DZone Refcardz

JBoss by Red Hat

CONTENTS INCLUDE:

Getting started with Infinispan

By Manik Surtani

OPEN SOURCE DATA GRIDS

What is Infinispan?

Infinispan is an open source data grid platform. Data grids are commonly used as low-latency, highly-available and elastic data storage backends, often as NoSQL solutions.

Data grids are often used in addition to traditional databases, as a distributed cache for fast data access.

Hot Tip Infinispan is LGPL licensed and is backed by an active, open and welcoming developer and user community!

For more information, visit http://www.infinispan.org

How can I get it?

The best way to use Infinispan in your project is via Maven. Infinispan's Maven coordinates are:

```
<dependencies>
 …
 <dependency>
     <groupId>org.infinispan</groupId>
     <artifactId>infinispan-core</artifactId>
     <version>LATEST_INFINISPAN_VERSION</version>
 </dependency>
 …
</dependencies>
```

However you would need to add the JBoss community projects Maven repository to your list of repositories to be able to locate and use Infinispan artifacts.

```
<repositories>
  …
  <repository>
      <id>jboss.org</id>
         <url>http://repository.jboss.org/nexus/content/groups/
         public</url>
      <releases><enabled>true</enabled></releases>
      <snapshots><enabled>true</enabled></snapshots>
 </repository>
 …
</repositories>
```

I don't use Maven. Where can I download binaries?

If you do not use Maven, or wish to run an Infinispan server (rather than in embedded, peer-to-peer mode), you can also download compiled binaries from http://www.jboss.org/infinispan/downloads

OPERATIONAL MODES

Client/server or peer-to-peer?

There are two ways in which you can interact with Infinispan. One is in embedded mode, where you start an Infinispan instance within your JVM. The other is client/server mode, where you start a remote Infinispan instance and connect to it using a client connector.

Your choice on which mode of interaction to use will depend on a number of factors, including whether you are using Infinispan as a clustering toolkit to cluster your own framework, whether you intend to use Infinispan to cache database lookups, or whether you plan to interact with Infinispan from a non-JVM environment. These are discussed in more detail at http://community.jboss.org/wiki/InfinispanServerModules

Embedded (p2p) mode

When used in this mode, Infinispan instances reside within your JVM alongside your code. You start up multiple instances of your application on different servers and each one starts and initializes an Infinispan instance. These Infinispan instances discover each other, form a data grid, and start sharing and distributing data.

This mode is what you will want to use if you are building an application or a framework that needs to be cluster-aware, or if you need an in-memory distributed cache to front a database or some other expensive data source.

A practical example

To see this in action, make sure the Infinispan libraries are available to your code—either via Maven, as above, or via downloading the zipped distribution and extracting the jar files.

Hot Tip Infinispan's core construct is an instance of the Cache interface. Extending java.util.Map, Cache exposes simple methods to store and retrieve objects.

Starting Infinispan instances from your code is simple. To start a local, non-clustered cache instance:

```
DefaultCacheManager m = new DefaultCacheManager();
Cache<String, String> c = m.getCache();
c.put("hello", "world");
```

To start a cluster-aware instance capable of detecting neighboring instances on the same local area network and sharing state between them:

```
GlobalConfiguration globalConf = GlobalConfiguration.
getClusteredDefault();
Configuration cfg = new Configuration();
Cfg.setCacheMode(Configuration.CacheMode.DIST_SYNC);
DefaultCacheManager m = new DefaultCacheManager(globalConf, cfg);
Cache<String, String> c = m.getCache();
c.put("hello", "world");
c.containsKey("hello"); // returns true
c.get("hello"); // returns "world"
c.remove("hello"); // returns "world"
```

This can also be done using a configuration file:

```
String configFile = "/path/to/my/infinispan_config.xml";
DefaultCacheManager m = new DefaultCacheManager(configFile);
Cache<String, String> c = m.getCache();
c.put("hello", "world");
c.containsKey("hello"); // returns true
c.get("hello"); // returns "world"
c.remove("hello"); // returns "world"
```

Embedded Infinispan and other JVM languages

Since Infinispan complies to Java byte code, it can be used by other JVM languages as well, such as Jython, JRuby, Scala and Groovy, provided the necessary libraries are available on the classpath. Here is an example of starting an embedded Infinispan instance in a Groovy console:

```
groovy:000> import org.infinispan.*
===> [import org.infinispan.*]
groovy:000> import org.infinispan.manager.*
===> [import org.infinispan.*, import org.infinispan.manager.*]
groovy:000> m = new DefaultCacheManager()
===> org.infinispan.manager.DefaultCacheManager@1a8fa0d1@Address:null
groovy:000> c = m.getCache()
===> Cache '___defaultcache'@574813774
groovy:000> c.put("hello", "world")
===> null
groovy:000> c.get("hello")
===> world
groovy:000> c.remove("hello")
===> world
```

Client/server mode

You may not always want Infinispan instances to reside in the same JVM as your application. Sometimes this is for security, sometimes for architectural reasons to maintain a separate data layer, but this can also be because your client application is not on a JVM platform. For example, .NET or C++ clients can also make use of Infinispan if Infinispan is run as a remote server.

Hot Tip A good discussion of remote data storage architectures using data grids can be found here: http://java.dzone.com/articles/data-service-data-fabric

Infinispan comes with several different server endpoints, speaking a variety of protocols. Here is a comparison of the protocols that can be used with Infinispan:

SERVER COMPARISON

	Protocol	Client Availability	Clustered	Smart Routing	Load Balancing / Failover
REST	Text	Tons	Yes	No	Any Http Load Balancer
Memcached	Text	Tons	Yes	No	Only with predefined list of servers
Hot Rod	Binary	Right now, only Java	Yes	Yes	Yes, dynamic via Hot Rod client
Websocket	Text	Javascript only	Yes	No	Any Http Load Balancer

STARTING AN INFINISPAN SERVER

Starting an Infinispan server is pretty easy. You need to download and unzip the Infinispan distribution and use the startServer script. E.g.,

```
$ bin/startServer.sh -r hotrod
```

The script takes in a set of switches to control the endpoint behavior:

```
$ bin/startServer.sh --help
usage: startServer [options]
options:
 -h, --help Show this help message
 -V, --version Show version information
 --  Stop processing options
 -p, --port=<num> TCP port number to listen on (default: 11211 for
Memcached, 11311 for Hot Rod and 8181 for WebSocket server)
 -l, --host=<host or ip> Interface to listen on (default: 127.0.0.1,
localhost)
 -m, --master_threads=<num> Number of threads accepting incoming
connections (default: unlimited while resources are available)
 -t, --work_threads=<num> Number of threads processing incoming
requests and sending responses (default: unlimited while resources
are available)
 -c, --cache_config=<filename> Cache configuration file (default:
creates cache with default values)
 -r, --protocol= Protocol to understand by the server. This is
a mandatory option and you should choose one of these options:
[memcached|hotrod|websocket]
 -i, --idle_timeout=<num> Idle read timeout, in seconds, used to
detect stale connections (default: -1). If no new messages have been
read within this time, the server disconnects the channel. Passing
-1 disables idle timeout.
 -n, --tcp_no_delay=[true|false] TCP no delay flag switch (default:
true).
 -s, --send_buf_size=<num> Send buffer size (default: as defined by
the OS).
 -e, --recv_buf_size=<num> Receive buffer size (default: as defined
by the OS).
 -o, --proxy_host=<host or ip> Host address to expose in topology
information sent to clients. If not present, it defaults to
configured host. Servers that do not transmit topology information
ignore this setting.
 -x, --proxy_port=<num> Port to expose in topology information sent
to clients. If not present, it defaults to configured port. Servers
that do not transmit topology information ignore this setting.
 -D<name>[=<value>] Set a system property
```

Starting the REST endpoint

Infinispan ships a REST endpoint as a web archive (.WAR file). To start the REST endpoint, simply deploy this WAR file in a servlet container of your choice, such as JBoss Application Server or Apache Tomcat.

Connecting using Java

You have a choice of protocols and clients you can use if connecting from a Java application.

If using the REST endpoint, all you need is a simple HTTP client library, such as Apache's HTTPClient.

First, you need to make sure you have Apache HTTPClient in your classpath — by declaring it as a Maven dependency or by downloading the jars. Also make sure you have the Infinispan REST module deployed and available.

Now you can connect to the REST endpoint:

```
HttpClient client = new HttpClient();
String cacheName =”___defaultcache”;
String key = “hello”;
String url = “http://infinispan_host/infinispan-server-rest/rest/” +
cacheName + “/” + key;

// Storing data
PutMethod put = new PutMethod(url);
put.setRequestHeader(“Content-type”, “text/plain”);
put.setRequestBody(“world”);
client.executeMethod(put);

// Retrieving data
GetMethod get = new GetMethod(url);
client.executeMethod(get);
System.out.printf(“Value of key %s is %s”, key, get.
getResponseBodyAsString());
```

If you choose to use the memcached protocol, the SpyMemcached library can be used. Again, you would need to make sure you have SpyMemcached in your classpath.

```
InetSocketAddress addr = new InetSocketAddress(“infinispan_host”,
portNum);
MemcachedClient c=new MemcachedClient(addr);
// Storing data
String key = “hello”
c.set(key, -1, “world”);
// Retrieving data
String result = c.get(key);
System.out.printf(“Value of key %s is %s”, key, result);
```

Finally, if you wish to use Hot Rod, you would need to declare a dependency on Infinispan's Hot Rod Java client library:

```
<dependencies>
 …
 <dependency>
 <groupId>org.infinispan</groupId>
 <artifactId>infinispan-client-hotrod</artifactId>
 <version>LATEST_INFINISPAN_VERSION</version>
 </dependency>
 …
</dependency>
```

The Hot Rod client jar files are also included in the Infinispan zipped distribution.

Using the client is very much like using the embedded API:

```
RemoteCacheManager rcm = new RemoteCacheManager(“infinispan_host”);
RemoteCache rc = rcm.getCache();
String key = “hello”;

// Storing data
rc.put(key, “world”);

// Retrieving data
System.out.printf(“Value of key %s is %s”, key, rc.get(key));
```

Connecting using non-JVM platforms

Infinispan supports connecting to the data grid from non-Java platforms. The simplest option is to use the REST endpoint. Here is an example of connecting to the Infinispan REST endpoint using a Python script:

```
import httplib

cache_name = “___defaultcache”
key = “hello”

// Storing data
conn = httplib.HTTPConnection(“infinispan_host”)
conn.request(“PUT”, “/infinispan-server-rest/rest/%s/%s” % (cache_
name, key), “world”, {“Content-Type”: “text/plain”})

// Retrieving data
conn = httplib.HTTPConnection(“infinispan_host”)
conn.request(“GET”, “/infinispan-server-rest/rest/%s/%s” % (cache_
name, key))
print “Value of key %s is %s” % (key, conn.getresponse().read())
```

If running the memcached endpoint, it is possible for clients to connect using any existing memcached client library. This example uses the memcached library for Python:

```
import memcache

conn = memcache.Client([“infinispan_host”])
key = “hello”

// Storing data
conn.set(key, “world”)

// Retrieving data
print “Value of key %s is %s” % (key, conn.get(key))
```

Hot Rod, too, can be used from non-JVM platforms. However, as of September 2010, the only known client libraries for Hot Rod are written in Java.

Hot Tip

The protocol specification is published online and the community is encouraged to write more client implementations for Hot Rod. The Java client can be used as a reference implementation, as its source code is open and publicly available

Load-balancing server endpoints

Infinispan's endpoints support load balancing and failover to some degree. Different endpoints offer different degrees of support.

The REST endpoint is the simplest, delegating all load balancing and failover responsibility to an external HTTP load balancer. Software load balancers such as mod_cluster and hardware load balancers could be used. You should refer to your servlet container documentation for details on load balancing.

The memcached endpoint — like all memcached servers — delegates the task of load balancing and failover to the memcached client. Most memcached client libraries have support for load balancing and failover, provided they are initialized with a static list of servers to connect to.

Hot Rod provides the most flexibility in terms of load balancing and failover.

Hot Tip The Hot Rod protocol has been designed specifically with load balancing and failover in mind.

Clients can be written to take advantage of the server topology that is provided to clients and regularly kept up-to-date. Clients can even be made aware of hash functions used on the back-end, so routing requests to a cluster of back-end nodes can be done in an intelligent fashion to minimize latency and remote lookup on the back-end. The reference implementation Java client makes use of such features and provides built-in load-balancing, failover, discovery of new backend nodes, as well as intelligent routing of requests. More details on the Java client can be found online.

ANATOMY OF AN INFINISPAN CONFIGURATION FILE

Infinispan's configuration file is written in XML. Sensible defaults are used throughout, and the simplest configuration file contains just the following:

```
<infinispan />
```

This defines local, non-clustered caches using defaults throughout.

A basic clustered configuration looks like:

```
<infinispan>
 <global>
      <transport />
 </global>

 <default>
      <clustering mode="R">
          <sync />
      </clustering>
 </default>
</infinispan>
```

The *default* configuration is used as a template configuration when calling DefaultCacheManager.getCache(). When calling DefaultCacheManager.getCache(cacheName), a clone of the default configuration is made. E.g.:

```
DefaultCacheManager cm =
new DefaultCacheManager("configuration.xml");

// returns the default cache
Cache<String, String> cache = cm.getCache();

// returns a new cache, with a configuration cloned from the default>
Cache<String, String> anotherCache = cm.getCache("another");
```

You can also name caches in your configuration file, such as:

```
<infinispan>
  <global>
      <transport />
  </global>

  <default>
      <clustering mode="R">
          <sync />
      </clustering>
  </default>

  <namedCache name="asyncCache">
      <clustering mode="R">
          <async />
      </clustering>
  </namedCache>

</infinispan>
```

With this configuration, DefaultCacheManager.getCache() would return a simple, synchronously replicated cache.

DefaultCacheManager.getCache("asyncCache") would, however, return an asynchronously replicated cache.

```
DefaultCacheManager cm = new DefaultCacheManager("configuration.
xml");

// returns the default cache – one that is synchronously replicated!
Cache<String, String> cache = cm.getCache();

// returns an asynchronously replicated cache
Cache<String, String> asyncCache = cm.getCache("asyncCache");
```

Named caches are hierarchical too, so they all inherit from the default. In the example below, the cache named "transactional" extends from the default cache. As such, the cache named "transactional" will also be synchronously replicated.

```
<infinispan>
  <global>
      <transport />
  </global>

  <default>
      <clustering mode="R">
          <sync />
      </clustering>
  </default>

  <namedCache name="transactional">
      <transaction
transactionManagerLookupClass="org.infinispan.transaction.lookup.
GenericTransactionManagerLookup"/>
  </namedCache>
</infinispan>
```

```
DefaultCacheManager cm = new DefaultCacheManager("configuration.
xml");

// returns the default cache – one that is synchronously replicated!
Cache<String, String> cache = cm.getCcahe();

// returns an transactional, synchronously replicated cache
Cache<String, String> txCache = cm.getCache("transactional");
```

XML Schemas

Infinspan makes use of an XML schema to validate configuration files.

The schema is packaged with the infinispan-core.jar archive, and is also available online at http://www.infinispan.org/schemas/infinispan-config-4.0.xsd

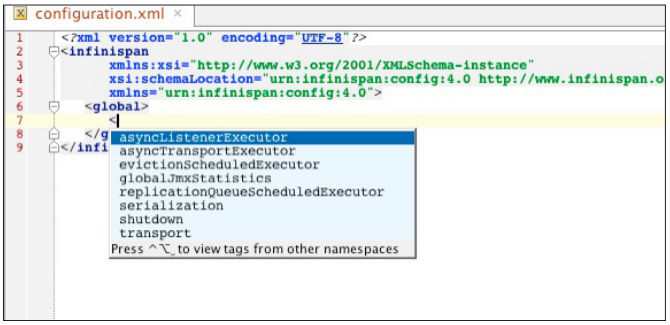
Typically, you would start your XML file with the following declaration:

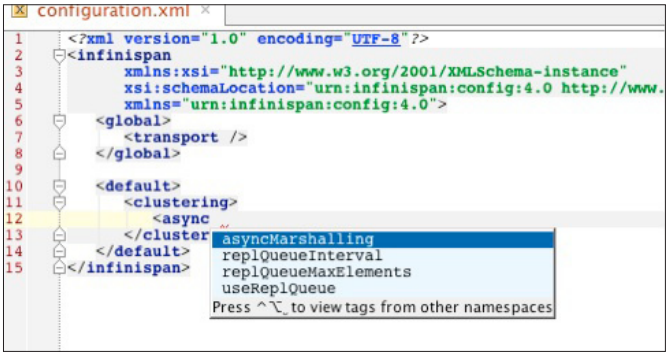
```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:infinispan:config:4.0 http://www.
infinispan.org/schemas/infinispan-config-4.0.xsd"
    xmlns="urn:infinispan:config:4.0">
… Your Infinispan configuration …
</infinispan>
```

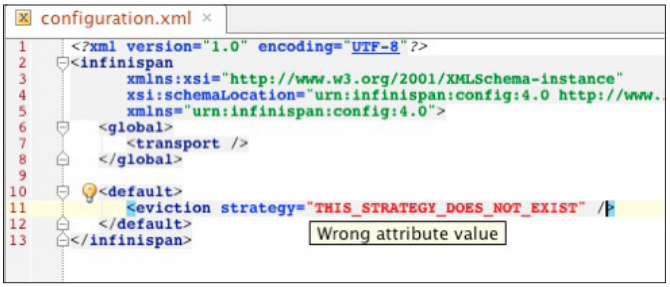
If you were to start your configuration file with a reference to this XML schema, XML authoring tools will help validate your configuration file.

Hot Tip Some tools even provide autocomplete suggestions!

For example:







Commonly used configuration elements

Global selection

Element Name	Element Description
transport	This element configures the transport used for network communications across the cluster.
serialization	Serialization and marshalling settings.
shutdown	This element specifies behavior when the JVM running the cache instance shuts down.
globalJmxStatistics	This element specifies whether global statistics are gathered and reported via JMX for all caches under this cache manager.
replicationQueueScheduledExecutor	Configuration for the scheduled executor service used to periodically flush replication queues, used if asynchronous clustering is enabled along with useReplQueue being set to true.
asyncTransportExecutor	Configuration for the executor service used for asynchronous work on the Transport, including asynchronous marshalling and Cache 'async operations' such as Cache.putAsync().
evictionScheduledExecutor	Configuration for the scheduled executor service used to periodically run eviction cleanup tasks.
asyncListenerExecutor	Configuration for the executor service used to emit notifications to asynchronous listeners.

Default/NamedCache sections

Element Name	Element Description
transaction	Defines transactional (JTA) characteristics of the cache.
invocationBatching	Defines whether invocation batching is allowed in this cache instance, and sets up internals accordingly to allow use of this API.
loaders	Holds the configuration for cache loaders and stores.
clustering	Defines clustered characteristics of the cache.
lazyDeserialization	A mechanism by which serialization and deserialization of objects is deferred until the point in time in which they are used and needed. This typically means that any deserialization happens using the thread context class loader of the invocation that requires deserialization, and is an effective mechanism to provide classloader isolation.
deadlockDetection	This element configures deadlock detection.
eviction	This element controls the eviction settings for the cache.
customInterceptors	Configures custom interceptors to be added to the cache.
unsafe	Allows you to tune various unsafe or non-standard characteristics. Certain operations such as Cache.put() that are supposed to return the previous value associated with the specified key according to the java.util.Map contract will not fulfill this contract if unsafe toggle is turned on. Use with care. See details at http://www.jboss.org/community/wiki/infinispantechnicalfaqs
jmxStatistics	This element specifies whether cache statistics are gathered and reported via JMX.

locking	Defines the local, in-VM locking and concurrency characteristics of the cache.
indexing	Configures indexing of entries in the cache for searching. Note that infinispan-query.jar and its dependencies needs to be available if this option is to be used.
expiration	This element controls the default expiration settings for entries in the cache.

MIGRATING FROM OTHER DATA GRID OR CACHE SYSTEMS

Infinispan provides a number of tools to help you migrate configurations from EHCache, Oracle Coherence and even JBoss Cache to Infinispan. These command-line tools help in the migration process.

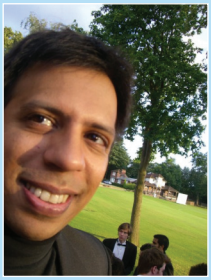
```
$ bin/importConfig.sh
Missing 'source', cannot proceed
Usage:
importConfig [-source <the file to be transformed>] [-destination
<where to store resulting XML>] [-type <the type of the source,
possible values being: [Coherence35x, Ehcache1x, JBossCache3x] >]
```

Further, Infinispan's Cache interface is compliant with JSR-107 (JCACHE), which means applications written against other JSR-107-like caches will work with minimal modifications.

More information

Please visit http://community.jboss.org/wiki/Infinispan for more detailed information on Infinispan, including an easy-to-use configuration reference.

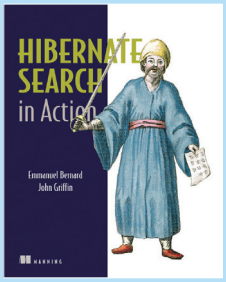
ABOUT THE AUTHOR



MANIK SURTANI, is a Principal Software Engineer and core JBoss research and development engineer at Red Hat. He is the founder of the Infinispan project, which he currently leads along with the JBoss Cache project. His interests lie in cloud and distributed computing, autonomous systems, and highly-available computing.

Manik has a background in artificial intelligence and neural networks, a field that he left behind when he moved from academic circles to the commercial world. Since then, he worked with Java-related technologies at a start-up company that focused on knowledge management and information exchange. He also worked as a technical lead focusing on e-commerce applications on large Java EE and peer-to-peer technology for a London-based consultancy. Manik is a strong proponent of open source development methodologies, ethos, and collaborative processes, and has been involved in open source since his first forays into computing.

RECOMMENDED BOOK



Good search capability is one of the primary demands of a business application. Engines like Lucene provide a great starting point, but with complex applications it can be tricky to implement. It's tough to keep the index up to date, deal with the mismatch between the index structure and the domain model, handle querying conflicts, and so on.

Hibernate Search is an enterprise search tool based on Hibernate Core and Apache Lucene. It provides full text search capabilities for Hibernate-based applications without the infrastructural code required by other search engines. With this free, open-source technology, you can quickly add high-powered search features in an intelligent, maintainable way.

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-00-4
ISBN-10: 1-936502-00-3
50795
9 781936 502004
$7.95

Version 1.0