

Project 1 (Part 1): IaaS

[CSE 546] [SPRING_2024] Cloud Computing

Due by **02/16/2024 at 11:59:59pm**

Summary

In the first project, we will build an elastic application that can automatically scale out and in 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 focus on familiarizing ourselves with AWS, its key IaaS resources, and the app development process on AWS; We will develop the web tier in Part 1, which will be used as the front end of our multi-tiered cloud app in Part 2.

We recommend you to follow the steps below to complete Part 1. But these are not exact step-by-step instructions. Check the AWS documentation for more information.

STEP-1: AWS Development Setup

1. Setting up the AWS Account

- a. Go to this [link](#) to Create an AWS account.
- b. Choose Account Type as “Personal” and fill in your Contact and Payment details
- c. In the Identity Verification step, confirm your identity.
- d. Select the “Basic Plan”

Important: Make sure to monitor your AWS usage and billing so you do not get charged. Use only the resources from the [AWS Free Tier](#). Follow [this tutorial](#) to use the CloudWatch to automatically generate billing alerts to you.

2. How to get AWS Access Key ID

- a. Go to the Amazon Web Services console and click on the name of your account (it is located in the top right corner of the console).
- b. In the expanded drop-down list, select *Security Credentials*.

- c. Click the “Generate new key” in “Access keys (access key ID and secret access key)”.
- d. Click “Show key”, and you will see the Access Key ID and the Secret Access Key.
- e. Copy your Access Key ID and the Secret Access Key to Eclipse for configuring the AWS account on the editor.

3. Creating IAM users for development and grading

An AWS Identity and Access Management (IAM) user is an entity you create in AWS. The IAM user represents the human user or workload who uses the IAM user to interact with AWS. A user in AWS consists of a name and credentials. An IAM user with administrator permissions differs from an AWS account root user.

We ask you to create two IAM users, one used by yourself for the development of your cloud app; and the other for the TA to use to check and grade your app. Use the following [link](#) to set up IAM user accounts. Follow the least privilege principle to give each IAM user only the necessary permissions.

- **Development IAM**

- Use this for all development tasks related to this course.
- Assign full permissions for any AWS resource you use for this project.

- **Grading IAM**

- The TA will use this only for grading and hence requires limited permissions.
- For **Project-1 Part-1**, the Grading IAM requires only this permission
 - AmazonEC2ReadOnlyAccess

4. Install and configure AWS CLI

- a. Based on your OS, install AWS Client as per the documentation [here](#)
- b. Configure AWS CLI using the command
 - i. To configure the CLI, execute the following AWS CLI command.

```
aws configure
```

- ii. You will be prompted to provide the access key, secret key, default region, and default output format (json/yaml). Provide the required details as shown below.

```
AWS Access Key ID : [*****LBSW]
AWS Secret Access Key: [*****QKwi]
```

```
Default region name: us-east-1
Default output format: json
```

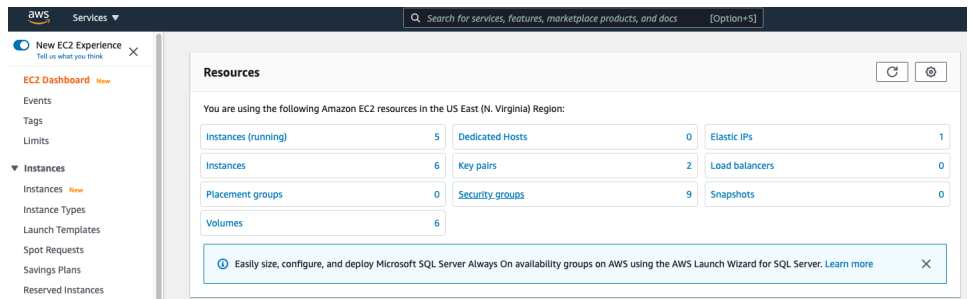
- iii. After configuration, a folder named **.aws** gets created in the user's home directory.
- iv. Inside the **.aws** directory, you will see the following two files.
 - 1. **config**: It contains all the default configs like region and output. You can change these values anytime and add new values as default.
 - 2. **credentials**: This file contains the access key and secret key as plain text.

5. Creating an AWS Project (Code examples shown with Python)

- a. **Creating EC2 Instances**: The following steps to create EC2 instances:

- i. **Define your security credentials**

- 1. Create your Key pairs in the EC2 dashboard



- 2. Copy the pem file into your `~/aws` folder

Create key pair

Key pair

A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.

Name

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

File format

☒ **pem**
For use with OpenSSH

☐ **ppk**
For use with PuTTY

Tags (Optional)

No tags associated with the resource.

3. Add SSH Inbound rules so that you can log in to your instance through an SSH session (**You may need to add more rules for your project**)

Inbound rules				
Type	Protocol	Port range	Source	Description - optional
SSH	TCP	22	0.0.0.0/0	-

4. Code example to define security credentials

```
AWS_ACCESS_KEY_ID      = "YOUR AccessKeyId"
AWS_SECRET_ACCESS_KEY  = "YOUR AWSSecretKey"
```

ii. Setup an Amazon EC2 Client:

1. Use the region of "US_EAST_1"

```
ec2 = boto3.resource(
    'ec2',
    region_name='US_EAST_1',
    aws_access_key_id=config('AWS_ACCESS_KEY_ID'),
    aws_secret_access_key=config('AWS_SECRET_ACCESS_KEY'))
```

iii. Launch an EC2 instance

1. Use AMI "ami-00ddb0e5626798373" for basic Ubuntu 18.04, as shown in this example. (You may use any Ubuntu AMI for webtier.)

```
ami_id = "ami-066ea1555baeb8dc4"
instance = ec2_client.create_instances(
    ImageId=ami_id,
    MinCount=1,
    MaxCount=1,
    InstanceType="t2.micro",
    TagSpecifications=[{'ResourceType': 'instance',
                        'Tags': [{
                            'Key': 'Name',
                            'Value': 'WebTier' } ]}]])
```

iv. Set tags for the EC2 Instance

```
CreateTagsRequest createTagsRequest = new
    CreateTagsRequest().withResources(
        instance.getInstanceId())
        .withTags(new Tag("Name", "Your Tag Name"));
ec2Client.createTags(createTagsRequest);
```

v. **Start the Instance**

```
instance = ec2_client.run_instances(  
    ImageId=ami_id,  
    MinCount=1,  
    MaxCount=1,  
    InstanceType="t2.micro",  
    TagSpecifications=[{'ResourceType': 'instance',  
                        'Tags': [{  
                            'Key': 'Name',  
                            'Value': 'WebTier Worker' }]}])
```

vi. **Check the state of the instance.**

You can see that the created EC2 instance is running

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability...	Public IPv4 DNS
<input type="checkbox"/>	cloud_computing_instance_test	i-045ffa498eb2194f6	Running	t2.micro	2/2 checks ...	No alarms +	us-west-2b	ec2-54-70-202-35.us

vii. **Connect to the Instance:**

You should be able to log into your instance with ssh session



Connect to instance [Info](#)

Connect to your instance i-0b2d180f9b7335644 (my-ubuntu-2) using any of these options


[EC2 Instance Connect](#) | [Session Manager](#) | **[SSH client](#)**

Instance ID

 [i-0b2d180f9b7335644](#) (my-ubuntu-2)

1. Open an SSH client.
2. Locate your private key file. The key used to launch this instance is open-tee.pem
3. Run this command, if necessary, to ensure your key is not publicly viewable.
 `chmod 400 open-tee.pem`
4. Connect to your instance using its Public DNS:
 `ec2-34-239-130-207.compute-1.amazonaws.com`

Example:

 `ssh -i "open-tee.pem" ubuntu@ec2-34-239-130-207.compute-1.amazonaws.com`

STEP-2: Developing the Web Tier

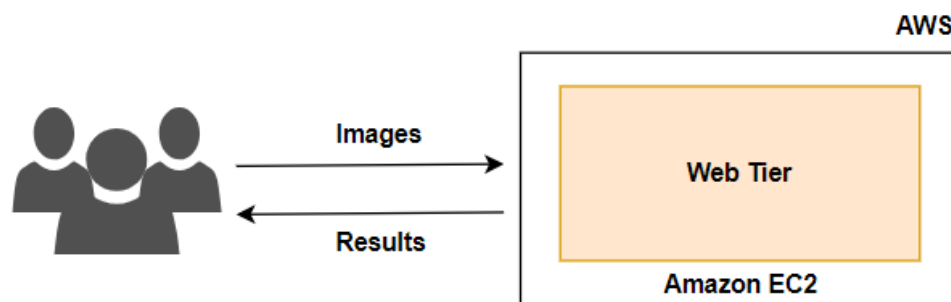
The web tier will receive face recognition requests from clients and return the classification results to the clients. In Part 1, instead of using a real classification model, which requires the app tier and data tier, we will use a lookup table to emulate the model inference process.

We will develop the web tier using a single EC2 micro instance. Assign this instance a static IP address so it does not change during development and testing.

An Elastic IP address is a static IPv4 address designed for dynamic cloud computing. It is allocated to your AWS account and yours until you release it. Follow the following help documents to determine how to assign an Elastic IP address to an EC2 Instance.

- [Elastic IP addresses - Amazon Elastic Compute Cloud](#)
- [How to Add a Static IP to an AWS EC2 Instance - DEV Community](#)
- [allocate_address - Boto3 1.34.27 documentation \(amazonaws.com\)](#)

Note: AWS has recently updated its pricing policy for [Elastic IPv4 addresses](#). Therefore, we recommend developing and testing the cloud app without associating with an Elastic IP. Once testing is complete, you can associate an Elastic IP for final testing. TAs will utilize the Elastic IP to grade your submission. This approach helps avoid any costs associated with using Elastic IPs, as the AWS Free Tier for EC2 now includes 750 hours of public IPv4 address usage per month for the first 12 months, effective February 1, 2024.



The web tier should fulfill the below requirements:

1. It should take images received from users as input and perform face recognition on these images by looking up the classification results provided to you. It should also return the recognition result as output to the users. The input from each request is a .jpg file, and the output in each response is the prediction 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>:<prediction_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](https://visa-lab.github.io/CSE546-Cloud-Computing/face_images_1000.zip)
- **Output:** [visa-lab/CSE546-Cloud-Computing/classification_face_images_1000.csv](https://visa-lab.github.io/CSE546-Cloud-Computing/classification_face_images_1000.csv)

2. The web tier should be able to handle multiple requests concurrently and as quickly as possible. The recognition results should all be correct. For 100 concurrent requests, the total runtime is about 13.67 seconds; for 1000 concurrent requests, the total runtime is about 2.04 seconds.
3. To facilitate testing, you **MUST** use only the resources from the US-East-1 region, and you **MUST** name your web-tier instance “**web-instance**”

Testing & Grading

- Use the provided [workload generator](#) to test your app thoroughly.
- The grading will be done using automated scripts and following the provided below

	Test Case	Test Criteria	Poor	Fair	Good	Excellent	Total Points
1	Validate EC2 Instance	To check if 1) If there exists an EC2 instance with the name "web-instance" 2) if exists, then check if the state of the web instance is "running"	There is no EC2 instance with the name "web-instance" (0)	The EC2 instance with the name "web-instance" exists, but is not in a "running" state (5)		The EC2 instance with the name "web-instance" exists and is in a "running" state (10)	10
2	Validate the completeness of the requests	1) Run workload generator script on the provided URL by the students with 30 requests 2) The HTTP Response code must be 200 for all the requests	Out of 30 requests, +1 is for completion of every request.				30
3	Validate the correctness of the classification result	1) Run workload generator script on the provided URL by the students with 30 requests	Out of 30 requests, +1 for the correct classification of each request				30

		2) The HTTP Response code must be 200 for all the requests 3) All the classification results must be correct					
4	Validate the total end to end latency of the cloud app	1) Run workload generator script on the provided URL by the students with 1000 requests 2) The HTTP Response code must be 200 for all the requests 3) All the classification results must be correct 4) Total runtime must be within the limits suggested in the project document	The total test duration for 1000 requests is greater than 30 seconds (7.5)	The test duration for 1000 requests is between 15 seconds - 30 seconds (15)	The total test duration for 1000 requests is between 5 sec - 15 sec (22.5)	The total test duration for 1000 requests is less than 5 sec (30)	30

- Test your web tier using the provided workload generator and [grading](#) script. If they fail to execute, you will receive **0** points.

Submission

The submission requires three components; all must be done by **02/16/2024 at 11:59:59pm**.

1. Provide the address of your web tier and your grading user credentials using a [Google Form](#).
2. 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.
3. Keep your web tier instance running until you are informed that grading is done.

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.