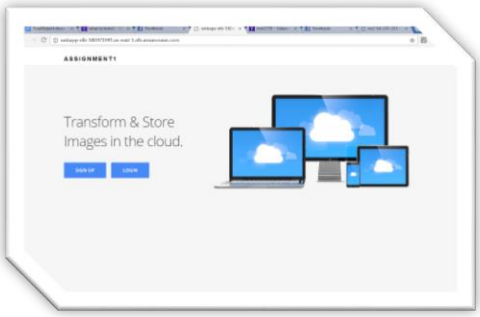
ECE1779 Project-1

Introduction to Cloud Computing

(Amazon EC2)



By:

Farshad Safavi **Jack Shengxi Xia**

March 16, 2017

Table of Contents

1. General Architecture

General architecture of our web application is shown in *Figure 1*. The project includes two web-based interfaces: User UI and Manager UI. Amazon Web Services are used for all backend supports including web application hosting, worker pool management and storage.

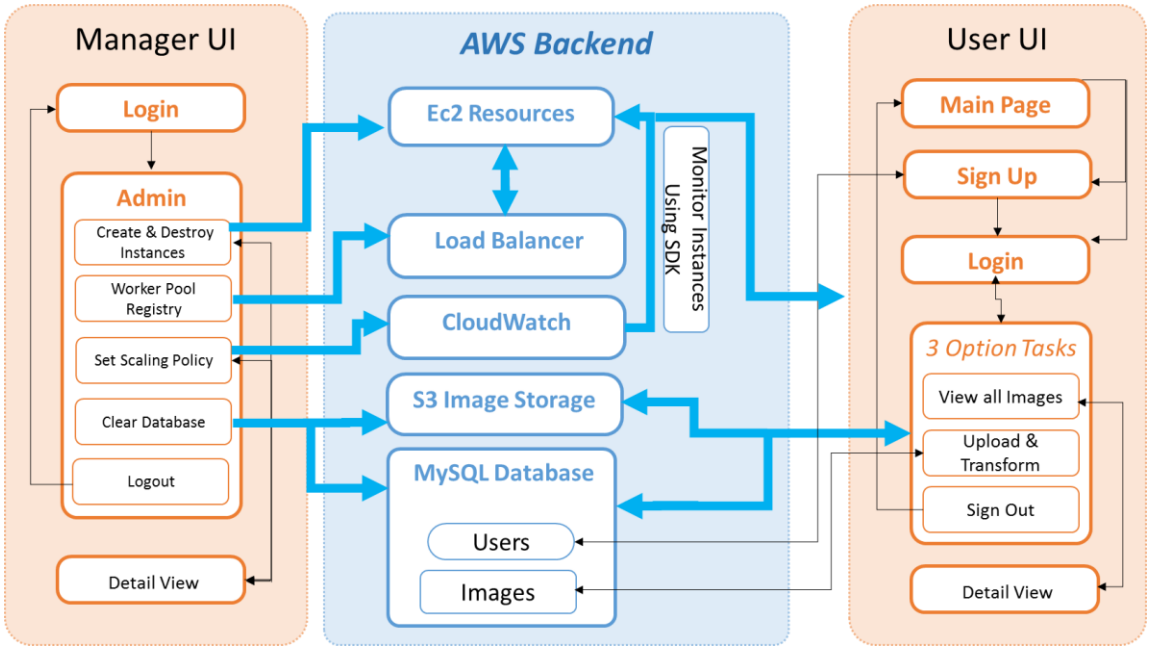


Figure 1: General Architecture of the application

1.1Manager UI

All web applications including manager UI are hosted in EC2 instances. There is only one manager instance which controls all web application instances (*Figure 2*). All workers share centralized RDS database and S3 Image Storage (*Figure 2*). This architecture allows all workers talk to a single centralized database and S3 image storage. All web development and backend support for Manager UI is located in "manager" folder project. Following is the brief explanation of project components and its implementations.

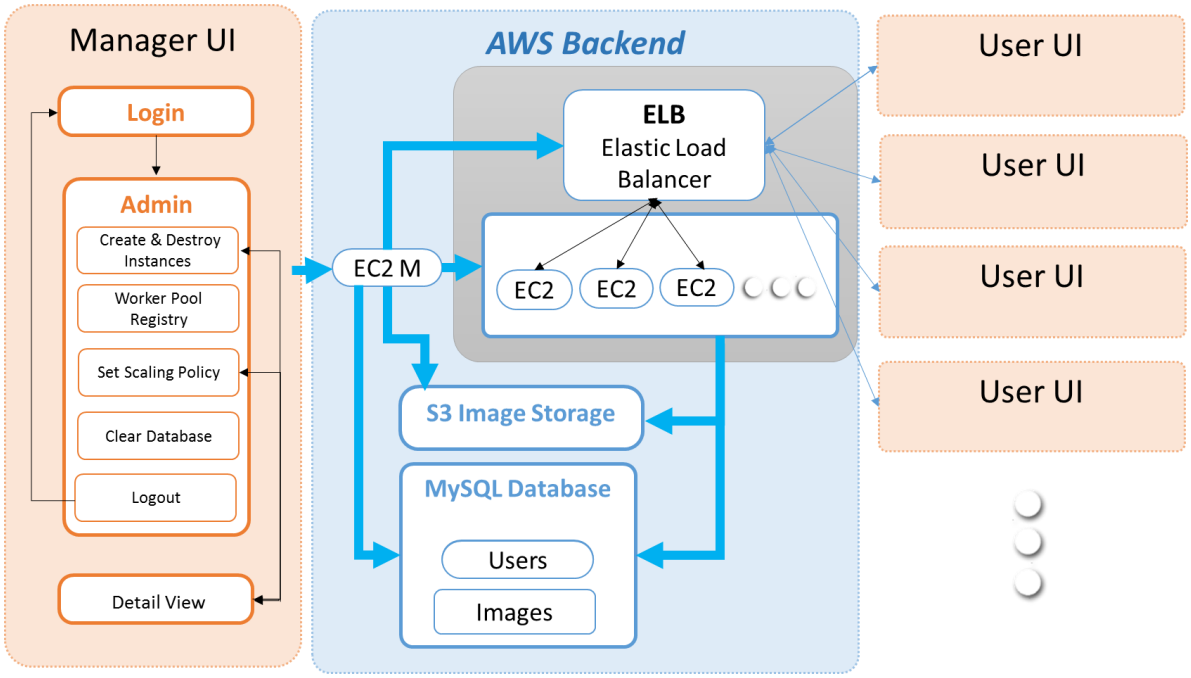


Figure 2: Manager UI Control Multiple User UI

Manager project includes five main components:

1. **Login & logout:** It allows the administrator to login and logout. This feature is implemented in "main.py" file in app folder.
2. **Create & Destroy Instances:** It allows to manually grow worker pool by creating new EC2 instances and manually register an EC2 instance to the load balancer. It gives an option to see CPU utilization or terminate a specified instance. All these functionalities are implemented by using Web Services (AWS) SDK for Python in "admin.py" and "detailcpu.py" in app folder.
3. **Worker Pool Registry:** It provides list of all EC2 instances which are behind the elastic load balancer (ELB). It provides an option to manually shrink the worker pool by deregister an instance and destroy it. The functionality is implemented in "elb.py" in app folder.

4. **Set Scaling Policy:** It offers autoscailing option. A python script in the manager instance applies cloudwatch services SDK to monitor and compute average CPU utilization of all instances behind ELB every five minutes. Subsequently, it allows users to fetch the real time upper bound or lower bound to decide to grow or shrink the worker pool. This function is implemented in "autoscale.py" in app folder.
5. **Clear Database:** This feature clears all images from S3 image storage and all data from images tables in mysql database. This function is implemented in "delete.py" in app folder.

1.2User UI

User UI is developed in Assignment1 project. User UI is a basic web application which allows users to create an account to transform and store their personal images. Assignment1 project includes following components:

1. **Main page:** It allows to create an account and login to the page. This feature is implemented using mysql database and located in "main.py", "signup.py" and "login.py" files in app folder.
2. **View All Images:** This feature queries all image names corresponding to specific UserID. Consequently, it shows the grid view of all images which are previously saved in S3 image storage. This functions are implemented in "gridview.py" file in app folder.
3. **Detail View:** By clicking on a specific image, you can see detail view of the image and its three transformations in a detail view. This function is implemented in "detail_view.py" in app folder.

4. **Image Store & Transformation:** This block uploads and transforms a chosen image. A unique image name; combination of username "_" image name, and three unique keys are stored in images table in mysql database. Actual image objects and their transformations are stored in S3 bucket using AWS SDK for python. These functions are implemented in "imagetransform.py" in app folder.

1.3Auto scaling Policy

It is important to note that AWS Auto Scaling feature is **Not used** in this assignment. Instead, threading task ("update_cpu_util" located in "run.py") runs as a background task which periodically checks the threshold and adjust the worker pool accordingly. The admin user has four options in the admin page:

- **CPU threshold for shrinking <:** The minimum CPU utilization Threshold in Percent. (e.g., 10 means if cpu utilization become less than 10%, manager automatically shrinks worker pool).
- **CPU threshold for growing >:** The maximum CPU utilization Threshold in Percent. (e.g., 80 means if cpu utilization become more than 80%, manager automatically grows worker pool).
- **Ratio by which to expand:** The application calculates number of all instances behind load balancer (ELB) and multiply expanding ration to that number (e.g., a ratio of 1 doubles the number of workers).
- **Ratio by which to shrink:** The application calculates number of all instances behind load balancer (ELB) and divide it by shrinking ratio. The result is the number of instances which will be remained behind ELB and reminder instances will be terminated. (e.g., a ratio of 1

shuts down 0% of the current workers, a ratio of 2 shuts down 50% of the current workers; and, a ratio of 4 shuts down 75% of the current workers).

When admin user fetch the data in real time for CPU threshold and ratio, the application log those data into files. Every five minutes “calculate_cpu_utilization_all_instances_2” function is called. This function calculates the CPU utilizations of all instances behind ELB. It passes CPU utilization of all workers to another function named “is_grow_or_shrink”. This function reads CPU threshold and ratio values from log files and decides to grow or shrink instances using specified create or destroy instance functions. As previously mentioned, this mechanism is implemented in “autoscale.py” in app folder.

1.4Validation & Evaluations

The admin page provides user with buttons to test and validate CPU utilization and autoscaling features:

- **CPU Utilization All Instances:** It shows current average CPU utilization across all instances behind ELB and number of active instances behind ELB.
- **Force Auto Scaling Once**: It allows the user test auto scaling once asynchronously when the application is in auto scaling mode. Otherwise, a user has to wait every 5 minutes to see the auto scaling action and results.

The application is saved at worker ***ami: ami-9c2b828a*** . Using userdata at the creation of each instance, supervisor will be launched at boot up. Supervisor is a linux program that can start the python application as a background program. It will keep running as long as the machine is powered.