Setting Up Cross-Site Replication with Infinispan 10.1

Table of Contents

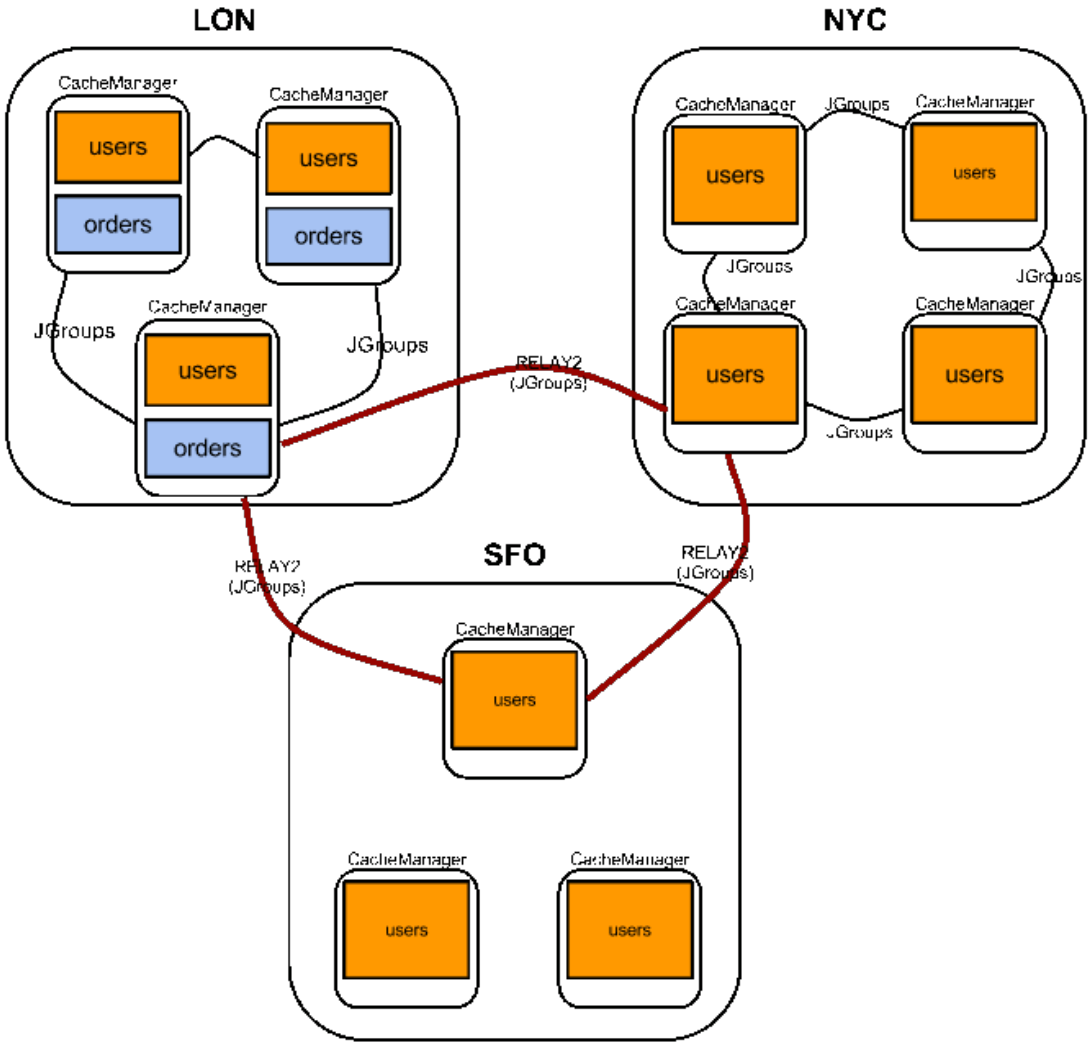
Chapter 1. Cross site replication

Cross site (x-site) replication allows backing up the data from one cluster to other clusters, potentially situated in different geographical location. The cross-site replication is built on top of JGroups' RELAY2 protocol . This document describes the technical design of cross site replication in more detail.

Cross site replication needs the backup cache running in the site master node(s) (i.e. node which receives the backup and applies it). The backup cache starts automatically when it receives the first backup request.

1.1. Sample deployment

The diagram below depicts a possible setup of replicated sites, followed by a description of individual elements present in the deployment. Options are then explained at large in future paragraphs. Comments on the diagram above:



- there are 3 sites: LON, NYC and SFO.
- in each site there is a running Infinispan cluster with a (potentially) different number of

physical nodes: 3 nodes in LON, 4 nodes in NYC and 3 nodes in SFO

- the "users" cache is active in LON, NYC and SFO. Updates on the "users" cache in any of these sites gets replicated to the other sites as well
- it is possible to use different replication mechanisms between sites. E.g. One can configure SFO to backup data synchronously to NYC and asynchronously to LON
- the "users" cache can have a different configuration from one site to the other. E.g. it might be configured as distributed with numOwners=2 in the LON site, REPL in the NYC site and distributed with numOwners=1 in the SFO site
- JGroups is used for both inter-site and intra-site communication. RELAY2 is used for inter-site communication
- "orders" is a site local to LON, i.e. updates to the data in "orders" don't get replicated to the remote sites The following sections discuss specific aspects of cross site replication into more detail. The foundation of the cross-site replication functionality is RELAY2 so it highly recommended to read JGroups' RELAY2 documentation before moving on into cross-site. Configuration

The cross-site replication configuration spreads over the following files:

1. the backup policy for each individual cache is defined in the Infinispan .xml configuration file (infinispan.xml)
2. cluster's JGroups xml configuration file: RELAY2 protocol needs to be added to the JGroups protocol stack (jgroups.xml)
3. RELAY2 configuration file: RELAY2 has an own configuration file (relay2.xml)
4. the JGroups channel that is used by RELAY2 has its own configuration file (jgroups-relay2.xml) Infinispan XML configuration file

The local site is defined in the the global configuration section. The local is the site where the node using this configuration file resides (in the example above local site is "LON").

infinispan.xml

```
<transport site="LON" />
```

The same setup can be achieved programmatically:

```
GlobalConfigurationBuilder lonGc = GlobalConfigurationBuilder.defaultClusteredBuilder
();
lonGc.site().localSite("LON");
```

The names of the site (case sensitive) should match the name of a site as defined within JGroups' RELAY2 protocol configuration file. Besides the global configuration, each cache specifies its backup policy in the "site" element:

infinispan.xml

```
<distributed-cache name="users">
  <backups>
    <backup site="NYC" failure-policy="WARN" strategy="SYNC" timeout="12000"/>
    <backup site="SFO" failure-policy="IGNORE" strategy="ASYNC"/>
    <backup site="LON" strategy="SYNC" enabled="false"/>
  </backups>
</distributed-cache>
```

The "users" cache backups its data to the "NYC" and "SFO" sites. Even though the "LON" appears as a backup site, it has the "enabled" attribute set to *false* so it will be ignored . For each site backup, the following configuration attributes can be specified:

- *strategy* - the strategy used for backing up data, either "SYNC" or "ASYNC". Defaults to "ASYNC".
- *failure-policy* - Decides what the system would do in case of failure during backup. Possible values are:
  - *IGNORE* - allow the local operation/transaction to succeed
  - *WARN* - same as IGNORE but also logs a warning message. Default.
  - *FAIL* - only in effect if "strategy" is "SYNC" - fails local cluster operation/transaction by throwing an exception to the user
  - *CUSTOM* - user provided, see "failurePolicyClass" below
- failurePolicyClass - If the 'failure-policy' is set to 'CUSTOM' then this attribute is required and should contain the fully qualified name of a class implementing org.infinispan.xsite.CustomFailurePolicy
- timeout - The timeout(milliseconds) to be used when backing up data remotely. Defaults to 10000 (10 seconds)

The same setup can be achieved programatically:

```
ConfigurationBuilder lon = new ConfigurationBuilder();
lon.sites().addBackup()
      .site("NYC")
      .backupFailurePolicy(BackupFailurePolicy.WARN)
      .strategy(BackupConfiguration.BackupStrategy.SYNC)
      .replicationTimeout(12000)
      .sites().addInUseBackupSite("NYC")
    .sites().addBackup()
      .site("SFO")
      .backupFailurePolicy(BackupFailurePolicy.IGNORE)
      .strategy(BackupConfiguration.BackupStrategy.ASYNC)
      .sites().addInUseBackupSite("SFO")
```

The "users" cache above doesn't know on which cache on the remote sites its data is being replicated. By default the remote site writes the backup data to a cache having the same name as

the originator, i.e. "users". This behaviour can be overridden with an "backupFor" element. For example the following configuration in SFO makes the "usersLONBackup" cache act as the backup cache for the "users" cache defined above in the LON site:

infinispan.xml

```
<infinispan>
  <cache-container default-cache="">
     <distributed-cache name="usersLONBackup">
        <backup-for remote-cache="users" remote-site="LON"/>
     </distributed-cache>
  </cache-container>
</infinispan>
```

The same setup can be achieved programatically:

```
ConfigurationBuilder cb = new ConfigurationBuilder();
cb.sites().backupFor().remoteCache("users").remoteSite("LON");
```

1.1.1. Local cluster's jgroups .xml configuration

This is the configuration file for the local (intra-site) Infinispan cluster. It is referred from the Infinispan configuration file, see "configurationFile" below:

infinispan.xml

```
<infinispan>
  <jgroups>
     <!-- Add custom JGroups stacks in external files. -->
     <stack-file name="prod-tcp" path="prod-jgroups-tcp.xml"/>
  </jgroups>
  <cache-container default-cache="replicatedCache">
    <!-- Add custom JGroups stacks to clustered caches. -->
    <transport stack="prod-tcp" />
    <replicated-cache name="replicatedCache"/>
  </cache-container>
  ...
</infinispan>
```

In order to allow inter-site calls, the RELAY2 protocol needs to be added to the protocol stack defined in the jgroups configuration (see attached jgroups.xml for an example).

1.1.2. RELAY2 configuration file

The RELAY2 configuration file is linked from the jgroups.xml (see attached relay2.xml). It defines the sites seen by this cluster and also the JGroups configuration file that is used by RELAY2 in order to communicate with the remote sites.

1.2. Data replication

For both transactional and non-transactional caches, the backup calls are performed in parallel with local cluster calls, e.g. if we write data to node N1 in LON then replication to the local nodes N2 and N3 and remote backup sites SFO and NYC happen in parallel.

1.2.1. Non transactional caches

In the case of non-transactional caches the replication happens during each operation. Given that data is sent in parallel to backups and local caches, it is possible for the operations to succeed locally and fail remotely, or the other way, causing inconsistencies

1.2.2. Transactional caches

For synchronous transactional caches, Infinispan internally uses a two phase commit protocol: lock acquisition during the 1st phase (prepare) and apply changes during the 2nd phase (commit). For asynchronous caches the two phases are merged, the "apply changes" message being sent asynchronously to the owners of data. This 2PC protocol maps to 2PC received from the JTA transaction manager. For transactional caches, both optimistic and pessimistic, the backup to remote sites happens during the prepare and commit phase only.

Synchronous local cluster with async backup

In this scenario the backup call happens during local commit phase(2nd phase). That means that if the local prepare fails, no remote data is being sent to the remote backup.

Synchronous local cluster with sync backup

In this case there are two backup calls:

- during prepare a message is sent across containing all the modifications that happened within this transaction
- if the remote backup cache is transactional then a transaction is started remotely and all these modifications are being written within this transaction’s scope. The transaction is not committed yet (see below)
- if the remote backup cache is not transactional, then the changes are applied remotely
- during the commit/rollback, a commit/rollback message is sent across
- if the remote backups cache is transactional then the transaction started at the previous phase is committed/rolled back
- if the remote backup is not transactional then this call is ignored

Both the local and the backup call(if the "backupFailurePolicy" is set to "FAIL") can veto transaction’s prepare outcome

Asynchronous local cluster

In the case of asynchronous local clusters, the backup data is sent during the commit phase. If the

backup call fails and the "backupFailurePolicy" is set to "FAIL" then the user is notified through an exception.

Replication of expired cache entries across sites

You can configure Infinispan to expire entries, which controls the amount of time entries remain in the cache.

- `lifespan` expiration is suitable for cross-site replication because entries expire without the need for Infinispan to determine if the entries have expired in clusters on other sites.
- `max-idle` expiration does not work with cross-site replication because Infinispan cannot determine if cache entries have reached the idle timeout in clusters on other sites.

1.3. Taking a site offline

If backing up to a site fails for a certain number of times during a time interval, then it is possible to automatically mark that site as offline. When a site is marked as offline the local site won't try to backup data to it anymore. In order to be taken online a system administrator intervention being required.

1.3.1. Configuration

The following configuration provides an example for taking sites offline:

infinispan.xml

```
<replicated-cache name="bestEffortBackup">
   ...
   <backups>
     <backup site="NYC" strategy="SYNC" failure-policy="FAIL">
         <take-offline after-failures="500" min-wait="10000"/>
     </backup>
   </backups>
    ...
</replicated-cache>
```

The *take-offline* element under the *backup* configures the taking offline of a site:

- *after-failures* - the number of consecutive failed backup operations after which this site should be taken offline. Defaults to 0. A zero or negative value would mean that the site will be taken offline after *minTimeToWait*
- *min-wait* - the number of milliseconds in which a site is not marked offline even if it is unreachable for 'afterFailures' number of times. If smaller or equal to 0, then only *afterFailures* is considered.

The equivalent programmatic configuration is:

```
lon.sites().addBackup()
      .site("NYC")
      .backupFailurePolicy(BackupFailurePolicy.FAIL)
      .strategy(BackupConfiguration.BackupStrategy.SYNC)
      .takeOffline()
         .afterFailures(500)
         .minTimeToWait(10000);
```

after-failures	*min-wait*	**Result**
`<=0`	`<=0`	The site will never be taken offline.
`N`	`<=0`	The site will be offline after `N` consecutive requests failures.
`<=0`	`M`	The site will be offline `M` milliseconds after the first request failure. A successful request rests this timer.
`N`	`M`	The site will be offline after `N` consecutive requests failures and after `M` milliseconds after the first consecutive failure.

1.3.2. Bringing Sites Back Online

After taking sites offline, invoke the `bringSiteOnline(siteName)` operation via the following JMX MBeans to bring sites back online:

- `XSiteAdmin` enables replication on caches across clusters in a remote site.
- `GlobalXSiteAdminOperations` enables replication on cache containers across clusters in a remote site.

The `bringSiteOnline(siteName)` operation enables replication only and does not do a full update. For this reason, when you bring sites back online, they only contain new entries. You should push transfer to sites after you bring them online to synchronize the most recent data.

Pushing state transfer brings sites back online. You can do that instead of invoking `bringSiteOnline(siteName)`.

1.4. Pushing State Transfer to Sites

Transferring state from one site to another synchronizes the data between the two sites.

You should always transfer state from the currently active site, which contains the most up-to-date data, to another site. The site to which you push state transfer can be online or offline. If you push state transfer to an offline site, it brings that site back online.

Caches on sites to which you transfer state from another site do not need to be empty. However, state transfer only overwrites existing keys on receiving sites and does not delete keys.

For example, key K exists on site A and site B. You transfer state from site A to site B. In this case, Infinispan overwrites key K. However, key Y exists on site B but not site A. After you transfer state from site A to site B, key Y still exists on site B.

You can receive state transfer from only one site at a time. Likewise, if you invoke a state transfer operation on a site, Infinispan ignores subsequent invocations on that site.

Invoke the `pushState(String)` operation via the following JMX MBeans to bring sites back online:

- `XSiteAdminOperations` synchronizes state for caches between the site on which you invoke the operation and a remote site. Brings the site online if it is offline.
- `GlobalXSiteAdminOperations` synchronizes state for cache containers between the site on which you invoke the operation and a remote site. Brings the site online if it is offline.

When you invoke the `pushState(String)` operation on the site that transfers state, you specify the name of the site that receives the state.

The following figure shows the pushState(String) operation in JConsole:

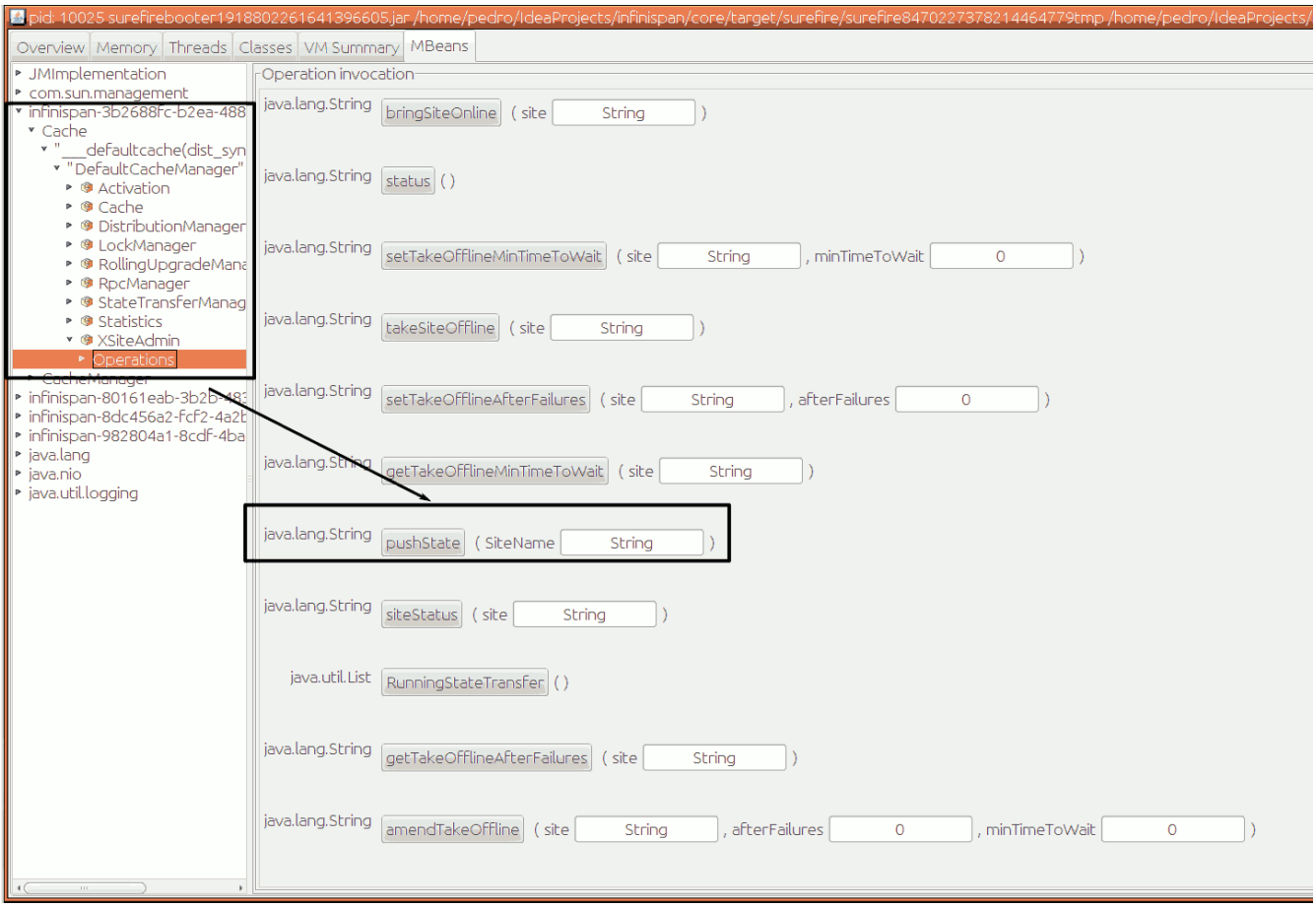


Figure 1. Pushing state via JConsole

1.4.1. Handling join/leave nodes

The current implementation automatically handles the topology changes in producer or consumer site. Also, the cross-site state transfer can run in parallel with a local site state transfer.

1.4.2. Handling broken link between sites

A System Administrator action is needed if the link between the producer and consumer site is broken during the cross-site state transfer (data consistency is not ensured in consumer site). The producer site retries for a while before giving up. Then, it gets back to normal state. However, the consumer site is not able to get back to normal state and, here, an action from System Administrator is need. The System Administrator should use the operation cancelReceiveState(String siteName) to bring the consumer site to normal state.

1.4.3. System Administrator Operations

A set of operations can be performed to control the cross-site state transfer:

- pushState(String siteName) - It starts the cross-site state transfer to the site name specified;
- cancelPushState(String siteName) - It cancels the cross-site state transfer to the site name specified;
- getRunningStateTransfer() - It returns a list of site name to which this site is pushing the state;
- getSendingSiteName() - It returns the site name that is pushing state to this site;
- cancelReceiveState(String siteName) - It restores the site to normal state. Should be used when the link between the sites is broken during the state transfer (as described above);
- getPushStateStatus() - It returns the status of completed cross-site state transfer;
- clearPushStateStatus() - It clears the status of completed cross-site state transfer.

For more technical information, you can check the Cross Site design document. See Reference.

1.4.4. Configuration

State transfer between sites cannot be enabled or disabled but it allows to tune some parameters. The values shown below are the default values:

infinispan.xml

```
<replicated-cache name="xSiteStateTransfer">
   ...
   <backups>
      <backup site="NYC" strategy="SYNC" failure-policy="FAIL">
         <state-transfer chunk-size="512" timeout="1200000" max-retries="30" wait-
time="2000" />
      </backup>
   </backups>
    ...
</replicated-cache>
```

The equivalent programmatic configuration is:

```
lon.sites().addBackup()
      .site("NYC")
      .backupFailurePolicy(BackupFailurePolicy.FAIL)
      .strategy(BackupConfiguration.BackupStrategy.SYNC)
      .stateTransfer()
         .chunkSize(512)
         .timeout(1200000)
         .maxRetries(30)
         .waitingTimeBetweenRetries(2000);
```

Below, it is the parameters description:

- *chunk-size* - The number of keys to batch before sending them to the consumer site. A negative or a zero value is **not** a valid value. Default value is 512 keys.
- *timeout* - The time (in milliseconds) to wait for the consumer site acknowledge the reception and appliance of a state chunk. A negative or zero value is **not** a valid value. Default value is 20 minutes.
- *max-retries* - The maximum number of retries when a push state command fails. A negative or a zero value means that the command will not retry in case of failure. Default value is 30.
- *wait-time* - The waiting time (in milliseconds) between each retry. A negative or a zero value is **not** a valid value. Default value is 2 seconds.

1.5. Reference

This document describes the technical design of cross site replication in more detail.