JBoss Data Grid lab guides

Lab 5

Thomas Qvarnström

v1.1 2015-12-06

Table of Contents

This guide explains the steps for running lab 5, either follow the steps in the step-by-step section or if you feel adventurous try to accomplish goals without the step-by-step guide.

1. Background

The `todo` application is a huge success and over 100 000 users are today using it to synchronize the task list between different devices. Even though each entry are typically small the data is growing fast. By mid next year numbers of users and expected to grow to 500 000 and 1 million at the end of the year. However the success comes with new challenges, the memory used to store tasks is growing fast and causing performance issues. Assigning more java heap is not an option since that will lead to long garbage collection pauses that will affect the performance of the application server.

Acme has three options to improve memory management:

1. Configure eviction to evict cache entries to disk or database
2. Switch to use distributed mode instead of replication data where data is distributed between the nodes.
3. Move to Client/Server mode where server nodes can be configured to use larger heaps.

1.1. Memory usage today and in the future

The average length of a title is 20 characters (about 80 bytes). Together with two Date objects (32 bytes each) and the other fields we can assume that each Task object will take about 200 bytes of memory. The average number of tasks per user is 50. Today the effective data storage (data without meta-data) is about 1 GB of data [100 000 x 50 tasks per user x 200 bytes per task / (1024^3)].

NOTE Best practice for memory sizing of JBoss Data Grid says that JVM should be minimum double the expected max storage. Given the data above the Java Heap Max setting should be 2GB or more.

Student tasks:

1. What's the expected storage need for data mid next year?
2. What's the expected storage need for data end of next year?

Given that the `todo` application is currently using replicated mode all nodes must be sized to stored all objects. Switching to distributed mode would mean that we can grow the in-memory storage by adding more nodes.

Library mode will share the same java heap as the application for storage. Client/Server mode has the benefit of better memory management where resources can be optimized for JBoss Data Grid. Java EE applications needs CPU resources and may actually behave better with a lower java heap, while JDG

Server uses less CPU and can be optimized to use larger java heaps.

Red Hat recommends the customer to move to Client/Server mode using distribution to meet future requirements.

2. Use-case

Rewrite the application to only use JDG Client/Server mode.

3. These are the main tasks of lab 5

1. Setup the lab environment
2. Add dependencies to HotRod client
3. Rewrite the `TaskService` to use HotRod instead of client mode
4. Remove dependencies to jdg-core

3.1. Setup the lab environment

To assist with setting up the lab environment we have provided a shell script that does this.

1. Run the shell script by standing in the jdg lab root directory (~/jdg-labs) execute a command like this

   ```
   $ sh init-lab.sh --lab=5
   ```

   Stop any existing servers from previous labs and Start the servers in separate consoles using the following commands

   EAP Server:

   ```
   $ ./target/jboss-eap-6.3/bin/standalone.sh
   ```

   JDG Server:

   ```
   $ ./target/jboss-datagrid-6.3.0-server/bin/standalone.sh -Djboss.socket.binding.port
   -offset=100
   ```

4. Step-by-Step

1. Discussion pros and cons with different options for the solution to the memory usage issues with your colleagues.
2. Open project lab5 in JBoss Developer Studio
3. Add HotRod client development and runtime dependencies by opening pom.xml and uncomment the following lines:

```
<dependency>
    <groupId>org.infinispan</groupId>
    <artifactId>infinispan-commons</artifactId>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>org.infinispan</groupId>
    <artifactId>infinispan-client-hotrod</artifactId>
    <scope>compile</scope>
</dependency>
```

NOTE We are here using the scope `compile` since we want maven to add these jars and it’s transient dependencies to WEB-INF/lib

4. Before we rewrite the `TaskService` we need to configure the HotRod client using CDI to produce a `RemoteCache` object. Open `Config` class and add the following to it:

```
@Produces
public RemoteCache<Long, Task> getRemoteCache() {
    ConfigurationBuilder builder = new ConfigurationBuilder();
    builder.addServer().host("localhost").port(11322);
    return new RemoteCacheManager(builder.build(), true).getCache("default");
}
```

NOTE We are reusing the default cache in this lab, in lab 6 we will configure our own cache instead.

5. You also need to add the following import statement if your IDE doesn’t fix that

```
import javax.enterprise.inject.Produces;

import org.infinispan.client.hotrod.RemoteCache;
import org.infinispan.client.hotrod.RemoteCacheManager;
import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
import org.jboss.infinispan.demo.model.Task;
```

6. Open the `TaskService` class
7. Inject a `RemoteCache` object like this:

```
@Inject
RemoteCache<Long, Task> cache;
```

8. You also need to add the following import statement if you IDE doesn’t fix that

```
import javax.inject.Inject;
```

9. Implement the `findAll()` method like this:

```
public Collection<Task> findAll() {
    return cache.getBulk().values();
}
```

10. Implement the `insert(Task)` method like this:

```
public void insert(Task task) {
    if(task.getCreatedOn()==null) {
        task.setCreatedOn(new Date());
    }
    task.setId(System.nanoTime());
    cache.putIfAbsent(task.getId(), task);
}
```

11. Implement the `update(Task)` method like this:

```
public void update(Task task) {
    cache.replace(task.getId(), task);
}
```

12. Implement the `delete(Long)` method like this:

```java
public void delete(Long id) {
    cache.remove(id);
}
```

13. Save the `TaskServer.java` file
14. Open `TaskServiceTest.java` and uncomment the the `File[] jars = ....` and `.addAsLibraries(...)`
15. Run the JUnit test and verify that everything works.
16. Deploy the application using the following command from lab5 dir

```
$ mvn clean package jboss-as:deploy
```

17. Congratulations you are done with lab 5.