Assignment 2

Amazon Web Services

Due Date: Nov 16

Objective

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

Description

Your task is to extend the web site you developed in A1 into an elastic web application that can resize its worker pool based on demand. Your application should provide two web-based interfaces: ***userUI*** and ***managerUI***.

userUI should include the following functionality:

1. A welcome page that lets returning users authenticate and enables new users to create a new account.
2. A home page that lets authenticated users browse thumbnails of their photos. Use the popular ImageMagick tool to create thumbnails.
3. An option for uploading a new photo.
4. After a new image is uploaded, your application should automatically create 3 different transformations (e.g, scale down, enlarge, black and white, sepia). Use the popular ImageMagick tool to transform the image. It is up to you to decide what transformations to implement.
5. Clicking on a photo's thumbnail should produce the full resolution version of the photo as well as its transformations.

managerUI should support the following functionality:

1. List workers, and their CPU utilization
2. Manually growing the worker pool by 1.
3. Manually shrinking the worker pool by 1.
4. Functionality for configuring a simple auto-scaling policy by setting the following parameters:
    - CPU threshold for growing the worker pool
    - CPU threshold for shrinking the worker pool
    - Ratio by which to expand the worker pool (e.g., a ratio of 2 doubles the number of workers).

    - Ratio by which to shrink the worker pool (e.g., a ratio of 4 shuts down 75% of the current workers).
5. Functionality for deleting all data. Executing this function should delete application data stored on the database as well as all images stored on S3.

Requirements

1. All images should be stored in S3
2. Store information about user accounts and the location of images (and their pre-computed transformations) owned by a user in a relational database. Do not store the images themselves in the database. It is up to you to design the database schema, but make sure that you follow design practicesand is properly normalized. **Important:** Run only 1 instance of the database (i.e., do not replicate the database); all workers should communicate with this single database instance.
3. User passwords should not be stored in clear text in the database. Instead, store the hash of the password concatenated with a per-user salt value. Details for how to do this are available here.
4. Use a separate EC2 instance to run each worker node. All worker nodes should run the ***userUI*** functionality.
5. The ***managerUI*** should run on a single EC2 instance separate from the instances used to run the worker nodes.
6. Your application should monitor the load on its pool of workers using the AWS CloudWatch API. When the load exceeds (or drops bellow) the configurable threshold, your application should use the AWS EC2 API to resize its worker pool. Do **NOT** use AWS Auto Scaling feature for this assignment. ***Important***: in order to use AWS CloudWatch, you need to enable monitoring of your EC2 instances.
7. Use only EC2 t2.small or t2.micro instances for your worker pool
8. Use EC2 Load Balancing to distribute request among your workers. Your application should use the AWS Elastic Load Balancing API to add and remove workers from the load balancing pool as you reconfigure your worker pool in response to load.
9. To simplify testing of your application under load you applications should also provide an entry point that conforms to the following interface:

```
relative URL = /test/FileUpload
enctype = multipart/form-data
method = post
field1 name = userID type = string
field2 name = password type = string
field3 name = uploadedfile type = file
```

This URI expects 3 arguments encoded with the multipart/form-data encoding. You can assume that the userID and password match the account of an existing user.

A sample form that conform to this specification is available here.

A load generator that conforms to this interfaces is available here. **Warning:** the load generator creates a lot of network traffic. To minimize bandwidth charges, please use the load generator inside EC2 only.

To run the program untar the file, and cd into ece1779LoadGenerator/bin

Run as:

java -cp . ece1779.loadgenerator.LoadGenerator server_ip_address port userid password

Deliverables

We will test your application using your AWS account. For this purpose, your application should be pre-loaded on an number of EC2 instances (e.g., 1 for the managerIU, 1 for the userUI, 1 for the database server). Please suspend your instances to prevent charges from accruing while the TA gets around to grading your submission.

Submit the assignment only once per group. Clearly identify the names and student IDs of group members in the documentation.

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

1. Developer documentation. Include a description of the general architecture of your application and your database schema.
2. Deployment instructions. Provide any information necessary for starting your site (e.g., commands to start flask after resuming EC2 instance, port in which flask is running). **Important:** by default your instances will have different IP addresses after they get resumed by the TA. Make sure that your documentation explains clearly how to start the various elements of your application to account for the new network address.
3. User documentation. Brief description of how to use your application.
4. AWS credentials. The TA 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 instances.
6. Anything else you think is needed to understand your code and how it works.

Resources

Amazon provides free credits for students to experiment with its cloud infrastructure, including EC2. To apply for an educational account go to Amazon Educate.

The following video tutorials show how to :

- Create an EC2 instance
- Connect to an instance using a VNC Client.
- Suspend an instance

To help you get started, an AMI (id ***ami-e3f432f5***) is provided with the following software:

Note: the AMI is only available in the N. Virginia AWS Region

- Python 3.5
- PyCharm IDE
- Firefox
- MySQL Server (root password ece1779pass)
- mysql-workbench
- vncserver (password ece1779pass)
- Database and AWS Flask examples covered in lectures and tutorials.
- ImageMagick
- Python Wand ImageMagick bindings
- gunicorn
  This is high performance server for Python-based applications. For an example of how to run it, look at the run.sh file inside Desktop/ece1779/databases
- In addition the directory in Desktop/ece1779 contains the following:
  - **databases:** A PyCharm project with all examples from the databases lecture and tutorial
  - **aws:** A PyCharm project with all examples from the AWS lecture and tutorial. **Important:** To run this application you will need to first configure your AWS credentials. This is a two stage process: First, create a set of credentials by logging into the AWS Console. Click on your user name (top right corner) and select "My Security Credential." Next, click on "Access Keys" and push the button "Create New Access Key." Click on "Show Access Key" and copy the access key id and secure access key. Second: open a shell and type "aws configure". Paste your access key id and secret access key when prompted.
  - **extras:** A PyCharm project with code for transcoding images and a sample form that conforms to the load testing interface

- **loadgenerator:** Contains the Java-based load generator for testing your application under load. To run, cd into loadgenerator and type:
  java -cp . ece1779.loadgenerator.LoadGenerator server_ip_address port userid password