cacheManager = new DefaultCacheManager(global.build());
// Create an embedded cache configuration.
ConfigurationBuilder builder = new ConfigurationBuilder();
                    builder.clustering().cacheMode(CacheMode.DIST_SYNC);
// Obtain a volatile cache.
Cache<String, String> cache = cacheManager.administration().withFlags
(CacheContainerAdmin.AdminFlag.VOLATILE).getOrCreateCache("myCache", builder.build());
```

Additional resources

- EmbeddedCacheManager
- EmbeddedCacheManager Configuration
- org.infinispan.configuration.global.GlobalConfiguration
- org.infinispan.configuration.cache.ConfigurationBuilder

Chapter 3. Enabling and configuring Infinispan statistics and JMX monitoring

Infinispan can provide Cache Manager and cache statistics as well as export JMX MBeans.

3.1. Enabling statistics in embedded caches

Configure Infinispan to export statistics for Cache Managers and caches.

Procedure

Enable Infinispan statistics in your cache configuration in one of the following ways:

- Declarative: Add the `statistics="true"` attribute.
- Programmatic: Call the `.statistics()` method.

Declarative

```
<!-- Enable Cache Manager statistics. -->
<cache-container statistics="true">
  <!-- Enables cache statistics. -->
  <local-cache name="mycache" statistics="true"/>
</cache-container>
```

Programmatic

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()
  //Enables statistics for the Cache Manager.
  .cacheContainer().statistics(true)
  .build();

Configuration config = new ConfigurationBuilder()
  //Enables statistics for the named cache.
  .statistics().enable()
  .build();
```

3.2. Configuring Infinispan metrics

Infinispan generates metrics that are compatible with the MicroProfile Metrics API.

- Gauges provide values such as the average number of nanoseconds for write operations or JVM uptime.
- Histograms provide details about operation execution times such as read, write, and remove times.

By default, Infinispan generates gauges when you enable statistics but you can also configure it to generate histograms.

Procedure

- Enable gauges and histograms as appropriate in the Cache Manager configuration.

```
<cache-container statistics="true">
  <!-- Enables gauge and histogram metrics for Infinispan statistics. -->
  <metrics gauges="true" histograms="true" />
</cache-container>
```

For embedded caches, you can configure metrics programmatically with the `GlobalMetricsConfigurationBuilder` API as follows:

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder()
  //Computes and collects statistics for the Cache Manager.
  .statistics().enable()
  //Exports collected statistics as gauge and histogram metrics.
  .metrics().gauges(true).histograms(true)
  .build();
```

Verification

For embedded caches, you must add the necessary MicroProfile API and provider JARs to your classpath to export Infinispan metrics.

Additional resources

- Eclipse MicroProfile Metrics

3.3. Registering JMX MBeans

Infinispan can register JMX MBeans that you can use to collect statistics and perform administrative operations. You must also enable statistics otherwise Infinispan provides `0` values for all statistic attributes in JMX MBeans.

Procedure

1. Open `infinispan.xml` for editing.
2. Add the `<jmx enabled="true" />` element to the cache container.

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

For embedded caches, you can invoke the `.jmx().enable()` method to programmatically enable JMX.

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

3.3.1. Enabling JMX remote ports

Provide unique remote JMX ports to expose Infinispan MBeans through connections in JMXServiceURL format.

Procedure

- Pass the following system properties to Infinispan at startup:

```
-Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=9999
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

3.3.2. Infinispan MBeans

Infinispan 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 Infinispan cache and cluster health statistics.

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

Additional resources

- Infinispan JMX Components

3.3.3. Registering MBeans in custom MBean servers

Infinispan includes an `MBeanServerLookup` interface that you can use to register MBeans in custom MBeanServer instances.

Procedure

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

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

For embedded caches, you can use the `mBeanServerLookup()` method to programmatically specify the fully qualified name of your class.

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

Chapter 4. Setting Up Infinispan Clusters

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

4.1. Default JGroups Stacks

Infinispan provides default JGroups stack files, `default-jgroups-*.xml`, in the `default-configs` directory inside the `infinispan-core-13.0.0.CR2.jar` file.

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-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-ec2.xml`	`ec2`	Uses TCP for transport and `NATIVE_S3_PING` for discovery. Suitable for Amazon EC2 nodes where UDP multicast is not available. Requires additional dependencies.
`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. Requires additional dependencies.
`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. Requires additional dependencies.

Additional resources

- JGroups Protocols

4.2. Cluster Discovery Protocols

Infinispan 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 Infinispan 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 Infinispan clusters.
  For instance the DNS_PING protocol performs discovery through DNS server records.

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

Additional resources

- JGroups Discovery Protocols

4.2.1. 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 Infinispan 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

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

Additional resources

- JGroups PING

4.2.2. 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 Infinispan cluster as part of the JGroups stack, rather than letting nodes discover each other dynamically.

TCPPING configuration example

```
<TCP bind_port="7800" />
<TCPPING timeout="3000"
         initial_hosts="${jgroups.tcpping.initial_hosts:hostname1[port1],hostname2[por
t2]}"
          port_range="0"
          num_initial_members="3"/>
```

Additional resources

- JGroups TCPPING

4.2.3. MPING

MPING uses IP multicast to discover the initial membership of Infinispan 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

```
<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}"/>
```

Additional resources

- JGroups MPING

4.2.4. TCPGOSSIP

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

You inject the address (`IP:PORT`) of the Gossip router into Infinispan 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

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

Additional resources

- JGroups Gossip Router

4.2.5. JDBC_PING

JDBC_PING uses shared databases to store information about Infinispan 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 Infinispan cluster on the network. When nodes leave Infinispan clusters, they delete their IP addresses from the shared database.

JDBC_PING configuration example

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

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

Additional resources

- JDBC_PING
- JDBC_PING Wiki

4.2.6. DNS_PING

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

DNS_PING configuration example

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

Additional resources

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

4.2.7. Cloud Discovery Protocols

Infinispan includes default JGroups stacks that use discovery protocol implementations that are specific to cloud providers.

Discovery protocol	Default stack file	Artifact	Version
NATIVE_S3_PING	default-jgroups-ec2.xml	org.jgroups.aws.s3:native-s3-ping	1.0.0.Final

Discovery protocol	Default stack file	Artifact	Version
GOOGLE_PING2	default-jgroups-google.xml	org.jgroups.google:jgroups-google	1.0.0.Final
AZURE_PING	default-jgroups-azure.xml	org.jgroups.azure:jgroups-azure	1.3.0.Final

Providing Dependencies for Cloud Discovery Protocols

To use NATIVE_S3_PING, GOOGLE_PING2, or AZURE_PING cloud discovery protocols, you need to provide dependent libraries to Infinispan.

Procedure

- Add the artifact dependencies to your project pom.xml.

You can then configure the cloud discovery protocol as part of a JGroups stack file or with system properties.

Additional resources

- JGroups NATIVE_S3_PING
- JGroups GOOGLE_PING2
- JGroups AZURE_PING

4.3. Using the Default JGroups Stacks

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

Infinispan 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

Do one of the following to use one of the default JGroups stacks:

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

```
<infinispan>
  <cache-container default-cache="replicatedCache">
    <!-- Use the default UDP stack for cluster transport. -->
    <transport cluster="${infinispan.cluster.name}"
               stack="udp"
               node-name="${infinispan.node.name:}"/>
  </cache-container>
</infinispan>
```

- Use the addProperty() method to set the JGroups stack file:

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

Verification

Infinispan logs the following message to indicate which stack it uses:

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

4.4. Customizing JGroups Stacks

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

Infinispan 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.
2. Add the `extends` attribute and specify a JGroups stack to inherit properties from.
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.
5. Specify the stack name as the value for the `stack` attribute in the `transport` configuration.

   For example, you might evaluate using a Gossip router and symmetric encryption with the default TCP stack as follows:

```
<infinispan>
  <jgroups>
    <!-- Creates a custom JGroups stack named "my-stack". -->
    <!-- Inherits properties from the default TCP stack. -->
    <stack name="my-stack" extends="tcp">
      <!-- Uses TCPGOSSIP as the discovery mechanism instead of MPING -->
      <TCPGOSSIP initial_hosts=
"${jgroups.tunnel.gossip_router_hosts:localhost[12001]}"
             stack.combine="REPLACE"
             stack.position="MPING" />
      <!-- Removes the FD_SOCK protocol from the stack. -->
      <FD_SOCK stack.combine="REMOVE"/>
      <!-- Modifies the timeout value for the VERIFY_SUSPECT protocol. -->
      <VERIFY_SUSPECT timeout="2000"/>
      <!-- Adds SYM_ENCRYPT to the stack after VERIFY_SUSPECT. -->
      <SYM_ENCRYPT sym_algorithm="AES"
                   keystore_name="mykeystore.p12"
                   keystore_type="PKCS12"
                   store_password="changeit"
                   key_password="changeit"
                   alias="myKey"
                   stack.combine="INSERT_AFTER"
                   stack.position="VERIFY_SUSPECT" />
    </stack>
    <cache-container name="default" statistics="true">
      <!-- Uses "my-stack" for cluster transport. -->
      <transport cluster="${infinispan.cluster.name}"
                 stack="my-stack"
                 node-name="${infinispan.node.name:}"/>
   </cache-container>
  </jgroups>
</infinispan>
```

6. Check Infinispan logs to ensure it uses the stack.

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

4.4.1. 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.
INSERT_BEFORE	Inserts a protocols into the stack before another protocol. Affects the protocol that you specify as the insertion point.
REMOVE	Removes protocols from the stack.

4.5. Using JGroups System Properties

Pass system properties to Infinispan 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
```

When you embed Infinispan clusters in clustered WildFly applications, JGroups system properties can clash or override each other.

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

4.5.1. Cluster Transport Properties

Use the following properties to customize JGroups cluster transport.

System Property	Description	Default Value	Required/Optional
jgroups.bind.address	Bind address for cluster transport.	SITE_LOCAL	Optional

System Property	Description	Default Value	Required/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

Reference

- JGroups System Properties
- JGroups Protocol List

4.5.2. System Properties for Cloud Discovery Protocols

Use the following properties to configure JGroups discovery protocols for hosted platforms.

Amazon EC2

System properties for configuring NATIVE_S3_PING.

System Property	Description	Default Value	Required/Optional
jgroups.s3.region_name	Name of the Amazon S3 region.	No default value.	Optional
jgroups.s3.bucket_name	Name of the Amazon S3 bucket. The name must exist and be unique.	No default value.	Optional

Google Cloud Platform

System properties for configuring `GOOGLE_PING2`.

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

Azure

System properties for `AZURE_PING`.

System Property	Description	Default Value	Required/Optional
`jboss.jgroups.azure_ping.storage_account_name`	Name of the Azure storage account. The name must exist and be unique.	No default value.	Required
`jboss.jgroups.azure_ping.storage_access_key`	Name of the Azure storage access key.	No default value.	Required
`jboss.jgroups.azure_ping.container`	Valid DNS name of the container that stores ping information.	No default value.	Required

Kubernetes

System properties for `DNS_PING`.

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

4.6. 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>
  <!-- Contains one or more JGroups stack definitions. -->
  <jgroups>
    <!-- Defines a custom JGroups stack named "prod". -->
    <stack name="prod">
      <TCP bind_port="7800" port_range="30" recv_buf_size="20000000" send_buf_size
="640000"/>
      <MPING 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="${jgroups.udp.ip_ttl:2}"/>
      <MERGE3 />
      <FD_SOCK />
      <FD_ALL timeout="3000" interval="1000" timeout_check_interval="1000" />
      <VERIFY_SUSPECT timeout="1000" />
      <pbcast.NAKACK2 use_mcast_xmit="false" xmit_interval="100"
xmit_table_num_rows="50"
                      xmit_table_msgs_per_row="1024"
xmit_table_max_compaction_time="30000" />
      <UNICAST3 xmit_interval="100" xmit_table_num_rows="50"
xmit_table_msgs_per_row="1024"
                xmit_table_max_compaction_time="30000" />
      <pbcast.STABLE desired_avg_gossip="2000" max_bytes="1M" />
      <pbcast.GMS print_local_addr="false" join_timeout=
"${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">
    <!-- Uses "prod" for cluster transport. -->
    <transport cluster="${infinispan.cluster.name}"
           stack="prod"
           node-name="${infinispan.node.name:}"/>
  </cache-container>
</infinispan>
```

4.7. 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>
     <!-- Creates a "prod-tcp" stack that references an external file. -->
     <stack-file name="prod-tcp" path="prod-jgroups-tcp.xml"/>
  </jgroups>
  <cache-container default-cache="replicatedCache">
    <!-- Use the "prod-tcp" stack for cluster transport. -->
    <transport stack="prod-tcp" />
    <replicated-cache name="replicatedCache"/>
  </cache-container>
  <!-- Cache configuration goes here. -->
</infinispan>
```

You can also use the `addProperty()` method in the `TransportConfigurationBuilder` class to specify a custom JGroups stack file as follows:

```
GlobalConfiguration globalConfig = new GlobalConfigurationBuilder().transport()
        .defaultTransport()
        .clusterName("prod-cluster")
        //Uses a custom JGroups stack for cluster transport.
        .addProperty("configurationFile", "my-jgroups-udp.xml")
        .build();
```

In this example, `my-jgroups-udp.xml` references a UDP stack with custom properties such as the following:

Custom UDP stack example

```
<config xmlns="urn:org:jgroups"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:org:jgroups http://www.jgroups.org/schema/jgroups-
4.2.xsd">
    <UDP bind_addr="${jgroups.bind_addr:127.0.0.1}"
         mcast_addr="${jgroups.udp.mcast_addr:192.0.2.0}"
         mcast_port="${jgroups.udp.mcast_port:46655}"
         tos="8"
         ucast_recv_buf_size="20000000"
         ucast_send_buf_size="640000"
         mcast_recv_buf_size="25000000"
         mcast_send_buf_size="640000"
         max_bundle_size="64000"
         ip_ttl="${jgroups.udp.ip_ttl:2}"
         enable_diagnostics="false"
         thread_naming_pattern="pl"
         thread_pool.enabled="true"
         thread_pool.min_threads="2"
         thread_pool.max_threads="30"
         thread_pool.keep_alive_time="5000" />
    <!-- Other JGroups stack configuration goes here. -->
</config>
```

Additional resources

- org.infinispan.configuration.global.TransportConfigurationBuilder

4.8. 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());
```

Infinispan cannot use custom JChannels that are already connected.

Reference

JGroups JChannel

4.9. Encrypting Cluster Transport

Secure cluster transport so that nodes communicate with encrypted messages. You can also

configure Infinispan clusters to perform certificate authentication so that only nodes with valid identities can join.

4.9.1. Infinispan Cluster Security

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

Infinispan 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 Infinispan configuration. This allows Infinispan clusters to generate and distribute secret keys.

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 Infinispan 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 Infinispan configuration. This allows Infinispan clusters to obtain secret keys from keystores that you provide.

1. Nodes install the secret key from a keystore on the Infinispan 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. Infinispan 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.

4.9.2. Configuring Cluster Transport with Asymmetric Encryption

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

Procedure

1. Create a keystore with certificate chains that enables Infinispan to verify node identity.
2. Place the keystore on the classpath for each node in the cluster.

   For Infinispan Server, you put the keystore in the $ISPN_HOME directory.

3. Add the `SSL_KEY_EXCHANGE` and `ASYM_ENCRYPT` protocols to a JGroups stack in your Infinispan configuration, as in the following example:

```
<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.
-->
      <!-- Uses the stack.combine and stack.position attributes to insert
SSL_KEY_EXCHANGE into the default TCP stack after VERIFY_SUSPECT. -->
      <SSL_KEY_EXCHANGE keystore_name="mykeystore.jks"
                        keystore_password="changeit"
                        stack.combine="INSERT_AFTER"
                        stack.position="VERIFY_SUSPECT"/>
      <!-- Configures ASYM_ENCRYPT -->
      <!-- Uses the stack.combine and stack.position attributes to insert
ASYM_ENCRYPT into the default TCP stack before pbcast.NAKACK2. -->
      <!-- The use_external_key_exchange = "true" attribute configures nodes to use
the `SSL_KEY_EXCHANGE` protocol for certificate authentication. -->
      <ASYM_ENCRYPT asym_keylength="2048"
                    asym_algorithm="RSA"
                    change_key_on_coord_leave = "false"
                    change_key_on_leave = "false"
                    use_external_key_exchange = "true"
                    stack.combine="INSERT_BEFORE"
                    stack.position="pbcast.NAKACK2"/>
    </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 Infinispan 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>
```

Infinispan 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 Infinispan 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

4.9.3. Configuring Cluster Transport with Symmetric Encryption

Configure Infinispan 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 Infinispan Server, you put the keystore in the $ISPN_HOME directory.

3. Add the SYM_ENCRYPT protocol to a JGroups stack in your Infinispan configuration.

```
<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 Infinispan 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>
```

Infinispan 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 Infinispan 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

4.10. TCP and UDP Ports for Cluster Traffic

Infinispan 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

Infinispan uses the following ports for the JGroups RELAY2 protocol:

`7900`
: For Infinispan clusters running on Kubernetes.

`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.