In Memory Storage Essentials Labs

Before You Begin

1. Set up your Hazelcast cluster following the directions in the Lab Setup Guide for the environment of your choice. You can run the cluster locally on your laptop/desktop machine, or deploy a cloud-based cluster via cloud.hazelcast.com.

2. Download the labs for this course from GitHub at https://github.com/hazelcast/training-courses/tree/master/In_Memory_Storage_Essentials. Import the lab into your IDE. You will have the most success importing them as a Maven project.

> *If you get stuck at any point in these labs, you can go to the Solutions subdirectory under each lab, and see our lab engineer's code.*

Lab 1: Basic Get/Put

In this hands-on exercise, you will:

- Use an instance of a Hazelcast client to add data to an IMap
- Use a separate instance of a Hazelcast client to retrieve data from an IMap

Code "skeletons" for this lab are in `IMap-Get_Put/src/main/java`.

1. Open the files Client1 and Client2.

2. Modify the configuration of both clients based on your lab environment.
    a. If you are using a local cluster, comment out all the configuration lines.
    b. If you are using a cloud-based cluster, modify the config section to include your specific cloud credentials.

3. In Client1, create a map in your cluster called "training".

4. Populate the map with integers. We've written the code to generate the integers for you; you just need to write the `put` statement to move the data into the map.

5. Optional: Add a line that prints a message to the system output when the map is populated.

6. In Client2, reference the map called "training".

> *Note: you use the same syntax to create and to reference a data structure. If it doesn't exist, Hazelcast will create it. If it does exist, Hazelcast will use the existing structure.*

7. Retrieve and store the entry with key value 42.

8. Print the entry to the system output.

9. Extra credit suggestions:
    a. Overwrite a specific entry, then retrieve it to verify it was overwritten.
    b. Retrieve an entry, perform a transformation (a simple `value + 1` works!), then put the entry back in the map. Verify that the transformed data was stored in the map.
    c. Write a loop that retrieves a selection of entries, performs a transformation, then puts the entries back in the map.
    d. Create and populate a new map with other data types (`String`, for example). Retrieve stored data.

Lab 2: Expiration/Eviction – Local Cluster

In this hands-on exercise, you will

- Observe the effects of an eviction configuration on a map.

Code for this lab is in `IMap-Eviction/src/main/java`.

Recall that in the lab setup, you added a configuration for the map called `training-eviction`.

```
<map name="training-eviction">
    <eviction eviction-policy="LFU" max-size-policy="USED_HEAP_SIZE" size="1"/>
</map>
```

This configuration limits the map to using no more than 1% of the total heap memory available.

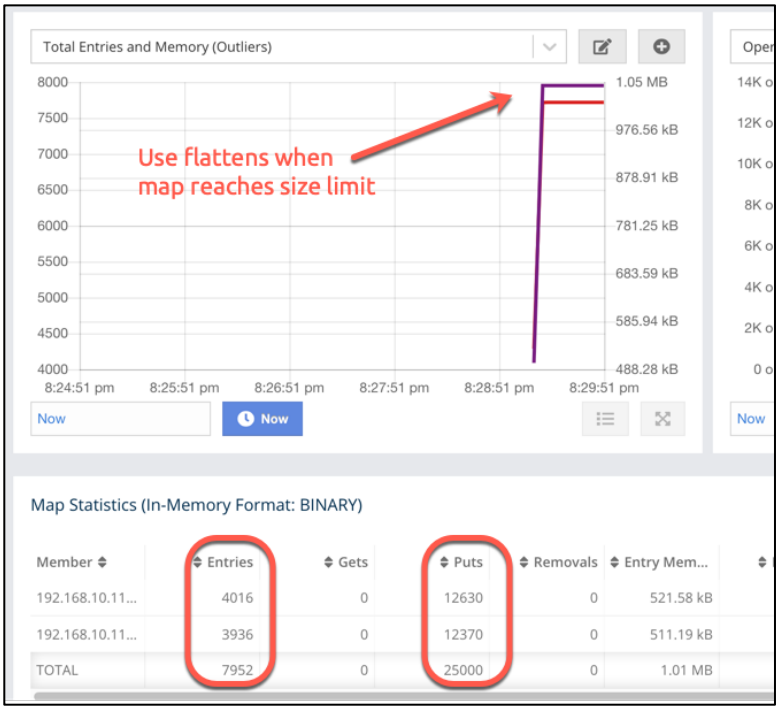
We're now going to run a client that fills this map past capacity and observe as eviction limits the map size.

1. Open Management Center.

2. Open the Client file. Modify the configuration as for Lab 1.
   a. The client creates the map called `training-eviction`.
   b. This client populates the map with 25000 entries.

3. Run the client. When the client is finished, note the number of entries available in the map. It should be significantly less than 25000, depending on the total memory available to your local cluster.

4. In Management Center, click on Maps on the left side of the screen, then click on training-eviction to open a dashboard for the specific map.



Note that the usage graph rises at a constant rate, then flattens out as the map reaches the configured maximum capacity.

You can see from the statistics that all 25000 entries were written to the map, but only a subset are currently in the map. Eviction removed the rest.

5. Extra credit: Open the console window in Management Center.
    a. Change the name space to training-eviction
       ```
       $ ns training-eviction
       ```
    b. Display all the entries in the map
       ```
       $ m.entries
       ```
    c. Use `m.get` and `m.put` to display and add individual entries to the map.
    d. Type `help` to see what other functions are available via the console.

> *Note: The Management Center console is a quick way to test data structure functionality without having to re-run client code.*

Lab 2: Expiration/Eviction – Cloud-Based Cluster

In this hands-on exercise, you will

- Observe the effects of an eviction configuration on a map.

Code for this lab is in `IMap-Eviction/src/main/java`.

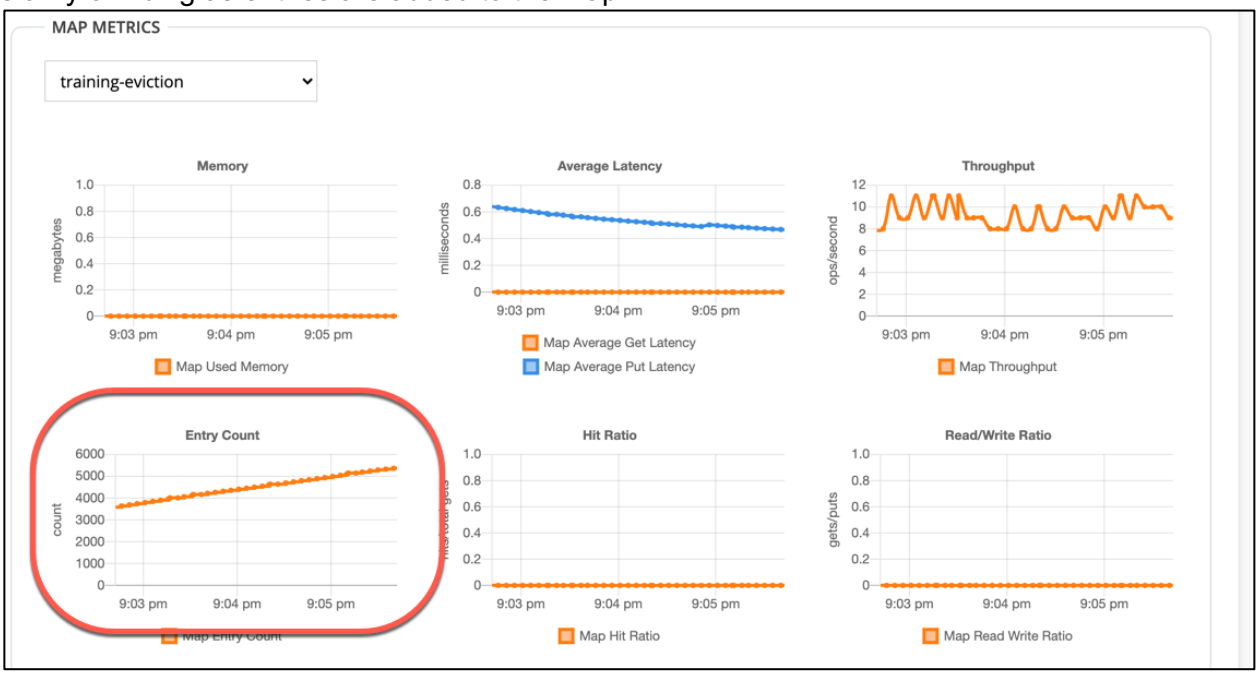
Recall that in the lab setup, you added a configuration for the map called `training-eviction`. You set the policy to evict least frequently used entries (LFU), and limited the map size to no more than 1% of the total heap memory available.

We're now going to run a client that fills this map past capacity and observe as eviction limits the map size.

1. Open the Client file. Modify the configuration as for Lab 1 to connect to your cloud cluster.
    a. The client creates the map called `training-eviction`.
    b. This client populates the map with 25000 entries.

2. Run the client. Go get a cup of your favorite beverage - this is going to take a while. (Actual performance will depend on the speed of your connection to the cloud. Remember, you're using a free implementation – performance isn't optimal.)

3. While you are waiting, you can use the cloud console to observe the map memory utilization slowly climbing as entries are added to the map.



Eventually, the memory utilization number will level out at a fixed count, and throughput will drop to 0. This indicates that you are no longer adding new entries to the map. You'll also note that the memory used by the map maxed out at 2.0 megabytes – one percent of the 200 MB available to a free cloud cluster.

4. Extra credit: Start a second client that pulls entries from the map. You may have to try several keys to get a valid one. Note that invalid keys – keys that have been removed from the map – return a null.

Lab 3: Queries

In this hands-on exercise, you will

- Populate a data store in the Hazelcast cluster.
- Create a query that pulls data from the data store.

Code "skeleton" for this lab is in `IMap-Querying/src/main/java/hazelcast`.

1. Modify the configuration and code loading section of the code based on your lab environment.
    a. If you are using a local cluster, comment out all the configuration lines and the user code deployment lines.
    b. If you are using a cloud-based cluster, modify the config section to include your specific cloud credentials.

    > *Note: The UserCodeDeployment section is needed to load the Employee data to the cloud-based instance.*

2. Create a query that retrieves records for all salaries between 0 and 2000. You can use either the Criteria API or SQL to create your query. The code will display the contents of the collection in the system output.

3. Run your code.
    a. The code will first populate the Employee map and return a time for how long it took to push the data. Note the duration.
    b. The code then runs your query and displays the output. Note how long the system took to complete your query.

    > *We will compare the load and query duration to an indexed map in the next lab.*

4. Extra credit: create a query using whatever method you did not use in step 2.

Lab 4: Indexes

In this hands-on exercise, you will

- Populate an indexed data store in the Hazelcast cluster.
- Create a query that pulls data from the data store.

Code "skeleton" for this lab is in `IMap-Indexing/src/main/java/hazelcast`.

1. Modify the configuration and code loading section of the code based on your lab environment.
    a. If you are using a local cluster, comment out all the configuration lines and the user code deployment lines.
    b. If you are using a cloud-based cluster, modify the config section to include your specific cloud credentials.

    > *Note: The UserCodeDeployment section is needed to load the Employee data to the cloud-based instance.*

2. We're using the same Employee data as the previous lab, but we want this new map to sort the data based on salary. Use map.addIndex to dynamically add this configuration to the cluster.

    > *Why add the configuration here? Because we don't want to take down our cluster to add it to the hazelcast.xml configuration. This index is specific to the map `training-index`, so adding it here does not affect any other operations.*

3. Create a query that retrieves records for all salaries between 0 and 2000. You can use either the Criteria API or SQL to create your query. The code will display the contents of the collection in the system output.

4. Run your code.
    a. The code will first populate the Employee map and return a time for how long it took to push the data. Note the duration.
    b. The code then runs your query and displays the output. Note how long the system took to complete your query.

    > *Why did populating the data take so much longer? Because the cluster had to rebuild the index each time an entry was added. The trade-off is that the query itself ran much faster.*
    >
    > *Takeaway: Use indexes for mostly-stable data, or accept the additional latency introduced by building the index.*