**Hazelcast Persistence with MySQL**

**What is hazelcast?**

Before going into details of hazelcast we can just talk about java collections. java collections are used to store data in a different formats. like List, Set and **Map** etc.

but these are restricted to single Thread. :(

what if we want to share same collection in different Threads we go for concurrent implementations like **ConcurrentHashMap**.

in the next level what if we want to share the same collection Object across different JVMs?

the answer is **Hazelcast.**

**definition**: hazelcast is a in-memory datagrid based on java. data is evenly distributed among the nodes. This allows for horizontal scalability both in terms of available storage space and processing power. hence it being used as most popular cache framework for java based applications. it is highly available for distributed cache for applications.

**Why Hazelcast?**

**There are three main reasons why we prefer to go for hazelcast**

1. One of the main features of Hazelcast is not having a master node. Each node in the cluster is configured to be the same in terms of functionality. The oldest node manages the cluster members, i.e. automatically performs the data assignment to nodes. When a new node joins to the cluster or a node goes down, this data assignment is repeated across the nodes and the data distribution comes to a balance again.

2. Another main feature is the data being persisted entirely in-memory. This is fast. In the case of a failure, such as a node crash, no data will be lost since Hazelcast keeps copies of data across all the nodes of cluster. Data is kept in partition slices and each partition slice is owned by a node and backed up on another node.

3. there is an option to persist data permanently to relational database like MySQL.

4. Hazelcast supports a number of distributed collections and features.like Maps ,List, Set and Queue.

5. it is open source and entire framework comes in a small JAR file which you can easily add to your classpath. it also supports client server model.

**Hazelcast's Distinctive Strengths**

* It is open source.
* It is a small JAR file. You do not need to install software.
* It is a library, it does not impose an architecture on Hazelcast users.
* It provides out of the box distributed data structures (i.e. Map, Queue, MultiMap, Topic, Lock, Executor, etc.).
* There is no "master", so no single point of failure in Hazelcast cluster; each node in the cluster is configured to be functionally the same.
* When the size of your memory and compute requirement increases, new nodes can be dynamically joined to the cluster to scale elastically.
* Data is resilient to node failure. Data backups are distributed across the cluster. This is a big benefit when a node in the cluster crashes: data will not be lost.
* Nodes are always aware of each other: they communicate, unlike traditional key-value caching solutions.
* You can build your own custom distributed data structures using the Service Programming Interface (SPI) if you are not happy with the data structures provided.

Hazelcast is a fit when you need:

* analytic applications requiring big data processing by partitioning the data,
* to retain frequently accessed data in the grid,
* a cache, particularly an open source JCache provider with elastic distributed scalability,
* a primary data store for applications with utmost performance, scalability and low-latency requirements,
* an In-Memory NoSQL Key Value Store,
* publish/subscribe communication at highest speed and scalability between applications,
* applications that need to scale elastically in distributed and cloud environments,
* a highly available distributed cache for applications,
* an alternative to Coherence, Gemfire and Terracotta.

## Hazelcast Cluster Discovery

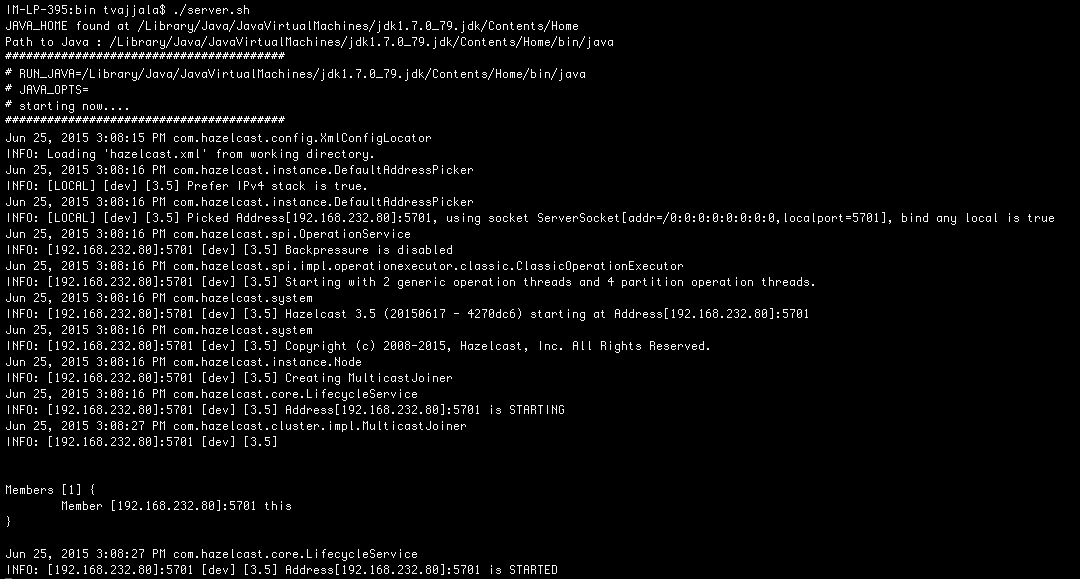
A Hazelcast cluster is a network of cluster members that run Hazelcast. Cluster members (also called nodes) automatically join together to form a cluster. This automatic joining takes place with various discovery mechanisms that the cluster members use to find each other. Hazelcast uses the following discovery mechanisms.

* [Multicast Auto-discovery](http://docs.hazelcast.org/docs/latest/manual/html/hazelcastclusters.html#multicast-auto-discovery)
* [Discovery by TCP](http://docs.hazelcast.org/docs/latest/manual/html/hazelcastclusters.html#discovery-by-tcp)
* [EC2 Cloud Auto-discovery](http://docs.hazelcast.org/docs/latest/manual/html/hazelcastclusters.html#ec2-cloud-auto-discovery)

**Getting Started with Terminal**

1. make sure you have JDK installed and configured in your system.
2. download hazelcast binary distribution from [http://www.hazelcast.org/download](http://www.hazelcast.org/download/)
3. Extract the archive and go to lib folder which contains different jars file
4. go to bin folder and run the script file and console look like this.
5. This script basically runs the below java class (which runs as server)

**com.hazelcast.core.server.StartServer**



6. There is another way we can run or connect to the hazelcast using below class

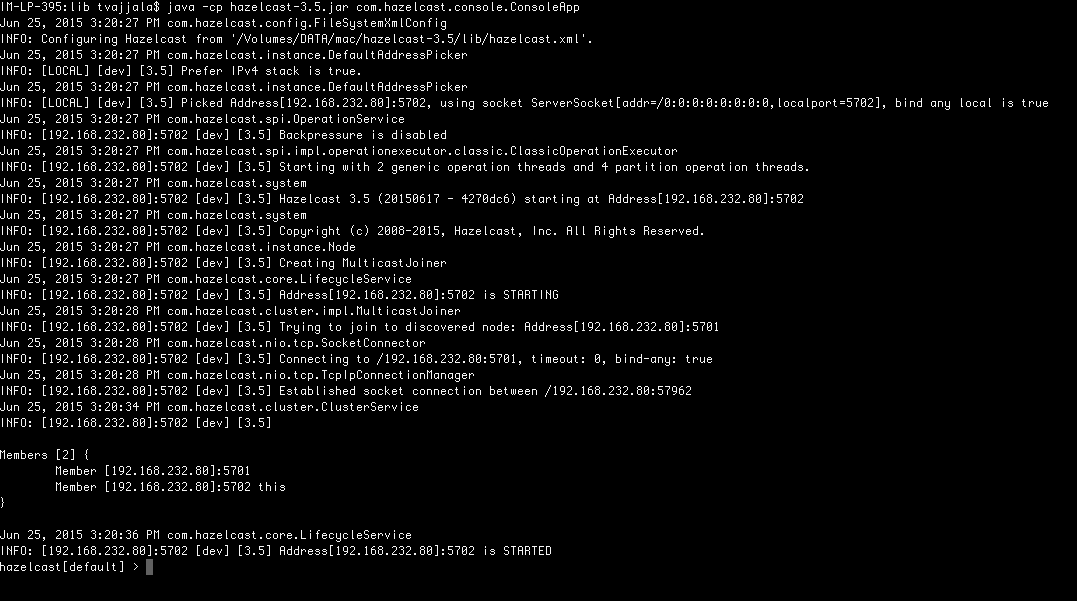
**com.hazelcast.console.ConsoleApp ( on hazelcast-3.5 )**

**NOTE** : older versions it was **com.hazelcast.examples.TestApp**

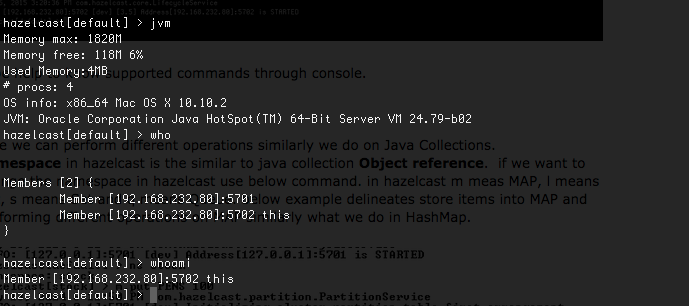
7. Go to lib folder and run the below command and could see the terminal output where you can perform some operations as well . ( create / add /delete data into collection)

**java -cp hazelcast-3.5.jar com.hazelcast.console.ConsoleApp**

8. Below screen it opens hazelcast console to perform operations as we do through programmatically.



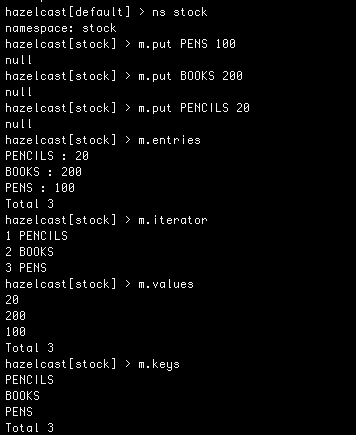
Type help to know supported commands through console.



There is concept called **namespace** which similar to object reference in java programing.

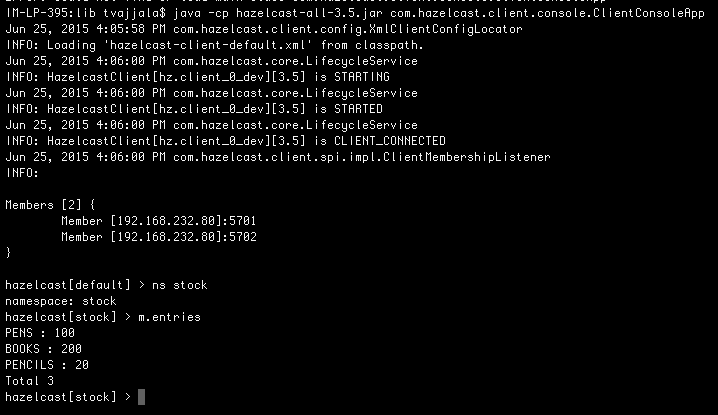
when first time start console it will have namespace value ***default***. below are some simple commands to know about your cluster information via console.

you can change namespace using **ns.**  and add elements into it. and it support four major collections types those short names are **m,l,s,q** and **mm** which is for **map, list, set, queue** and **multimap** respectively. below example show adding some elements to the map with namespace stock and iterate all the elements. and i’ll write java client which reads and print data what we added via console.



now we have some data stored in the hazelcast through console Application. we need to read this data through java client.

NOTE: hazelcast provides default console client which just connects to existing cluster but doesn’t create new node in the cluster.



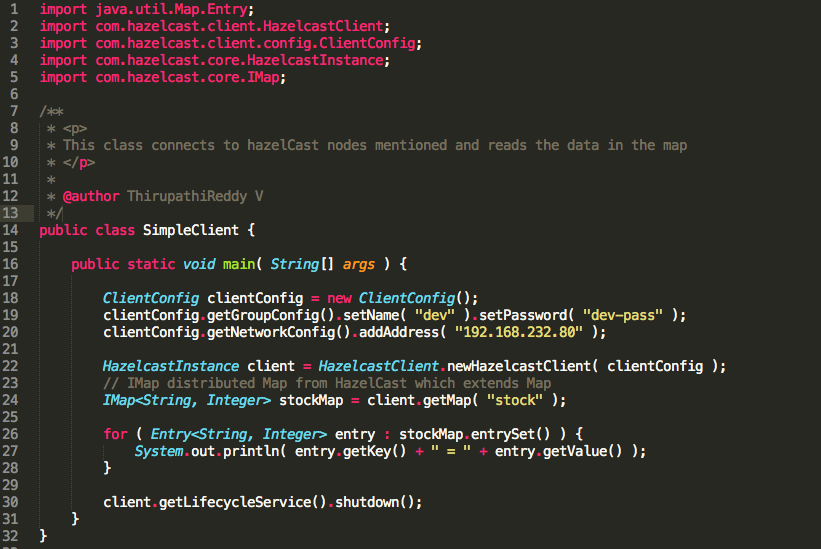
**Hazelcast Java Client Application**

Client configuration can be possible in two ways.

1. Using java **ClientConfig.java** file which used to set the required data.
2. Using hazelcast-client.xml you can file here.

for both the scenarios you have to use **HazelcastClient.java** class which has static method newHazelcastClient().

below is the code snippet which uses java based configuration to connect to node and reads the data.



in the second approach it looks for hazelcast-client.xml file in the class path.

what you need is just call the zero argument method and place xml file in the classpath.

**Adding Spring Flavor** 

spring support easy way of integrating hazelcast into our applications. use below dependency along with hazelcast dependencies to work with spring-hazelcast schema.

<dependency>

<groupId>com.hazelcast</groupId>

<artifactId>hazelcast-spring</artifactId>

<version>3.5</version>

</dependency>

you can get the latest version of maven dependecy or direct JAR from <http://mvnrepository.com/artifact/com.hazelcast/hazelcast-spring>

add required namespace to your spring configuration xml file.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:hz="http://www.hazelcast.com/schema/spring"

xmlns:cache="http://www.springframework.org/schema/cache"

xsi:schemaLocation=

"http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.hazelcast.com/schema/spring

http://www.hazelcast.com/schema/spring/hazelcast-spring-3.5.xsd

http://www.springframework.org/schema/cache http://www.springframework.org/schema/cache/spring-cache.xsd">

</beans>

now you can add the possible hazelcast configuration inside spring configuration file with the namespace prefix **hz.**

**Persisting in-memory datastore into MySQL**

implement **com.hazelcast.core.MapStore** interface and write db persistence logic which we store map data into database.

package com.trvajjala.model.store;

import java.io.ByteArrayInputStream;

import java.io.ObjectInputStream;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.Collection;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Set;

import org.apache.log4j.Logger;

import org.springframework.dao.DataAccessException;

import org.springframework.dao.EmptyResultDataAccessException;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.PreparedStatementCallback;

import com.hazelcast.core.MapStore;

/\*\*

\*

\* mapstore will persist into mysql Database

\*

\*/

public class MySQLMapStore implements MapStore<String, Object>, SQLQuery {

private static Logger logger = Logger.getLogger(MySQLMapStore.class

.getSimpleName());

{

logger.info("Hazelcast MySQL MapStore Instance Created. ");

}

/\*\*

\* jdbcTemplate will handle DB connection

\*/

private JdbcTemplate jdbcTemplate;

/\*\*

\* setting method injected through IOC

\*

\* @param jdbcTemplate

\*/

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

/\*\*

\* load object from Database using given KEY

\*/

public Object load(String key) {

logger.info(" loading persisted entry " + key);

try {

byte[] bytes = jdbcTemplate.queryForObject(SELECT\_QUERY,

new Object[] { key }, byte[].class);

return deserialize(bytes);

} catch (EmptyResultDataAccessException e) {

return null;

}

}

public Map<String, Object> loadAll(Collection<String> keys) {

Map<String, Object> map = new HashMap<String, Object>();

for (Iterator<String> iterator = keys.iterator(); iterator.hasNext();) {

String map\_key = iterator.next();

Object map\_value = load(map\_key);

map.put(map\_key, map\_value);

}

logger.info(" total " + map.size() + " objects loaded ");

return map;

}

public Set<String> loadAllKeys() {

List<String> list = jdbcTemplate.queryForList(ALL\_KEYS, String.class);

logger.info(" total keys found " + list.size());

return new HashSet<String>(list);

}

public void delete(final String key) {

jdbcTemplate.execute(DELETE\_KEY,

new PreparedStatementCallback<Boolean>() {

public Boolean doInPreparedStatement(PreparedStatement ps)

throws SQLException, DataAccessException {

ps.setString(1, key);

return ps.execute();

}

});

logger.info(" entry with key " + key + "deleted ");

}

public void deleteAll(Collection<String> keys) {

for (Iterator<String> iterator = keys.iterator(); iterator.hasNext();) {

String map\_key = iterator.next();

delete(map\_key);

}

if (!keys.isEmpty()) {

logger.info(" total " + keys.size() + "objects deleted. ");

}

}

/\*\*

\* store method save object into DB . if KEY already exist it will update

\* the record with new value

\*/

public void store(final String key, final Object value) {

try {

jdbcTemplate.execute(INSERT\_QUERY,

new PreparedStatementCallback<Boolean>() {

public Boolean doInPreparedStatement(

PreparedStatement ps) throws SQLException,

DataAccessException {

ps.setString(1, key);

ps.setObject(2, value);

return ps.execute();

}

});

logger.info(" entry persisted successfully [ " + key + " = "

+ value + " ]");

} catch (org.springframework.dao.DuplicateKeyException dke) {

logger.info(" updating value with existing key " + key);

update(key, value);

}

}

private void update(final String key, final Object value) {

jdbcTemplate.execute(UPDATE\_QUERY,

new PreparedStatementCallback<Boolean>() {

public Boolean doInPreparedStatement(PreparedStatement ps)

throws SQLException, DataAccessException {

ps.setObject(1, value);

ps.setString(2, key);

return ps.execute();

}

});

}

public void storeAll(Map<String, Object> map) {

Set<Entry<String, Object>> set = map.entrySet();

for (Iterator<Entry<String, Object>> iterator = set.iterator(); iterator

.hasNext();) {

Entry<String, Object> entry = iterator.next();

store(entry.getKey(), entry.getValue());

}

logger.info(" total " + map.size() + " objects saved.");

}

/\*\*

\* convert byte stream into Object

\*

\* @param objectBytes

\* @return

\*/

public Object deserialize(byte[] objectBytes) {

try {

ObjectInputStream objectIn = new ObjectInputStream(

new ByteArrayInputStream(objectBytes));

return objectIn.readObject();

} catch (Exception e) {

return null;

}

}

}

**Queries are placed under below interface**

package com.trvajjala.model.store;

/\*\*

\* MySQL Queries which are used in MySQLMapStore

\*

\*/

public interface SQLQuery {

String INSERT\_QUERY = "INSERT INTO tbl\_hazelmap(map\_key,map\_value) values(?,?)";

String UPDATE\_QUERY = "UPDATE tbl\_hazelmap SET map\_value=? WHERE map\_key=?";

String SELECT\_QUERY = "SELECT map\_value FROM tbl\_hazelmap WHERE map\_key=?";

String ALL\_KEYS = "SELECT map\_key FROM tbl\_hazelmap";

String DELETE\_KEY = "DELETE FROM tbl\_hazelmap WHERE map\_key=?";

}

you can have DB script as show below to support any kind Map value . or change db datatypes as per your requirement

CREATE DATABASE IF NOT EXISTS `hazel\_db`;

USE `hazel\_db`;

CREATE TABLE `tbl\_hazelmap` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`map\_key` varchar(45) DEFAULT NULL,

`map\_value` blob,

PRIMARY KEY (`id`),

UNIQUE KEY `map\_key\_UNIQUE` (`map\_key`)

) ENGINE=InnoDB AUTO\_INCREMENT=16 DEFAULT CHARSET='utf8'

final spring configuration file looks like this

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"

xmlns:hz="http://www.hazelcast.com/schema/spring"

xsi:schemaLocation="http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.hazelcast.com/schema/spring

http://www.hazelcast.com/schema/spring/hazelcast-spring-3.2.xsd">

<context:property-placeholder location="classpath:hazel-server.properties" />

<hz:hazelcast id="instance">

<hz:config>

<hz:group name="${hazel.username}" password="${hazel.password}" />

<hz:properties>

<hz:property name="hazelcast.merge.first.run.delay.seconds">5</hz:property>

<hz:property name="hazelcast.merge.next.run.delay.seconds">5</hz:property>

<hz:property name="hazelcast.logging.type">log4j</hz:property>

<!-- reference : http://hazelcast.org/docs/latest/manual/html/logging.html -->

</hz:properties>

<hz:network port="${hazel.network.port}"

port-auto-increment="${hazel.port.autoincrement}">

<hz:join>

<hz:multicast enabled="false" />

<hz:tcp-ip connection-timeout-seconds="5"

enabled="${hazel.tcp.enabled}">

<hz:member>${hazel.tcp.member}</hz:member>

</hz:tcp-ip>

</hz:join>

</hz:network>

<hz:map name="jobTracker" backup-count="1" in-memory-format="BINARY"

statistics-enabled="true">

<hz:map-store enabled="true" implementation="mysqlMapStore"

write-delay-seconds="${hazel.mapstore.delay}">

</hz:map-store>

</hz:map>

<hz:map name="tmpQueueTracker" backup-count="1"

in-memory-format="BINARY">

<hz:map-store enabled="true" implementation="mysqlMapStore"

write-delay-seconds="${hazel.mapstore.delay}">

</hz:map-store>

</hz:map>

</hz:config>

</hz:hazelcast>

<hz:map instance-ref="instance" id="jobTracker" name="jobTracker" />

<hz:map instance-ref="instance" id="tmpQueueTracker" name="tmpQueueTracker" />

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">

<property name="dataSource" ref="dataSource" />

</bean>

<bean name="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="${db.driverClassName}" />

<property name="url" value="${db.url}" />

<property name="username" value="${db.username}" />

<property name="password" value="${db.password}" />

</bean>

<bean id="mysqlMapStore" class="com.tvajjala.model.store.MySQLMapStore">

<property name="jdbcTemplate" ref="jdbcTemplate" />

</bean>

</beans>

hazel-server.properties FILE

#DB PROPERTIES

db.driverClassName=com.mysql.jdbc.Driver

db.url=jdbc:mysql://localhost:3306/hazel\_db

db.username=root

db.password=XXXXX

#HAZELCAST PROPERTIES

hazel.username=dev

hazel.password=dev

hazel.tcp.enabled=true

hazel.tcp.member=127.0.0.1

hazel.network.port=5701

hazel.port.autoincrement=true

#mysql store delay seconds

hazel.mapstore.delay=1

now whatever the map entries that you store into hazelcast map instance will be persisted into DB with 1 sec delay.