Project 1 (Part I): IaaS

Due by **09/28/2025 11:59 PM (firm deadline)**

Summary

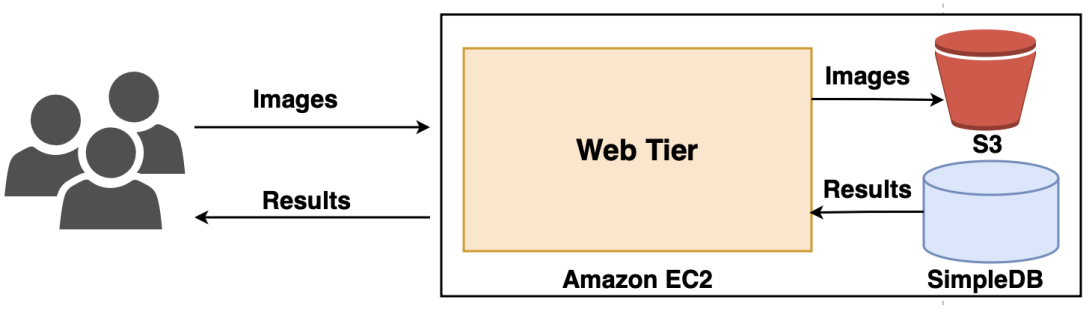
In the first project, we will build an elastic application that can automatically scale in and out on-demand and cost-effectively by using the IaaS cloud. Specifically, we will build this application using the IaaS resources from Amazon Web Services (AWS). AWS is the most widely used IaaS provider and offers a variety of compute, storage, and message services. Our application will offer a meaningful cloud service to users, and the technologies and techniques that we learn will be useful for us to build many others in the future.

The project is divided into two parts. In the first part, we will familiarize ourselves with AWS, its key IaaS resources, and the app development process on AWS; We will develop the web tier in Part I, which will be used as the front end of our multi-tiered cloud app in Part II.

We recommend you to follow the steps below to complete Part I. But these are not exact step-by-step instructions. Check the AWS documentation for more information, and ask on Ed Discussion.

Description

The web tier will receive face recognition requests from clients, store the images in an S3 input bucket, and return the classification results to the clients. In Part I, instead of using a real classification model, we will use a lookup table stored in SimpleDB to emulate the model inference process.



The web tier should implement the following (as illustrated in the diagram above):

1. Extract images received from HTTP requests
    - The web tier MUST handle HTTP POST requests to the root endpoint ("/") on port 8000.
    - 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.
2. Store the received images in an input bucket on S3
    - 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.
    - You MUST follow the naming convention below in naming your S3 bucket
        - Input bucket: <ASU ID>-in-bucket
        - For example, if your ASU ID is “1225754101”, your bucket names will be 1225754101-in-bucket.
3. Perform emulated “face recognition” by looking up a SimpleDB domain
    - Use AWS boto3 SDK to create and populate the SimpleDB. You can use the create_domain API to create a domain.
    - You MUST name your SimpleDB domain as <ASU ID>-simpleDB. E.g., if your ASU ID is 1225754101, your SimpleDB domain should be 1225754101-simpleDB.
    - Populate the SimpleDB domain with classification results. This is available at dataset/classification_face_images_1000.csv. You can use put_attributes to create or replace attributes in a SimpleDB item.
    - The web tier will query this SimpleDB domain and fetch the “recognition” result for each input image.
4. Return the results to the HTTP requests
    - The web tier will return the “recognition” result as the response to the original HTTP request. You are not allowed to use additional HTTP requests to fetch the results.
    - The output MUST be in plain text and in the format <filename>:<prediction_results>.

      For example, for input “test_00.jpg”, the output should be “test_00:Paul” in plain text.
    - You need to implement the handling of concurrent requests in your web tier.

The web tier implementation needs to meet the following requirements:

- Use a **single** EC2 micro instance to run the entire web tier. Make sure to monitor your AWS usage and stay within the free tier limits for EC2, S3, and SimpleDB. If you incur any charge by mistake, you can always request AWS to waive it.
- Implement your web tier program using Python; no other programming languages are allowed.
- To facilitate testing, you MUST use only the resources from the US-East-1 region, and follow all the naming conventions as described in the document.
- Assign your EC2 instance a static IP address so it does not change during testing. You MUST name your EC2 instance "web-instance". Note that you can use a dynamic IP address during development to not incur any cost, but you MUST use Elastic IP to assign a static IP address to your web tier instance before you submit your project for grading.
- The web tier should be able to handle multiple requests concurrently and as quickly as possible. As a reference point, the average response time during the TA's testing was 0.116 seconds for a workload with 1000 requests.

You MUST use Python to implement the web tier and you MUST name your program "server.py", but it is up to you to decide how to implement your web tier, as long as it meets the above requirements.

Testing

Make sure that you use the provided autograder and follow the instructions below to test your project submission. Failure to do so may cause you to lose all the project points and there will be **absolutely no second chance**. Use a Linux-based system to test your project.

1. Download the zip file you submitted from Canvas.
2. Download the autograder from GitHub:
   https://github.com/nehavadnere/CSE546-FALL-2025.git
   In order to clone the Github repository follow the below steps:
   a. git clone https://github.com/nehavadnere/CSE546-FALL-2025.git
   b. cd CSE546-FALL-2025/
   c. git checkout project-1-part-1
   d. Create a directory "submissions" in the CSE546-FALL-2025 directory and move your zip file to the submissions directory.
2. To facilitate the testing, a standard face dataset and the expected recognition output of each image are provided to you at:
   - Input: dataset/face_images_1000.zip
   - Output: dataset/classification_face_images_1000.csv

3. Prepare to run the autograder
    a. Install Python: sudo apt install python3
    b. Populate the class_roster.csv
        i. If you are a student; replace the given template only with your details.
        ii. If you are a grader; use the class roster for the entire class
4. Run the autograder
    - To run the autograder: python3 autograder.py --num_requests 100 --img_folder="<dataset folder path>" --pred_file="<output classification csv file path>"
    - The autograder will look for submissions for each entry present in the class_roster.csv
    - For each submission the autograder will
        - Validate if the zip file adheres to the submission guidelines as mentioned in the project document.
            - If Yes; proceed to next step
            - If No; allocate 0 grade points and proceed to the next submission
        - The autograder extracts the credentials.txt from the submission and parses the entries.
        - Use the Grader IAM credentials to test the project as per the grading rubrics and allocate grade points.
    - The autograder has a workload generator component to generate requests to your web tier.

Sample Output:

```
------------------------------------------------------------
---------------- Executing Test-Case:2 ----------------
[EC2-log] AmazonEC2ReadOnlyAccess policy attached with grading IAM
[EC2-log] EC2-state Pass. web-tier instance found. With state: running.Points deducted: 0
[S3-log] AmazonS3FullAccess policy attached with grading IAM
[S3-log] - WARN: If there are objects in the S3 buckets; they will be deleted
[S3-log] ------------------------------------------------------
[S3-log] S3 Bucket:1225754101-in-bucket has 0 object(s). Points deducted:0
[DB-log] SimpleDB Domain: 1225754101-simpleDB exist
[DB-log] SimpleDB Domain: 1225754101-simpleDB exist with 1000 items. Points deducted:0
---------------- Executing Test-Case:3 ----------------
[Workload-gen] Attempt-1 1000/1000 requests successful.
[Workload-gen] All requests have been processed or retried.
[Workload-gen] ----- Workload Generator Statistics -----
[Workload-gen] Total number of requests: 1000
[Workload-gen] Total number of requests completed successfully: 1000
[Workload-gen] Total number of failed requests: 0
[Workload-gen] Total number of correct predictions : 1000
[Workload-gen] Total number of wrong predictions: 0
[Workload-gen] Total Test Duration: 5.070960998535156 (seconds)
[Workload-gen] ----------------------------------

[S3-log] Bucket:1225754101-in-bucket is now EMPTY !!
[Test-Case-3-log] 1000/1000 entries in S3 bucket:1225754101-in-bucket.Points:[20.0/20]
[Test-Case-3-log] 1000/1000 completed successfully.Points:[20.0/20]
[Test-Case-3-log] 1000/1000 correct predictions.Points:[20.0/20]
[Test-Case-3-log] Test Latency: 5.070960998535156 sec. `latency<=10`.Points:[40/40]
Total Grade Points: 100.0
Removed extracted folder: extracted
Execution Time for Doe John ASUID: 1225754101: 70.24327301979065 seconds
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Grading complete for Project-1. Check the Project-1-grades.csv file.
```

Submission

1. Create a Grading IAM User. The TA will use this only for grading.
    a. You MUST name the grading IAM user as “cse546-AutoGrader”
    b. For Project-1, the Grading IAM requires only these permission IAMReadOnlyAccess, AmazonEC2ReadOnlyAccess, AmazonS3FullAccess, SecurityAudit
2. Submit the following in a zip file to Canvas:
    a. Credentials: Make a directory named “credentials”. In the ‘credentials’ folder, create a txt file “credentials.txt” with following parameter values separated by commas in the order as mentioned below:
        i. ACCESS KEY ID of the grading IAM user
        ii. SECRET ACCESS KEY of the grading IAM user
        iii. IPv4 address of the web tier

```
kjha9@en4113732l:~/git/GTA-CSE546-SPRING-2025/Project-1/part-1/grader$
kjha9@en4113732l:~/git/GTA-CSE546-SPRING-2025/Project-1/part-1/grader$ cat credentials/credentials.txt
AKIA5GLEVSDVHWPNQX5H,dygP3gamUan3YAYHQj2RO52zM68OMn/HIV1XFRGl,3.89.222.7
kjha9@en4113732l:~/git/GTA-CSE546-SPRING-2025/Project-1/part-1/grader$
```

    b. Web tier code: Make a directory named “web-tier” and include your web tier code file named “server.py”. Do not include any code that is not implemented by you.
    c. Do not submit any other files or you will lose points.
    d. You MUST name the zip file following the below naming convention: “Project1-<Your ASU ID>.zip”, e.g., Project1-1225754101.zip
    You can use the following command to create your zip file:
    zip -r Project1-1225754101.zip credentials/ web-tier/
    Note: Extensions automatically generated by Canvas due to multiple submission attempts are allowed.
3. Keep your web tier instance running until you are informed that grading is done. If your web tier is not reachable when your project is being graded, you will receive zero point for the entire project.
4. 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.
5. Do not do any kind of testing on your web tier after the submission deadline; otherwise, you may lose CPU credit, causing your instance to be slow during the grading and deductions to your project grade.

Grading

Note: The grader may use a workload different from the one provided to test your project during grading. Passing all the test cases does not guarantee all the points. Your code will be checked for correctness, plagiarism, and use of AI-generated code.

#	Test Case	Test Criteria	Test Rubrics	Points
1	Unzip submission and check folders/files	1) Submission can be unzipped 2) Required folders/files can be found 3) No extra files	1) Deduct all points if unzip fails 2) Deduct all points if credentials.txt can't be found 3) Deduct all points if web-tier code can't be found 4) Deduct 10 if there are extra files and another 10 if extra binary files	
2	Validate the initial state of the resources (EC2, S3 and SimpleDB)	To check if 1) If there exists an EC2 instance with the name "web-instance" and if the state of the web instance is "running" 2) The input S3 bucket must exist and should be empty 3) SimpleDB domain must exist and with 1000 elements	Deduct 33.33 points each if any of the resources does not exist or not in the correct initial state.	
3	Validate the completeness, correctness and the latency of the requests	Run workload generator script on the provided URL by the students with 100 requests 1) The HTTP response code must be 200 for all the requests 2) The input bucket should have all the input images 3) All the classification results must be correct 4) Total runtime must be within the limits suggested in the project document	Out of 100 requests 1) +0.2 for completion of every requests (20 total) 2) +0.2 correct classification of the request (20 total) 3) +0.2 storing the input in the S3 bucket (20 total) 4) Total test duration for 100 requests (40 total) - less than 3 sec (40) - between 3 sec - 6 sec (20) - between 6 seconds - 9 seconds (10) - greater than 9 seconds (0)	100

Policies

- 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 for your work than to submit late for no credit.
- Every student needs to **work independently** on this exercise. We encourage high-level discussions among students to help each other understand the concepts and principles. However, a code-level discussion is prohibited and plagiarism will directly lead to failure of this course. We will use anti-plagiarism tools to detect violations of this policy.
- **The use of generative AI tools is not allowed** to complete any portion of the assignment.