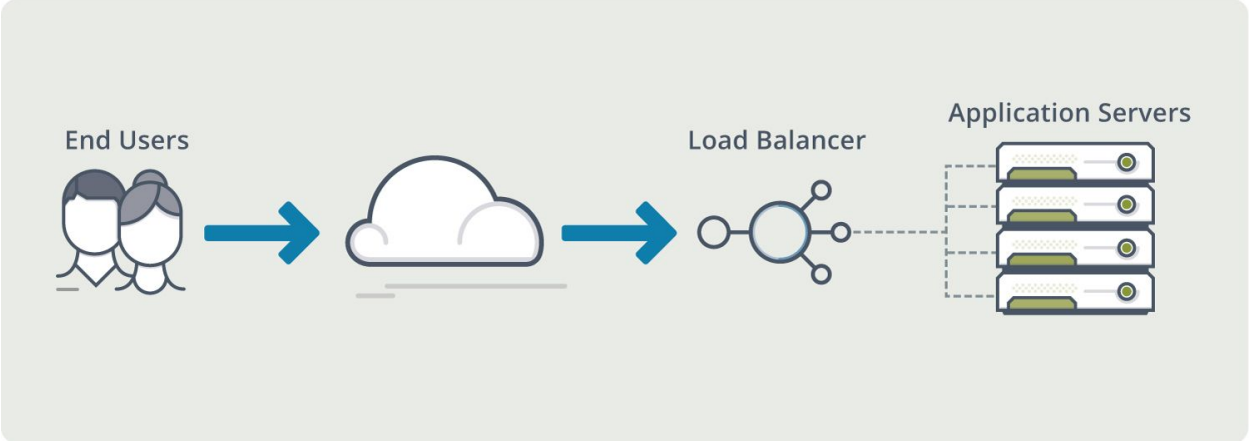
ECE1779: Introduction to Cloud Computing

Winter/Spring 2019

Assignment 2

Dynamic Resource Management



Due Date

March 15

Objective

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

Description

In this assignment, your task is to extend the application you developed in assignment 1 into an elastic web application that can resize its worker pool based on demand. "worker pool" is the set of ec2 instances that run the user-app, shown in the figure as the "application servers." Your application consists of two parts:

The user-app which is what you already have from Assignment 1, instances running as the worker pool:

1. A welcome page with login and register forms. New users can register by providing a username and password. Previously registered users can login using their username and password.
2. A page for uploading a new photo. After a new photo is uploaded, your application should automatically draw a rectangle around human faces in the photo and save the result as well as the original photo.
3. A page that lets logged-in users browse thumbnails of their uploaded photos. Clicking on a photo's thumbnail should produce the original version of the photo as well the photos with rectangles around faces.

The *manager-app* which controls the worker pool and through its user interface, should support the following functionalities:

1. A page with the list of workers, and a chart showing the CPU utilization of each one (you can get the actual chart from CloudWatch)
2. For each worker, show the number of HTTP requests the worker has received per minute. For each worker there should be a chart that has the rate (HTTP requests per minute) on the y-axis and time on the x-axis.
3. Manually growing the worker pool by 1.
4. Manually shrinking the worker pool by 1.
5. Functionality for configuring a simple auto-scaling policy by setting the following parameters:
    - CPU threshold (average for all workers) for growing the worker pool
    - CPU threshold (average for all workers) 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).
6. 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 photos should be stored in S3.
2. Store information about user accounts and the location of photos owned by a user on **AWS RDS**. Do **not** store the photos themselves in the database. It is up to you to design the database schema, but make sure that you follow design practices and that your database schema is properly normalized. It is advised that you use the smallest possible instance of RDS to save on credit.

3. How you implement the requests rate feature (for each worker) and showing in the manager-app is up to you. You can try using CloudWatch or save a log on RDS whenever a new request is received and then draw the chart using the logs.
4. Follow basic web application security principles. All input should be validated. In addition, 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.
5. Use a separate EC2 instance to run each worker node. All worker nodes should run the *user-app* functionality.
6. The *manager-app* should run on a single EC2 instance separate from the instances used to run the worker nodes.
7. **Auto Scaling**: Your application should monitor the load on its pool of workers using the AWS CloudWatch API. When the load exceeds (or drops below) the configurable threshold, your application should use the AWS EC2 API to resize its worker pool. You can write a simple python program on the manager, that monitors the worker pools, reads the auto-scaling policy info set by the manager interface from the database and re-scales the worker pool based on load. Checking for the CPU usage of workers every 1 minute is good enough. 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.
8. Use only EC2 t2.small or t2.micro instances for your worker pool
9. 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.
10. This is from assignment 1: To simplify testing, your user-app application should **also (in addition to your web interface)** include two URLs that makes it possible for the TA to automatically register users and upload photos. The URLs should conform to the following interfaces:

    Register:

    ```
    relative URL = /api/register
    method = POST
    POST parameter: name = username, type = string
    POST parameter: name = password, type = string
    ```

    Upload:

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

```
POST parameter: name = password, type = string
POST parameter: name = file, type = file
```

Sample forms that conform to the above specification are available here and here.

A load generator that conforms to the upload interface is available here. You can use the load generator to test your application. **Warning:** the load generator creates a lot of network traffic. To minimize bandwidth charges, you can use the load generator inside EC2 only.

To run the load generator, run the python script using:

```
python3 gen.py upload_url username password upload_rate(upload_per_second)
files_folder number_of_uploads
```

for example:

```
python3 gen.py http://9.9.9.9:5000/api/upload user pass 1 ./photos/ 100
```

11. All code should be properly formatted and documented

Deliverables

We will test your application using your AWS account. For this purpose, your application should be pre-loaded on an EC2 instance. Please suspend the instance to prevent charges from occuring while the TA gets around to grading your submission.

You should write a shell or bash script called **start.sh on the manager** that initializes your manager web application. This script should be put in Desktop folder inside the EC2 instance. Your web application should run on port **5000** and be accessible from outside the instance. So, 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.

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 for 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 report include a section called **Results.** This section should show that your auto-scaling policy works (design test cases and scenarios that demonstrate the functionality of your auto-scaling algorithm as load increases on workers)
3. If you are using the AWS educate account (link provided in the class) with the $50 credit, then we don't need your AWS credentials as we already have access to your AWS console.

   Otherwise, please send us your AWS credentials (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.

4. Key-pair used to ssh into the EC2 instance (keypair.pem).

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

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

Marking Scheme

Manager UI: 5 Points - Providing the appropriate menus, bars and buttons for the admin to control the system
Basic Cluster Operations: 7 Points - HTTP requests to the load balancer should be correctly forwarded to the workers and the workers should detect faces on images. Saving images on S3 and user data on RDS. Registering the workers on the ELB correctly, etc.
Manager Functionality: 8 Points: Showing worker info, correct charts, ability to add/remove workers manually, option for deleting all data, having a page for tuning the Auto-scaler policy, etc
AutoScaling: 7 Points: Implementation of the autoscaling policy. Keep in mind your auto scaling policy should converge, meaning if the load is constant, you shouldn't be adding or removing workers.
Documentation: 8 Points: Having the proper instructions, system architecture, database information, other implementation details, result section

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-080bb209625b6f74f) 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
- OpenCV libraries
- Python Wand ImageMagick bindings
- Python OpenCV binding (cv2)
- 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
  - **extras:** A PyCharm project with code for transcoding photos and a sample form that conforms to the load testing interface