GET STARTED WITH INFINISPAN

Sandeep Khemani | Jim Tyrrell
April / 18 / 2011

Through this lab, application developers who are seeking a scalable caching & compute data grid solution, will get a hands on introduction to Infinispan. A fully certified, supported offering based on the Infinispan project will manifest itself in the near future – please stay tuned for details.

Contents

1 INTRODUCTION TO INFINISPAN

The Infinispan solution is a 100% open source solution that is written in Java and aims to provide a scalable & available, data grid platform. It will also be highly concurrent, clustered compute data grid that will make the most of modern multi-processor/multi-core architectures while at the same time providing distributed cache capabilities. It will be optionally backed by a peer-to-peer network architecture to distribute state efficiently around a data grid running on multiple physical servers.

Offering high availability of state across a network as well as persisting state to configurable data stores, Infinispan offers enterprise features such as efficient eviction for efficient memory usage as well as JTA compatibility.

In addition to the peer-to-peer architecture of Infinispan, on the roadmap is the ability to run farms of Infinispan instances as servers and connecting to them using a plethora of clients - both written in Java as well as other popular platforms.

Here are some of the features of Infinispan:

State-of-the-Art Core - Infinispan's core is a specialized data structure, tuned to and geared for concurrency - especially on multi-CPU/multi-core architectures. The internals are intended to be lock and synchronization free, favoring state-of-the-art non-blocking algorithms and techniques wherever possible.

Massive Heap - If you have 100 blade servers with 2GB nodes each in a ***replicated cache***, you end with 2 GB of total data. Every server is just a copy. On the other hand, with a distributed grid - assuming you want a total of 2 copies per data item - you get a 100 GB memory backed virtual heap that is efficiently accessible from anywhere in the grid. If a server fails, the grid simply creates new copies of the lost data, and puts them on other servers. This means that applications looking for ultimate performance are no longer forced to delegate the majority of their data lookups to slow backend data repositories (such as databases, flatfiles and mainframes) That expensive bottleneck of data lookups and joins exists in over 80% of enterprise applications!

Extreme Scalability - Since data is evenly distributed, there is essentially no major limit to the size of the grid, except group communication on the network - which is minimized to discovery of new nodes. All data access patterns use peer-to-peer communication where nodes directly speak to each other, which Infinspan has a stated goal of near linear scalability.

Not Just for Java (PHP, Python, Ruby, C, etc.) - The roadmap has a plan for a language-independent server module. This will support both the popular memcached protocol - with existing clients for almost every popular programming language - as well as an optimized Infinispan-specific protocol. This means that Infinispan is not just useful to Java. Any major website or application that wants to take advantage of a fast data grid will be able to do so. If this will be supported in the product at the time of this writing is still being determined.

Support for Compute Grids - Also on the roadmap is the ability to pass a Runnable around the grid. You will be able to push complex processing towards the server where data is local, and pull back results using a Future. This map/reduce style paradigm is common in applications where a large amount of data is needed to compute relatively small results.

Grid Management - When running a grid on several servers, management is no longer an after thought, it becomes a necessity. This is on Infinispan's roadmap and rich tooling in this area, with many integration opportunities is planned.

1.1 Audience for this document

IT professionals interested in a data grid solution (Project Infinispan) as a means to reduce the load on backend data stores, boost the performance of applications by pushing data to the edge. Professionals may include architects, application developers and application administrators. If you are already familiar with proprietary data grid solutions – you are at the right place and can potentially leverage a supported JBoss Enterprise solution in the future which will be based on Infinispan.

1.2 Prerequisites

This is a hands on introduction to Infinispan. We will be writing Java code and launching some pre-built Infinispan examples. It is highly recommended to be Java coder, although the instructions in this document are intended to be lucid and should be able to give you a perspective to the product even if you are not a Java coder on a day to day basis.

2 INFINISPAN LAB: INTRODUCTION

This lab should take between 60-90 minutes of time.

2.1 Pre-requisites to this Lab

Inspect Lab Artifacts: All required files are in the **"Downloads"** directory on your Desktop, with the following contents / structure:

```
Downloads
      |
      |__ Infinispan
             |
             |__    Infinispan.pdf
             |
             |__    Readme_Infinispan_Lab_Commands
             |
             |
             |__    src
             |          |__ org
             |                |__infinispan
             |                        |__examples
             |                             |
             |                             |_ Node1.java
             |                             |
             |                             |_ Node2.java
             |                             |
             |                             |_ Node3.java
             |                             |
             |                             |_ CacheListener.java
             |                             |
             |                             |_ Person.java
             |
             |
             |__   classes
             |
             |
             |__   solutions
             |            |__ org
             |                |__ infinispan
             |                        |__ examples
             |                             |
             |                             |_ Node1.java
             |                 |           |
             |                             |_ Node2.java
             |                             |
             |                             |_ Node3.java
             |                             |
             |                             |_ CacheListener.java
             |                             |
             |                             |_ Person.java
             |
             |
             |__  Infinispan
                       |
                       |_ infinispan-5.0.0.BETA2-all.zip
```

Infinispan.pdf – this lab guide you are reading. Lab commands for cut-paste convenience are in the document **Readme_Infinispan_Lab_Commands**

The "**src**" directory is your working directory and contains stub classes - you will be implementing your code in these classes.

The "**classes**" directory is where your code will be compiled.

The "**solutions**" directory contains lab solutions you may refer to.

The "**Infinispan**" directory contains Infinispan software you will be installing.

2.2 What will we do in today's Infinispan workshop

As mentioned in the introductory section, Infinispan is a highly scalable, resilient data grid. In this workshop we will illustrate some introductory concepts of Infinispan. As part of the labs, we will -

1. Launch the GUI example application that comes out of the box with Infinispan. We will create a grid or cluster of multiple cache nodes, and insert literal / primitive data into the grid via the GUI interface. Once a grid is created with data loaded, we will demonstrate data resilience and data distribution by removing / adding node members from the grid.

2. Get under the hood – we will write Java classes to programmatically create a data grid and insert a simple key-value (String) data into the cache

3. We will update the cache to insert a serializable object to the data grid – moving away from using literals.

4. Attach event listeners to caches – meaning data changes are captured realtime – potentially for alerting realtime clients. We will also now cluster our Infinispan node with a set of existing nodes from the GUI demo

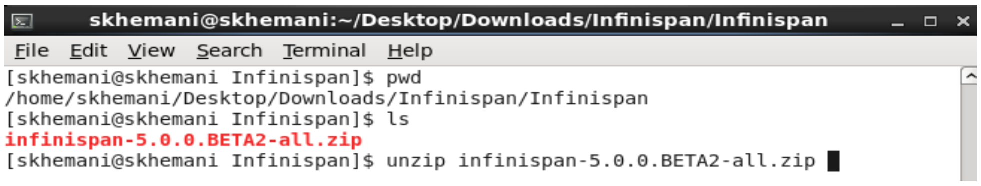
5. Discuss configuration options for the data grid.

6. Pointers to additional online resources to extend your learning of Infinispan

2.3 Set Up Lab Environment: Unzip Infinispan

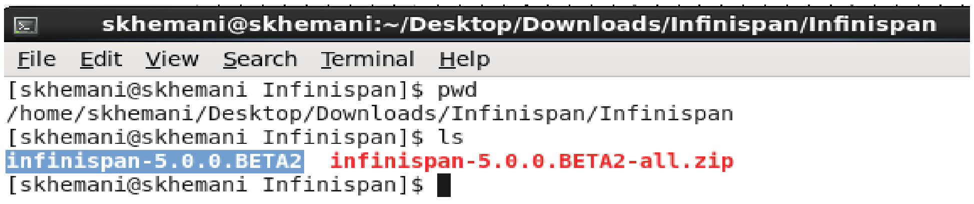
Since JAVA_HOME and the PATH environment variables are already set on your machine you are ready to install Infinispan. Note that JAVA_HOME is set to your JDK home directory and is generally required to run most JBoss projects and products. Infinispan requires JDK 1.6 or higher, and you can check this via java -version, and see which version is installed.

The next thing to do is to unzip **infinispan-5.0.0.BETA2-all.zip** – this is available at ~student/Desktop/Downloads/Infinispan/Infinispan.


skhemani@skhemani:~/Desktop/Downloads/Infinispan/Infinispan
File Edit View Search Terminal Help

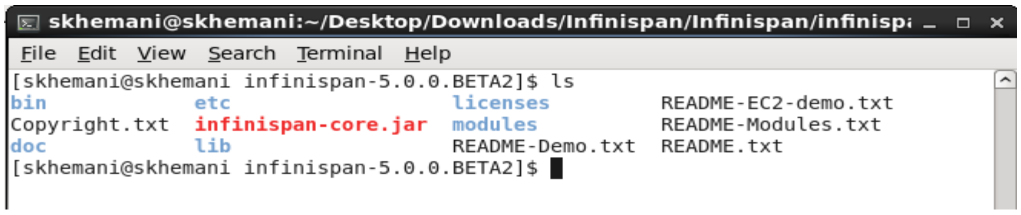
```
[skhemani@skhemani Infinispan]$ pwd
/home/skhemani/Desktop/Downloads/Infinispan/Infinispan
[skhemani@skhemani Infinispan]$ ls
infinispan-5.0.0.BETA2-all.zip
[skhemani@skhemani Infinispan]$ unzip infinispan-5.0.0.BETA2-all.zip
```

This should create a directory called **infinispan-5.0.0.BETA2** where you unzipped it (as follows)


skhemani@skhemani:~/Desktop/Downloads/Infinispan/Infinispan
File Edit View Search Terminal Help

```
[skhemani@skhemani Infinispan]$ pwd
/home/skhemani/Desktop/Downloads/Infinispan/Infinispan
[skhemani@skhemani Infinispan]$ ls
infinispan-5.0.0.BETA2  infinispan-5.0.0.BETA2-all.zip
[skhemani@skhemani Infinispan]$
```

Congratulations, you just installed **INFINISPAN**. Inspect the contents of the unzipped (installed) directory locations.


skhemani@skhemani:~/Desktop/Downloads/Infinispan/Infinispan/infinispa
File Edit View Search Terminal Help

```
[skhemani@skhemani infinispan-5.0.0.BETA2]$ ls
bin            etc                 licenses          README-EC2-demo.txt
Copyright.txt  infinispan-core.jar modules           README-Modules.txt
doc            lib                 README-Demo.txt   README.txt
[skhemani@skhemani infinispan-5.0.0.BETA2]$
```

– **infinispan-core.jar:** This binary is the core library for Infinispan. This alongwith the Jars in the lib directory mentioned below are almost always required for development and for running your Infinispan cluster
– **lib:** This directory contains the jar files that are typically required to be in the classpath along with the infinispan-core.jar mentioned above
– **bin:** This directory contains the various scripts used by infinispan – there are scripts for starting an infinispan node, starting several other examples. We will be using the runGuiDemo.sh in our lab (adjust to runGuiDemo.bat if you are using a windows development environment)
– **doc:** The Javadocs for this version of Infinispan for reference

Now a few more checks -

Check JAVA_HOME – The JAVA_HOME environment variable should be already set for you on your desktop. You can validate its value by opening a shell window [Right click on the Desktop and choose “Open in Terminal”] and run the following command ...

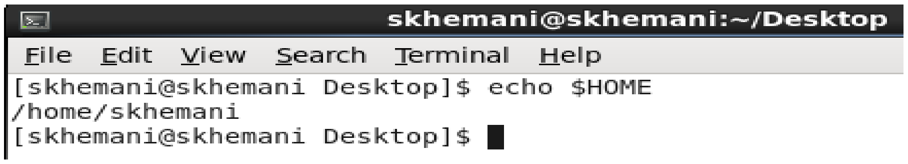
```
echo $JAVA_HOME
```

You should see the home directory location of the provided JDK 1.6+ that has already been installed on your system.

Check HOME – The HOME environment variable should also be already set for you on your desktop. You can validate its value by running the following command on your shell window ...

```
echo $HOME
```

You should see something like the following screenshot, for the provided computers it will be /home/student:


skhemani@skhemani:~/Desktop
File Edit View Search Terminal Help

```
[skhemani@skhemani Desktop]$ echo $HOME
/home/skhemani
[skhemani@skhemani Desktop]$
```

COPY - PASTE CAUTION: When copying and pasting commands from this pdf to the shell window you will see characters dropped or spaces added – your commands will not work. You should instead copy and paste from the provided document - **Readme_Infinispan_Lab_Commands**

2.4 Concept: JBoss JGroups & its Relevance to Infinispan

As mentioned above, Infinispan enables you to create a data grid – meaning that you can use memory and compute power of available physical machines to create one or more contiguous data grid/s. This distributed architecture provides resilience, dynamic scale, optimal resource utilization to store (and access) objects that otherwise would require fetches from multiple data sources and the marshalling/unmarshalling overhead. Your consuming applications (other processes or the presentation tier) will perform better and you will reduce the load on backend data sources.

However, running a data grid (or a cluster of a large number of data / cache nodes) means that the inter-process communication between the nodes must be optimized and efficient – inter-process communication is used for data transfer & balancing, grid memberships (joins and leaves), running queries, object updates, etc.

If you are familiar with JBoss, you probably are familiar with JGroups – . Infinispan re-uses JGroups for inter process communication for the cache/data grid. Lets talk a little bit more about JGroups.

What is JGroups? The JBoss JGroups framework provides services to enable peer-to-peer communications between nodes in a cluster. It is built on top a stack of network communication protocols that provide transport, discovery, reliability and failure detection, and cluster membership management services. Essentially it is a stable and mature toolkit for reliable multicast communication. That this doesn't necessarily mean IP Multicast only. JGroups can also use transports such as TCP, leveraging a point to point topology. It can be used to create groups of processes whose members can send messages to each other. The main features include -

- Group creation and deletion. Group members can be spread across LANs or WANs
- Joining and leaving of groups
- Membership detection and notification about joined/left/crashed members
- Detection and removal of crashed members
- Sending and receiving of member-to-group messages (point-to-multipoint)
- Sending and receiving of member-to-member messages (point-to-point)

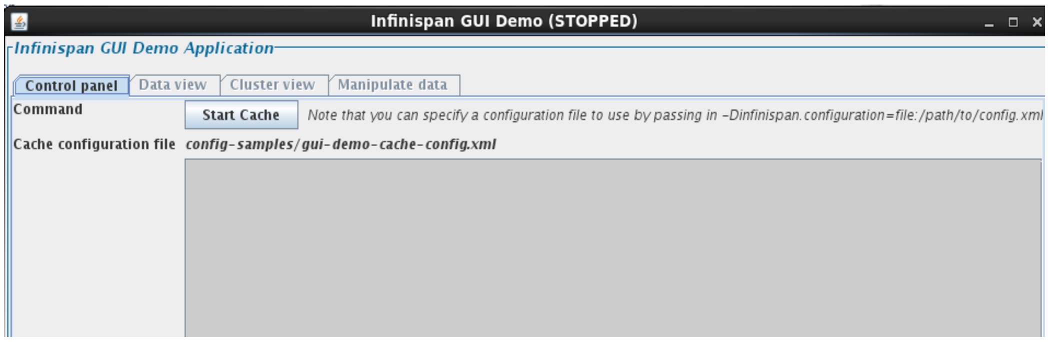
3 LAB: THE OUT OF THE BOX INFINISPAN GUI EXAMPLE

We are now ready to see some action. We will start with the sample GUI application that ships out of the box -

STEP 1**:** Start the Out of the Box Demo GUI . Open a shell window (unless already open) and execute the following commands:

```
cd ~/Desktop/Downloads/Infinispan/Infinispan/
cd infinispan-5.0.0.BETA2/bin
./runGuiDemo.sh
```

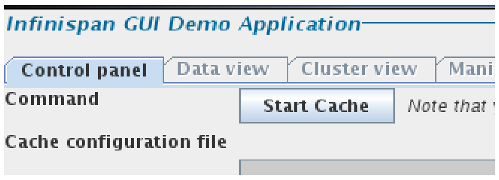
This should bring up the following swing window.



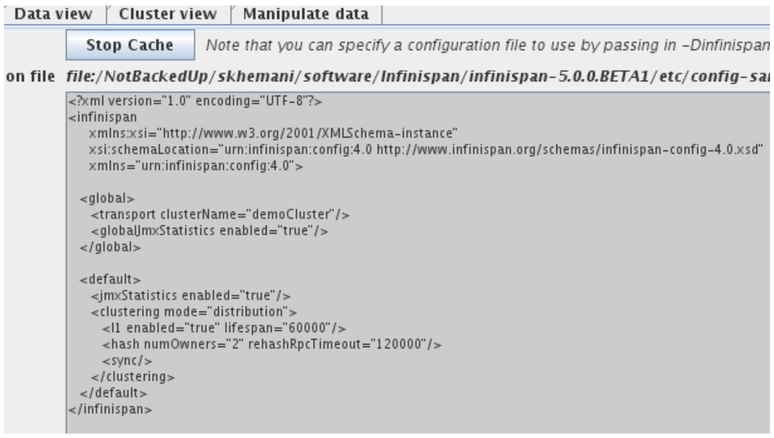
Note that you can specify a configuration file to use by passing in -Dinfinispan.configuration=file:/path/to/config.xml

config-samples/gui-demo-cache-config.xml

STEP 2**:** Start the cache in the GUI above, using the Start Cache button.



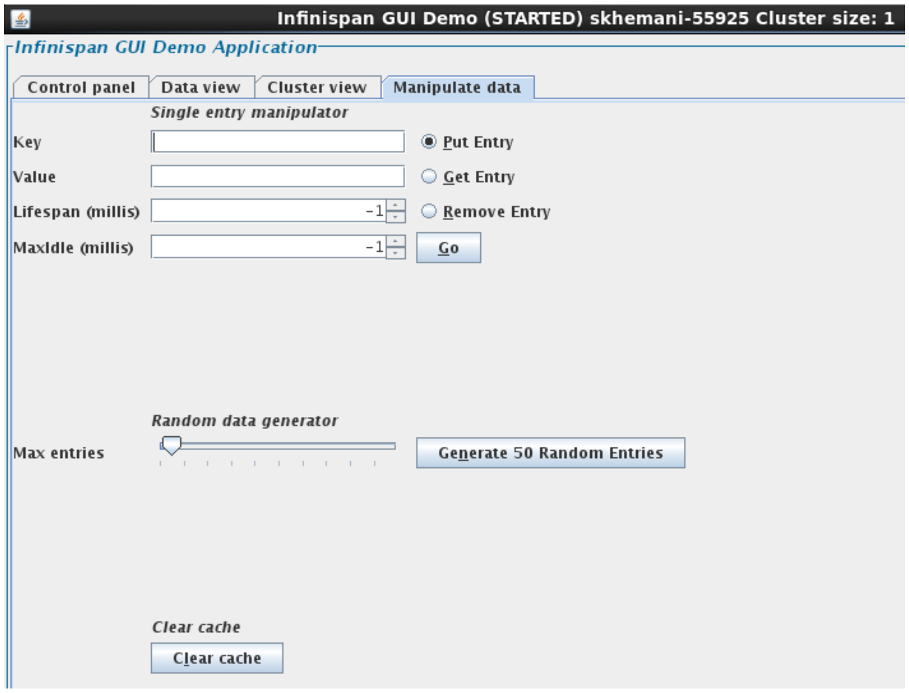
You should see that the cache starts (as screenshot below) and displays the configuration file it is using.


Data view
Cluster view
Manipulate data
Stop Cache
Note that you can specify a configuration file to use by passing in -Dinfinispan
on file
file:/NotBackedUp/skhemani/software/Infinispan/infinispan-5.0.0.BETA1/etc/config-sa

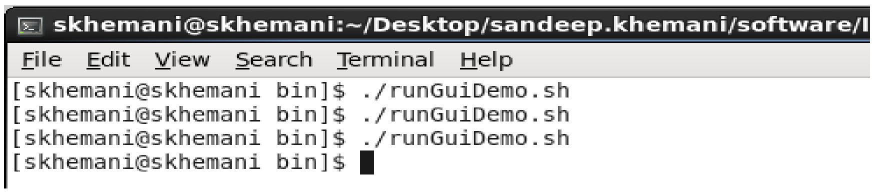
```
<?xml version="1.0" encoding="UTF-8"?>
<infinispan
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:infinispan:config:4.0 http://www.infinispan.org/schemas/infinispan-config-4.0.xsd"
    xmlns="urn:infinispan:config:4.0">

  <global>
    <transport clusterName="demoCluster"/>
    <globalJmxStatistics enabled="true"/>
  </global>

  <default>
    <jmxStatistics enabled="true"/>
    <clustering mode="distribution">
      <l1 enabled="true" lifespan="60000"/>
      <hash numOwners="2" rehashRpcTimeout="120000"/>
      <sync/>
    </clustering>
  </default>
</infinispan>
```

Click on the manipulate data button.

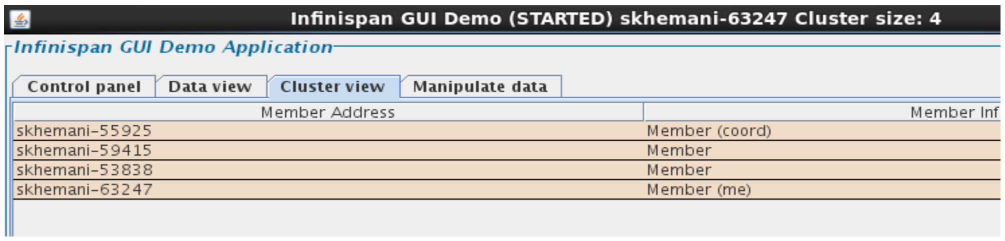
STEP 3: Manipulate data: In the Manipulate Data tab, add entries, generate random data, etc.
It might be suggest to put in 10,10: 20, 20 and 30,30



STEP 4: Start more cache instances: Repeat Steps 1 and 2 above to launch and start up more caches (ensure you hit the Start Cache button on each new window as specified in Step 2). Watch cluster formation in the Cluster View tab.


skhemani@skhemani:~/Desktop/sandeep.khemani/software/I
File Edit View Search Terminal Help

```
[skhemani@skhemani bin]$ ./runGuiDemo.sh
[skhemani@skhemani bin]$ ./runGuiDemo.sh
[skhemani@skhemani bin]$ ./runGuiDemo.sh
[skhemani@skhemani bin]$ 
```

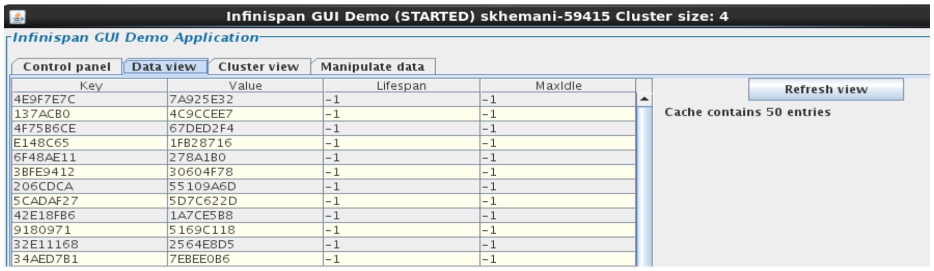

Infinispan GUI Demo (STARTED) skhemani-63247 Cluster size: 4
Infinispan GUI Demo Application
Control panel
Data view
Cluster view
Manipulate data

Member Address	Member Inf
skhemani-55925	Member (coord)
skhemani-59415	Member
skhemani-53838	Member
skhemani-63247	Member (me)

From the screenshot above, you can see that we launched the `./runGuiDemo.sh` three more times; we hit the start cache button on each and hence we see that the total of 4 nodes that started and formed a cluster.

STEP 5: Manipulate more data: Add and remove data on any of the nodes, and watch state being distributed (Note the number of entries each node contains, ie data is not copied to all the other nodes – see the statement below the "Refresh view" button).

Shut nodes down to witness data durability.


Infinispan GUI Demo (STARTED) skhemani-59415 Cluster size: 4
Infinispan GUI Demo Application
Control panel
Data view
Cluster view
Manipulate data
Refresh view

Key	Value	Lifespan	MaxIdle
4E9F7E7C	7A925E32	-1	-1
137ACB0	4C9CCEE7	-1	-1
4F75B6CE	67DED2F4	-1	-1
E148C65	1FB28716	-1	-1
6F48AE11	278A1B0	-1	-1
3BFE9412	30604F78	-1	-1
206CDCA	55109A6D	-1	-1
5CADAF27	5D7C622D	-1	-1
42E18FB6	1A7CE5B8	-1	-1
9180971	5169C118	-1	-1
32E11168	2564E8D5	-1	-1
34AED7B1	7EBEE0B6	-1	-1

Cache contains 50 entries

QUIZ:

– What happens to the data when you shut down all except 1 node?
[HINT:] Ideally if the last node has enough JVM capacity, it will hold all the data that was inserted from any where else in the data grid ...

– What happens when you re-start new nodes and kill the original node?
[HINT:] When new nodes are started, data is re-distributed to ensure additional capacity is utilized. The node holds some primary and some backup data. When other nodes die, the backup data for their primaries residing on other nodes is promoted to becoming primary

– What happens when you kill the (coord) node
[HINT:] Another node takes that responsibility

– In the control panel, you see the XML configuration details. What does **`<clustering mode="distribution">`** mean?
[HINT:] Means data is not replicated or copied to each node. Data is distributed across the available members in the data grid – each member holds its primary data and 1 or more nodes can hold copies of another member's data (for high availability)

– Inspect the contents of the XML shown – we will discuss this in a later section
– What are some usecases for Infinispan in your company?

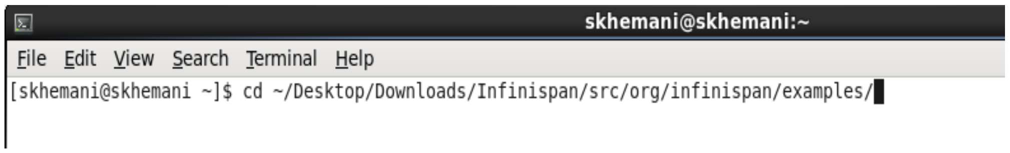
At this time, please shutdown all the GUI windows that are open to stop the running cache nodes. In the shutdown process, you can see data being re-distributed and the cluster details changing in the GUI.

4 LAB: IMPLEMENT THE CODE TO START AN INFINISPAN CLUSTER

In this lab, we will use an existing stubbed Java class (Node1.java) that uses Infinispan APIs to programmatically create an Infinispan node and place a simple key-value based data in the grid. Your lab implementation will also print out the size of the cluster. This means that as an Infinispan node comes up, it will try to cluster with existing Infinispan nodes. Follow the steps below:

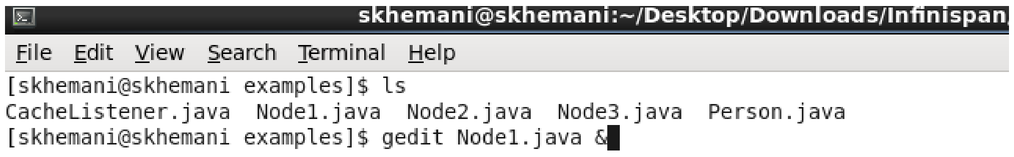
STEP 1: Open a shell window and navigate to the stubbed out lab class **Node1.java** provided to you:

```
cd ~/Desktop/Downloads/Infinispan/src/org/infinispan/examples/
```



```
[skhemani@skhemani ~]$ cd ~/Desktop/Downloads/Infinispan/src/org/infinispan/examples/
```

STEP 2: Open the file **Node1.java** using the editor of your choice (gedit, vi, emacs, nano, etc).
nano -w Node1.java


skhemani@skhemani:~/Desktop/Downloads/Infinispan
File Edit View Search Terminal Help

```
[skhemani@skhemani examples]$ ls
CacheListener.java  Node1.java  Node2.java  Node3.java  Person.java
[skhemani@skhemani examples]$ gedit Node1.java &
```

STEP 3: Add code to Node1.java (which you just opened) to implement an instance of an Infinispan node.

- To the (stubbed) **init** method add the following code to create the cache manager and get a handle to the cache we will use in the next example. A cache manager is the mechanism for retrieving a cache instance, and is used as a starting point to using the cache. Cache managers are heavyweight objects, and no more than one cache manager is to be used per JVM. Constructing a cache manager is done via one of its constructors, which optionally take in a configuration or a path or URL to a configuration XML file. Please read the code carefully before adding to the init method of Node1.java

```
//---------------------BEGIN------------------------
try{
       GlobalConfiguration gc =
                      GlobalConfiguration.getClusteredDefault();
       Configuration c = new Configuration();

       // DIST_SYNC – the cache will not replicate data across
       // all members of the cluster but distribute it
       // Updates will be synchronous to the primary and the
       // secondary copies

       c.setCacheMode(Configuration.CacheMode.DIST_SYNC);
       cacheManager = new DefaultCacheManager(gc, c);
}
catch (Exception e){
       System.out.println("Error creating custom Cache Manager."
                             + " Creating DefaultCache!");
       this.cacheManager = new DefaultCacheManager();
}
//-----------------------END-------------------------
```

- To the (stubbed) **placeItemInCache** method we need to add the following code. Please read the code carefully – it should be self explanatory.

```
//---------------------BEGIN------------------------
Cache<String, Object> cache = this.cacheManager.getCache();

cache.put("key1" , "value1");

System.out.println("key-value String pair inserted into"
                      + " infinispan cluster!");

System.out.println("The size of the cluster is: " +
                      this.cacheManager.getClusterSize());
//-----------------------END-------------------------
```

- **Compile** your code using the following command:

```
javac -cp $
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/lib/*:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/infinispan-core.jar:$
{HOME}/Desktop/Downloads/Infinispan/classes -Xlint:deprecation -d $
{HOME}/Desktop/Downloads/Infinispan/classes Node1.java
```

What did you just do?

You compiled Node1.java using a classpath that included the required libraries at:

  - `{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/lib/*`
  - `$ {HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/infinispan-core.jar`
  - `${HOME}/Desktop/Downloads/Infinispan/classes`

Your class file was generated at -

  - `{HOME}/Desktop/Downloads/Infinispan/classes`

STEP 4: Execute (and start the first node in the Infinispan cluster) using the following command:

```
java -cp $
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/lib/*:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/infinispan-core.jar:$
{HOME}/Desktop/Downloads/Infinispan/classes
-Dbind.address=127.0.0.1 -Djava.net.preferIPv4Stack=true
-Dlog4j.configuration=file:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/etc/log4j.xml org.infinispan.examples.Node1
```

What did you just do?

You executed class `org.infinispan.examples.Node1` using a classpath that included the required libraries at:

- {HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/lib/*

- $
  {HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/infinispan-core.jar

- ${HOME}/Desktop/Downloads/Infinispan/classes

You also provided the following System Parameters (-D parameters) to the JVM (explanations below)

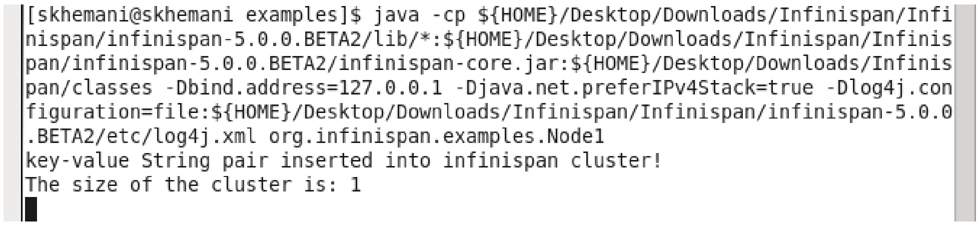
- **bind.address=127.0.0.1**
  Infinispan requires the bind.address system property to be set to function as a distributed cache

- **java.net.preferIPv4Stack=true**
  It ensures that the program uses IPv4, a system of addresses used to identify devices on a network.

- **log4j.configuration=file:${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/etc/log4j.xml**
  The default logging configuration through the log4j.xml file used by Infinispan node

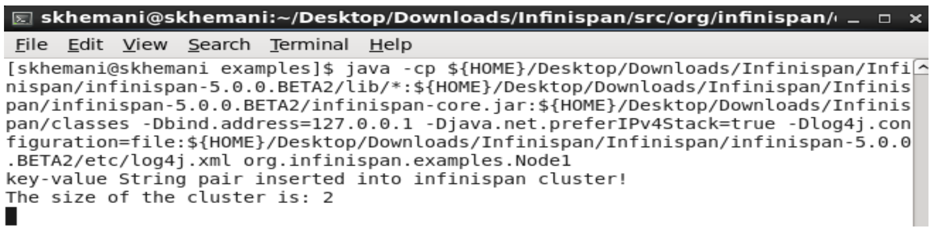
See following screenshots on what to expect ...

You should see the first node in the cluster starting up (as shown in the screenshot below) – it shows that the node started and added a simple piece of data to the cluster. It also shows that the size of the cluster is 1, given that this is the first node in the cluster.



```
[skhemani@skhemani examples]$ java -cp ${HOME}/Desktop/Downloads/Infinispan/Infi
nispan/infinispan-5.0.0.BETA2/lib/*:${HOME}/Desktop/Downloads/Infinispan/Infinis
pan/infinispan-5.0.0.BETA2/infinispan-core.jar:${HOME}/Desktop/Downloads/Infinis
pan/classes -Dbind.address=127.0.0.1 -Djava.net.preferIPv4Stack=true -Dlog4j.con
figuration=file:${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0
.BETA2/etc/log4j.xml org.infinispan.examples.Node1
key-value String pair inserted into infinispan cluster!
The size of the cluster is: 1
```

STEP 5: Open a new shell window (cntrl-shift-t) make this really easy, and start a second node in the Infinispan cluster using the same java command as in STEP 4. You should see that a second node started up and it shows that the size of the cluster now is 2. You may choose to launch 1-2 more nodes using the same command (WARNING: The number of nodes you can launch is a function of the available memory and the compute capacity of the system. See screenshot below.


skhemani@skhemani:~/Desktop/Downloads/Infinispan/src/org/infinispan/
File Edit View Search Terminal Help

```
[skhemani@skhemani examples]$ java -cp ${HOME}/Desktop/Downloads/Infinispan/Infi
nispan/infinispan-5.0.0.BETA2/lib/*:${HOME}/Desktop/Downloads/Infinispan/Infinis
pan/infinispan-5.0.0.BETA2/infinispan-core.jar:${HOME}/Desktop/Downloads/Infinis
pan/classes -Dbind.address=127.0.0.1 -Djava.net.preferIPv4Stack=true -Dlog4j.con
figuration=file:${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0
.BETA2/etc/log4j.xml org.infinispan.examples.Node1
key-value String pair inserted into infinispan cluster!
The size of the cluster is: 2
```

At this time, please shutdown all running Infinispan cache nodes.

5 LAB: NO STRINGS ATTACHED – WORKING WITH OBJECTS

Thus far data inserted into the data grid has been Strings – lets insert a serializable object into the data grid.

For this section, we will use the **Person.java** class at the following location
~/Desktop/Downloads/Infinispan/src/org/infinispan/examples/Person.java

This is a simple Java class that just has 2 String attributes – firstname and lastname, a constructor and mutators (setXXX) and accessors (getXXX) implemented. The Person class implements:

```
java.io.Serializable
```

Open a shell window and navigate to

```
~/Desktop/Downloads/Infinispan/src/org/infinispan/examples/
```

Open (gedit, nano, or vi) and Update **Node2.java** at the location and ensure you are inserting a Person object in the API **`placeItemInCache()`**
`nano -w Node2.java` :

```
cache.put("person1" , new Person("a", "b"));
```

Inspect and compile Person.java using the following command:

```
javac -d ${HOME}/Desktop/Downloads/Infinispan/classes Person.java
```

Compile Node2.java using the following command:

```
        javac -cp ${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/lib/*:${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/infinispan-core.jar:${HOME}/Desktop/Downloads/Infinispan/classes
-Xlint:deprecation -d ${HOME}/Desktop/Downloads/Infinispan/classes Node2.java
```

Execute Node2 using the following command:

```
        java -cp ${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/lib/*:${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/infinispan-core.jar:${HOME}/Desktop/Downloads/Infinispan/classes
-Dbind.address=127.0.0.1 -Djava.net.preferIPv4Stack=true
-Dlog4j.configuration=file:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/etc/log4j.xml org.infinispan.examples.Node2
```

"You should see that the Person object was inserted into the infinispan cluster."

At this time, please shutdown all running Infinispan cache nodes.

6 LAB: USING A CONFIG XML FOR A CUSTOM CACHE CONFIGURATION

In your projects it's likely that you want to provide a custom configuration. You have two options to configure Infinispan: programmatically (as above Labs illustrate) or by using a configuration file. In this Lab we will be using the same configuration (via XML) as used for the GUI demo. This configuration file is located at

~/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/etc/config-samplesgui-demo-cache-config.xml

Lets explain the XML code in the file – refer to the XML comments below for explanations:

```
<?xml version="1.0" encoding="UTF-8"?>

           <!-- ------------------------------------------------------
           'infinispan' is the root element: the minimum required
           for a configuration file
           ----------------------------------------------------- -->

<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">

           <!-- ------------------------------------------------------
           'global' contains System wide local settings
           ------------------------------------------------------ -->

   <global>

           <!-- ------------------------------------------------------
           'transport' configures transport used for network
           communications across the cluster. 'clusterName'defines the
           cluster name & nodes connect to clusters sharing the same name.
           'globalJmxStatistics' - if global statistics are gathered and
           reported via JMX for all caches under this cache manager.
           ------------------------------------------------------ -->

      <transport clusterName="demoCluster"/>
      <globalJmxStatistics enabled="true"/>
   </global>

           <!-- ------------------------------------------------------
           'default' is the configuration used by all Infinispan caches.
           ------------------------------------------------------ -->

   <default>
      <jmxStatistics enabled="true"/>

           <!-- ------------------------------------------------------
           'clustering' defines clustered characteristics of the cache.
           ------------------------------------------------------ -->
      <clustering mode="distribution">

           <!-- ------------------------------------------------------
           'l1'configures the L1 cache behavior in 'distributed' caches
           instances. 'lifespan' is the max lifespan of an entry placed in
           the cache. 'hash' allows fine-tuning of rehashing characteristics
           in distributed cache mode. 'numOwners' is # of cluster-wide
           replicas for each cache entry. 'sync' means communications are
           synchronous and blocking until it receives an acknowledgement
           ------------------------------------------------------ -->

         <l1 enabled="true" lifespan="60000"/>
         <hash numOwners="2" rehashRpcTimeout="120000"/>
         <sync/>
      </clustering>
   </default>
</infinispan>
```

A Configuration file can also defined named caches as follows. Named caches define custom specialized caches:

```
<infinispan>
    <global />
    <default />
```

```
        <namedCache  name="A">
        <namedCache  name="B">
    </infinispan>
```

If you want to define a custom cache for evicting data from the cache you could use the following syntax:

```
<infinispan>
  <namedCache name="evictionCache">
     <eviction wakeUpInterval="500" maxEntries="5000" strategy="FIFO" />
     <expiration lifespan="60000" maxIdle="1000"/>
  </namedCache>
</infinispan>
```

If you want to refer all the configuration options, please refer to the document at -
http://docs.jboss.org/infinispan/4.0/apidocs/config.html#ce__infinispan

More details about configuring a cache programmatically: This is accomplished through an instance of the org.infinispan.config.Configuration class where you can set custom attributes.

```
Configuration c = new Configuration();
c.setExpirationLifespan(10000);
manager.defineConfiguration("myconfig", c);
Cache cache = manager.getCache("myconfig");
```

STEP 1: In this lab we will enable our java program to use the same config XML file as was used by the GUI example / lab we completed in SECTION 3 LAB: THE OUT OF THE BOX INFINISPAN GUI EXAMPLE

Open a shell window.

Open the lab source code (**Node3.java**) at -
~/Desktop/Downloads/Infinispan/src/org/infinispan/examples

The location of this XML file will be passed to the java process (Node3) as a commandline program argument.

In the API **init()** instantiate the variable cacheManager using the **configurationFileLocation -** the following is the code to do so ...

```
this.cacheManager
       = new DefaultCacheManager(configurationFileLocation, true);
```

Explaining the parameters above:

- **configurationFile** - name of configuration file to use as a template for all caches created
- **start** - if true, the cache manager is started

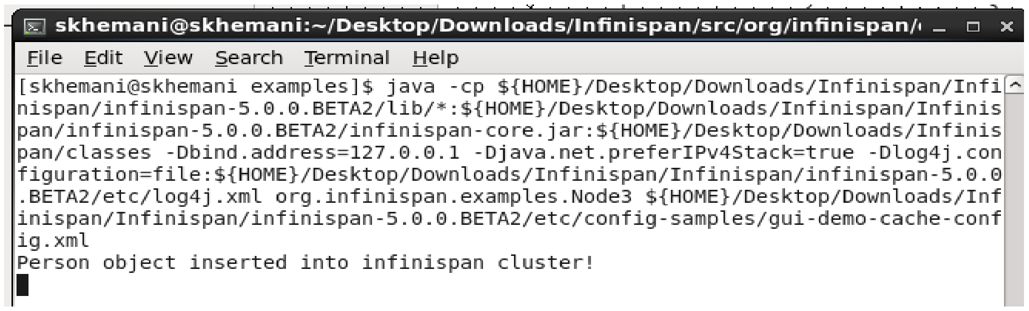
STEP 2: On a shell window and at the right location where your lab version of Node3.java exists, compile Node3.java using the following command:

```
javac -cp ${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/lib/*:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/infinispan-core.jar:$
{HOME}/Desktop/Downloads/Infinispan/classes -Xlint:deprecation -d $
{HOME}/Desktop/Downloads/Infinispan/classes Node3.java
```

STEP 3: Execute Node3 using the following command (pay attention to the commandline program argument after the classname

```
java -cp ${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/lib/*:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/infinispan-core.jar:$
{HOME}/Desktop/Downloads/Infinispan/classes -Dbind.address=127.0.0.1
-Djava.net.preferIPv4Stack=true -Dlog4j.configuration=file:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/etc/log4j.xml org.infinispan.examples.Node3 $
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/etc/config-samples/gui-demo-cache-config.xml
```

Your cache node should come up with a Person object inserted as below.


skhemani@skhemani:~/Desktop/Downloads/Infinispan/src/org/infinispan/
File Edit View Search Terminal Help

```
[skhemani@skhemani examples]$ java -cp ${HOME}/Desktop/Downloads/Infinispan/Infi
nispan/infinispan-5.0.0.BETA2/lib/*:${HOME}/Desktop/Downloads/Infinispan/Infinis
pan/infinispan-5.0.0.BETA2/infinispan-core.jar:${HOME}/Desktop/Downloads/Infinis
pan/classes -Dbind.address=127.0.0.1 -Djava.net.preferIPv4Stack=true -Dlog4j.con
figuration=file:${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0
.BETA2/etc/log4j.xml org.infinispan.examples.Node3 ${HOME}/Desktop/Downloads/Inf
inispan/Infinispan/infinispan-5.0.0.BETA2/etc/config-samples/gui-demo-cache-conf
ig.xml
Person object inserted into infinispan cluster!
```

STEP 4: We will now start a GUI demo to cluster our programmatic node with a GUI launched node.

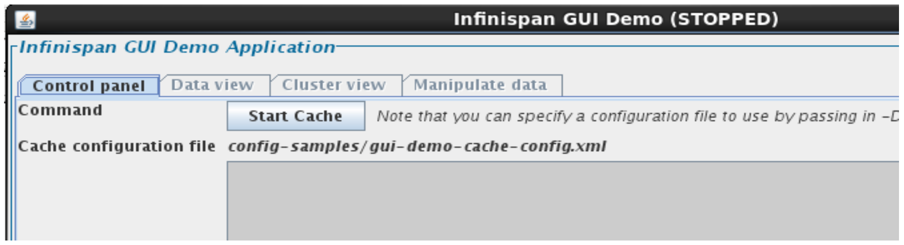
Open another shell window. Change the directory location to where the demo script is location (Do not launch it yet)

```
cd ~/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/bin
```

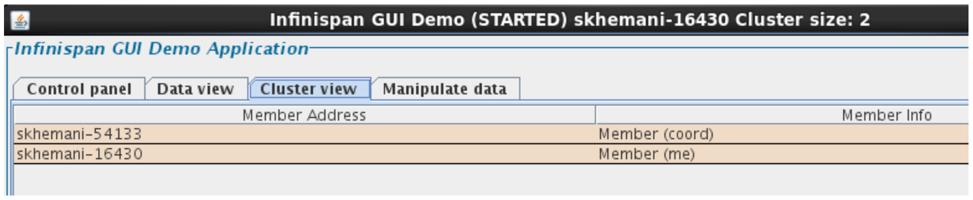
Run the following command to add the location of our compiled Person.class for the GUI demo – the demo script internally will append its classpath to this CP variable.

```
export CP=${HOME}/Desktop/Downloads/Infinispan/classes

./runGuiDemo.sh
```

Click on the **Start Cache** button.

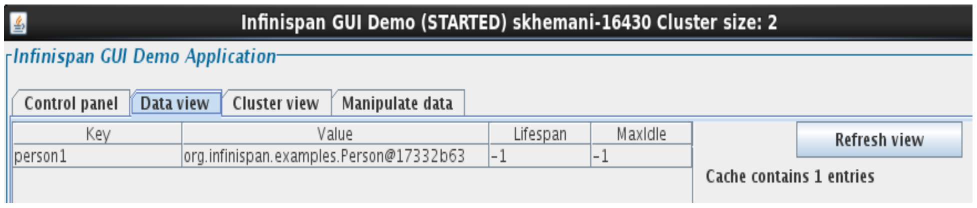

Infinispan GUI Demo (STOPPED)
Infinispan GUI Demo Application
Control panel
Data view
Cluster view
Manipulate data
Command
Start Cache
Note that you can specify a configuration file to use by passing in -D
Cache configuration file
config-samples/gui-demo-cache-config.xml

Click on the **Cluster View** tab to check if the GUI demo node has clustered with your Node3


Infinispan GUI Demo (STARTED) skhemani-16430 Cluster size: 2
Infinispan GUI Demo Application
Control panel
Data view
Cluster view
Manipulate data

Member Address	Member Info
skhemani-54133	Member (coord)
skhemani-16430	Member (me)

Click on the **Data View** tab to check if the Person object inserted from Node3 appears in the data view.


Infinispan GUI Demo (STARTED) skhemani-16430 Cluster size: 2
Infinispan GUI Demo Application
Control panel
Data view
Cluster view
Manipulate data
Refresh view

Key	Value	Lifespan	MaxIdle
person1	org.infinispan.examples.Person@17332b63	-1	-1

Cache contains 1 entries

Feel free to add additional data from the GUI demo, open mutiple GUI windows and test the resilience of data by stopping some cache nodes.

At this time, please shutdown all running Infinispan cache nodes.

7 LAB: REGISTERING EVENT LISTENERS TO OBSERVE EVENTS

In this section we will discuss Infinispan's event notification system and actually attach a simple Listener on a cache we create.

This mechanism allows you to receive notifications when interesting things happen. The API in question is encapsulated by the `Listenable` interface. Both `Cache` and `CacheManager` interfaces extend `Listenable`.

Registering listeners: Listeners themselves are simple POJO instances. These POJOs need to be annotated with `@Listener`. Listener instances themselves should expose one or more methods which are invoked when events happen. These methods should then be annotated with the event it is interested in.

Events that occur on the `Cache` interface are represented by annotations in the `org.infinispan.notifications.cachelistener.annotation` package, such as the `@CacheEntryModified` annotation.

Similarly, events that occur on the `CacheManager` interface are represented by annotations in the `org.infinispan.notifications.cachemanagerlistener.annotation` package, such as `@CacheStarted`.

Methods on listeners: Listener methods annotated with these events must be public, return a void, and take in a single parameter representing the event type, or something that the event type can be assigned to. For example, a method annotated with @CacheEntryModified may look like the following:

```
@CacheEntryModified
public void handle(CacheEntryModifiedEvent e) {}
```

or even:

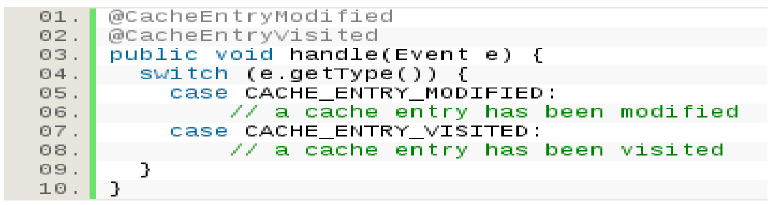
```
@CacheEntryModified
public void handle(Event e) {}
```

Multiple annotations can be placed on the same method, too:

```
@CacheEntryModified
@CacheEntryVisited
public void handle(Event e) {}
```

... because both `CacheEntryModifiedEvent` and `CacheEntryVisitedEvent` can be assigned to `Event.`

You may want to know what type of event you've received. You have two ways of doing this; either testing the type of the event passed in (using `instanceof`) or by inspecting the results of `Event.getType()` which returns an enumeration, which allows use within a switch block.



```
@CacheEntryModified
@CacheEntryVisited
public void handle(Event e) {
    switch (e.getType()) {
        case CACHE_ENTRY_MODIFIED:
            // a cache entry has been modified
        case CACHE_ENTRY_VISITED:
            // a cache entry has been visited
    }
}
```

Event ordering: Event notifications are fired both before and after an event happens. So if you register a listener interested in, say, `@CacheEntryModified`, your listener will be called both before and after the event takes place. Querying `Event.isPre()` will tell you whether the callback is before or after the event takes place.

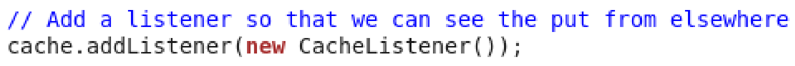
Threads and notification dispatching: Notifications are, by default, dispatched synchronously. That is, the callback your listener receives happens on the same thread that causes the event. This means that if your listener performs tasks that could be slow, the original caller's thread that triggered the event will block until your listener completes.

This side-effect can be undesirable for certain applications, so the general recommendation is that your listener implementations must not perform any long-running tasks, or tasks that could block. If you do need to perform such tasks, annotate your listener with `@Listener(sync = false)` to force asynchronous dispatch of notifications for this listener. This means that notifications will be invoked by a separate thread pool, and won't block the original caller's thread that triggered the event.

Let us introduce you to the event listener technology within Infinispan.

STEP 1: Open and inspect the `CacheListener.java,` a class available at ~/Desktop/Downloads/Infinispan/src/org/infinispan/examples. Based on the concepts explained above, uncomment the annotations for data added (`@CacheEntryCreated`) and removed (`@CacheEntryRemoved`) from the cache. Also uncomment @Listener at the class level and the 2 `System.out.println` statements so we can (later) visually see our APIs being called.
You will uncomment five lines.

STEP 2: Associate that listener class to the API placeItemInCache() in class Node4.java [~/Desktop/Downloads/Infinispan/src/org/infinispan/examples]. The code to be added just below line - `Cache<String, Object> cache = this.cacheManager.getCache()` is :



```
// Add a listener so that we can see the put from elsewhere
cache.addListener(new CacheListener());
```

STEP 3: Open a shell window, navigate to location **/Desktop/Downloads/Infinispan/src/org/infinispan/examples** and compile classes CacheListener.java and Node4.java using the command below:

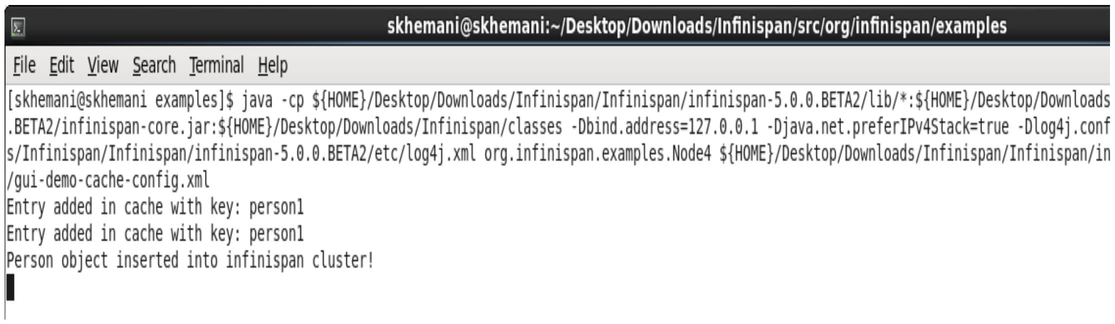
```
                javac -cp $
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/lib/*:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/infinispan-
core.jar:${HOME}/Desktop/Downloads/Infinispan/classes -Xlint:deprecation -d $
{HOME}/Desktop/Downloads/Infinispan/classes CacheListener.java
```

```
                javac -cp $
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/lib/*:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/infinispan-
core.jar:${HOME}/Desktop/Downloads/Infinispan/classes -Xlint:deprecation -d $
{HOME}/Desktop/Downloads/Infinispan/classes Node4.java
```

STEP 4: Run Node4 using the following command -

```
                java -cp $
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/lib/*:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/infinispan-
core.jar:${HOME}/Desktop/Downloads/Infinispan/classes -Dbind.address=127.0.0.1
-Djava.net.preferIPv4Stack=true -Dlog4j.configuration=file:$
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-
5.0.0.BETA2/etc/log4j.xml org.infinispan.examples.Node4 $
{HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/etc/config-
samples/gui-demo-cache-config.xml
```

You should see something like the following screenshot – you should notice that the event was fired upon inserting the Person object to the cache

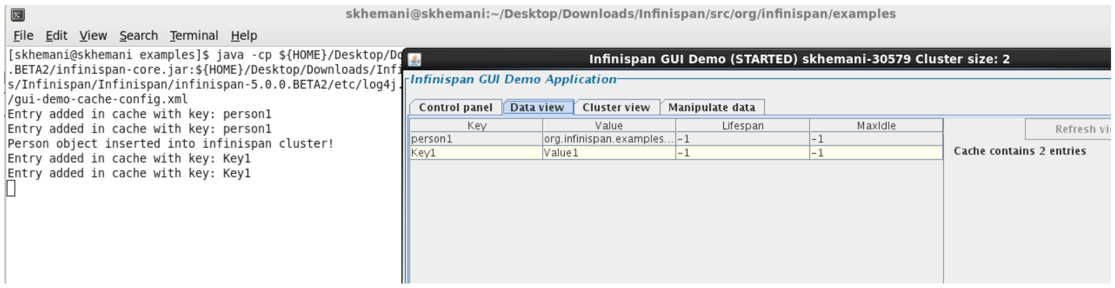

skhemani@skhemani:~/Desktop/Downloads/Infinispan/src/org/infinispan/examples
File Edit View Search Terminal Help

```
[skhemani@skhemani examples]$ java -cp ${HOME}/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/lib/*:${HOME}/Desktop/Downloads
.BETA2/infinispan-core.jar:${HOME}/Desktop/Downloads/Infinispan/classes -Dbind.address=127.0.0.1 -Djava.net.preferIPv4Stack=true -Dlog4j.conf
s/Infinispan/Infinispan/infinispan-5.0.0.BETA2/etc/log4j.xml org.infinispan.examples.Node4 ${HOME}/Desktop/Downloads/Infinispan/Infinispan/in
/gui-demo-cache-config.xml
Entry added in cache with key: person1
Entry added in cache with key: person1
Person object inserted into infinispan cluster!
```

STEP 5: Bring up GUI demo (instructions below). Start by opening up a shell command.

```
cd ~/Desktop/Downloads/Infinispan/Infinispan/infinispan-5.0.0.BETA2/bin

export CP=${HOME}/Desktop/Downloads/Infinispan/classes

./runGuiDemo.sh
```

Start the cache and insert data from the GUI window. You should see your `CacheListener` picking up the data inserted event and events printed on your command line shell.


skhemani@skhemani:~/Desktop/Downloads/Infinispan/src/org/infinispan/examples
File Edit View Search Terminal Help
Infinispan GUI Demo (STARTED) skhemani-30579 Cluster size: 2
Infinispan GUI Demo Application
Control panel
Data view
Cluster view
Manipulate data
Refresh vi
Cache contains 2 entries

```
[skhemani@skhemani examples]$ java -cp ${HOME}/Desktop/Do
.BETA2/infinispan-core.jar:${HOME}/Desktop/Downloads/Infi
s/Infinispan/Infinispan/infinispan-5.0.0.BETA2/etc/log4j.
/gui-demo-cache-config.xml
Entry added in cache with key: person1
Entry added in cache with key: person1
Person object inserted into infinispan cluster!
Entry added in cache with key: Key1
Entry added in cache with key: Key1
```

Key	Value	Lifespan	MaxIdle
person1	org.infinispan.examples...	-1	-1
Key1	Value1	-1	-1

You have successfully completed all the hands on labs. At this time, please shutdown all running Infinispan cache nodes.

8 ADDITIONAL RESOURCES ON PROJECT INFINISPAN

What is Infinispan? http://community.jboss.org/wiki/WhatisInfinispan

The Infinispan user guide is at - http://community.jboss.org/wiki/infinispan

The Project Infinispan page is at http://www.jboss.org/infinispan

A JBoss application can be configured to use Infinispan as the Hibernate 2nd-level cache, replacing JBoss Cache. William DeCoste explains in more detail at -
http://community.jboss.org/wiki/InfinispanasHibernate2nd-LevelCacheinJBossAS5x

Infinispan can optionally be configured with one or several cache stores allowing it to store data in a persistent location such as shared JDBC database, a local filesystem, etc. Using a Cache store requires to define a Cache with a loader element. Infinispan can handle updates to the cache store in two different ways:

Write-Through (Synchronous)
Write-Behind (Asynchronous)

More details at - http://community.jboss.org/wiki/Write-ThroughAndWrite-BehindCaching

A tutorial that defines a cache which uses a Flat file Cache store is available at the following location (courtesy Francesco Marchioni)
http://www.mastertheboss.com/jboss-application-server/249-infinispan-tutorial-part-2.html