IaaS Project

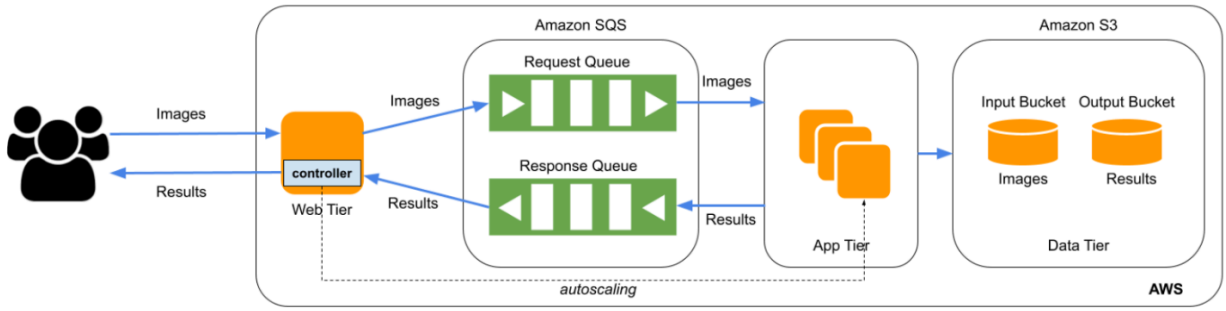
[CSE 546] [SPRING_2024] Cloud Computing

Summary

In this Project, we will complete the development of the elastic face recognition application using the IaaS resources from AWS. We will implement the App Tier and Data Tier of our multi-tiered cloud application, use a machine learning model to perform face recognition and implement autoscaling to allow the App Tier to dynamically scale on demand.

Description

We will strictly follow the architecture below to develop the application.



Web Tier

1. You **MUST** use the same web tier EC2 instance from Part 1 of the project. The web tier **Must** listen for requests on port number **8000**.
2. It should take images received from users as input and forward it to the App Tier for model inference. It should also return the recognition result from the App Tier as output to the users. The input from each request is a .jpg file, and the output in each response is the recognition result. Find more details below.

   Input:

   - The key to the HTTP payload **MUST** be defined as "**inputFile**" and should be used as the same. In the case of a Python backend, it denotes a standard Python file object.

- For example, the user uploads an image named “test_00.jpg”.
- Use the provided workload generator to generate requests to your Web Tier.

Output:

- The Web Tier will handle HTTP POST requests to the root endpoint ("/").
- The output **MUST** be in plain text, and the format <filename>:<classification_results>
- For the above example request, the output should be “test_00:Paul” in plain text.
- You need to implement the handling of concurrent requests in your Web Tier.

To facilitate the testing, a standard face dataset and the expected recognition output of each image are provided to you at

Input: visa-lab/CSE546-Cloud-Computing/face_images_1000.zip

Output: visa-lab/CSE546-Cloud-Computing/classification_face_images_1000.csv

3. The Web Tier uses only **one** instance. You **MUST** name your web-tier instance “**web-instance**”
4. The Web Tier sends requests to the App Tier and receives results using two **SQS Queues**. You **MUST** follow the following naming convention in naming your S3 Buckets:
    a. **Request Queue**: <ASU ID>-req-queue
    b. **Response Queue**: <ASU ID>-resp-queue

    For example, if your ASU ID is “12345678910”, your queue names will be 12345678910-req-queue and 12345678910-resp-queue
5. The Web Tier also runs the autoscaling **controller,** which determines how to scale the App Tier. You are **not** allowed to use AWS features for autoscaling. You **MUST** implement your own auto scaling algorithm.

App Tier

1. The App Tier will use the provided deep learning model for model inference. The deep learning model code is available here. The first step is to create an AMI, which will be used to launch App Tier instances.
    a. Launch a base EC2 instance. You can use the AWS Linux or Ubuntu AMI.
    b. Install the required package on the instance using the following command.

```
pip3 install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cpu
```

    c. Copy the provided deep learning model code and model weights folder to the EC2 instance using scp.
    d. Refer to the README.md on how to use the deep learning model code.

e. Create an AMI using this EC2 instance, following the instructions here. Now, you can use this AMI to create your App Tier instances. You **MUST** name your app-tier instances "**app-tier-instance-<instance#>**"

2. The App Tier should automatically scale out when the request demand increases and automatically scale in when the demand drops. The number of App Tier instances should be 0 when there are no requests being processed or waiting to be processed. The number of App Tier instances can scale to **at most 20** because we have limited resources from the free tier.

Data Tier

1. All the inputs (images) and outputs (recognition results) should be stored in separate buckets on S3 for persistence.
2. S3 stores all the objects as key-value pairs. For the Input bucket, each object's key is the input file's name, e.g., test_00.jpg, and the value is the image file. For the Output bucket, the key is the image name (e.g., test_00), and the value is the classification result (e.g., "Paul").
3. You **MUST** follow the following naming convention in naming your S3 Buckets
    a. **Input Bucket**: <ASU ID>-in-bucket
    b. **Output Bucket**: <ASU ID>-out-bucket

    For example, if your ASU ID is "1234567890", your bucket names will be 1234567890-in-bucket and 1234567890-out-bucket

Testing & Grading

- Use the provided workload generator to test your app thoroughly. Check the following:
    - The output of the workload generator is correct.
    - The contents in the S3 buckets are correct;
    - While processing the workload, the number of EC2 instances is correct.
    - All the requests are processed within a reasonable amount of time. As a reference point, for a workload of **100** concurrent requests using the TAs' implementation of the project, completed within **80** seconds (average of 5 iterations).
- Test your application using the provided workload generator and grading script. If they fail to execute, you will receive **0** points. Refer to the Readme on how to use the grading script.

- We will use the same Grading IAM user credentials we collected in Part 1 of the project. You **MUST** update the Grading IAM user with the following permissions:
    - AmazonEC2ReadOnlyAccess
    - AmazonS3FullAccess
    - AmazonSQSFullAccess
- Make sure you test everything using the Grading IAM Credentials before making a submission.
- To facilitate testing, you **MUST** use only the resources from the US-East-1 region.
- The grading will be done using the workload generator and grading script and following the rubrics below:

#	Test Objective	Test Criteria	Test Command Sequence	Fail	Pass	Total Points
1	Validate EC2 initial state	1) Single Web Tier instance exists and the state of the web-instance is "running" 2) There are 0 running App Tier instances with name	Run p2_grader.py & use selection choice:1 for validating the instances	There is no EC2 instance with name "web-instance" (0) There are app tier instances in "running" state (0)	The EC2 instance with the name "web-instance" exists; and is in "running" state (2.5) There are no app tier instance in "running" state (2.5)	5
2	Validate S3 initial state	The input and the output buckets must exist and should be empty	Run p2_grader.py script & use selection choice:2 for validating S3 Buckets	The input and the output buckets either do not exists or are not empty(0)	The input and the output buckets both exists and are empty (2.5)	2.5
3	Validate SQS initial state	The SQS Request and Response Queues must exist and have no messages in the queue	Run p2_grader.py script & use selection choice:3 for validating SQS Queues	The request and the response queues either do not exists or are not empty(0)	The request and the response queue both exist and are empty (2.5)	2.5
4	Validate the completeness of the 10-request workload	The HTTP Response code must be 200 for all the requests	1) Start the p2_grader.py before the workload generator and use selection choice: 4 1) Run workload_generator.py with 10 requests 2) After the workload generator is over, check the stats "Total number of requests completed successfully"	Total number of requests completed successfully stats from workload generator is not 10 (0)	Total number of requests completed successfully stats from workload generator is 10 (5) +0.5 for every request completed successfully.	5

	Validate the correctness of the classification results	All the classification results must be correct	After the workload generator is over, check the stats "Total number of correct predictions"	Total number of correct predictions stats from workload_generator.py is not 10 (0)	The Total number of correct predictions stats from workload generator.py is 10 (5) +0.5 for every correct classification.			5
	Validate S3 buckets	The S3 input and Output Buckets should each have 10 objects	Run p2_grader.py & use selection choice:2 for validating S3 Buckets	The S3 Input and Output Buckets do not have 10 objects. (5)	The S3 Input and Output Buckets have 10 objects (5)			5
	Validate autoscaling	1) The # of app tier instances should gradually scale from 0 to 10 and reduce back to 0 2) The # of SQS messages in Request Queue should increase from 0 and then reduce back to 0	Validate the prints from p2_grader.py when the selection choice: 4 was made from Step-1	The # of app tier instances do not gradually scale from 0 to 10 and reduce back to 0 (0) The # of SQS messages in the Request Queue do not gradually increase from 0 and reduce back to 0 (0)	The # of app tier instances gradually scale from 0 to 10 and reduce back to 0 (10) The # of SQS messages in Request Queue gradually increases from 0 and reduce back to 0 (5)			15
	Validate total latency	Total end to end latency must be reasonable	Check the "Total Test Duration" stats from the workload_generator.py	The total test duration for 10 requests is greater than 4 min (0)	The Total test duration for 10 requests is between 3 and 4 min (5)	The Total test duration for 10 requests is between 2 and 3 min (10)	The Total test duration for 10 requests is less than 2 min (15)	15
5	Validate the completeness of the 50-request workload	The HTTP Response code must be 200 for all the requests	1) Start the p2_grader.py before the workload generator and use selection choice: 4 1) Run workload_generator.py with 50 requests 2) After the workload generator is over, check the stats "Total number of requests completed successfully"	Total number of requests completed successfully stats from workload generator is not 50(0)	Total number of requests completed successfully stats from workload generator is 50 (5) +0.1 for every request completed successfully.			5
	Validate the	All the classification	After the workload	The Total number of	Total number of correct predictions stats			5

	correctness of the classification result	results must be correct	generator is over, check the stats "Total number of correct predictions"	correct predictions stats from workload generator is not 50 (0)	from workload generator is 50 (5) +0.1 for every correct classification.			
	Validate S3 buckets	The S3 input and Output Buckets should each have 10 objects	Run p2_grader.py & use selection choice:2 for validating S3 Buckets	The S3 Input and Output Buckets do not have 50 objects. (0)	The S3 Input and Output Buckets have 50 objects (5)			5
	Validate autoscaling	1) The # of app tier instances should gradually scale from 0 to 20 and reduce back to 0 2) The # of SQS messages in Request Queue should increase from 0 and then reduce back to 0	Validate the prints from p2_grader.py when the selection choice: 4 was made from Step-1	The # of app tier instances do not gradually scale from 0 to 20 and reduce back to 0 (0) The # of SQS messages in Request Queue do not gradually increase from 0 and then reduce back to 0 (0)	The # of app tier instances gradually scale from 0 to 20 and reduce back to 0 (10) The # of SQS messages in Request Queue gradually increase from 0 and reduce back to 0 (5)			15
	Validate total latency	Total end to end latency must be reasonable	Check the "Total Test Duration" stats from the workload_generator.py	Total test duration for 50 requests is greater than 5 min (0)	Total test duration for 50 requests is between 4 and 5 min (5)	Total test duration for 50 requests is between 3 and 4 min (10)	Total test duration for 50 requests is less than 3 min (15)	15

Submission

The submission requires **four** components; all must be completed by ***03/15/2024 at 11:59:59 pm***.

1. Upload your source code to Canvas as a single zip file named by your full name: <lastname><firstname>.zip. Submit only your source code. Do not include any code not developed by you. Do not include any binary files.
2. Share your web tier Elastic IP and AWS Grading Credentials using the Google Form
3. Keep your web tier instance running until you are informed that grading is done.
4. Before you submit, make sure you empty the Input and Output Buckets.

IMPORTANT:

- Failure to follow the submission instructions will cause a penalty to your grade.

- Do not change your code after the submission deadline. The grader will compare the code you have submitted on Canvas and the code you run on AWS. If any discrepancy is detected, it will be considered as academic integrity violations.

Policies

1) Late submissions will absolutely not be graded (unless you have verifiable proof of emergency). It is much better to submit partial work on time and get partial credit than to submit late for no credit.

2) You need to work independently on this project. We encourage high-level group discussions to help others understand the concepts and principles. However, code-level discussion is prohibited, and plagiarism will directly lead to failure in this course. We will use anti-plagiarism tools to detect violations of this policy.