Red Hat

Red Hat Data Grid 8.4

Data Grid Cross-Site Replication

Back up data between Data Grid clusters

Last Updated: 2024-04-19

Red Hat Data Grid 8.4 Data Grid Cross-Site Replication

Back up data between Data Grid clusters

Abstract

Data Grid can form global clusters to replicate data across geographic locations. This guide explains how to configure backup locations for caches and perform cross-site operations.

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 8.4 Documentation
- Data Grid 8.4 Component Details
- Supported Configurations for Data Grid 8.4
- 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. CROSS-SITE REPLICATION

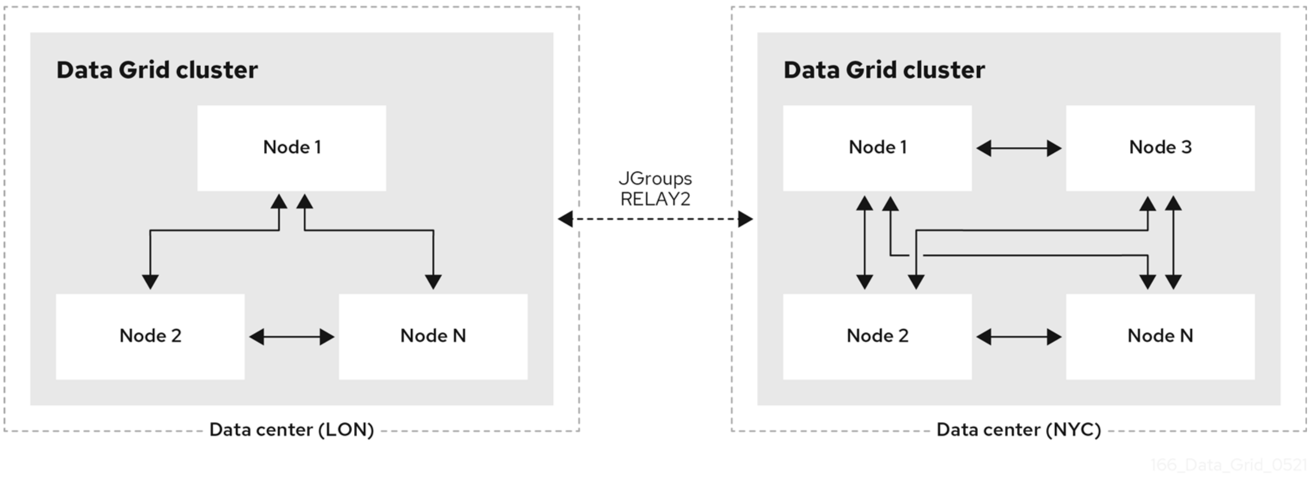
This section explains Data Grid cross-site replication capabilities, including details about relay nodes, state transfer, and client connections for remote caches.

1.1. CROSS-SITE REPLICATION

Data Grid can back up data between clusters running in geographically dispersed data centers and across different cloud providers. Cross-site replication provides Data Grid with a global cluster view and:

- Guarantees service continuity in the event of outages or disasters.
- Presents client applications with a single point of access to data in globally distributed caches.

Figure 1.1. Cross-site replication



1.2. RELAY NODES

Relay nodes are the nodes in Data Grid clusters that are responsible for sending and receiving requests from backup locations.

If a node is not a relay node, it must forward backup requests to a local relay node. Only relay nodes can send requests to backup locations.

For optimal performance, you should configure all nodes as relay nodes. This increases the speed of backup requests because each node in the cluster can backup to remote sites directly without having to forward backup requests to local relay nodes.

NOTE

Diagrams in this document illustrate Data Grid clusters with one relay node because this is the default for the JGroups RELAY2 protocol. Likewise, a single relay node is easier to illustrate because each relay node in a cluster communicates with each relay node in the remote cluster.

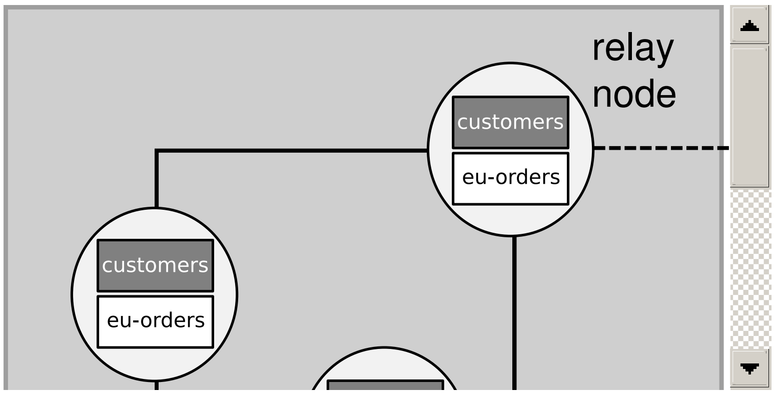
NOTE

JGroups configuration refers to relay nodes as "site master" nodes. Data Grid uses relay node instead because it is more descriptive and presents a more intuitive choice for our users.

1.3. DATA GRID CACHE BACKUPS

Data Grid caches include a **backups** configuration that let you name remote sites as backup locations.

For example, the following diagram shows three caches, "customers", "eu-orders", and "us-orders":



- In **LON**, "customers" names **NYC** as a backup location.
- In **NYC**, "customers" names **LON** as a backup location.
- "eu-orders" and "us-orders" do not have backups and are local to the respective cluster.

1.4. BACKUP STRATEGIES

Data Grid replicates data between clusters at the same time that writes to caches occur. For example, if a client writes "k1" to **LON**, Data Grid backs up "k1" to **NYC** at the same time.

To back up data to a different cluster, Data Grid can use either a synchronous or asynchronous strategy.

Synchronous strategy

When Data Grid replicates data to backup locations, it writes to the cache on the local cluster and the cache on the remote cluster concurrently. With the synchronous strategy, Data Grid waits for both write operations to complete before returning.

You can control how Data Grid handles writes to the cache on the local cluster if backup operations fail. Data Grid can do the following:

- Ignore the failed backup and silently continue the write to the local cluster.
- Log a warning message or throw an exception and continue the write to the local cluster.
- Handle failed backup operations with custom logic.

Synchronous backups also support two-phase commits with caches that participate in optimistic transactions. The first phase of the backup acquires a lock. The second phase commits the modification.

IMPORTANT

Two-phase commit with cross-site replication has a significant performance impact because it requires two round-trips across the network.

Asynchronous strategy

When Data Grid replicates data to backup locations, it does not wait until the operation completes before writing to the local cache.

Asynchronous backup operations and writes to the local cache are independent of each other. If backup operations fail, write operations to the local cache continue and no exceptions occur. When this happens Data Grid also retries the write operation until the remote cluster disconnects from the cross-site view.

Synchronous vs asynchronous backups

Synchronous backups offer the strongest guarantee of data consistency across sites. If **strategy=sync**, when **cache.put()** calls return you know the value is up to date in the local cache and in the backup locations.

The trade-off for this consistency is performance. Synchronous backups have much greater latency in comparison to asynchronous backups.

Asynchronous backups, on the other hand, do not add latency to client requests so they have no performance impact. However, if **strategy=async**, when **cache.put()** calls return you cannot be sure of that the value in the backup location is the same as in the local cache.

1.5. AUTOMATIC OFFLINE PARAMETERS FOR BACKUP LOCATIONS

Operations to replicate data across clusters are resource intensive, using excessive RAM and CPU. To avoid wasting resources Data Grid can take backup locations offline when they stop accepting requests after a specific period of time.

Data Grid takes remote sites offline based on the number of failed sequential requests and the time interval since the first failure. Requests are failed when the target cluster does not have any nodes in the cross-site view (JGroups bridge) or when a timeout expires before the target cluster acknowledges the request.

Backup timeouts

Backup configurations include timeout values for operations to replicate data between clusters. If operations do not complete before the timeout expires, Data Grid records them as failures.

In the following example, operations to replicate data to NYC are recorded as failures if they do not complete after 10 seconds:

XML

```
<distributed-cache>
  <backups>
    <backup site="NYC"
            strategy="ASYNC"
            timeout="10000" />
  </backups>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "backups": {
      "NYC" : {
        "backup" : {
          "strategy" : "ASYNC",
          "timeout" : "10000"
        }
      }
    }
  }
}
```

YAML

```
distributedCache:
  backups:
    NYC:
      backup:
        strategy: "ASYNC"
        timeout: "10000"
```

Number of failures

You can specify the number of **consecutive** failures that can occur before backup locations go offline.

In the following example, if a cluster attempts to replicate data to NYC and five consecutive operations fail, NYC automatically goes offline:

XML

```
<distributed-cache>
  <backups>
    <backup site="NYC"
            strategy="ASYNC"
            timeout="10000">
      <take-offline after-failures="5"/>
    </backup>
  </backups>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "backups": {
      "NYC" : {
        "backup" : {
          "strategy" : "ASYNC",
          "timeout" : "10000",
          "take-offline" : {
            "after-failures" : "5"
          }
        }
        }
      }
    }
  }
```

YAML

```
distributedCache:
  backups:
    NYC:
      backup:
        strategy: "ASYNC"
        timeout: "10000"
        takeOffline:
          afterFailures: "5"
```

Time to wait
You can also specify how long to wait before taking sites offline when backup operations fail. If a backup request succeeds before the wait time runs out, Data Grid does not take the site offline.

One or two minutes is generally a suitable time to wait before automatically taking backup locations offline. If the wait period is too short then backup locations go offline too soon. You then need to bring clusters back online and perform state transfer operations to ensure data is in sync between the clusters.

A negative or zero value for the number of failures is equivalent to a value of **1**. Data Grid uses only a minimum time to wait to take backup locations offline after a failure occurs, for example:

```
<take-offline after-failures="-1"
              min-wait="10000"/>
```

In the following example, if a cluster attempts to replicate data to NYC and there are more than five consecutive failures and 15 seconds elapse after the first failed operation, NYC automatically goes offline:

XML

```
<distributed-cache>
  <backups>
    <backup site="NYC"
            strategy="ASYNC"
            timeout="10000">
      <take-offline after-failures="5" min-wait="15000"/>
    </backup>
  </backups>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "backups": {
      "NYC" : {
          "backup" : {
            "strategy" : "ASYNC",
            "timeout" : "10000",
            "take-offline" : {
              "after-failures" : "5",
              "min-wait" : "15000"
            }
          }
        }
      }
    }
}
```

YAML

```
distributedCache:
  backups:
    NYC:
      backup:
        strategy: "ASYNC"
        timeout: "10000"
        takeOffline:
          afterFailures: "5"
          minWait: "15000"
```

1.6. STATE TRANSFER

State transfer is an administrative operation that synchronizes data between sites.

For example, **LON** goes offline and **NYC** starts handling client requests. When you bring **LON** back online, the Data Grid cluster in **LON** does not have the same data as the cluster in **NYC**.

To ensure the data is consistent between **LON** and **NYC**, you can push state from **NYC** to **LON**.

- State transfer is bidirectional. For example, you can push state from **NYC** to **LON** or from **LON** to **NYC**.
- Pushing state to offline sites brings them back online.
- State transfer overwrites only data that exists on both sites, the originating site and the receiving site. Data Grid does not delete data.
  For example, "k2" exists on **LON** and **NYC**. "k2" is removed from **NYC** while **LON** is offline. When you bring **LON** back online, "k2" still exists at that location. If you push state from **NYC** to **LON**, the transfer does not affect "k2" on **LON**.

TIP

To ensure contents of the cache are identical after state transfer, remove all data from the cache on the receiving site before pushing state.

Use the **clear()** method or the **clearcache** command from the CLI.

- State transfer does not overwrite updates to data that occur after you initiate the push.

For example, "k1,v1" exists on **LON** and **NYC**. **LON** goes offline so you push state transfer to **LON** from **NYC**, which brings **LON** back online. Before state transfer completes, a client puts "k1,v2" on **LON**.

In this case the state transfer from **NYC** does not overwrite "k1,v2" because that modification happened after you initiated the push.

Automatic state transfer

By default, you must manually perform cross-site state transfer operations with the CLI or via JMX or REST.

However, when using the asynchronous backup strategy, Data Grid can automatically perform cross-site state transfer operations.

When a backup location comes back online, and the network connection is stable, Data Grid initiates bidirectional state transfer between backup locations. For example, Data Grid simultaneously transfers state from **LON** to **NYC** and **NYC** to **LON**.

NOTE

To avoid temporary network disconnects triggering state transfer operations, there are two conditions that backup locations must meet to go offline. The status of a backup location must be offline and it must not be included in the cross-site view with JGroups RELAY2.

The automatic state transfer is also triggered when a cache starts.

In the scenario where **LON** is starting up, after a cache starts, it sends a notification to **NYC**. Following this, **NYC** starts a unidirectional state transfer to **LON**.

Additional resources

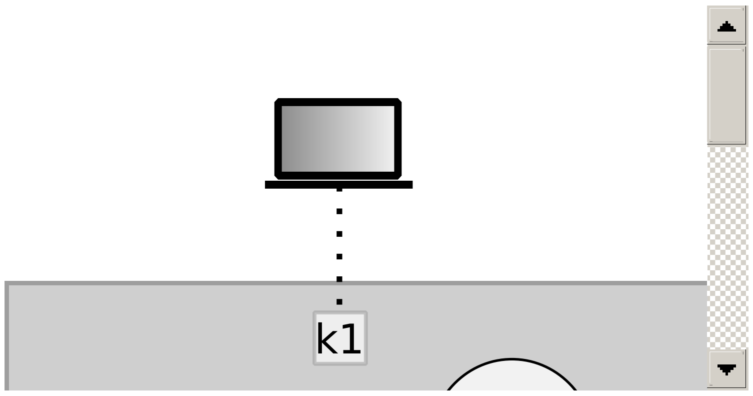
- org.infinispan.Cache.clear()
- Using the Data Grid Command Line Interface
- Data Grid REST API

1.7. CLIENT CONNECTIONS ACROSS SITES

Clients can write to Data Grid clusters in either an Active/Passive or Active/Active configuration.

Active/Passive

The following diagram illustrates Active/Passive where Data Grid handles client requests from one site only:



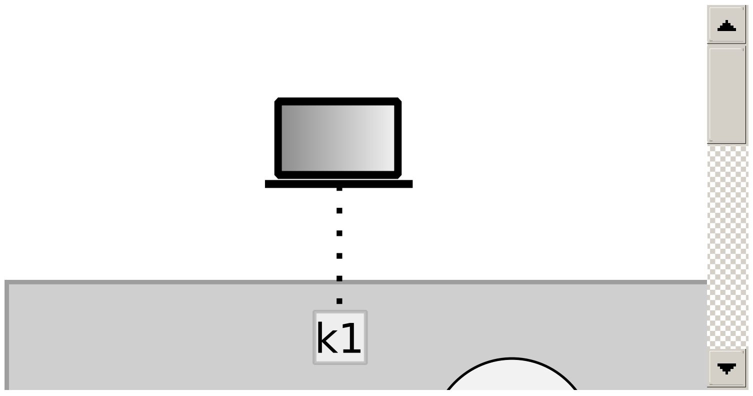
In the preceding image:

1. Client connects to the Data Grid cluster at **LON**.
2. Client writes "k1" to the cache.
3. The relay node at **LON**, "n1", sends the request to replicate "k1" to the relay node at **NYC**, "nA".

With Active/Passive, **NYC** provides data redundancy. If the Data Grid cluster at **LON** goes offline for any reason, clients can start sending requests to **NYC**. When you bring **LON** back online you can synchronize data with **NYC** and then switch clients back to **LON**.

Active/Active

The following diagram illustrates Active/Active where Data Grid handles client requests at two sites:



In the preceding image:

1. Client A connects to the Data Grid cluster at **LON**.
2. Client A writes "k1" to the cache.
3. Client B connects to the Data Grid cluster at **NYC**.
4. Client B writes "k2" to the cache.
5. Relay nodes at **LON** and **NYC** send requests so that "k1" is replicated to **NYC** and "k2" is replicated to **LON**.

With Active/Active both **NYC** and **LON** replicate data to remote caches while handling client requests. If either **NYC** or **LON** go offline, clients can start sending requests to the online site. You can then bring offline sites back online, push state to synchronize data, and switch clients as required.

Backup strategies and client connections

IMPORTANT

An asynchronous backup strategy (**strategy=async**) is recommended with Active/Active configurations.

If multiple clients attempt to write to the same entry concurrently, and the backup strategy is synchronous (**strategy=sync**), then deadlocks occur. However you can use the synchronous backup strategy with an Active/Passive configuration if both sites access different data sets, in which case there is no risk of deadlocks from concurrent writes.

1.7.1. Concurrent writes and conflicting entries

Conflicting entries can occur with Active/Active site configurations if clients write to the same entries at the same time but at different sites.

For example, client A writes to "k1" in **LON** at the same time that client B writes to "k1" in **NYC**. In this case, "k1" has a different value in **LON** than in **NYC**. After replication occurs, there is no guarantee which value for "k1" exists at which site.

To ensure data consistency, Data Grid uses a vector clock algorithm to detect conflicting entries during backup operations, as in the following illustration:

```
            LON       NYC

k1=(n/a)    0,0       0,0

k1=2        1,0  -->  1,0   k1=2

k1=3        1,1  <--  1,1   k1=3

k1=5        2,1       1,2   k1=8

                 -->  2,1 (conflict)
(conflict)  1,2  <--
```

Vector clocks are timestamp metadata that increment with each write to an entry. In the preceding example, **0,0** represents the initial value for the vector clock on "k1".

A client puts "k1=2" in **LON** and the vector clock is **1,0**, which Data Grid replicates to **NYC**. A client then puts "k1=3" in **NYC** and the vector clock updates to **1,1**, which Data Grid replicates to **LON**.

However if a client puts "k1=5" in **LON** at the same time that a client puts "k1=8" in **NYC**, Data Grid detects a conflicting entry because the vector value for "k1" is not strictly greater or less between **LON** and **NYC**.

When it finds conflicting entries, Data Grid uses the Java **compareTo(String anotherString)** method to compare site names. To determine which key takes priority, Data Grid selects the site name that is lexicographically less than the other. Keys from a site named **AAA** take priority over keys from a site named **AAB** and so on.

Following the same example, to resolve the conflict for "k1", Data Grid uses the value for "k1" that originates from **LON**. This results in "k1=5" in both **LON** and **NYC** after Data Grid resolves the conflict and replicates the value.

TIP

Prepend site names with numbers as a simple way to represent the order of priority for resolving conflicting entries; for example, **1LON** and **2NYC**.

Backup strategies

Data Grid performs conflict resolution with the asynchronous backup strategy (**strategy=async**) only.

You should never use the synchronous backup strategy with an Active/Active configuration. In this configuration concurrent writes result in deadlocks and you lose data. However you can use the synchronous backup strategy with an Active/Active configuration if both sites access different data sets, in which case there is no risk of deadlocks from concurrent writes.

Cross-site merge policies

Data Grid provides an **XSiteEntryMergePolicy** SPI in addition to cross-site merge policies that configure Data Grid to do the following:

- Always remove conflicting entries.
- Apply write operations when write/remove conflicts occur.
- Remove entries when write/remove conflicts occur.

Additional resources

- **XSiteMergePolicy** enum lists all merge polices that Data Grid provides
- **XSiteEntryMergePolicy** SPI
- java.lang.String#compareTo()

1.8. EXPIRATION WITH CROSS-SITE REPLICATION

Expiration removes cache entries based on time. Data Grid provides two ways to configure expiration for entries:

Lifespan

The **lifespan** attribute sets the maximum amount of time that entries can exist. When you set **lifespan** with cross-site replication, Data Grid clusters expire entries independently of remote sites.

Maximum idle

The **max-idle** attribute specifies how long entries can exist based on read or write operations in a given time period. When you set a **max-idle** with cross-site replication, Data Grid clusters send touch commands to coordinate idle timeout values with remote sites.

NOTE

Using maximum idle expiration in cross-site deployments can impact performance because the additional processing to keep **max-idle** values synchronized means some operations take longer to complete.

CHAPTER 2. CONFIGURING DATA GRID CROSS-SITE REPLICATION

Set up cluster transport so Data Grid clusters can discover each other and relay nodes can send messages for cross-site replication. You can then add backup locations to Data Grid caches.

2.1. CONFIGURING CLUSTER TRANSPORT FOR CROSS-SITE REPLICATION

Add JGroups RELAY2 to your transport layer so that Data Grid can replicate caches to backup locations.

Procedure

1. Open your Data Grid configuration for editing.
2. Add the RELAY2 protocol to a JGroups stack.
3. Specify the stack name with the **stack** attribute for the transport configuration so the Data Grid cluster uses it.
4. Save and close your Data Grid configuration.

JGroups RELAY2 stacks

The following configuration shows a JGroups RELAY2 stack that:

- Uses the default JGroups UDP stack for inter-cluster transport, which refers to communication between nodes at the local site.
- Uses the default JGroups TCP stack for cross-site replication traffic.
- Names the local site as **LON**.
- Specifies a maximum of 1000 nodes in the cluster that can send cross-site replication requests.
- Specifies the names of all backup locations that participate in cross-site replication.

```
<infinispan>
  <jgroups>
    <stack name="xsite" extends="udp">
      <relay.RELAY2 xmlns="urn:org:jgroups"
                    site="LON"
                    max_site_masters="1000"/>
      <remote-sites default-stack="tcp">
        <remote-site name="LON"/>
        <remote-site name="NYC"/>
      </remote-sites>
    </stack>
  </jgroups>
  <cache-container>
    <transport cluster="${cluster.name}" stack="xsite"/>
  </cache-container>
</infinispan>
```

Additional resources

- JGroups RELAY2 Stacks
- Data Grid configuration schema reference

2.1.1. Custom JGroups RELAY2 stacks

You can add custom JGroups RELAY2 stacks to Data Grid clusters to use different transport properties for cross-site replication. For example, the following configuration uses TCPPING instead of MPING for discovery and extends the default TCP stack:

```
<infinispan>
  <jgroups>
    <stack name="relay-global" extends="tcp">
      <TCPPING initial_hosts="192.0.2.0[7800]"
               stack.combine="REPLACE"
               stack.position="MPING"/>
    </stack>
    <stack name="xsite" extends="udp">
      <relay.RELAY2 site="LON" xmlns="urn:org:jgroups"
                    max_site_masters="10"
                    can_become_site_master="true"/>
      <remote-sites default-stack="relay-global">
        <remote-site name="LON"/>
        <remote-site name="NYC"/>
      </remote-sites>
    </stack>
  </jgroups>
</infinispan>
```

Additional resources

- JGroups RELAY2
- Relaying between multiple sites (RELAY2)

2.2. ADDING BACKUP LOCATIONS TO CACHES

Specify the names of remote sites so Data Grid can replicate data to caches on those clusters.

Procedure

1. Open your Data Grid configuration for editing.
2. Add the **backups** element to your cache configuration.
3. Specify the name of the remote site as the backup location.
   For example, in the **LON** configuration, specify **NYC** as the backup.
4. Repeat the preceding steps on each cluster so that each site is a backup for other sites.
   For example, if you add **LON** as a backup for **NYC** you should also add **NYC** as a backup for **LON**.
5. Save and close your Data Grid configuration.

Backup configuration

The following example shows the "customers" cache configuration for the **LON** cluster:

XML

```xml
<replicated-cache name="customers">
  <backups>
    <backup site="NYC"
            strategy="ASYNC" />
  </backups>
</replicated-cache>
```

JSON

```json
{
  "replicated-cache": {
    "name": "customers",
    "backups": {
      "NYC": {
        "backup" : {
          "strategy" : "ASYNC"
        }
      }
    }
  }
}
```

YAML

```yaml
replicatedCache:
  name: "customers"
  backups:
    NYC:
      backup:
        strategy: "ASYNC"
```

The following example shows the "customers" cache configuration for the **NYC** cluster:

XML

```xml
<distributed-cache name="customers">
  <backups>
    <backup site="LON"
            strategy="ASYNC" />
  </backups>
</distributed-cache>
```

JSON

```json
{
  "distributed-cache": {
      "name": "customers",
      "backups": {
        "LON": {
          "backup": {
            "strategy": "ASYNC"
          }
        }
      }
    }
  }
```

YAML

```
distributedCache:
  name: "customers"
  backups:
    LON:
      backup:
        strategy: "ASYNC"
```

Additional resources

- Data Grid configuration schema reference

2.3. BACKING UP TO CACHES WITH DIFFERENT NAMES

Data Grid replicates data between caches that have the same name by default. If you want Data Grid to replicate between caches with different names, you can explicitly declare the backup for each cache.

Procedure

1. Open your Data Grid configuration for editing.
2. Use **backup-for** or **backupFor** to replicate data from a remote site into a cache with a different name on the local site.
3. Save and close your Data Grid configuration.

Backup for configuration

The following example configures the "eu-customers" cache to receive updates from the "customers" cache on the **LON** cluster:

XML

```
<distributed-cache name="eu-customers">
  <backups>
    <backup site="LON"
            strategy="ASYNC" />
  </backups>
  <backup-for remote-cache="customers"
              remote-site="LON" />
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "name": "eu-customers",
    "backups": {
      "LON": {
        "backup": {
          "strategy": "ASYNC"
        }
      }
    },
    "backup-for" : {
      "remote-cache" : "customers",
      "remote-site" : "LON"
    }
  }
}
```

YAML

```
distributedCache:
  name: "eu-customers"
  backups:
    LON:
      backup:
        strategy: "ASYNC"
  backupFor:
    remoteCache: "customers"
    remoteSite: "LON"
```

2.4. CONFIGURING CROSS-SITE STATE TRANSFER

Change cross-site state transfer settings to optimize performance and specify whether operations happen manually or automatically.

Procedure

1. Open your Data Grid configuration for editing.
2. Configure state transfer operations as appropriate.
    a. Specify the number of entries to include in each state transfer operation with **chunk-size** or **chunkSize**.
    b. Specify the time to wait, in milliseconds, for state transfer operations to complete with **timeout**.
    c. Set the maximum number of attempts for Data Grid to retry failed state transfers with **max-retries** or **maxRetries**.
    d. Specify the time to wait, in milliseconds, between retry attempts with **wait-time** or **waitTime**.

e. Specify if state transfer operations happen automatically or manually with **mode**.

3. Open your Data Grid configuration for editing.

State transfer configuration

XML

```
<distributed-cache name="eu-customers">
  <backups>
    <backup site="LON"
            strategy="ASYNC">
      <state-transfer chunk-size="600"
                      timeout="2400000"
                      max-retries="30"
                      wait-time="2000"
                      mode="AUTO"/>
    </backup>
  </backups>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "name": "eu-customers",
    "backups": {
      "LON": {
        "backup": {
          "strategy": "ASYNC",
          "state-transfer": {
            "chunk-size": "600",
            "timeout": "2400000",
            "max-retries": "30",
            "wait-time": "2000",
            "mode": "AUTO"
          }
        }
      }
    }
  }
}
```

YAML

```
distributedCache:
  name: "eu-customers"
  backups:
    LON:
      backup:
        strategy: "ASYNC"
        stateTransfer:
          chunkSize: "600"
        timeout: "2400000"
        maxRetries: "30"
        waitTime: "2000"
        mode: "AUTO"
```

2.5. CONFIGURING CONFLICT RESOLUTION ALGORITHMS

Configure Data Grid to use a different algorithm to resolve conflicting entries between backup locations.

Procedure

1. Open your Data Grid configuration for editing.
2. Specify one of the Data Grid algorithms or a custom implementation as the merge policy to resolve conflicting entries.
3. Save and close your Data Grid configuration for editing.

Data Grid algorithms

TIP

Find all Data Grid algorithms and their descriptions in the **org.infinispan.xsite.spi.XSiteMergePolicy** enum.

The following example configuration uses the **ALWAYS_REMOVE** algorithm that deletes conflicting entries from both sites:

XML

```
<distributed-cache>
  <backups merge-policy="ALWAYS_REMOVE">
    <backup site="LON" strategy="ASYNC"/>
  </backups>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "backups": {
      "merge-policy": "ALWAYS_REMOVE",
      "LON": {
        "backup": {
          "strategy": "ASYNC"
        }
      }
    }
  }
}
```

YAML

```
distributedCache:
  backups:
    mergePolicy: "ALWAYS_REMOVE"
    LON:
      backup:
        strategy: "ASYNC"
```

Custom conflict resolution algorithms

If you create a custom **XSiteEntryMergePolicy** implementation, you can specify the fully qualified class name as the merge policy.

XML

```
<distributed-cache>
   <backups merge-policy="org.mycompany.MyCustomXSiteEntryMergePolicy">
      <backup site="LON" strategy="ASYNC"/>
   </backups>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "backups": {
      "merge-policy": "org.mycompany.MyCustomXSiteEntryMergePolicy",
      "LON": {
        "backup": {
          "strategy": "ASYNC"
        }
      }
    }
  }
}
```

YAML

```
distributedCache:
  backups:
    mergePolicy: "org.mycompany.MyCustomXSiteEntryMergePolicy"
    LON:
      backup:
        strategy: "ASYNC"
```

Additional resources

- org.infinispan.xsite.spi.XSiteEntryMergePolicy
- org.infinispan.xsite.spi.XSiteMergePolicy
- org.infinispan.xsite.spi.SiteEntry
- Data Grid configuration schema reference

2.6. CLEANING TOMBSTONES FOR ASYNCHRONOUS BACKUPS

With the asynchronous backup strategy Data Grid stores metadata, known as tombstones, when it removes keys. Data Grid periodically runs a task to remove these tombstones and reduce excessive memory usage when backup locations no longer require the metadata. You can configure the frequency for this task by defining a target size for tombstone maps as well as the maximum delay between task runs.

Procedure

1. Open your Data Grid configuration for editing.
2. Specify the number of tombstones to store with the **tombstone-map-size** attribute.
   If the number of tombstones increases beyond this number then Data Grid runs the cleanup task more frequently. Likewise, if the number of tombstones is less than this number then Data Grid does not run the cleanup task as frequently.
3. Add the **max-cleanup-delay** attribute and specify the maximum delay, in milliseconds, between tombstone cleanup tasks.
4. Save the changes to your configuration.

Tombstone cleanup task configuration

XML

```
<distributed-cache>
  <backups tombstone-map-size="512000" max-cleanup-delay="30000">
    <backup site="LON" strategy="ASYNC"/>
  </backups>
</distributed-cache>
```

JSON

```
{
  "distributed-cache": {
    "backups": {
      "tombstone-map-size": 512000,
      "max-cleanup-delay": 30000,
      "LON": {
        "backup": {
          "strategy": "ASYNC"
        }
      }
    }
  }
}
```

YAML

```
distributedCache:
  backups:
    tombstoneMapSize: 512000
    maxCleanupDelay: 30000
    LON:
      backup:
        strategy: "ASYNC"
```

Additional resources

- Data Grid configuration schema reference

2.7. VERIFYING CROSS-SITE VIEWS

When you set up Data Grid to perform cross-site replication, you should check log files to ensure that Data Grid clusters have successfully formed cross-site views.

Procedure

1. Open Data Grid log files with any appropriate editor.
2. Check for **ISPN000439: Received new x-site view** messages.

For example, if a Data Grid cluster in **LON** has formed a cross-site view with a Data Grid cluster in **NYC**, logs include the following messages:

```
INFO  [org.infinispan.XSITE] (jgroups-5,<server-hostname>) ISPN000439: Received new x-site view:
[NYC]
INFO  [org.infinispan.XSITE] (jgroups-7,<server-hostname>) ISPN000439: Received new x-site view:
[LON, NYC]
```

2.8. CONFIGURING HOT ROD CLIENTS FOR CROSS-SITE REPLICATION

Configure Hot Rod clients to use Data Grid clusters at different sites.

hotrod-client.properties

```
# Servers at the active site
infinispan.client.hotrod.server_list = LON_host1:11222,LON_host2:11222,LON_host3:11222

# Servers at the backup site
infinispan.client.hotrod.cluster.NYC =
NYC_hostA:11222,NYC_hostB:11222,NYC_hostC:11222,NYC_hostD:11222
```

ConfigurationBuilder

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.addServers("LON_host1:11222;LON_host2:11222;LON_host3:11222")
       .addCluster("NYC")

.addClusterNodes("NYC_hostA:11222;NYC_hostB:11222;NYC_hostC:11222;NYC_hostD:11222")
```

TIP

Use the following methods to switch Hot Rod clients to the default cluster or to a cluster at a different site:

- **RemoteCacheManager.switchToDefaultCluster()**
- **RemoteCacheManager.switchToCluster(${site.name})**

Additional resources

- org.infinispan.client.hotrod.configuration package description
- org.infinispan.client.hotrod.configuration.ConfigurationBuilder
- org.infinispan.client.hotrod.RemoteCacheManager

CHAPTER 3. PERFORMING CROSS-SITE OPERATIONS WITH THE CLI

Use the Data Grid command line interface (CLI) to connect to Data Grid Server clusters, manage sites, and push state transfer to backup locations.

3.1. BRINGING BACKUP LOCATIONS OFFLINE AND ONLINE

Take backup locations offline manually and bring them back online.

Prerequisites

- Create a CLI connection to Data Grid.

Procedure

1. Check if backup locations are online or offline with the **site status** command:

   ```
   site status --cache=cacheName --site=NYC
   ```

   > **NOTE**
   >
   > **--site** is an optional argument. If not set, the CLI returns all backup locations.

   TIP

   Use the **--all-caches** option to get the backup location status for all caches.

2. Manage backup locations as follows:
   - Bring backup locations online with the **bring-online** command:

     ```
     site bring-online --cache=customers --site=NYC
     ```

   - Take backup locations offline with the **take-offline** command:

     ```
     site take-offline --cache=customers --site=NYC
     ```

TIP

Use the **--all-caches** option to bring a backup location online, or take a backup location offline, for all caches.

For more information and examples, run the **help site** command.

3.2. CONFIGURING CROSS-SITE STATE TRANSFER MODES

You can configure cross-site state transfer operations to happen automatically when Data Grid detects that backup locations come online. Alternatively you can use the default mode, which is to manually perform state transfer.

Prerequisites

- Create a CLI connection to Data Grid.

Procedure

1. Use the **site** command to configure state transfer modes, as in the following examples:
    - Retrieve the current state transfer mode.

      ```
      site state-transfer-mode get --cache=cacheName --site=NYC
      ```

    - Configure automatic state transfer operations for a cache and backup location.

      ```
      site state-transfer-mode set --cache=cacheName --site=NYC --mode=AUTO
      ```

TIP

Run the **help site** command for more information and examples.

3.3. PUSHING STATE TO BACKUP LOCATIONS

Transfer cache state to backup locations.

Prerequisites

- Create a CLI connection to Data Grid.

Procedure

- Use the **site push-site-state** command to push state transfer, as in the following example:

  ```
  site push-site-state --cache=cacheName --site=NYC
  ```

TIP

Use the **--all-caches** option to push state transfer for all caches.

For more information and examples, run the **help site** command.

CHAPTER 4. PERFORMING CROSS-SITE OPERATIONS WITH THE REST API

Data Grid Server provides a REST endpoint that exposes methods for performing cross-site operations.

4.1. GETTING STATUS OF ALL BACKUP LOCATIONS

Retrieve the status of all backup locations with **GET** requests.

```
GET /rest/v2/caches/{cacheName}/x-site/backups/
```

Data Grid responds with the status of each backup location in JSON format, as in the following example:

```
{
  "NYC": {
    "status": "online"
  },
  "LON": {
    "status": "mixed",
    "online": [
      "NodeA"
    ],
    "offline": [
      "NodeB"
    ]
  }
}
```

Table 4.1. Returned Status

Value	Description
online	All nodes in the local cluster have a cross-site view with the backup location.
offline	No nodes in the local cluster have a cross-site view with the backup location.
mixed	Some nodes in the local cluster have a cross-site view with the backup location, other nodes in the local cluster do not have a cross-site view. The response indicates status for each node.

4.2. GETTING STATUS OF SPECIFIC BACKUP LOCATIONS

Retrieve the status of a backup location with **GET** requests.

```
GET /rest/v2/caches/{cacheName}/x-site/backups/{siteName}
```

Data Grid responds with the status of each node in the site in JSON format, as in the following example:

```
{
  "NodeA":"offline",
  "NodeB":"online"
}
```

Table 4.2. Returned Status

Value	Description
online	The node is online.
offline	The node is offline.
failed	Not possible to retrieve status. The remote cache could be shutting down or a network error occurred during the request.

4.3. TAKING BACKUP LOCATIONS OFFLINE

Take backup locations offline with **POST** requests and the **?action=take-offline** parameter.

```
POST /rest/v2/caches/{cacheName}/x-site/backups/{siteName}?action=take-offline
```

4.4. BRINGING BACKUP LOCATIONS ONLINE

Bring backup locations online with the **?action=bring-online** parameter.

```
POST /rest/v2/caches/{cacheName}/x-site/backups/{siteName}?action=bring-online
```

4.5. PUSHING STATE TO BACKUP LOCATIONS

Push cache state to a backup location with the **?action=start-push-state** parameter.

```
POST /rest/v2/caches/{cacheName}/x-site/backups/{siteName}?action=start-push-state
```

4.6. CANCELING STATE TRANSFER

Cancel state transfer operations with the **?action=cancel-push-state** parameter.

```
POST /rest/v2/caches/{cacheName}/x-site/backups/{siteName}?action=cancel-push-state
```

4.7. GETTING STATE TRANSFER STATUS

Retrieve status of state transfer operations with the **?action=push-state-status** parameter.

```
GET /rest/v2/caches/{cacheName}/x-site/backups?action=push-state-status
```

Data Grid responds with the status of state transfer for each backup location in JSON format, as in the following example:

```
{
  "NYC":"CANCELED",
  "LON":"OK"
}
```

Table 4.3. Returned status

Value	Description
SENDING	State transfer to the backup location is in progress.
OK	State transfer completed successfully.
ERROR	An error occurred with state transfer. Check log files.
CANCELLING	State transfer cancellation is in progress.

4.8. CLEARING STATE TRANSFER STATUS

Clear state transfer status for sending sites with the **?action=clear-push-state-status** parameter.

```
POST /rest/v2/caches/{cacheName}/x-site/local?action=clear-push-state-status
```

4.9. MODIFYING TAKE OFFLINE CONDITIONS

Sites go offline if certain conditions are met. Modify the take offline parameters to control when backup locations automatically go offline.

Procedure

1. Check configured take offline parameters with **GET** requests and the **take-offline-config** parameter.

   ```
   GET /rest/v2/caches/{cacheName}/x-site/backups/{siteName}/take-offline-config
   ```

   The Data Grid response includes **after_failures** and **min_wait** fields as follows:

   ```
   {
    "after_failures": 2,
    "min_wait": 1000
   }
   ```

2. Modify take offline parameters in the body of **PUT** requests.

   ```
   PUT /rest/v2/caches/{cacheName}/x-site/backups/{siteName}/take-offline-config
   ```

   If the operation successfully completes, the service returns **204 (No Content)**.

4.10. CANCELING STATE TRANSFER FROM RECEIVING SITES

If the connection between two backup locations breaks, you can cancel state transfer on the site that is receiving the push.

Cancel state transfer from a remote site and keep the current state of the local cache with the **?action=cancel-receive-state** parameter.

```
POST /rest/v2/caches/{cacheName}/x-site/backups/{siteName}?action=cancel-receive-state
```

4.11. GETTING STATUS OF BACKUP LOCATIONS

Retrieve the status of all backup locations from Cache Managers with **GET** requests.

```
GET /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/
```

Data Grid responds with status in JSON format, as in the following example:

```
{
  "SFO-3":{
    "status":"online"
  },
  "NYC-2":{
    "status":"mixed",
    "online":[
      "CACHE_1"
    ],
    "offline":[
      "CACHE_2"
    ],
    "mixed": [
      "CACHE_3"
    ]
  }
}
```

Table 4.4. Returned status

Value	Description
online	All nodes in the local cluster have a cross-site view with the backup location.
offline	No nodes in the local cluster have a cross-site view with the backup location.
mixed	Some nodes in the local cluster have a cross-site view with the backup location, other nodes in the local cluster do not have a cross-site view. The response indicates status for each node.

```
GET /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{site}
```

Returns the status for a single backup location.

4.12. TAKING BACKUP LOCATIONS OFFLINE

Take backup locations offline with the **?action=take-offline** parameter.

```
POST /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=take-offline
```

4.13. BRINGING BACKUP LOCATIONS ONLINE

Bring backup locations online with the **?action=bring-online** parameter.

```
POST /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=bring-
online
```

4.14. RETRIEVING THE STATE TRANSFER MODE

Check the state transfer mode with **GET** requests.

```
GET /rest/v2/caches/{cacheName}/x-site/backups/{site}/state-transfer-mode
```

4.15. SETTING THE STATE TRANSFER MODE

Configure the state transfer mode with the **?action=set** parameter.

```
POST /rest/v2/caches/{cacheName}/x-site/backups/{site}/state-transfer-mode?action=set&mode=
{mode}
```

4.16. STARTING STATE TRANSFER

Push state of all caches to remote sites with the **?action=start-push-state** parameter.

```
POST /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=start-push-
state
```

4.17. CANCELING STATE TRANSFER

Cancel ongoing state transfer operations with the **?action=cancel-push-state** parameter.

```
POST /rest/v2/cache-managers/{cacheManagerName}/x-site/backups/{siteName}?action=cancel-
push-state
```

CHAPTER 5. PERFORMING CROSS-SITE OPERATIONS VIA JMX

Perform cross-site operations such as pushing state transfer and bringing sites online via JMX.

5.1. REGISTERING JMX MBEANS

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

Procedure

1. Open your Data Grid configuration for editing.
2. Add the **jmx** element or object to the cache container and specify **true** as the value for the **enabled** attribute or field.
3. Add the **domain** attribute or field and specify the domain where JMX MBeans are exposed, if required.
4. Save and close your client configuration.

JMX configuration

XML

```
<infinispan>
  <cache-container statistics="true">
    <jmx enabled="true"
         domain="example.com"/>
  </cache-container>
</infinispan>
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "statistics" : "true",
      "jmx" : {
        "enabled" : "true",
        "domain" : "example.com"
      }
    }
  }
}
```

YAML

```
infinispan:
  cacheContainer:
```

```
statistics: "true"
jmx:
  enabled: "true"
  domain: "example.com"
```

5.2. PERFORMING CROSS-SITE OPERATIONS WITH JMX CLIENTS

Perform cross-site operations with JMX clients.

Prerequisites

- Configure Data Grid to register JMX MBeans

Procedure

1. Connect to Data Grid with any JMX client.
2. Invoke operations from the following MBeans:
   - **XSiteAdmin** provides cross-site operations for caches.
   - **GlobalXSiteAdminOperations** provides cross-site operations for Cache Managers.
     For example, to bring sites back online, invoke **bringSiteOnline(siteName)**.

Additional resources

- XSiteAdmin MBean
- GlobalXSiteAdminOperations MBean

5.3. JMX MBEANS FOR CROSS-SITE REPLICATION

Data Grid provides JMX MBeans for cross-site replication that let you gather statistics and perform remote operations.

The **org.infinispan:type=Cache** component provides the following JMX MBeans:

- **XSiteAdmin** exposes cross-site operations that apply to specific cache instances.
- **RpcManager** provides statistics about network requests for cross-site replication.
- **AsyncXSiteStatistics** provides statistics for asynchronous cross-site replication, including queue size and number of conflicts.

The **org.infinispan:type=CacheManager** component includes the following JMX MBean:

- **GlobalXSiteAdminOperations** exposes cross-site operations that apply to all caches in a cache container.

For details about JMX MBeans along with descriptions of available operations and statistics, see the *Data Grid JMX Components* documentation.

Additional resources

- Data Grid JMX Components