Upgrading Infinispan 11.0

Table of Contents

Chapter 1. Infinispan Version Details

Learn about changes in Infinispan versions before you upgrade.

1.1. Upgrading from 10.1 to 11.0

1.1.1. Protobuf Marshalling

Infinispan includes an upgraded version of the ProtoStream API that can affect upgrade from previous Infinispan versions.

In previous versions, the ProtoStream API did not correctly nest message types with the result that the messages were generated as top-level only. For this reason, if you have Protobuf messages in a persistent cache store and upgrade to Infinispan 11.0.4, then you should modify Java classes so that Protobuf annotations are at top-level. This ensures that the nesting in your persisted messages matches the nesting in your Java classes, otherwise data incompatibility issues can occur.

1.1.2. HotRod Client

The `GenericJBossMarshaller` is no longer automatically configured if the `infinispan-jboss-marshalling` module is on the classpath. If jboss-marshalling is required, it's necessary for the aforementioned jar to be on the classpath and for the `org.infinispan.jboss.marshalling.commons.GenericJBossMarshaller` to be explicitly configured when creating the RemoteCacheManager.

1.1.3. Wildfly modules

The Wildfly modules are now deprecated. The `jgroups`, `infinispan` and `endpoint` extensions have been removed and all the components are now in a single `org.infinispan` module.

1.1.4. Cross Site Replication

- The `org.infinispan.xsite.CustomFailurePolicy` interface has been deprecated and it will be replaced by `org.infinispan.configuration.cache.CustomFailurePolicy`.
- Cross Site Replication was disabled for local caches. They are unable to send or receive updates.

1.1.5. Total Order transaction protocol removed

Total Order transaction protocol was deprecated in 10.0 and now it is removed.

1.1.6. OSGi

OSGi support has been deprecated and will be removed in a future release.

1.1.7. Search

Indexing

- The Infinispan Lucene Directory, the `InfinispanIndexManager` and `AffinityIndexManager` index managers, and the Infinispan Directory provider for Hibernate Search were deprecated and are now removed.
- The `auto-config` attribute is deprecated and will be removed in a future version.
- The index mode configuration `index()` is no longer necessary. The system will automatically choose the best way to manage indexing once it is enabled and several previously supported values are no longer supported and will result in a fatal configuration error when used. The following substitutions should be done:
  - `.indexing().index(Index.NONE)` → `indexing().enabled(false)`
  - All the other enum values → `indexing().enabled(true)`

In the XML configuration it is possible to omit `enabled="true"` if the configuration contains others sub-elements. Programmatic and JSON configurations must use it.

It is forbidden to use both the `.indexing().enabled( )` and the deprecated `.indexing().index( )` configuration.

- Indexed types required: starting with version 11 it is mandatory to declare all indexed types in the indexing configuration or else warnings will be logged when the undeclared type is firstly used. This requirement exists solely for indexed caches and applies to both Java classes and protobuf types. Please consider updating your cache configurations in order to avoid these warnings now. Starting with next major such incomplete configurations will be considered invalid and will result in a fatal error at cache startup time.

Querying

- The `SearchManager` has been deprecated and no longer supports Lucene and Hibernate Search native objects:
  - `.getQuery()` methods that take a Lucene Query have been removed. The alternative methods that take an Ickle query from the `org.infinispan.query.Search` entry point should be used instead.
  - `.buildQueryBuilderForClass()`, that allows to build Hibernate Search queries directly has been removed. Ickle queries should be used instead.
  - It is not possible anymore to specify multiple target entity(ies) class(es) when calling `.getQuery()`. The entity will come from the Ickle query string, so multi-entity queries are not supported anymore.
- `CacheQuery` has been deprecated and `org.infinispan.query.dsl.Query` obtained from `Search.getQueryFactory()` should be used instead.
- Instances of `org.infinispan.query.dsl.Query` don't cache query results anymore and allow queries to be re-executed when calling methods such as `list()`.

Entity Mappings

- It is now required to annotate a field that requires sorting with `@SortableField`, both for

embedded and remote queries.

- Several features have been deprecated:
  - Custom bridges, declared with `@FieldBridge` and `@ClassBridge`
  - Analyzer definitions, declared with `@AnalyzerDef` and `@NormalizerDef`
  - Dynamic selection of analyzers based on a discriminator, declared with `@AnalyzerDiscriminator`
  - Index time boosting, declared as an attribute of the `@Field` annotation
  - Definition of a default analyzer, either using the configuration property `hibernate.search.analyzer` or using the `@Analyzer` annotation
  - `indexNullAs`, used as an attribute in the `@Field` declaration
  - The configuration `hibernate.search.index_uninverting_allowed`, that allows regular `@Field` to be sortable by un-inverting them at runtime

1.1.8. ThreadFactoryConfiguration changes

The ThreadGroup configuration setting has been removed and only thread group name is allowed now. This configuration was inconsistent between programmatic and declarative configuration and is now consistent.

1.1.9. Persistence

Single File Store

The `<file-store relative-to=""">` attribute has been removed. This attribute will be ignored in pre 11.0 schemas with only the `path` attribute being taking into account when configuring the stores location.

ClusterLoader

The `ClusterLoader` has been deprecated and will be removed in a future release with no direct replacement.

1.1.10. HotRod

The `LAZY_RETRIEVAL` option utilises the now deprecated `ClusterLoader` and consequently has also been deprecated. It will be removed in a future release with no direct replacement.

1.1.11. Memory configuration changes

- The `BINARY` storage has been deprecated, and will no longer store primitives and String mixed with `byte[]`, but only `byte[]`.
- The child elements `<object>`, `<binary>` and `<off-heap>` are also deprecated. The following config changes should be done:
  - On heap storage:

```
10.1
```

```
<memory>
   <object size="1000000" strategy="REMOVE"/>
</memory>
```

```
11.0
```

```
<memory max-count="1000000" when-full="REMOVE"/>
```

- Binary, on heap storage:

```
10.1
```

```
<cache>
   <memory>
      <binary size="500000000" strategy="EXCEPTION" eviction="MEMORY"/>
   </memory>
</cache>
```

```
11.0
```

```
<cache>
   <!-- Or any other binary format -->
   <encoding media-type="application/x-protostream"/>
   <!-- Sizes are human-readable, e.g.: "1 GB", "0.5TB" -->
   <memory max-size="500 MB" when-full="EXCEPTION"/>
</cache>
```

- Off-heap:

```
10.1
```

```
<cache>
   <memory>
      <off-heap size="10000000" eviction="COUNT"/>
   </memory>
</cache>
```

```
11.0
```

```
<memory storage="OFF_HEAP" max-count="10000000"/>
```

- Due to the changes above, cache configurations serialized to XML or JSON (for example, when using REST) will always be in the new format.

1.1.12. Encoding in Server Caches

- Server caches should configure the MediaType for keys and values, or else a WARN will be logged. Usage of operations that require data conversion or indexing will not be supported for caches without encoding configuration in a future version. These operations include search, task execution, remote filters/converter/listeners, REST/Hot Rod reading/writing in different data formats

1.1.13. Security

Server security

The server is now secure by default. Use the `user-tool` to add users or remove the security realm attribute from the endpoint to allow anonymous connections.

Cache authorization roles

If you do not explicitly specify roles when enabling cache authorization, all roles declared in the global configuration apply.

1.1.14. REST

- REST API calls that have the extra URL parameter `?action` to perform operations with side effects now also support the POST method (returning 200 when the response has content or 204 otherwise). Support for using `GET` method on those calls will be removed in a future version.

1.2. Upgrading from 10.0 to 10.1 and 10.0 to 11.0

1.2.1. Maximum Idle Timeouts with Clustered Cache Modes

Maximum idle expiration has been changed to improve data consistency with clustered cache modes when Infinispan nodes fail.

- `Cache.get()` calls do not return until the touch commands complete. This synchronous behavior increases latency of client requests and reduces performance.
- Maximum idle expiration, `max-idle`, does not currently work with entries stored in off-heap memory.
- Likewise, `max-idle` does not work if caches use cache stores as a persistence layer.

See Maximum Idle Expiration for complete details.

1.3. Upgrading from 10.0 to 10.1

1.3.1. REST Store

The following configurations were removed from the REST store: `append-cache-name-to-path` and `path`.

To specify the remote server endpoint path, a single configuration `cache-name` should be used.

1.3.2. Infinispan Lucene Directory is deprecated

The Infinispan Lucene directory is now deprecated and will be removed in a future release. Consequently, the Infinispan Directory provider for Hibernate Search will also be discontinued, with no replacement.

Both IndexManagers that rely on the Lucene Directory are also deprecated, the InfinispanIndexManager and the AffinityIndexManager. Users are encouraged to reconfigure their indexes as non-shared, using the Near Real Time IndexManager, with file system storage:

```
<distributed-cache name="default">
    <indexing index="PRIMARY_OWNER">
        <property name="default.indexmanager">near-real-time</property>
        <property name="default.indexBase">
/opt/infinispan/server/data/indexes</property>
    </indexing>
</distributed-cache>
```

Queries need to be adjusted to use the `BROADCAST` runtime option.

1.3.3. Security role mappers and audit loggers

The security role mapper implementations have been moved from the `org.infinispan.security.impl` package to the `org.infinispan.security.mappers` package:

- `org.infinispan.security.impl.CommonNameRoleMapper` ⇒ `org.infinispan.security.mappers.CommonNameRoleMapper`
- `org.infinispan.security.impl.ClusterRoleMapper` ⇒

`org.infinispan.security.mappers.ClusterRoleMapper`

- `org.infinispan.security.impl.IdentityRoleMapper` ⇒ `org.infinispan.security.mappers.IdentityRoleMapper`

The security audit logger implementations have been moved from the `org.infinispan.security.impl` package to the `org.infinispan.security.audit` package:

- `org.infinispan.security.impl.LoggingAuditLogger` ⇒ `org.infinispan.security.audit.LoggingAuditLogger`
- `org.infinispan.security.impl.NullAuditLogger` ⇒ `org.infinispan.security.audit.NullAuditLogger`

1.3.4. Memcached protocol server is deprecated

The Memcached protocol server is now deprecated and will be removed in a future release. This is being done because Infinispan only implements the very dated text-only protocol instead of the binary protocol which means no security (authentication / encryption), no support for some new Memcached features and no integration with Infinispan features like single-port. If someone in the community wishes to implement the binary protocol, we would revert the decision.

1.3.5. Hot Rod client default mechanism changed to SCRAM-SHA-512

The default Hot Rod client authentication mechanism has been changed from `DIGEST-MD5` to `SCRAM-SHA-512`. If you are using `property` user realms, you must make sure you are using `plain-text` storage.

1.3.6. Transactions

The Map implementation `EntryVersionsMap` has been removed and replaced with a `Map<Object, IncrementalEntryVersion>`. If the old `EntryVersionsMap#merge` logic is required, it can be replaced with `org.infinispan.transaction.impl.WriteSkewHelper#mergeEntryVersions`, however users should not rely on this code as it could be removed in the future without notice,

1.4. Upgrading from 9.4 to 10.0

1.4.1. Marshalling

The internal marshalling capabilities of Infinispan have undergone a significant refactoring in 10.0. The marshalling of internal Infinispan objects and user objects are now truly isolated. This means that it's now possible to configure Marshaller implementations in embedded mode or on the server, without having to handle the marshalling of Infinispan internal classes. Consequently, it's possible to easily change the marshaller implementation, in a similar manner to how users of the HotRod client are accustomed.

As a consequence of the above changes, the default marshaller used for marshalling user types is no longer based upon JBoss Marshalling. Instead we now utilise the ProtoStream library to store user types in the language agnostic Protocol Buffers format. It is still possible to utilise the old default, JBossUserMarshaller, however it's necessary to add the `org.infinispan:infinispan-jboss-marshalling` artifact to your application's classpath.

Externalizer Deprecations

The following interfaces/annotations have been deprecated as a consequence of the marshalling refactoring:

- Externalizer,
- AdvancedExternalizer
- SerializeWith

For cluster communication any configured Externalizer's are still utilised to marshall objects, however they are ignored when persisting data to cache stores unless the JBossUserMarshaller is explicitly configured via the global SerializationConfiguration.

It's highly recommended to migrate from the old Externalizer and JBoss marshalling approach to the new ProtoStream based marshalling, as the interfaces listed above and the JBossUserMarshaller implementation will be removed in future versions.

Store Migration

Unfortunately, the extensive marshalling changes mean that the binary format used by Infinispan stores in `9.4.x` is no longer compatible with `10.0.x`. Therefore, it's necessary for any existing stores to be migrated to the new format via the StoreMigrator tool.

Whilst we regret that 9.4.x stores are no longer binary compatible, these extensive changes should ensure binary compatibility across future major versions.

Store Defaults

Stores now default to being segmented if the property is not configured. Some stores do not support being segmented, which will result in a configuration exception being thrown at startup. The moving forward position is to use segmented stores when possible to increase cache wide performance and reduce memory requirements for various operations including state transfer.

The file based stores (SingleFileStore and SoftIndexFileStore) both support being segmented, but their current implementation requires opening file descriptors based on how many segments there are. This may cause issues in some configurations and users should be aware. Infinispan will print a single WARN message when such a configuration is found.

1.4.2. CacheContainterAdmin

Caches created through the CacheContainerAdmin API will now be `PERMANENT` by default. Use the `VOLATILE` flag to obtain the previous behaviour.

1.4.3. Hot Rod 3.0

Older versions of the Hot Rod protocol treated expiration values greater than the number of milliseconds in 30 days as Unix time. Starting with Hot Rod 3.0 this adjustment no longer happens and expiration is taken literally.

1.4.4. Total Order transaction protocol is deprecated

Total Order transaction protocol is going to be removed in a future release. Use the default protocol (2PC).

1.4.5. Removed the infinispan.server.hotrod.workerThreads system property

The `infinispan.server.hotrod.workerThreads` property was introduced as a hack to work around the fact that the configuration did not expose it. The property has been removed and endpoint worker threads must now be exclusively configured using the `worker-threads` attribute.

1.4.6. Removed AtomicMap and FineGrainedAtomicMap

AtomicMapLookup, AtomicMap and FineGrainedAtomicMap have been removed. Please see FunctionalMaps or Cache#Merge for similar functionality.

1.4.7. Removed Delta and DeltaAware

The previously deprecated Delta and DeltaAware interfaces have been removed.

1.4.8. Removed compatibility mode

The previously deprecated Compatibility Mode has been removed.

1.4.9. Removed the implicit default cache

The default cache must now be named explicitly via the GlobalConfigurationBuilder#defaultCacheName() method.

1.4.10. Removed DistributedExecutor

The previously deprecated DistributedExecutor is now removed. References should be updated to use ClusterExecutor.

1.4.11. Removed the Tree module

TreeCache has been unsupported for a long time and was only intended as a quick stopgap for JBossCache users. The module has now been removed completely.

1.4.12. The JDBC PooledConnectionFactory now utilises Agroal

Previously the JDBC PooledConnectionFactory provided c3p0 and HikariCP based connection pools. From 10.0 we only provide a PooledConnectionFactory based upon the Agroal project. This means that it is no longer possible to utilise `c3p0.properties` and `hikari.properties` files to configure the pool, instead an agroal compatiblet properties file can be provided.

1.4.13. XML configuration changes

Several configuration elements and attributes that were deprecated since 9.0 have been removed:

- `<eviction>` - replaced with `memory`
- `<versioning>` - automatically enabled
- `<data-container>` - no longer customizable
- `deadlock-detection-spin` - always disabled
- `write-skew` - enabled automatically

The xsite state transfer chunk size (`<backup><state-transfer chunk-size="X"/></backup>`) can no longer be `>= 0`, same as the regular state transfer chunk size. Previously a value <= 0 would transfer the entire cache in a single batch, which is almost always a bad idea.

1.4.14. RemoteCache Changes

Marshalling Changes

The default marshaller is no longer GenericJbossMarshaller. We now utilise the ProtoStream library as the default. If Java Serialization is required by clients, we strongly recommend utilising the link:JavaSerializationMarshaller instead. However if the GenericJbossMarshaller must be used, it's necessary to add the `org.infinispan:infinispan-jboss-marshalling` artifact to your client's classpath and for the `GenericJbossMarshaller` to be configured as the marshaller.

The getBulk methods have been removed

The getBulk method is an expensive method as it requires holding all keys in memory at once and requires a possibly very single result to populate it. The new retrieveEntries, entrySet, keySet and values methods handle this in a much more efficient way. Therefore the getBulk methods have been removed in favor of them.

1.4.15. Persistence changes

- File-based cache stores (SingleFileStore, SoftIndexFileStore, RocksDBStore) filesystem layout has been normalized so that they will use the `GlobalStateConfiguration` persistent location as a default location. Additionally, all stores will now use the cache name as part of the data file/directory naming allowing multiple stores to avoid conflicts and ambiguity.
- The CLI loader (`infinispan-persistence-cli`) has been removed.
- The LevelDB store (`infinispan-cachestore-leveldb`) has been removed. Use the RocksDB store instead, as it is fully backwards compatible.
- The deprecated `singleton` store configuration option and the wrapper class `SingletonCacheWriter` have been removed.

  Using `shared=true` is enough, as only the primary owner of each key will write to a shared store.

1.4.16. Client/Server changes

- The Hot Rod client and server only support protocol versions 2.0 and higher. Support for Hot Rod versions 1.0 to 1.3 has been dropped.

1.4.17. SKIP_LISTENER_NOTIFICATION flag

`SKIP_LISTENER_NOTIFICATION` notification flag has been added in the hotrod client. This flag only works when the client and the server version is 9.4.15 or higher. Spring Session integration uses this flag when a session id has changed. If you are using Spring Session with Infinispan 9.4, consider upgrading the client and the server.

1.4.18. performAsync header removed from REST

The `performAsync` header was removed from the REST server. Clients that want to perform async operations with the REST server should manage the request and response on their side to avoid blocking.

1.4.19. REST status code change

REST operations that don't return resources and are used with `PUT`, `POST` and `DELETE` methods now return status `204` (No content) instead of `200`.

1.4.20. Default JGroups stacks in the XML configuration

With the introduction of inline XML JGroups stacks in the configuration, two default stacks are always enabled: `udp` and `tcp`. If you are declaring your own stacks with the same names, an exception reporting the conflict will be thrown. Simply rename your own configurations to avoid the conflict.

1.4.21. JGroups S3_PING replaced with NATIVE_S3_PING

Because of changes in AWS's access policy regarding signatures, S3_PING will not work in newer regions and will stop working in older regions too. For this reason, you should migrate to using NATIVE_S3_PING instead.

1.4.22. Cache and Cache Manager Listeners can now be configured to be non blocking

Listeners in the past that were sync, always ran in the thread that caused the event. We now allow a Listener method to be non-blocking in that it will still fire in the original thread, under the assumption that it will return immediately. Please read the Listener Javadoc for information and examples on this.

1.4.23. Distributed Streams operations no longer support null values

Distributed Streams has parts rewritten to utilize non blocking reactive streams based operations. As such null values are not supported as values from operations as per the reactive streams spec. Please utilize other means to denote a null value.

1.4.24. Removed the infinispan-cloud module

The infinispan-cloud module has been removed and the `kubernetes`, `ec2`, `google` and `azure` default configurations have been included in `infinispan-core` and can be referenced as default named JGroups configurations.

1.4.25. Removed experimental flag GUARANTEED_DELIVERY

Almost as soon as GUARANTEED_DELIVERY was added, UNICAST3 and NAKACK2.resend_last_seqno removed the need for it. It was always documented as experimental, so we removed it without deprecation and we also removed the RSVP protocol from the default JGroups stacks.

1.4.26. Cache Health

The possible statuses of the cache health are now HEALTHY, HEALTHY_REBALANCING and DEGRADED to better reflect the fact that `rebalancing` doesn't mean a cluster is unhealthy.

1.4.27. Multi-tenancy

When using multi-tenancy in the WildFly based server, it's necessary to specify the `content-path` for each of the REST connectors, to match the `prefix` element under `multi-tenancy\rest\prefix`.

1.4.28. OffHeap Automatic Resizing

Off Heap memory containers now will dynamically resize based on number of entries in the container. Due to this the address count configuration value is now deprecated for APIs and has been removed from the xml parser.

1.4.29. Deprecated methods from DataContainer removed

The deprecated methods keySet, values, entrySet and executeTask has been removed.

1.5. Upgrading from 9.3 to 9.4

1.5.1. Client/Server changes

Compatibility mode deprecation

Compatibility mode has been deprecated and will be removed in the next Infinispan version.

To use a cache from multiple endpoints, it is recommended to store data in binary format and to configure the MediaType for keys and values.

If storing data as unmarshalled objects is still desired, the equivalent of compatibility mode is to configure keys and values to store object content:

```
<encoding>
   <key media-type="application/x-java-object"/>
   <value media-type="application/x-java-object"/>
</encoding>
```

Memcached storage

For better interoperability between endpoints, the Memcached server no longer stores keys as `java.lang.String`, but as UTF-8 `byte[]`.

If using memcached, it's recommended to run a rolling upgrade from 9.3 to store data in the new format, or reload the data in the cache.

Scripts Response

Distributed scripts with text-based data type no longer return `null` when the result from each server is null. The response is now a JSON array with each individual result, e.g. `"[null, null]"`

WebSocket endpoint removal

The WebSocket endpoint has been unmaintained for several years. It has been removed.

Hot Rod client connection pool properties

Since the Hot Rod client was overhauled in 9.2, the way the connection pool configuration is handled has changed. Infinispan 9.4 introduces a new naming scheme for the connection pool properties which deprecates the old *commons-pool* names. For a complete reference of the available configuration options for the properties file please refer to remote client configuration javadoc.

Server thread pools

The threads that handle the child Netty event loops have been renamed from *-ServerWorker to *-ServerIO

1.5.2. Persistence Changes

Shared and Passivation

A store cannot be configured as both shared and having passivation enabled. Doing so can cause data inconsistencies as there is no way to synchronize data between all the various nodes. As such this configuration will now cause a startup exception. Please update your configuration as appropriate.

1.5.3. Query changes

AffinityIndexManager

The default number of shards is down to 4, it was previously equals to the number of segments in

the cache.

1.6. Upgrading from 9.2 to 9.3

1.6.1. AdvancedCacheLoader changes

The AdvancedCacheLoader SPI has been enhanced to provide an alternative method to process and instead allows reactive streams based publishKeys and publishEntries methods which provide benefits in performance, threading and ease of use. Note this change will only affect you if you wish take advantage of it in any custom CacheLoaders you may have implemented.

1.6.2. Partition Handling Configuration

In 9.3 the default MergePolicy is now MergePolicy.NONE, opposed to MergePolicy.PREFERRED_ALWAYS.

1.6.3. Stat Changes

We have reverted the stat changes introduced in 9.1, so average values for read, write and removals are once again returned as milliseconds.

1.6.4. Event log changes

Several new event log messages have been added, and one message has been removed (ISPN100013).

1.6.5. Max Idle Expiration Changes

The max idle entry expiration information is sent between owners in the cluster. However when an entry expires via max idle on a given node, this was not replicated (only removing it locally). Max idle has been enhanced to now expire an entry across the entire cluster, instead of per node. This includes ensuring that max idle expiration is applied across all owners (meaning if another node has accessed the entry within the given time it will prevent that entry from expiring on other nodes that didn't have an access).

Max idle in a transactional clustered cache does not remove expired entries on access (although it will not be returned). These entries are only removed via the expiration reaper.

Iteration in a clustered cache will still show entries that are expired via maxIdle to ensure good performance, but could be removed at any point due to expiration reaper.

1.6.6. WildFly Modules

The Infinispan WildFly modules are now located in the `system/add-ons/{moduleprefix}` dir as per the WildFly module conventions.

1.6.7. Deserialization Whitelist

Deserialization of content sent by clients to the server are no longer allowed by default. This applies to JSON, XML, and marshalled byte[] that, depending on the cache configuration, will cause the server to convert it to Java Objects either to store it or to perform any operation that cannot be done on a byte[] directly.

The deserialization needs to be enabled using system properties, ether by class name or regular expressions:

```
// Comma separated list of fully qualified class names
-Dinfinispan.deserialization.whitelist.classes=java.time.Instant,com.myclass.Entity

// Regex expression
-Dinfinispan.deserialization.whitelist.regexps=.*
```

1.7. Upgrading from 9.0 to 9.1

1.7.1. Kubernetes Ping changes

The latest version of Kubernetes Ping uses unified environmental variables for both Kubernetes and OpenShift. Some of them were shortened for example `OPENSHIFT_KUBE_PING_NAMESPACE` was changed to `KUBERNETES_NAMESPACE`. Please refer to Kubernetes Ping documentation.

1.7.2. Stat Changes

Average values for read, write and removals are now returned in Nanoseconds, opposed to Milliseconds.

1.7.3. (FineGrained)AtomicMap reimplemented

Infinispan now contains a new implementation of both `AtomicMap` and `FineGrainedAtomicMap`, but the semantics has been preserved. The new implementation does not use `DeltaAware` interface but the Functional API instead.

There are no changes needed for `AtomicMap`, but it now supports non-transactional use case as well.

`FineGrainedAtomicMap` now uses the Grouping API and therefore you need to enable groups in configuration. Also it holds entries as regular cache entries, plus one cache entry for cached key set (the map itself). Therefore the cache size or iteration/streaming results may differ. Note that fine grained atomic maps are still supported on transactional caches only.

1.7.4. RemoteCache keySet/entrySet/values

RemoteCache now implements all of the collection backed methods from `Map` interface. Previously `keySet` was implemented, however it was a deep copy. This has now changed and it is a backing set. That is that the set retrieves the updated values on each invocation or updates to the backing remote cache for writes. The entrySet and values methods are also now supported as backing

variants as well.

If you wish to have a copy like was provided before it is recommended to copy the contents into a in memory local set such as

```
Set<K> keysCopy = remoteCache.keySet().stream().collect(Collectors.toSet());
```

1.7.5. DeltaAware deprecated

Interfaces `DeltaAware`, `Delta` and `CopyableDeltaAware` have been deprecated. Method `AdvancedCache.applyDelta()` has been deprecated and the implementation does not allow custom set of locked keys. `ApplyDeltaCommand` and its uses in interceptor stack are deprecated.

Any partial updates to an entry should be replaced using the Functional API.

1.7.6. Infinispan Query Configuration

The configuration property `directory_provider` now accepts a new value `local-heap`. This value replaces the now deprecated `ram`, and as its predecessor will cause the index to be stored in a `org.apache.lucene.store.RAMDirectory`.

The configuration value `ram` is still accepted and will have the same effect, but failing to replace `ram` with `local-heap` will cause a warning to be logged. We suggest to perform this replacement, as the `ram` value will no longer be recognised by Infinispan in a future version.

This change was made as the team believes the `local-heap` name better expresses the storage model, especially as this storage method will not allow real-time replication of the index across multiple nodes. This index storage option is mostly useful for single node integration testing of the query functionality.

1.7.7. Store Batch Size Changes

TableManipulation::batchSize and JpaStoreConfiguration::batchSize have been deprecated and replaced by the higher level AbstractStoreConfiguration::maxBatchSize.

1.7.8. Partition Handling changes

In Infinispan 9.1 partition handling has been improved to allow for automatic conflict resolution on partition merges. Consequently, PartitionHandlingConfiguration::enabled has been deprecated in favour of PartitionHandlingConfiguration::whenSplit. Configuring whenSplit to the DENY_READ_WRITES strategy is equivalent to setting enabled to true, whilst specifying ALLOW_READ_WRITES is equivalent to disabling partition handling (default).

Furthermore, during a partition merge with ALLOW_READ_WRITES, the default EntryMergePolicy is MergePolicies.PREFERRED_ALWAYS which provides a deterministic way of tie-breaking CacheEntry conflicts. If you require the old behaviour, simply set the merge-policy to null.

1.8. Upgrading from 8.x to 9.0

1.8.1. Default transaction mode changed

The default configuration for transactional caches changed from `READ_COMMITTED` and `OPTIMISTIC` locking to `REPEATABLE_READ` and `OPTIMISTIC` locking with `write-skew` enabled.

Also, using the `REPEATABLE_READ` isolation level and `OPTIMISTIC` locking without `write-skew` enabled is no longer allowed. To help with the upgrade, `write-skew` will be automatically enabled in this case.

The following configuration has been deprecated:

- `write-skew`: as said, it is automatically enabled.
- `<versioning>` and its attributes. It is automatically enabled and configured when needed.

1.8.2. Removed eagerLocking and eagerLockingSingleNode configuration settings

Both were deprecated since version 5.1. `eagerLocking(true)` can be replaced with `lockingMode(LockingMode.PESSIMISTIC)`, and `eagerLockingSingleNode()` does not need a replacement because it was a no-op.

1.8.3. Removed async transaction support

Asynchronous mode is no longer supported in transactional caches and it will automatically use the synchronous cache mode. In addition, the second phase of a transaction commit is done synchronously. The following methods (and related) are deprecated:

- `TransactionConfigurationBuilder.syncCommitPhase(boolean)`
- `TransactionConfigurationBuilder.syncRollbackPhase(boolean)`

1.8.4. Deprecated all the dummy related transaction classes.

The following classes have been deprecated and they will be removed in the future:

- `DummyBaseTransactionManager`: replaced by `EmbeddedBasedTransactionManager`;
- `DummyNoXaXid` and `DummyXid`: replaced by `EmbeddedXid`;
- `DummyTransaction`: replaced by `EmbeddedTransaction`;
- `DummyTransactionManager`: replaced by `EmbeddedTransactionManager`;
- `DummyTransactionManagerLookup` and `RecoveryDummyTransactionManagerLookup`: replaced by `EmbeddedTransactionManagerLookup`;
- `DummyUserTransaction`: replaced by `EmbeddedUserTransaction`;

1.8.5. Clustering configuration changes

The `mode` attribute in the XML declaration of clustered caches is no longer mandatory. It defaults to SYNC.

1.8.6. Default Cache changes

Up to Infinispan 8.x, the default cache always implicitly existed, even if not declared in the XML configuration. Additionally, the default cache configuration affected all other cache configurations, acting as some kind of base template. Since 9.0, the default cache only exists if it has been explicitly configured. Additionally, even if it has been specified, it will never act as base template for other caches.

1.8.7. Marshalling Enhancements and Store Compatibility

Internally Infinispan 9.x has introduced many improvements to its marshalling codebase in order to improve performance and allow for greater flexibility. Consequently, data marshalled and persisted by Infinispan 8.x is no longer compatible with Infinispan 9.0. To aid you in migrating your existing stores to 9.0, we have provided a Store Migrator, however at present this only allows the migration of JDBC stores.

1.8.8. New Cloud module for library mode

In Infinispan 8.x, cloud related configuration were added to `infinispan-core` module. Since 9.0 they were moved to `infinispan-cloud` module.

1.8.9. Entry Retriever is now removed

The entry retriever feature has been removed. Please update to use the new Streams feature detailed in the User Guide. The `org.infinispan.filter.CacheFilters` class can be used to convert `KeyValueFilter` and `Converter` instances into proper Stream operations that are able to be marshalled.

1.8.10. Map / Reduce is now removed

Map reduce has been removed in favor of the new Streams feature which should provide more features and performance. There are no bridge classes to convert to the new streams and all references must be rewritten.

1.8.11. Spring 4 support is now removed

Spring 4 is no longer supported.

1.8.12. Function classes have moved packages

The class `SerializableSupplier` has moved from the `org.infinispan.stream` package to the `org.infinispan.util.function` package.

The class `CloseableSupplier` has moved from the `org.infinispan.util` package to the `org.infinispan.util.function` package.

The classes `TriConsumer`, `CloseableSupplier`, `SerializableRunnable`, `SerializableFunction` & `SerializableCallable` have all been moved from the `org.infinispan.util` package to the `org.infinispan.util.function` package.

1.8.13. SegmentCompletionListener interface has moved

The interface `SegmentCompletionListener` has moved from the interface `org.infinispan.CacheStream` to the new `org.infinispan.BaseCacheStream`.

1.8.14. Spring module dependency changes

All Infinispan, Spring and Logger dependencies are now in the `provided` scope. One can decide whether to use small jars or uber jars but they need to be added to the classpath of the application. It also gives one freedom in choosing Spring (or Spring Boot) version.

Here is an example:

```
<dependencies>
   <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-embedded</artifactId>
   </dependency>
   <dependency>
      <groupId>org.infinispan</groupId>
      <artifactId>infinispan-spring5-embedded</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
   </dependency>
   <dependency>
      <groupId>org.springframework.session</groupId>
      <artifactId>spring-session</artifactId>
   </dependency>
</dependencies>
```

Additionally there is no Logger implementation specified (since this may vary depending on use case).

1.8.15. Total order executor is now removed

The total order protocol now uses the `remote-command-executor`. The attribute `total-order-executor` in `<container>` tag is removed.

1.8.16. HikariCP is now the default implementation for JDBC PooledConnectionFactory

HikariCP offers superior performance to c3p0 and is now the default implementation. Additional properties for HikariCP can be provided by placing a `hikari.properties` file on the classpath or by specifying the path to the file via `PooledConnectionFactoryConfiguration.propertyFile` or `properties-file` in the connection pool's xml config. N.B. a properties file specified explicitly in the configuration is loaded instead of the `hikari.properties` file on the class path and Connection pool characteristics which are explicitly set in PooledConnectionFactoryConfiguration always override

the values loaded from a properties file.

Support for c3p0 has been deprecated and will be removed in a future release. Users can force c3p0 to be utilised as before by providing the system property `-Dinfinispan.jdbc.c3p0.force=true`.

1.8.17. RocksDB in place of LevelDB

The LevelDB cache store was replaced with a RocksDB. RocksDB is a fork of LevelDB which provides superior performance in high concurrency scenarios. The new cache store can parse old LevelDB configurations but will always use the RocksDB implementation.

1.8.18. JDBC Mixed and Binary stores removed

The JDBC Mixed and Binary stores have been removed due to the poor performance associated with storing entries in buckets. Storing entries in buckets is non-optimal as each read/write to the store requires an existing bucket for a given hash to be retrieved, deserialised, updated, serialised and then re-inserted back into the db. If you were previously using one of the removed stores, we have provided a migrator tool to assist in migrating data from an existing binary table to a JDBC string based store.

1.8.19. @Store Annotation Introduced

A new annotation, `@Store`, has been added for persistence stores. This allows a store's properties to be explicitly defined and validated against the provided store configuration. Existing stores should be updated to use this annotation and the store's configuration class should also declare the `@ConfigurationFor` annotation. If neither of these annotations are present on the store or configuration class, then a your store will continue to function as before, albeit with a warning that additional store validation cannot be completed.

1.8.20. Server authentication changes

The no-anonymous policy is now automatically enabled for Hot Rod authentication unless explicitly specified.

1.8.21. Package org.infinispan.util.concurrent.jdk8backported has been removed

Moved classes

Classes regarding EntrySizeCalculator have now been moved down to the org.infinispan.util package.

Removed classes

The *ConcurrentHashMapV8 classes and their supporting classes have all been removed. The CollectionFactory#makeBoundedConcurrentMap method should be used if you desire to have a bounded ConcurrentMap.

1.8.22. Store as Binary is deprecated

Store as Binary configuration is now deprecated and will be removed in a future release. This is replaced by the new memory configuration.

1.8.23. DataContainer collection methods are deprecated

The keySet, entrySet and values methods on DataContainer have been deprecated. These behavior of these methods are very inconsistent and will be removed later. It is recommended to update references to use iterator or iteratorIncludingExpired methods intead.

1.9. Upgrading from 8.1 to 8.2

1.9.1. Entry Retriever is deprecated

Entry Retriever is now deprecated and will be removed in Infinispan 9. This is replaced by the new Streams feature.

1.9.2. Map / Reduce is deprecated

Map reduce is now deprecated and will be removed in Infinispan 9. This is replaced by the new Streams feature.

1.10. Upgrading from 8.x to 8.1

1.10.1. Packaging changes

CDI module split

CDI module (GroupId:ArtifactId `org.infinispan:infinispan-cdi`) has been split into `org.infinispan:infinispan-cdi-embedded` and `org.infinispan:infinispan-cdi-remote`. Please make sure that you use proper artifact.

Spring module split

Spring module (GroupId:ArtifactId `org.infinispan:infinispan-spring5`) has been split into `org.infinispan:infinispan-spring5-embedded` and `org.infinispan:infinispan-spring5-remote`. Please make sure that you use proper artifact.

1.10.2. Spring 3 support is deprecated

Spring 3 support (GroupId:ArtifactId `org.infinispan:infinispan-spring`) is deprecated. Please consider migrating into Spring 4 support.

1.11. Upgrading from 7.x to 8.0

1.11.1. Configuration changes

Removal of Async Marshalling

Async marshalling has been entirely dropped since it was never reliable enough. The "async-marshalling" attribute has been removed from the 8.0 XML schema and will be ignored when parsing 7.x configuration files. The programmatic configuration methods related to asyncMarshalling/syncMarshalling are now deprecated and have no effect aside from producing a WARN message in the logs.

Reenabling of isolation level configurations in server

Because of the inability to configure write skew in the server, the isolation level attribute was ignored and defaulted to READ_COMMITTED. Now, when enabling REPEATABLE_READ together with optimistic locking, write skew is enabled by default in local and synchronous configurations.

Subsystem renaming in server

In order to avoid conflict and confusion with the similar subsystems in WildFly, we have renamed the following subsystems in server: * infinispan → datagrid-infinispan * jgroups → datagrid-jgroups * endpoint → datagrid-infinispan-endpoint

Server domain mode

We no longer support the use of standalone mode for running clusters of servers. Domain mode (bin/domain.sh) should be used instead.

1.12. Upgrading from 6.0 to 7.0

1.12.1. API Changes

Cache Loader

To be more inline with JCache and java.util.collections interfaces we have changed the first argument type for the CacheLoader.load & CacheLoader.contains methods to be Object from type K.

Cache Writer

To be more inline with JCache and java.util.collections interfaces we have changed the first argument type for the CacheWriter.delete method to be Object from type K.

Filters

Over time Infinispan added 2 interfaces with identical names and almost identical methods. The org.infinispan.notifications.KeyFilter and org.infinispan.persistence.spi.AdvancedCacheLoader$KeyFilter interfaces.

Both of these interfaces are used for the sole purpose of filtering an entry by it’s given key. Infinispan 7.0 has also introduced the KeyValueFilter which is similar to both but also can filter on the entries value and/or metadata.

As such all of these classes have been moved into a new package org.infinispan.filter and all of their related helper classes.

The new org.infinispan.filter.KeyFilter interface has replaced both of the previous interfaces and all previous references use the new interface.

1.12.2. Declarative configuration

The XML schema for the embedded configuration has changed to more closely follow the server configuration. Use the `config-converter.sh` or `config-converter.bat` scripts to convert an Infinispan 6.0 to the current format.

1.13. Upgrading from 5.3 to 6.0

1.13.1. Declarative configuration

In order to use all of the latest features, make sure you change the namespace declaration at the top of your XML configuration files as follows:

```
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"urn:infinispan:config:6.0 https://infinispan.org/schemas/infinispan-config-6.0.xsd"
xmlns="urn:infinispan:config:6.0">
   ...
</infinispan>
```

1.13.2. Deprecated API removal

- Class `org.infinispan.persistence.remote.wrapperEntryWrapper`.
- Method `ObjectOutput startObjectOutput(OutputStream os, boolean isReentrant)` from class `org.infinispan.commons.marshall.StreamingMarshaller`.
- Method `CacheEntry getCacheEntry(Object key, EnumSet<Flag> explicitFlags, ClassLoader explicitClassLoader)` from class `org.infinispan.AdvancedCache`. Please use instead: `AdvanceCache.withFlags(Flag… flags).with(ClassLoader classLoader).getCacheEntry(K key)`.
- Method `AtomicMap<K, V> getAtomicMap(Cache<MK, ?> cache, MK key, FlagContainer flagContainer)` from class `org.infinispan.atomic.AtomicMapLookup`. Please use instead `AtomicMapLookup.getAtomicMap(cache.getAdvancedCache().withFlags(Flag… flags), MK key)`.
- Package `org.infinispan.config` (and all methods involving the old configuration classes). All methods removed has an overloaded method which receives the new configuration classes as parameters.

  

  This only affects the programmatic configuration.

- Class `org.infinispan.context.FlagContainer`.
- Method `boolean isLocal(Object key)` from class `org.infinispan.distribution.DistributionManager`. Please use instead

`DistributionManager.getLocality(Object key)`.

- JMX operation `void setStatisticsEnabled(boolean enabled)` from class `org.infinispan.interceptors.TxInterceptor` Please use instead the `statisticsEnabled` attribute.
- Method `boolean delete(boolean synchronous)` from class `org.infinispan.io.GridFile`. Please use instead `GridFile.delete()`.
- JMX attribute `long getLocallyInterruptedTransactions()` from class `org.infinispan.util.concurrent.locks.DeadlockDetectingLockManager`.

1.14. Upgrading from 5.2 to 5.3

1.14.1. Declarative configuration

In order to use all of the latest features, make sure you change the namespace declaration at the top of your XML configuration files as follows:

```
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"urn:infinispan:config:5.2 https://infinispan.org/schemas/infinispan-config-5.2.xsd"
xmlns="urn:infinispan:config:5.3">
   ...
</infinispan>
```

1.15. Upgrading from 5.1 to 5.2

1.15.1. Declarative configuration

In order to use all of the latest features, make sure you change the namespace declaration at the top of your XML configuration files as follows:

```
<infinispan xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
"urn:infinispan:config:5.2 https://infinispan.org/schemas/infinispan-config-5.2.xsd"
xmlns="urn:infinispan:config:5.2">
   ...
</infinispan>
```

1.15.2. Transaction

The default transaction enlistment model has changed (ISPN-1284) from `XAResource` to `Synchronization`. Also now, if the `XAResource` enlistment is used, then recovery is enabled by default.

In practical terms, if you were using the default values, this should not cause any backward compatibility issues but an increase in performance of about 5-7%. However in order to use the old configuration defaults, you need to configure the following:

```
<transaction useSynchronization="false">
   <recovery enabled="false"/>
</transaction>
```

or the programmatic configuration equivalent:

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.transaction().useSynchronization(false).recovery().enabled(false)
```

1.15.3. Cache Loader and Store configuration

Cache Loader and Store configuration has changed greatly in Infinispan 5.2.

1.15.4. Virtual Nodes and Segments

The concept of Virtual Nodes doesn't exist anymore in Infinispan 5.2 and has been replaced by Segments.

1.16. Upgrading from 5.0 to 5.1

1.16.1. API

The cache and cache manager hierarchies have changed slightly in 5.1 with the introduction of `BasicCache` and `BasicCacheContainer` , which are parent classes of existing `Cache` and `CacheContainer` classes respectively. What's important is that Hot Rod clients must now code against `BasicCache` and `BasicCacheContainer` rather than `Cache` and `CacheContainer`. So previous code that was written like this will no longer compile.

WontCompile.java

```
import org.infinispan.Cache;
import org.infinispan.manager.CacheContainer;
import org.infinispan.client.hotrod.RemoteCacheManager;
...
CacheContainer cacheContainer = new RemoteCacheManager();
Cache cache = cacheContainer.getCache();
```

Instead, if Hot Rod clients want to continue using interfaces higher up the hierarchy from the remote cache/container classes, they'll have to write:

Correct.java

```
import org.infinispan.BasicCache;
import org.infinispan.manager.BasicCacheContainer;
import org.infinispan.client.hotrod.RemoteCacheManager;
...
BasicCacheContainer cacheContainer = new RemoteCacheManager();
BasicCache cache = cacheContainer.getCache();
```

However, previous code that interacted against the `RemoteCache` and `RemoteCacheManager` will work as it used to:

AlsoCorrect.java

```
import org.infinispan.client.hotrod.RemoteCache;
import org.infinispan.client.hotrod.RemoteCacheManager;
...
RemoteCacheManager cacheContainer = new RemoteCacheManager();
RemoteCache cache = cacheContainer.getCache();
```

1.16.2. Eviction and Expiration

- The eviction XML element no longer defines the `wakeUpInterval` attribute. This is now configured via the `expiration` element:

```
<expiration wakeUpInterval="60000"... />
```

Eviction's `maxEntries` is used as guide for the entire cache, but eviction happens on a per cache segment, so when the segment is full, the segment is evicted. That's why `maxEntries` is a theoretical limit but in practical terms, it'll be a bit less than that. This is done for performance reasons.

1.16.3. Transactions

- A cache marked as `TRANSACTIONAL` cannot be accessed outside of a transaction, and a `NON_TRANSACTIONAL` cache cannot be accessed within a transaction. In 5.0, a transactional cache would support non-transactional calls as well. This change was done to be in-line with expectations set out in JSR-107 as well as to provide more consistent behavior.
- In 5.0, commit and rollback phases were asynchronous by default. Starting with 5.1, these are now synchronous by default, to provide the guarantees required by a single lock-owner model.

1.16.4. State transfer

One of the big changes we made in 5.1 was to use the same push-based state transfer we introduced in 5.0 both for rehashing in distributed mode and for state retrieval in replicated mode. We even borrow the consistent hash concept in replicated mode to transfer state from all previous cache members at once in order to speed up transfer.

As a consequence we've unified the state transfer configuration as well, there is now a `stateTransfer` element containing a simplified state transfer configuration. The corresponding attributes in the `stateRetrieval` and `hash` elements have been deprecated, as have been some attributes that are no longer used.

1.16.5. Configuration

If you use XML to configure Infinispan, you shouldn't notice any change, except a much faster startup, courtesy of the StAX based parser. However, if you use programmatic configuration, read on for the important differences.

Configuration is now packaged in `org.infinispan.configuration`, and you must use a fluent, builder style:

```
Configuration c1 = new ConfigurationBuilder()
   // Adjust any configuration defaults you want
   .clustering()
      .l1()
         .disable()
      .mode(DIST_SYNC)
      .hash()
         .numOwners(5)
   .build();
```

- The old javabean style configuration is now deprecated and will be removed in a later version.
- Configuration properties which can be safely changed at runtime are mutable, and all others are immutable.
- To copy a configuration, use the `read()` method on the builder, for example:

```
Configuration c2 = new ConfigurationBuilder()
   // Read in C1 to provide defaults
   .read(c1)
   .clustering()
      .l1()
         .enable()
   // This cache is DIST_SYNC, will have 5 owners, with L1 cache enabled
   .build();
```

This completely replaces the old system of defining a set of overrides on bean properties. Note that this means the behaviour of Infinispan configuration is somewhat different when used programmatically. Whilst before, you could define a default configuration, and any overrides would be applied on top of *your* defaults when defined, now you must explicitly read in your defaults to the builder. This allows for much greater flexibility in your code (you can have a as many "default" configurations as you want), and makes your code more explicit and type safe (finding references works).

The schema is unchanged from before. Infinispan 4.0 configurations are currently not being

parsed. To upgrade, just change the schema definition from:

```
<infinispan
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:infinispan:config:4.1
https://infinispan.org/schemas/infinispan-config-4.1.xsd"
     xmlns="urn:infinispan:config:4.1">
```

to

```
<infinispan
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="urn:infinispan:config:5.1
https://infinispan.org/schemas/infinispan-config-5.1.xsd"
     xmlns="urn:infinispan:config:5.1">
```

The schema documentation has changed format, as it is now produced using the standard tool `xsddoc`. This should be a significant improvement, as better navigation is offered. Some elements and attributes are missing docs right now, we are working on adding this. As an added benefit, your IDE should now show documentation when an xsd referenced (as above)

We are in the process of adding in support for this configuration style for modules (such as cache stores). In the meantime, please use the old configuration or XML if you require support for cache store module configuration.

1.16.6. Flags and ClassLoaders

The `Flags` and `ClassLoader` API has changed. In the past, the following would work:

```
 cache.withFlags(f1, f2); cache.withClassLoader(cl); cache.put(k, v);
```

In 5.1.0, these `withX()` methods return a new instance and not the cache itself, so thread locals are avoided and the code above will not work. If used in a fluent manner however, things still work:

```
cache.withFlags(f1, f2).withClassLoader(cl).put(k, v);
```

The above pattern has always been the intention of this API anyway.

1.16.7. JGroups Bind Address

Since upgrading to JGroups 3.x, `-Dbind.address` is ignored. This should be replaced with `-Djgroups.bind_addr`.

Chapter 2. Performing Rolling Upgrades for Infinispan Servers

Perform rolling upgrades of your Infinispan clusters to change between versions without downtime or data loss. Rolling upgrades migrate both your Infinispan servers and your data to the target version over Hot Rod.

2.1. Setting Up Target Clusters

Create a cluster that runs the target Infinispan version and uses a remote cache store to load data from the source cluster.

Prerequisites

- Install a Infinispan cluster with the target upgrade version.

Ensure the network properties for the target cluster do not overlap with those for the source cluster. You should specify unique names for the target and source clusters in the JGroups transport configuration. Depending on your environment you can also use different network interfaces and specify port offsets to keep the target and source clusters separate.

Procedure

1. Add a `RemoteCacheStore` on the target cluster for each cache you want to migrate from the source cluster.

   Remote cache stores use the Hot Rod protocol to retrieve data from remote Infinispan clusters. When you add the remote cache store to the target cluster, it can lazily load data from the source cluster to handle client requests.

2. Switch clients over to the target cluster so it starts handling all requests.
   a. Update client configuration with the location of the target cluster.
   b. Restart clients.

2.1.1. Remote Cache Stores for Rolling Upgrades

You must use specific remote cache store configuration to perform rolling upgrades, as follows:

```
<persistence passivation="false"> ①
   <remote-store xmlns="urn:infinispan:config:store:remote:11.0"
                 cache="myDistCache" ②
                 protocol-version="2.5" ③
                 hotrod-wrapping="true" ④
                 raw-values="true" ⑤
                 segmented="false"> ⑥
      <remote-server host="127.0.0.1" port="11222"/> ⑦
   </remote-store>
</persistence>
```

① Disables passivation. Remote cache stores for rolling upgrades must disable passivation.

② Matches the name of a cache in the source cluster. Target clusters load data from this cache using the remote cache store.

③ Matches the Hot Rod protocol version of the source cluster. `2.5` is the minimum version and is suitable for any upgrade paths. You do not need to set another Hot Rod version.

④ Ensures that entries are wrapped in a suitable format for the Hot Rod protocol.

⑤ Stores data in the remote cache store in raw format. This ensures that clients can use data directly from the remote cache store.

⑥ Disables segmentation for the remote cache store. You should enable segmentation for remote cache stores only if the number of segments in the target cluster matches the number of segements for the cache in the source cluster.

⑦ Points to the location of the source cluster.

Reference

- Remote cache store configuration schema
- RemoteStore
- RemoteStoreConfigurationBuilder

2.2. Synchronizing Data to Target Clusters

When your target cluster is running and handling client requests using a remote cache store to load data on demand, you can synchronize data from the source cluster to the target cluster.

This operation reads data from the source cluster and writes it to the target cluster. Data migrates to all nodes in the target cluster in parallel, with each node receiving a subset of the data. You must perform the synchronization for each cache in your Infinispan configuration.

Procedure

1. Start the synchronization operation for each cache in your Infinispan configuration that you want to migrate to the target cluster.

   Use the Infinispan REST API and invoke `POST` requests with the `?action=sync- data` parameter. For example, to synchronize data in a cache named "myCache" from a source cluster to a target cluster, do the following:

```
POST /v2/caches/myCache?action=sync-data
```

When the operation completes, Infinispan responds with the total number of entries copied to the target cluster.

Alternatively, you can use JMX by invoking `synchronizeData(migratorName=hotrod)` on the `RollingUpgradeManager` MBean.

2. Disconnect each node in the target cluster from the source cluster.

   For example, to disconnect the "myCache" cache from the source cluster, invoke the following `POST` request:

   ```
   POST /v2/caches/myCache?action=disconnect-source
   ```

   To use JMX, invoke `disconnectSource(migratorName=hotrod)` on the `RollingUpgradeManager` MBean.

Next steps

After you synchronize all data from the source cluster, the rolling upgrade process is complete. You can now decommission the source cluster.

Chapter 3. Patching Infinispan Server Installations

Install and manage patches for Infinispan server installations.

You can apply patches to multiple Infinispan servers with different versions to upgrade to a desired target version. However, patches do not take effect if Infinispan servers are running. For this reason you install patches while servers are offline. If you want to upgrade Infinispan clusters without downtime, create a new cluster with the target version and perform a rolling upgrade to that version instead of patching.

3.1. Infinispan Server Patches

Infinispan server patches are `.zip` archives that contain artifacts that you can apply to your `$ISPN_HOME` directory to fix issues and add new features.

Patches also provide a set of rules for Infinispan to modify your server installation. When you apply patches, Infinispan overwrites some files and removes others, depending on if they are required for the target version.

However, Infinispan does not make any changes to configuration files that you have created or modified when applying a patch. Server patches do not modify or replace any custom configuration or data.

3.2. Creating Server Patches

You can create patches for Infinispan servers from an existing server installation.

You can create patches for Infinispan servers starting from 10.1.7. You can patch any 10.1 or later server installation. However you cannot patch 9.4.x or earlier servers with 10.1.7 or later.

You can also create patches that either upgrade or downgrade the Infinispan server version. For example, you can create a patch from version 10.1.7 and use it to upgrade version 10.1.5 or downgrade version 11.0.0.

Procedure

1. Navigate to `$ISPN_HOME` for a Infinispan server installation that has the target version for the patch you want to create.
2. Start the CLI.

```
$ bin/cli.sh
[disconnected]>
```

3. Use the `patch create` command to generate a patch archive and include the `-q` option with a meaningful qualifier to describe the patch.

```
[disconnected]> patch create -q "this is my test patch" path/to/mypatch.zip \
path/to/target/server/home path/to/source/server/home
```

The preceding command generates a `.zip` archive in the specified directory. Paths are relative to `$ISPN_HOME` for the target server.

Create single patches for multiple different Infinispan versions, for example:

```
[disconnected]> patch create -q "this is my test patch"
path/to/mypatch.zip \
path/to/target/server/home \
path/to/source/server1/home path/to/source/server2/home
```

Where `server1` and `server2` are different Infinispan versions where you can install "mypatch.zip".

4. Describe the generated patch archive.

```
[disconnected]> patch describe path/to/mypatch.zip

Infinispan patch target=$target_version(my test patch)  source=$source_version
created=$timestamp
```

   - `$target_version` is the Infinispan server version from which the patch was created.
   - `$source_version` is one or more Infinispan server versions to which you can apply the patch.

     You can apply patches to Infinispan servers that match the `$source_version` only. Attempting to apply patches to other versions results in the following exception:

     ```
     java.lang.IllegalStateException: The supplied patch cannot be applied to
     `$source_version`
     ```

3.3. Installing Server Patches

Apply patches to Infinispan servers to upgrade or downgrade an existing version.

Prerequisites

- Create a server patch for the target version.

Procedure

1. Navigate to `$ISPN_HOME` for the Infinispan server you want to patch.
2. Stop the server if it is running.

If you patch a server while it is running, the version changes take effect after restart. If you do not want to stop the server, create a new cluster with the target version and perform a rolling upgrade to that version instead of patching.

3. Start the CLI.

```
$ bin/cli.sh
[disconnected]>
```

4. Install the patch.

```
[disconnected]> patch install path/to/patch.zip

Infinispan patch target=$target_version source=$source_version \
created=$timestamp installed=$timestamp
```

   - `$target_version` displays the Infinispan version that the patch installed.
   - `$source_version` displays the Infinispan version before you installed the patch.

5. Start the server to verify the patch is installed.

```
$ bin/server.sh
...
ISPN080001: Infinispan Server $version
```

   If the patch is installed successfully `$version` matches `$target_version`.

Use the `--server` option to install patches in a different `$ISPN_HOME` directory, for example:

```
[disconnected]> patch install path/to/patch.zip
--server=path/to/server/home
```

3.4. Rolling Back Server Patches

Remove patches from Infinispan servers by rolling them back and restoring the previous Infinispan version.

If a server has multiple patches installed, you can roll back the last installed patch only.

Rolling back patches does not revert configuration changes you make to Infinispan server. Before you roll back patches, you should ensure that your configuration is compatible with the version to which you are rolling back.

Procedure

1. Navigate to `$ISPN_HOME` for the Infinispan server installation you want to roll back.
2. Stop the server if it is running.
3. Start the CLI.

```
$ bin/cli.sh
[disconnected]>
```

4. List the installed patches.

```
[disconnected]> patch ls

Infinispan patch target=$target_version source=$source_version
created=$timestamp installed=$timestamp
```

   - `$target_version` is the Infinispan server version after the patch was applied.
   - `$source_version` is the version for Infinispan server before the patch was applied. Rolling back the patch restores the server to this version.

5. Roll back the last installed patch.

```
[disconnected]> patch rollback
```

6. Quit the CLI.

```
[disconnected]> quit
```

7. Start the server to verify the patch is rolled back to the previous version.

```
$ bin/server.sh
...
ISPN080001: Infinispan Server $version
```

   If the patch is rolled back successfully `$version` matches `$source_version`.

Use the `--server` option to rollback patches in a different `$ISPN_HOME` directory, for example:

```
[disconnected]> patch rollback --server=path/to/server/home
```

Chapter 4. Migrating Data Between Cache Stores

Infinispan provides a Java utility for migrating persisted data between cache stores.

In the case of upgrading Infinispan, functional differences between major versions do not allow backwards compatibility between cache stores. You can use `StoreMigrator` to convert your data so that it is compatible with the target version.

For example, upgrading to Infinispan 10.0 changes the default marshaller to Protostream. In previous Infinispan versions, cache stores use a binary format that is not compatible with the changes to marshalling. This means that Infinispan 10.1 cannot read from cache stores with previous Infinispan versions.

In other cases Infinispan versions deprecate or remove cache store implementations, such as JDBC Mixed and Binary stores. You can use `StoreMigrator` in these cases to convert to different cache store implementations.

4.1. Cache Store Migrator

Infinispan provides the `StoreMigrator.java` utility that recreates data for the latest Infinispan cache store implementations.

`StoreMigrator` takes a cache store from a previous version of Infinispan as source and uses a cache store implementation as target.

When you run `StoreMigrator`, it creates the target cache with the cache store type that you define using the `EmbeddedCacheManager` interface. `StoreMigrator` then loads entries from the source store into memory and then puts them into the target cache.

`StoreMigrator` also lets you migrate data from one type of cache store to another. For example, you can migrate from a JDBC String-Based cache store to a Single File cache store.

`StoreMigrator` cannot migrate data from segmented cache stores to:

- Non-segmented cache store.
- Segmented cache stores that have a different number of segments.

4.2. Getting the Store Migrator

`StoreMigrator` is available as part of the Infinispan tools library, `infinispan-tools`, and is included in the Maven repository.

Procedure

- Configure your `pom.xml` for `StoreMigrator` as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.infinispan.example</groupId>
    <artifactId>jdbc-migrator-example</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
      <dependency>
        <groupId>org.infinispan</groupId>
        <artifactId>infinispan-tools</artifactId>
      </dependency>
      <!-- Additional dependencies -->
    </dependencies>

    <build>
      <plugins>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>1.2.1</version>
          <executions>
            <execution>
              <goals>
                <goal>java</goal>
              </goals>
            </execution>
          </executions>
          <configuration>
            <mainClass>
org.infinispan.tools.store.migrator.StoreMigrator</mainClass>
            <arguments>
              <argument>path/to/migrator.properties</argument>
            </arguments>
          </configuration>
        </plugin>
      </plugins>
    </build>
</project>
```

4.3. Configuring the Store Migrator

Set properties for source and target cache stores in a `migrator.properties` file.

Procedure

1. Create a `migrator.properties` file.
2. Configure the source cache store in `migrator.properties`.
   a. Prepend all configuration properties with `source.` as in the following example:

```
source.type=SOFT_INDEX_FILE_STORE
source.cache_name=myCache
source.location=/path/to/source/sifs
```

3. Configure the target cache store in `migrator.properties`.
   a. Prepend all configuration properties with `target.` as in the following example:

```
target.type=SINGLE_FILE_STORE
target.cache_name=myCache
target.location=/path/to/target/sfs.dat
```

4.3.1. Store Migrator Properties

Configure source and target cache stores in a `StoreMigrator` properties.

Table 1. Cache Store Type Property

Property	Description	Required/Optional

`type`	Specifies the type of cache store type for a source or target.
	
	`.type=JDBC_STRING`
	
	`.type=JDBC_BINARY`
	
	`.type=JDBC_MIXED`
	
	`.type=LEVELDB`
	
	`.type=ROCKSDB`
	
	`.type=SINGLE_FILE_STORE`
	
	`.type=SOFT_INDEX_FILE_STORE`
	
	`.type=JDBC_MIXED`
		Required

Table 2. Common Properties

Property	Description	Example Value	Required/Optional
`cache_name`	Names the cache that the store backs.	`.cache_name=myCache`	Required
`segment_count`	Specifies the number of segments for target cache stores that can use segmentation. The number of segments must match `clustering.hash.numSegments` in the Infinispan configuration. In other words, the number of segments for a cache store must match the number of segments for the corresponding cache. If the number of segments is not the same, Infinispan cannot read data from the cache store.	`.segment_count=256`	Optional

Table 3. JDBC Properties

Property	Description	Required/Optional
`dialect`	Specifies the dialect of the underlying database.	Required
`version`	Specifies the marshaller version for source cache stores. Set one of the following values: * `8` for Infinispan 8.x * `9` for Infinispan 9.x * `10` Infinispan 10.x	Required for source stores only. For example: `source.version=9`
`marshaller.class`	Specifies a custom marshaller class.	Required if using custom marshallers.

Property	Description	Required/Optional
marshaller.externalizers	Specifies a comma-separated list of custom AdvancedExternalizer implementations to load in this format: [id]:<Externalizer class>	Optional
connection_pool.connection_url	Specifies the JDBC connection URL.	Required
connection_pool.driver_class	Specifies the class of the JDBC driver.	Required
connection_pool.username	Specifies a database username.	Required
connection_pool.password	Specifies a password for the database username.	Required
db.major_version	Sets the database major version.	Optional
db.minor_version	Sets the database minor version.	Optional
db.disable_upsert	Disables database upsert.	Optional
db.disable_indexing	Specifies if table indexes are created.	Optional
table.string.table_name_prefix	Specifies additional prefixes for the table name.	Optional
table.string.<id|data|timestamp>.name	Specifies the column name.	Required
table.string.<id|data|timestamp>.type	Specifies the column type.	Required
key_to_string_mapper	Specifies the TwoWayKey2StringMapper class.	Optional

To migrate from Binary cache stores in older Infinispan versions, change table.string.* to table.binary.* in the following properties:

- source.table.binary.table_name_prefix
- source.table.binary.<id\|data\|timestamp>.name
- source.table.binary.<id\|data\|timestamp>.type

```
# Example configuration for migrating to a JDBC String-Based cache store
target.type=STRING
target.cache_name=myCache
target.dialect=POSTGRES
target.marshaller.class=org.example.CustomMarshaller
target.marshaller.externalizers=25:Externalizer1,org.example.Externalizer2
target.connection_pool.connection_url=jdbc:postgresql:postgres
target.connection_pool.driver_class=org.postrgesql.Driver
target.connection_pool.username=postgres
target.connection_pool.password=redhat
target.db.major_version=9
target.db.minor_version=5
target.db.disable_upsert=false
target.db.disable_indexing=false
target.table.string.table_name_prefix=tablePrefix
target.table.string.id.name=id_column
target.table.string.data.name=datum_column
target.table.string.timestamp.name=timestamp_column
target.table.string.id.type=VARCHAR
target.table.string.data.type=bytea
target.table.string.timestamp.type=BIGINT
target.key_to_string_mapper=org.infinispan.persistence.keymappers.
DefaultTwoWayKey2StringMapper
```

Table 4. RocksDB Properties

Property	Description	Required/Optional
`location`	Sets the database directory.	Required
`compression`	Specifies the compression type to use.	Optional

```
# Example configuration for migrating from a RocksDB cache store.
source.type=ROCKSDB
source.cache_name=myCache
source.location=/path/to/rocksdb/database
source.compression=SNAPPY
```

Table 5. SingleFileStore Properties

Property	Description	Required/Optional
`location`	Sets the directory that contains the cache store `.dat` file.	Required

```
# Example configuration for migrating to a Single File cache store.
target.type=SINGLE_FILE_STORE
target.cache_name=myCache
target.location=/path/to/sfs.dat
```

Table 6. SoftIndexFileStore Properties

Property	Description	Value
Required/Optional	`location`	Sets the database directory.
Required	`index_location`	Sets the database index directory.

```
# Example configuration for migrating to a Soft-Index File cache store.
target.type=SOFT_INDEX_FILE_STORE
target.cache_name=myCache
target.location=path/to/sifs/database
target.location=path/to/sifs/index
```

4.4. Migrating Cache Stores

Run `StoreMigrator` to migrate data from one cache store to another.

Prerequisites

- Get `infinispan-tools.jar`.
- Create a `migrator.properties` file that configures the source and target cache stores.

Procedure

- If you build `infinispan-tools.jar` from source, do the following:
  1. Add `infinispan-tools.jar` and dependencies for your source and target databases, such as JDBC drivers, to your classpath.
  2. Specify `migrator.properties` file as an argument for `StoreMigrator`.
- If you pull `infinispan-tools.jar` from the Maven repository, run the following command:

  ```
  mvn exec:java
  ```