Security Guide for Infinispan 13.0

Table of Contents

Chapter 1. Infinispan Security

Infinispan provides security for components as well as data across different layers:

- Within the core library to provide role-based access control (RBAC) to CacheManagers, Cache instances, and stored data.
- Over remote protocols to authenticate client requests and encrypt network traffic.
- Across nodes in clusters to authenticate new cluster members and encrypt the cluster transport.

The Infinispan core library uses standard Java security libraries such as JAAS, JSSE, JCA, JCE, and SASL to ease integration and improve compatibility with custom applications and container environments. For this reason, the Infinispan core library provides only interfaces and a set of basic implementations.

Infinispan servers support a wide range of security standards and mechanisms to readily integrate with enterprise-level security frameworks.

Chapter 2. Configuring User Authorization

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

2.1. Enabling Authorization in Cache Configuration

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

Procedure

1. Open your `infinispan.xml` configuration for editing.
2. If it is not already declared, add the `<authorization />` tag inside the `security` elements for the `cache-container`.

   This enables authorization for the Cache Manager and provides a global set of roles and permissions that caches can inherit.

3. Add the `<authorization />` tag to each cache for which Infinispan restricts access based on user roles.

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

```
<infinispan>
  <cache-container default-cache="rbac-cache" name="restricted">
    <security>
      <!-- Enable authorization with the default roles and permissions. -->
      <authorization />
    </security>
    <local-cache name="rbac-cache">
      <security>
        <!-- Inherit authorization settings from the cache-container. -->
        <authorization/>
      </security>
    </local-cache>
  </cache-container>
</infinispan>
```

2.2. User Roles and Permissions

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

`ClusterRoleMapper` is the default mechanism that Infinispan uses to associate security principals to authorization roles.

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

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

Reference

- org.infinispan.security.AuthorizationPermission Enumeration
- Infinispan Configuration Schema Reference

2.3. How Security Authorization Works

Infinispan authorization secures your installation by restricting user access.

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

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

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

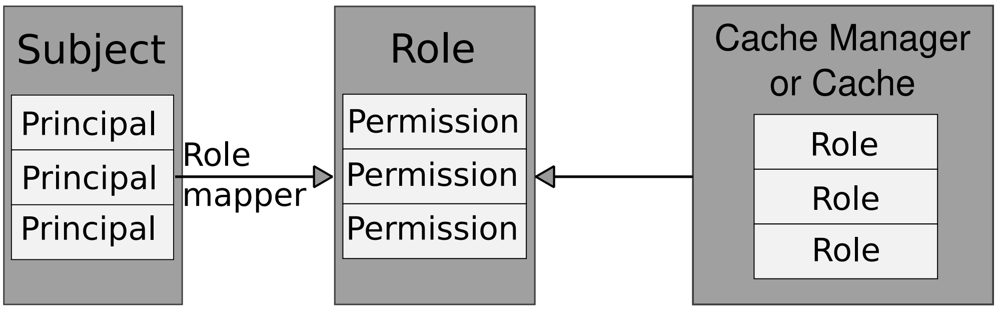
Identities

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

Identities to roles

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

The following image illustrates how security principals correspond to roles:



2.3.1. Permissions

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

Cache Manager permissions

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

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.

Permission	Function	Description
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.
MONITOR	`getStats`	Allows access to JMX statistics and the `metrics` endpoint.
ALL	-	Includes all cache permissions.
ALL_READ	-	Combines the READ and BULK_READ permissions.
ALL_WRITE	-	Combines the WRITE and BULK_WRITE permissions.

Reference

- Infinispan Security API

2.3.2. Role Mappers

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

Cluster role mappers

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

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

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

Identity role mappers

`IdentityRoleMapper` uses the Principal name as the role name.

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

CommonName role mappers

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

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

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

Custom role mappers

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

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

Reference

- Infinispan Security API
- org.infinispan.security.PrincipalRoleMapper

2.4. Access Control List (ACL) Cache

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

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

```
<security cache-size="1000" cache-timeout="300000">
  <authorization />
</security>
```

Reference

- Infinispan Configuration Schema Reference

2.5. Customizing Roles and Permissions

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

Procedure

1. Open your `infinispan.xml` configuration for editing.
2. Configure authorization for the `cache-container` by declaring a role mapper and a set of roles and permissions.
3. Configure authorization for caches to restrict access based on user roles.

The following configuration example shows how to configure security authorization with roles and permissions:

```
<infinispan>
  <cache-container default-cache="restricted" name="custom-authorization">
    <security>
      <authorization>
        <!-- Declare a role mapper that associates a security principal
             to each role. -->
        <identity-role-mapper />
        <!-- Specify user roles and corresponding permissions. -->
        <role name="admin" permissions="ALL" />
        <role name="reader" permissions="READ" />
        <role name="writer" permissions="WRITE" />
        <role name="supervisor" permissions="READ WRITE EXEC"/>
      </authorization>
    </security>
    <local-cache name="implicit-authorization">
      <security>
        <!-- Inherit roles and permissions from the cache-container. -->
        <authorization/>
      </security>
    </local-cache>
    <local-cache name="restricted">
      <security>
        <!-- Explicitly define which roles can access the cache. -->
        <authorization roles="admin supervisor"/>
      </security>
    </local-cache>
  </cache-container>
</infinispan>
```

2.6. Disabling Security Authorization

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

Procedure

1. Open your `infinispan.xml` configuration for editing.
2. Remove any `authorization` elements from the `security` configuration for the `cache-container` and each cache configuration.

2.7. Configuring Authorization with Client Certificates

Enabling client certificate authentication means you do not need to specify Infinispan user credentials in client configuration, which means you must associate roles with the Common Name (CN) field in the client certificate(s).

Prerequisites

- Provide clients with a Java keystore that contains either their public certificates or part of the certificate chain, typically a public CA certificate.
- Configure Infinispan Server to perform client certificate authentication.

Procedure

1. Enable the `common-name-role-mapper` in the security authorization configuration.
2. Assign the Common Name (`CN`) from the client certificate a role with the appropriate permissions.

```
<cache-container name="certificate-authentication" statistics="true">
   <security>
     <authorization>
       <!-- Declare a role mapper that associates the common name (CN) field
            in client certificate trust stores with authorization roles. -->
       <common-name-role-mapper/>
       <!-- In this example, if a client certificate contains `CN=Client1` then
            clients with matching certificates get ALL permissions. -->
       <role name="Client1" permissions="ALL"/>
     </authorization>
   </security>
</cache-container>
```

2.8. Programmatically Configuring Authorization

When using Infinispan 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 Infinispan 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. Infinispan denies any access requests from those users.

Reference

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

2.9. Code Execution with Secure Caches

When you configure Infinispan 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 Infinispan, however `Security.doAs()` provides better performance.

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

```
Security.getSubject();
```

Chapter 3. 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.

3.1. 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.

3.2. 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

3.3. 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

Chapter 4. Infinispan Ports and Protocols

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

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

4.1. Infinispan Server Ports and Protocols

Infinispan 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.

4.1.1. 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.

4.2. 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.