Upgrading to Infinispan 9.0

The Infinispan community

Table of Contents

This guide walks you through the process of upgrading Infinispan.

Chapter 1. Upgrading from 8.x to 9.0

1.1. 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.

1.2. 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.

1.3. 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.

1.4. 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.

1.5. Spring 3 support is now removed

Spring 3 is no longer supported.

1.6. 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.

1.7. SegmentCompletionListener interface has moved

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

1.8. 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-spring4-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).

1.9. Total order executor is not removed

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

1.10. 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`.

1.11. 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.

1.12. 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. See JDBC Migrator Guide for more details.

1.13. @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.

Chapter 2. Upgrading from 8.1 to 8.2

2.1. Entry Retriever is deprecated

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

2.2. Map / Reduce is deprecated

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

Chapter 3. Upgrading from 8.x to 8.1

3.1. Packaging changes

3.1.1. 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.

3.1.2. Spring module split

Spring module (GroupId:ArtifactId `org.infinispan:infinispan-spring4`) has been split into `org.infinispan:infinispan-spring4-embedded` and `org.infinispan:infinispan-spring4-remote`. Please make sure that you use proper artifact.

3.2. Spring 3 support is deprecated

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

Chapter 4. Upgrading from 7.x to 8.0

4.1. Configuration changes

4.1.1. 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.

4.1.2. 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.

4.1.3. 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

4.1.4. 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.

Chapter 5. Upgrading from 6.0 to 7.0

5.1. API Changes

5.1.1. 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.

5.1.2. 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.

5.1.3. 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.

5.2. 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.

Chapter 6. Upgrading from 5.3 to 6.0

6.1. 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 http://www.infinispan.org/schemas/infinispan-config-
6.0.xsd" xmlns="urn:infinispan:config:6.0">
   ...
</infinispan>
```

6.2. 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. Please refer to [_configuration] for more information about the new configuration classes.

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`.

Chapter 7. Upgrading from 5.2 to 5.3

7.1. 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 http://www.infinispan.org/schemas/infinispan-config-
5.2.xsd" xmlns="urn:infinispan:config:5.3">
   ...
</infinispan>
```

Chapter 8. Upgrading from 5.1 to 5.2

8.1. 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 http://www.infinispan.org/schemas/infinispan-config-
5.2.xsd" xmlns="urn:infinispan:config:5.2">
   ...
</infinispan>
```

8.2. 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)
```

8.3. Cache Loader and Store configuration

Cache Loader and Store configuration has changed greatly in Infinispan 5.2. Please refer to the Cache Loaders and Stores documentation.

8.4. Virtual Nodes and Segments

The concept of Virtual Nodes doesn't exist anymore in Infinispan 5.2 and has been replaced by Segments. Please refer to the Clustering modes documentation for details.

Chapter 9. Upgrading from 5.0 to 5.1

9.1. 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();
```

9.2. 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.

9.3. 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.

9.4. 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.

9.5. 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
http://www.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
http://www.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.

9.6. 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.

9.7. JGroups Bind Address

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