Cloud Computing (INGI2145) - Lab Session 2

Waleed Reda, Marco Canini

1. Background:

In this tutorial, you'll learn how to interact with a selection of Amazon's storage services (which we've covered during the beginning of this lab session). We'll start with Amazon S3, where you'll learn to create/delete buckets, add objects to them and download objects into files.

Following this exercise, you'll be introduced to Amazon's DynamoDB and how to create/delete tables as well as the insertion/updating/deletion of records. Then we'll integrate Amazon's ElastiCache with DynamoDB and see how it can be used to retrieve data items faster.

In both exercises, we'll provide you with a slew of coding scripts that you can run to attain some of the discussed functionality. However, you'll need to write a few lines of code to create some additional functionality.

2. Tutorial:

To setup the VM for these exercises, you need to re-provision the system as this will install node.js, memcached and the boto AWS Python library. You will need to boot the INGI2145-vm by running `vagrant up --provision`. Once puppet has concluded, login and go to the directory `/vagrant/lab2`. Then run `npm install` to install the node.js dependencies.

At this point you need to setup **two environment variables with your ACCESS ID and SECRET KEY** in order to access the AWS API. Based on your AWS access key, export the following environment variables:

```
# export AWS_ACCESS_KEY_ID='AKID'
# export AWS_SECRET_ACCESS_KEY='SECRET'
```

I. Amazon S3

As discussed previously, this section of the tutorial aims to get you more familiarized with utilizing Amazon's S3. The provided 'lab2-s3.py' file is a Python script that comprises a few commands that you should familiarize yourself with. It should give you a backbone for how to perform rudimentary tasks with Amazon's S3.

As you can see, this script executes a set of operations on an Amazon S3 store. The following commands are executed (in this particular order):

1. Initiate a connection with Amazon's S3 using your credentials.
2. Create a bucket on S3. Make sure to **give it a unique name** (e.g., ingi2145-<AWS user>) as the bucket namespace is shared between all S3 users.
3. Create some objects and insert them into the bucket.
4. Delete one of the objects.
5. Download all remaining objects to separate files on your local machine
6. Destroy the bucket. Note: Make sure you remove all objects before initiating a delete command, since non-empty buckets can't be deleted

Now try to modify this script in order to ***read your objects from local files*** instead of hardcoding their values. You may consult the Python S3 documentation found at: https://boto.readthedocs.org/en/latest/s3_tut.html.

You can also try to skip step 6, which destroys the bucket, and head to the AWS console at https://mcanini.signin.aws.amazon.com/console/ and use the S3 console to navigate to your bucket and confirm the contents are those that you think they are.

II. DynamoDB operations:

In this exercise, you'll be required to run a few operations on Amazon's DynamoDB. These include instructions such as: Insertions, deletions, queries, etc.

A – Setup:

Before starting your exercise, you need to set up a new table (which you'll be using vicariously throughout this exercise). However, **the table name must be unique** (since the namespace is shared between all students). As such, access the script file called 'lab2-commons.js' and modify the 'table_name' variable to whatever suits you. After setting up your table name, you should then run the following Javascript files in the appointed order:

1- `nodejs lab2-create-dynamodb-table.js`: Creates a table with your specified name with the following attributes: NOMA (hash key), Name (range key), UserAWS, Email
2- `nodejs lab2-insert-data.js`: Inserts student data of those currently enrolled in this course

B – Exercises:

For both these exercises, you'll run a node.js server, which is located in the script file called "lab2-node.js". You'll then interact with this server by sending GET requests in order to query/add/edit/delete items in your DynamoDB table. A list of the supported GET requests and their corresponding functionality is presented in part C (Command API).

i. Node.JS and DynamoDB:

In this section, we're going to operate a non-caching version of DynamoDB. You're required to run a few operations and see how they can be used to manipulate DynamoDB items (you can

reference Amazon's DynamoDB API for javascript for more insight into the inner-workings of the code at: http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/frames.html.) Proceed to execute the following commands through GET requests:

- Query for a specific student by name
- Retrieve all students from the table
- Add an attribute called "ProgName" to all students

Task:

After running these steps, try to add a function that allows you to ***delete students by either their NOMA or Name*** (Note: spaces are encoded as %20 in URLs while accented characters have their own encodings). In order to implement this, please refer to the following two files:

a) "lab2-commons.js": This comprises all the functions that are necessary to query/scan/update student data. You should add your deletion function here.
b) `nodejs lab2-node.js`: This is the node.js server file responsible for executing the required operations. Hint: the "adding attribute" instruction might give you some hints on how to perform the deletion operation.

ii. Node.JS and DynamoDB (with Memcached support):

Now that we've covered the basics of DynamoDB, we'll see how Memcached (which is equivalent to ElastiCache for the purposes of this exercise) can help speed-up the retrieval of data. Before starting, you can opt to wipe out the previously created table and recreate a new one. This can be done by running the `nodejs lab2-delete-dynamodb-table.js`. To recreate a new table, just redo the ***Setup*** step (in part A) done at the beginning of the exercise.

For this exercise, you'll need to modify the 'caching_mode' variable in the lab2-node.js file to 'true', in order to run the caching-version of the server. After doing so, carry out the same operations as in the preceding exercise and note the differences caused by introducing the caching feature.

Task:

Take note of how the caching feature works during each step. After executing these instructions, try to implement ***caching for queries that utilize the student name*** (as opposed to the NOMA) so that data received in this manner could also be cached (with their names acting as keys in this case). Also, try to update the delete student command (created in the preceding exercise) so as to **delete any pre-existing cache entries** as well. Similarly to the preceding exercise, you'll need to utilize the functions contained in "lab2-commons.js" to implement this.

BONUS task:

Try to implement a timer that measures how long it takes for DynamoDB to respond as opposed to your local cache. Obviously, given the locality of your Memcached cluster, the response times should be negligibly low compared to real Memcached clusters.

C – Command API:

This subsection comprises that list of GET requests that you're going to be utilizing to run the various operations in both DynamoDB exercises:

Command	Function
"/query?id=<NOMA>"	Query for a student by his/her NOMA
"/query?name=<NAME>"	Query for a student by his/her Name
"/list"	Retrieve all students
"/set?attr=<ATTR>&value=<VALUE>"	Add a new attribute/value to all records

Similarly to S3, DynamoDB also has a console under the AWS console at https://mcanini.signin.aws.amazon.com/console/. Use the DynamoDB console to navigate to your table and confirm the contents are those that you think they are.