ECE1779: Introduction to Cloud Computing

Winter 2022

Assignment 2

Dynamic Resource Management

Due Date

March 21, 2022

Objective

This assignment will expose you to the following AWS technologies: EC2, RDS, S3, and CloudWatch.

Description

In this assignment, your task is to extend the application you developed in assignment 1 into an elastic memory cache that can resize the pool of storage nodes on demand. Your application should consist of four components:

1.- A manager-app that controls the size of the memcache pool. The UI of this application should support the following functionality:

1. Use charts to show the number of workers as well as to aggregate statistics for the memcache pool including miss rate, hit rate, number of items in cache, total size of items in cache, number of requests served per minute. The charts should display data for the last 30 minutes at 1-minute granularity.
2. Configure the capacity and replacement policy used by memcache nodes. All memcache nodes in the pool will operate with the same configuration.
3. Selecting between 2 mutually-exclusive options for resizing the memcache pool:
    a. Manual mode. There should be two buttons for manually growing the pool size by 1 and shrinking the pool size by 1. The maximum and minimum sizes should be 1 and 8.
    b. Automatic. Configure a simple auto-scaling policy by setting the following parameters:
        i. Max Miss Rate threshold (**average for all nodes in the pool** over the past 1 minute) for growing the pool

   ii. Min Miss Rate threshold (**average for all nodes in the pool** over the past 1 minute) for shrinking the pool
   iii. Ratio by which to expand the pool (e.g., expand the ratio of 2.0 doubles the number of memcache nodes).
   iv. Ratio by which to shrink the pool (e.g., shrink ratio of 0.5 shuts down 50% of the current memcache nodes).
4. Deleting all application data: A button to delete image data stored on the database as well as all image files stored on S3.
5. Clearing memcache data: A button to clear the content of all memcache nodes in the pool.

2.- An *auto-scaler* component that automatically resizes the memcache pool based on configuration values set by the manager-app. It should monitor the miss rate of the mecache pool by getting this information using the AWS CloudWatch API. The auto-scaler should be implemented as a stand-alone program that runs in the background. It should implement the following functionality:

- There should be no need to manually restart the auto-scaler every time policy is changed.
- Check for the cache miss rate every 1 minute.
- Do **NOT** use the AWS Auto Scaling feature for this assignment.
- **Limit the maximum size of the worker pool set by auto-scaler to 8 and the minimum to 1**

3.- The web front end you developed for A1. Provide the functionality implemented for A1 with the following modifications:

1. Remove functionality to configure memcache settings.
2. Remove functionality that displays memcache statistics.
3. All image files should be stored in **S3**.
4. The mapping between keys and image files should be stored on **AWS RDS**. Do **not** store the images themselves in the database. It is advised that you use the smallest possible instance of RDS to save on credit. **Important:** RDS is an expensive service that can cost several dollars per day for larger instances. To ensure that you do not run out of credits, use one of the smaller instance types (e.g., burstable db.t2.small class) and remember to stop your RDS instance when you are not actively using it.
5. Route requests to the memcache pool using a consistent hashing approach using the MD5 hash. For simplicity, assume that the key space is partitioned into 16 equal-size regions which are then allocated to the pool of memcache nodes. Figures 1-3 illustrate how this assignment changes as the size pool goes from 1 node to 2 nodes, to 4 nodes.

6. Add a feature to the front-end that makes it possible to automatically notify it when the size of the memcache pool changes. In response, the front-end should rebalance the mapping of key regions to nodes, and use the new allocation to route requests to the memcache.

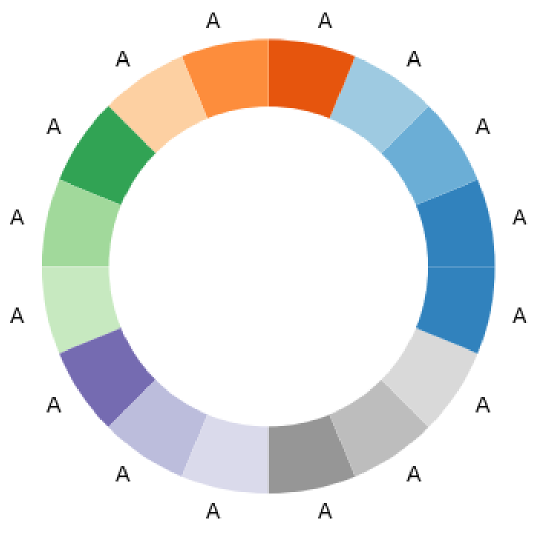

Figure 1

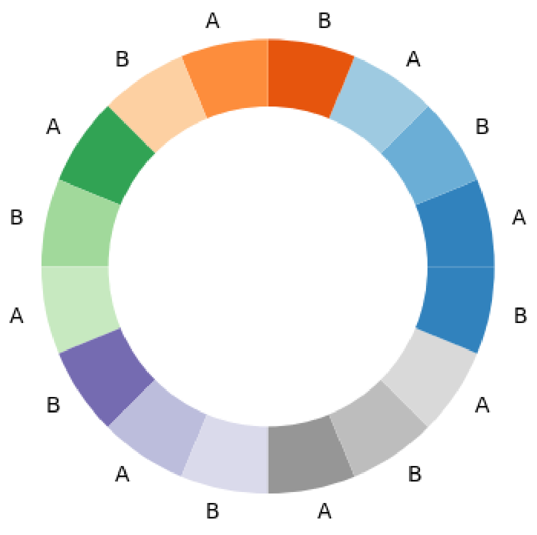

Figure 2

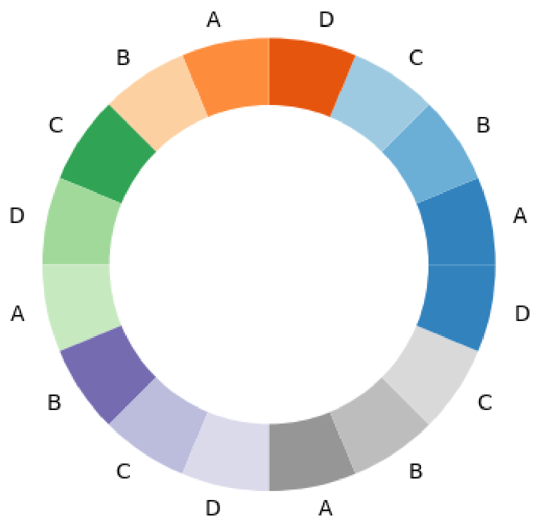

Figure 3

4.- The Key-Value Memory Cache developed in A1. Provide the functionality implemented for A1 with the following modifications:

1. Store mem-cache statistic every 5 seconds using [CloudWatch Custom Metrics] instead of the database as was done in A1. Also, look at [boto3 documentation] on how to publish these metrics.

Requirements

1. Implement all the required components of the application system. Starter code for a Flask-based mem-cache is available here. It uses a global variable to store data in memory as a Python dictionary and includes two endpoints (get, put) that return json.
2. To allow for automatic testing, your application should **(in addition to your web interface)** include three URL endpoints to automatically upload files and retrieve files. Our automatic testing script will send **HTTP requests** to your URL endpoints with the parameters in the body. Your implementation should not accept HTTP requests where parameters have been appended to the URL. See this link for more information about how HTTP POST requests work: https://www.w3schools.com/tags/ref_httpmethods.asp
3. The two automatic testing endpoints should conform to the following interfaces:

   Upload:

   ```
   relative URL = /api/upload
   enctype = multipart/form-data
   method = POST
   POST parameter: name = key, type = string
   POST parameter: name = file, type = file
   ```

   Retrieve all keys

   ```
   relative URL = /api/list_keys
   method = POST
   ```

   Retrieve an image associated with a specific key

   ```
   relative URL = /api/key/<key_value>
   method = POST
   ```

   - These endpoints should generate responses in the following JSON format:
     In case of failure:

     In case of success for the ***upload*** interface:
     ```
     {
           "success": "true"
     }
     ```

In case of success for the ***retrieve*** interface while fetching all keys

```
{
      "success": "true",
      "keys": [Array of keys(strings)]
}
```

In case of success for the ***retrieve*** interface while fetching a particular key

```
{
      "success": "true",
      “content” : file contents
}
```

In case of failure of any of the calls of the ***upload*** or ***retrieve*** interface

```
{
      "success": "false",
      "error": {
            "code": servererrorcode
            "message": errormessage
      }
}
```

Deliverables

We will test your application using your AWS account. For this purpose, you should pre-load the manager-app and your web front-end on one (or more) EC2 instances. Please **suspend** the instances when you submit your project to prevent charges from occurring while the TA gets around to grading your submission. Make sure to not restart the instance from the moment you submit your project. Also, make sure no other worker instances are running when the manager-app instance is suspended.

You should write a shell or bash script called **start.sh on the manager** that initializes the **manager-app and auto-scaler**. This script should be put in the Desktop folder inside the EC2 instance.

Once the manager starts, it should resize the mem-cache pool to 1.

The manager-app and the web front-end should run on port **5000** and be accessible from outside the instance. Make sure that you open this port on your EC2 instance. Documentation about how you can open the port can be found here.

Submit the assignment only once per group. Clearly identify the names and student IDs of group members in the documentation. In your report explain what each member of the group did towards the design and implementation of the project.

You will demo your project to a TA and the TA may ask questions about the parts each member did so having all the members of the group present at the project demo is mandatory. Information about how to arrange a demo time will be posted on Piazza.

To submit your project, upload to Quercus a single tar file with the following information:

1. User and developer documentation in a **PDF** file (documentation.pdf). Include a description of how to use your web application as well as the general architecture of your application. Also include a figure describing your database schema. Click here for tips on how to write documentation. Your documentation will be marked based on how cohesive it is and how well you are able to explain the technical details of your implementation.
2. In the documentation, include a section called **Results.** This section should show that your auto-scaler component works. Design test cases and scenarios that demonstrate the functionality of your auto-scaling algorithm and its reliability as load increases and decreases. During each scenario, stick to one auto-scaling policy (thresholds and ratios) and try to demonstrate that the size of the worker pool reasonably changes in response to changes in the incoming load. Show that your algorithm converges if the load stays at a particular level for a few minutes. Use charts showing the load and worker pool size over time to present your results.
3. In addition to the documentation, put the first name, last name, and student ID of each student in a separate line in a text file named group.txt. Please make sure that the first and last names in this group.txt file match exactly how your name is displayed in Quercus.
4. Include your AWS credentials in a text file (named credentials.txt). We will need these to log into your account. You can create credentials with limited privileges using AWS IAM if you don't want to share your password with the TA; however, make sure that you include permissions to start and stop EC2 instances. Test the credential to make sure they work.
5. Key-pair used to ssh into the EC2 instance (named keypair.pem).

   Do not forget to send the .pem files required for ssh-ing into your VM's.

6. Anything else you think is needed to understand your code and how it works.

Marking Scheme (35 points)

1. Manager UI: Providing the appropriate menus, bars and buttons for the manager to control the system and displays performance metrics. (**10 points)**
2. Manual pool resizing (**8 points**)
3. Automatic pool resizing (**8 points**)
4. Documentation: all codes should be properly formatted and documented, explanations about how to log in to your Amazon account, how to start the instance and how to run the codes should be included. You should also write about the general architecture of your codes and how different elements of your codes are connected to each other. Do not forget about the database schema and group members, and the result section described above **(9 points)**

Resources

Amazon provides limited free access to its infrastructure to new users through its free tier. Information about the AWS free tier, including how to apply for an account is available at https://aws.amazon.com/free. You should complete this assignment using exclusively resources from the free tier.

Important note: Your AWS account is fully functional which means that it will incur charges if you use resources that do not qualify for the free tier, or if you use free-tier qualifying resources beyond the quota that Amazon provides (e.g., a 750 hour per month of EC2 t2.micro, 30 GB of EBS). You will be responsible for any charges. To avoid incurring charges make sure to use only resources from the free tier and remember to suspend your EC2 instances when not in active use.

The following video tutorials show how to:

- Create an EC2 instance
- Connect to the new instance using ssh
- Suspend an instance

To help you get started, an AMI (id: ami-080ff70d8f5b80ba5) is provided with the following software:
Note: the AMI is only available in the N. Virginia AWS Region

- Python
- MySQL Server (root password ece1779pass)
- Database and AWS Flask examples covered in lectures
- gunicorn
  This is a high-performance server for Python-based applications. For an example of how to run it, look at the run.sh file inside /home/ubuntu/extras/run.sh

- In addition the directory in Desktop/ece1779 contains the following:
    - **databases:** A PyCharm project with all examples from the databases lecture and tutorial
    - **extras:** A PyCharm project with code for transcoding photos and a sample form that conforms to the load testing interface