Project 2 (Part II): Edge Computing

Due by **May 11th 10:00 AM (firm deadline)**

Summary

In the second part of Project 2, we will move part of the face recognition pipeline that we built in Part I out of AWS Lambda, and implement it using AWS' IoT and edge computing services.

Edge computing performs computations near where the data is collected, i.e., Internet of Things, and enables real-time responses by avoiding transferring raw data to the remote cloud and privacy protection by processing the raw data on user-trusted devices.

Instead of using real IoT devices, we will use EC2 instances to emulate them, but we will use real AWS IoT services including IoT Greengrass and MQTT for the development. Through this project, we will gain hands-on experience with IoTs and edge computing and gain skills that are essential for building modern, IoT-data-driven ML-based edge/cloud applications.

Description

Our application will use AWS IoT Greengrass and AWS Lambda to implement a distributed pipeline to recognize faces in video frames collected from Internet of Things (IoT) devices such as smart cameras (illustrated in Figure 1).

1. The pipeline starts with an IoT device, sending video frames to a Greengrass Core device using MQTT.
2. The face detection program, running as a Greengrass component on the Core device, receives video frames, performs face detection using a machine learning model (MTCNN), and produces the detected faces.
3. The detected faces are then sent to an SQS request queue, triggering the face-recognition function running on AWS Lambda.
4. The face recognition function performs face recognition on the detected faces using a ML model (FaceNet), and produces the classification results of the recognized faces.
5. The recognition results are to an SQS response queue and retrieved by the IoT device.

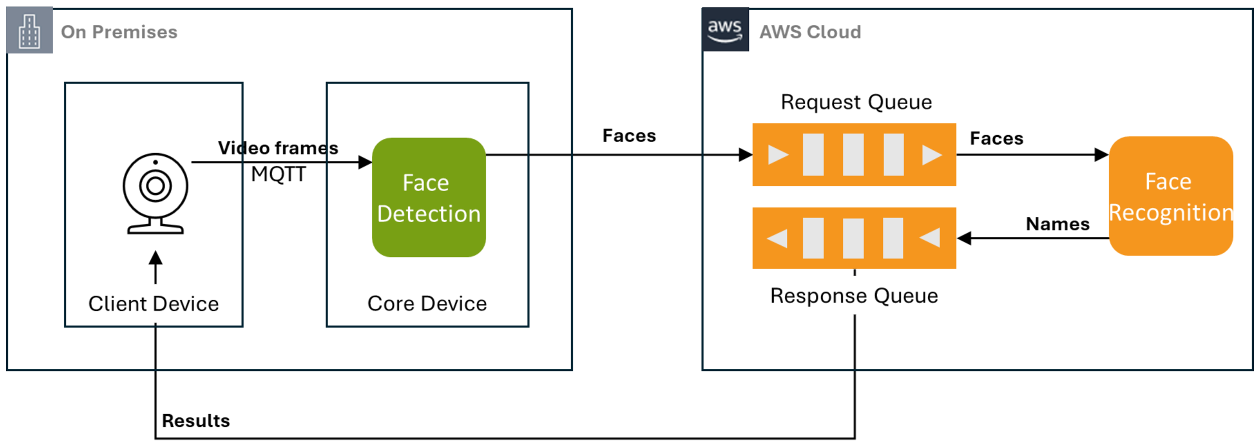


Figure 1: Architecture diagram of Project 2 (Part II)

Cost Estimation: This project may cost USD 2.34 for Greengrass, ECR, Lambda, and SQS services for 1 month. [Cost Estimation Link]

Note this project is quite complex as it involves multiple, distributed components. Make sure to follow the instructions below carefully.

Step 1: Setting up IoT Greengrass Core

AWS IoT Greengrass Core is the software that runs on edge devices to enable local computation, data management, and communication with other devices and the AWS cloud. We call the device that runs IoT Greengrass Core the *core device*. We will use an EC2 instance to emulate a core device and run the IoT Greengrass Core.

1. **Launch an EC2 Instance :**
   a. Launch an EC2 instance of type t2.micro with Amazon Linux 2023.
   b. For ease of use make sure the security group configuration allows all inbound traffic from any IP address (0.0.0.0/0) on all ports and protocols.
   c. You MUST name this EC2 instance: “IoT-Greengrass-Core”

2. **Set up the Linux instance for AWS Greengrass v2**
   a. Install the Java runtime:
      i. Install OpenJDK on your device

```
sudo dnf install java-11-amazon-corretto -y
```

   ii. Check the java version
       ```
       java -version
       ```
b. Create the default system user and group that runs components on the device.
   ```
   sudo useradd --system --create-home ggc_user
   sudo groupadd --system ggc_group
   ```
c. Verify that the user that runs the AWS IoT Greengrass Core software (typically root), has permission to run sudo with any user and any group.
   i. Open the `/etc/sudoers` file.
       ```
       sudo visudo
       ```
   ii. Verify that the permission for the user looks like the following example.
       ```
       root ALL=(ALL:ALL) ALL
       ```
d. Export AWS configurations as follows.
   ```
   export AWS_ACCESS_KEY_ID=<your-access-key>
   export AWS_SECRET_ACCESS_KEY=<your-secret-access-key>
   ```

3. **Install and configure the AWS IoT Greengrass Core software**
   a. Switch to the home directory. `cd ~`
   b. Download the AWS IoT Greengrass Core software:
      ```
      curl -s
      https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nu
      cleus-latest.zip > greengrass-nucleus-latest.zip
      ```
   c. Unzip the AWS IoT Greengrass Core software to a folder.
      i. Replace <*GreengrassInstaller*> with the folder that you want to use.
         ```
         unzip greengrass-nucleus-latest.zip -d
         <GreengrassInstaller> && rm greengrass-nucleus-latest.zip
         ```
   d. Run the following command to launch the AWS IoT Greengrass Core software installer. This command does the following:
      i. Create the AWS resources that the core device requires to operate.
      ii. Set up the AWS IoT Greengrass Core software as a system service that runs at boot.
      iii. Deploy the AWS IoT Greengrass CLI component, which is a command-line tool that enables you to develop custom Greengrass components on the core device.
      iv. Specify to use the `ggc_user` system user to run software components on the core device. On Linux devices, this command also specifies to use the `ggc_group` system group, and the installer creates the system user and group for you.
         ```
         sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
         ```

```
-jar ./GreengrassInstaller/lib/Greengrass.jar \
--aws-region region \
--thing-name MyGreengrassCore \
--thing-group-name MyGreengrassCoreGroup \
--thing-policy-name GreengrassV2IoTThingPolicy \
--tes-role-name GreengrassV2TokenExchangeRole \
--tes-role-alias-name
GreengrassCoreTokenExchangeRoleAlias \
--component-default-user ggc_user:ggc_group \
--provision true \
--setup-system-service true \
--deploy-dev-tools true
```

Replace argument values in the above command as follows.

1. */greengrass/v2*: The path to the root folder to use to install the AWS IoT Greengrass Core software.
2. *GreengrassInstaller*. The path to the folder where you unpacked the AWS IoT Greengrass Core software installer.
3. *region*. The AWS Region in which to find or create resources. You must use us-east-1 as the region.
4. *MyGreengrassCore*. The name of the AWS IoT thing for your Greengrass core device. If the thing doesn't exist, the installer creates it. The installer downloads the certificates to authenticate as the AWS IoT thing
5. *MyGreengrassCoreGroup*. The name of the AWS IoT thing group for your Greengrass core device. If the thing group doesn't exist, the installer creates it and adds the thing to it. If the thing group exists and has an active deployment, the core device downloads and runs the software that the deployment specifies.
6. *GreengrassV2IoTThingPolicy*. The name of the AWS IoT policy that allows the Greengrass core devices to communicate with AWS IoT and AWS IoT Greengrass. If the AWS IoT policy doesn't exist, the installer creates a permissive AWS IoT policy with this name.
7. *GreengrassV2TokenExchangeRole*. The name of the IAM role that allows the Greengrass core device to get temporary AWS credentials. If the role doesn't exist, the installer creates it and creates and attaches a policy named *GreengrassV2TokenExchangeRole* Access.

8. *GreengrassCoreTokenExchangeRoleAlias*. The alias to the IAM role that allows the Greengrass core device to get temporary credentials later. If the role alias doesn't exist, the installer creates it and points it to the IAM role that you specify.

v. When you run this command, you should see the following messages to indicate that the installer succeeded.
```
Successfully configured Nucleus with provisioned
resource details!
Configured Nucleus to deploy aws.greengrass.Cli
component
Successfully set up Nucleus as a system service
```
vi. Verify the Greengrass CLI installation on the device
```
/greengrass/v2/bin/greengrass-cli help
```

Step 2: Creating the Face Detection component on IoT Greengrass Core

AWS IoT Greengrass *components* are software modules that we deploy to Greengrass Core. Components can represent applications, runtime installers, libraries, or any code that you would run on a device. Every component is composed of a *recipe* and *artifacts*.

- **Recipes:** Every component contains a recipe file, which defines its metadata. The recipe also specifies the component's configuration parameters, component dependencies, lifecycle, and platform compatibility. The component lifecycle defines the commands that install, run, and shut down the component. You can define recipes in JSON or YAML format.
- **Artifacts:** Components can have any number of artifacts, which are component binaries. Artifacts can include scripts, compiled code, static resources, and any other files that a component consumes. Components can also consume artifacts from component dependencies.

Let's first create a sample HelloWorld component before we actually implement the face detection component. This HelloWorld component does not use MQTT for any message passing; instead a default message is defined in the recipe of the component. Follow the below steps to create the sample component.

1. Create a folder for your components with subfolders for recipes and artifacts.
```
mkdir -p ~/greengrassv2/{recipes,artifacts}
```
2. `cd ~/greengrassv2`

3. Create the recipe file that defines your component's metadata, parameters, dependencies, lifecycle, and platform capability
   ```
   vi recipes/com.example.HelloWorld-1.0.0.json
   ```
4. Paste the following in your recipe file. The recipe's `ComponentConfiguration` defines a `Message` parameter with a default value of "world." The `Manifests` section provides platform-specific install instructions. In the `Lifecycle`, the Greengrass core device runs the Hello World script using the Message value as an argument.
   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.HelloWorld",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "My first AWS IoT Greengrass component.",
     "ComponentPublisher": "Amazon",
     "ComponentConfiguration": {
       "DefaultConfiguration": {
         "Message": "world"
       }
     },
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
         }
       },
       {
         "Platform": {
           "os": "windows"
         },
         "Lifecycle": {
           "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
         }
       }
     ]
   ```

```
}
```

5. Create a folder for component artifacts
   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```
6. Create the artifact file
   ```
   vi artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```
7. Copy the following code in your py script.
   ```
   import sys
   message = "Hello, %s!" % sys.argv[1]
   # Print the message to stdout, which Greengrass saves in a log file.
   print(message)
   ```
8. Deploy the HelloWorld component using the following command
   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

9. Verify that the HelloWorld component runs and the message is logged. Use the following command
   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```
10. The following message should be printed in the log
   ```
   Hello, world!
   ```

Now, after familiarizing ourselves with the basic functionality of Greengrass components, we will proceed to develop the face detection component which will use MQTT for message passing. Follow the below steps to create your face detection component.

1) Create a folder for your components with subfolders for recipes and artifacts.
   ```
   mkdir -p ~/greengrassv2/{recipes,artifacts}
   ```
2) `cd ~/greengrassv2`
3) Create a recipe file
   ```
   vi recipes/com.clientdevices.FaceDetection-1.0.0.json
   ```
4) Use the recipe here as a sample to create the recipe for your component
5) Create a folder for the component artifacts.
   ```
   mkdir -p artifacts/com.clientdevices.FaceDetection/1.0.0
   ```
   `You have to store all the code that will be deployed in the component in the artifacts directory.`
6) Create a Python script artifact file for your FaceDetection component

7) `vi artifacts/com.clientdevices FaceDetection/1.0.0/fd_component.py`
8) Use this Python file to create the logic for the face detection function.

The face detection component must adhere to the following design considerations:

1) The face detection function MUST subscribe to a MQTT topic: "clients/<ASU-ID>-IoTThing"
2) When a message is published in the MQTT topic, the function should extract "encoded" (Base64-encoded input file), "request_id" (a unique ID to identify the requests), and "filename" (name of the input test image) from the JSON-formatted message.
3) It performs face detection on the received video frame using MTCNN (Multi-task Cascaded Convolutional Networks). It is a cascaded network that consists of three stages: face detection, facial landmark localization, and face alignment. These stages work together to detect faces and facial landmarks in an image.
4) You can refer to face-detection code to implement the ML logic of the face detection function. In this project, we will **not** install facenet_pytorch as a dependency, but rather import it as a code, like we did in Project-1. This facenet_pytorch code should be present in the artifacts directory of your component.
   To install Torch and other dependencies, use the following command in the recipe of your component
   ```
   python3 -m pip install --no-cache-dir --user awsiotsdk boto3
   numpy==1.24.4 torch==1.9.1+cpu torchvision==0.10.1+cpu
   torchaudio==0.9.1 --extra-index-url
   https://download.pytorch.org/whl/cpu
   ```
5) If a face is detected in the video frame the component will send the detected face to a SQS request queue (<ASU ID>-req-queue) to trigger the face recognition Lambda function on the AWS cloud.

Run the following command to deploy the component to the AWS IoT Greengrass core:
```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir ~/greengrassv2/recipes \
  --artifactDir ~/greengrassv2/artifacts \
  --merge "com.clientdevices.FaceDetection=1.0.0"
```

Note:

- The AWS IoT Greengrass Core software saves stdout from the component process to log files in the logs folder.

- Run the following command to verify Face Detection appears in the list of components of your core device:
  ```
  sudo /greengrass/v2/bin/greengrass-cli component list
  ```
- Use the following command to update the component with your changes.
  ```
  sudo /greengrass/v2/bin/greengrass-cli deployment create \
    --recipeDir ~/greengrassv2/recipes \
    --artifactDir ~/greengrassv2/artifacts \
    --merge "com.clientdevices.FaceDetection=1.0.0"
  ```
- Use the following command to restart the component. When you restart a component, the core device uses the latest changes.
  ```
  sudo /greengrass/v2/bin/greengrass-cli component restart \
    --names "com.clientdevices.FaceDetection"
  ```

Step 3: Setting up the IoT Greengrass Client Device

We will use another EC2 instance to emulate an IoT device which collects video frames for our application to process, as illustrated in Figure 1. The IoT device runs *AWS IoT Greengrass client* and connects to the IoT Greengrass Core. We call this device the *client device*.

1. Launch an EC2 Instance
   - Launch an EC2 instance of type t2.micro with Ubuntu.
   - For ease of use make sure the security group configuration allows all inbound traffic from any IP address (0.0.0.0/0) on all ports and protocols.
   - You MUST name this EC2 instance: “IoT-Greengrass-Client”

2. Install and Configure AWS CLI
   - Install AWS CLI
     - `curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"`
     - `unzip awscliv2.zip`
     - `sudo ./aws/install`
   - Configure AWS CLI using the command `aws configure` and enter the values from your account at the prompts displayed
   - Test your AWS CLI configuration using the below command. If it is configured correctly, the command should return an endpoint address from your AWS account.

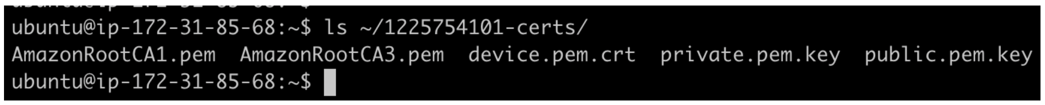
```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
```

3. Create AWS IoT resources for your IoT device
   Devices connected to AWS IoT are represented by *thing objects* in the AWS IoT registry. In the scope of this project, a *thing object* represents your IoT device (emulated by the EC2 instance).

   1) Create the thing object.
      ```
      aws iot create-thing --thing-name <name>
      ```
      You MUST name your thing object as "<ASU-ID>-IoTThing"

   2) Create and attach AWS IoT keys and certificates. These are used to authenticate the device to AWS IoT.
      a) Create a directory to store your certificate and key files.
         ```
         mkdir ~/<ASU-ID>-certs
         ```
      b) Download a copy of the Amazon certificate authority (CA) certificate by using this command
         ```
         curl -o ~/<ASU-ID>-certs/AmazonRootCA1.pem \
         https://www.amazontrust.com/repository/AmazonRootCA1.p
         em
         ```
      c) Run the following command to create your private key, public key, and X.509 certificate files. This command also registers and activates the certificate with AWS IoT.
         ```
         aws iot create-keys-and-certificate \
         --set-as-active \
         --certificate-pem-outfile "~/<ASU-ID>-certs/device.pem.crt"\
         --public-key-outfile "~/<ASU-ID>-certs/public.pem.key" \
         --private-key-outfile "~/<ASU-ID>-certs/private.pem.key"
         ```
         Save the *certificateArn* so that you can use it in subsequent commands.

         Make sure you follow the directory location and file naming conventions when you save certificates. The autograder relies on this and failure to do so will result in zero grade points. See an example below:



```
ubuntu@ip-172-31-85-68:~$ ls ~/1225754101-certs/
AmazonRootCA1.pem  AmazonRootCA3.pem  device.pem.crt  private.pem.key  public.pem.key
ubuntu@ip-172-31-85-68:~$ 
```

      d) Attach your thing object to the certificate you just created by using the following command and the *certificateArn* in the response from the previous command.

```
aws iot attach-thing-principal \
 --thing-name <ASU-ID>-IoTThing \
 --principal "certificateArn"
```

3) Create and attach a policy.
    a) Navigate to the AWS IoT Greengrass web console
    b) In the left navigation menu, Choose Security -> Policies
    c) There should exist a policy named 'GreengrassV2IoTThingPolicy', created from Step-1
    d) Make sure 'GreengrassV2IoTThingPolicy' grants full IoT Core operations (Connect, Publish, Subscribe, Receive) and Greengrass permissions (greengrass:*)
    e) In the left navigation menu, Choose Manage -> All devices -> Things
    f) Select <ASU-ID>-IoTThing
    g) Click on 'Certificates' and then on the Certificate ID attached to the thing
    h) Goto Policies -> Attach policies
    i) Select the 'GreengrassV2IoTThingPolicy' with correct permissions.

4. Associate the IoT greengrass client device to IoT greengrass core
   For a client device to use cloud discovery to connect to a core device, we must associate the devices. When we associate a client device to a core device, we enable that client device to retrieve the core device's IP addresses and certificates to use to connect.
   Follow the below steps:
    - Navigate to the AWS IoT Greengrass web console.
    - In the left navigation menu, choose Core devices.
    - On the Core devices page, choose the core device where you want to enable client device support.
    - On the core device details page, choose the Client devices tab.
    - On the Client devices tab, choose Configure cloud discovery.
      The Configure core device discovery page opens. On this page, you can associate client devices to a core device and deploy client device components.
        - Step 1: Select target core devices
            - Make sure the core device is selected created in the Step 1
        - Step 2: Associate client devices :
          Associating a client device's AWS IoT thing (<ASU-ID>-IoTThing) to the core device enables the client device to use cloud discovery to retrieve

the core device's connectivity information and certificates. Do the following:

- Click Associate client devices
- In the Associate client devices with core device, enter the name of the AWS IoT thing created in Step:2 to associate. (<ASU-ID>-IoTThing)
- Choose Add.
- Choose Associate.

▪ Step 3: Configure and deploy Greengrass components

- Greengrass nucleus - aws.greengrass.Nucleus
  - For the aws.greengrass.Nucleus component, choose Edit configuration.
  - For Component version, choose version 2.6.0 or later.
  - Choose Confirm.
- Client device auth - aws.greengrass.clientdevices.Auth
  - For the aws.greengrass.clientdevices.Auth component, choose Edit configuration
  - Under Configuration, in the Configuration to merge code block, it has the client device authorization policy
  - Replace the value of key 'thingName' with the name of the AWS IoT thing (<ASU-ID>-IoTThing) to connect as a client device. Refer to a sample configuration here
- MQTT 3.1.1 broker (Moquette) - aws.greengrass.clientdevices.mqtt.Moquette
  - Make sure it is enabled.
- MQTT bridge - aws.greengrass.clientdevices.mqtt.Bridge
  - Under Configuration, in the Configuration to merge code block, enter the following configuration
  - See the example configuration which specifies to relay MQTT messages on the clients/+/hello/world topic filter from client devices to the AWS IoT Core cloud service

```
{   "mqttTopicMapping": {
      "HelloWorldIotCoreMapping": {
      "topic":  "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
      }
    }
}
```

- Use the above sample configuration to filter for the topic clients/<ASU-ID>-IoTThing
- You also need to add a similar configuration to enable client-to-core communication via Pubsub. Add another JSON element with "target" as "Pubsub". Refer to a sample configuration here

- IP detector - aws.greengrass.clientdevices.IPDetector
  - Make sure it is enabled.

- Review and Deploy

5. Install AWS IoT Device SDK
   The IoT device can use the AWS IoT Device SDK to discover, connect, and communicate with a core device. We need to download and install the AWS IoT Device SDK v2 for Python to the IoT instance (the EC2 instance that we use to emulate the IoT device). The workload generator will use the SDK to communicate with our core device.
   - Clone the AWS IoT Device SDK v2 for Python repository to download it.
     ```
     git clone
     https://github.com/aws/aws-iot-device-sdk-python-v2.git
     ```
   - Install the AWS IoT Device SDK v2 for Python.
     ```
     python3 -m pip install --user ./aws-iot-device-sdk-python-v2
     ```

6. Generate Video Frames
   Instead of collecting video frames from a real video camera, we will emulate the generation of the video frames by publishing them to MQTT. Message Queuing Telemetry Transport (MQTT) is a lightweight publish/subscribe messaging protocol optimized for low-bandwidth and unreliable network environments. IoT devices publish MQTT messages that are identified by topics to communicate their state to AWS IoT.
   You can publish a message through the web console as follows. (For real testing, we will use a workload generator from the autograder.)
   - Navigate to the AWS IoT Greengrass web console.
   - Goto MQTT test client. This is to monitor the MQTT messages being passed.
   - Publish to the MQTT topic, a JSON-formatted message published in the MQTT topic, with key values as follows: "encoded" (Base64-encoded input file),

"request_id" (a unique ID to identify the requests), and "filename" (name of the input test image) from the JSON-formatted message.

- ○ Make sure that your face detection function has consumed the message, performed face detection, and triggered the face recognition Lambda function.
- ○ You can also verify the logs on the IoT Greengrass Core using
  `sudo tail -f /greengrass/v2/logs/com.example.clientdevices.FaceDetection.log`

Bonus

You have an opportunity to score a maximum of 20 bonus points in this project. Here's the bonus requirement: If the face detection component is passed a video frame where there are no faces then face detection the component will directly push the response to the SQS response queue (<ASU ID>-resp-queue); the response should have "No-Face" as the value of the 'result' key in the payload. By performing face detection on the edge, we can improve the response time and conserve the use of cloud in such a scenario.

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/CSE546-Cloud-Computing/CSE546-SPRING-2025.git
   In order to clone the Github repository follow the below steps:
   a. git clone https://github.com/CSE546-Cloud-Computing/CSE546-SPRING-2025.git
   b. cd CSE546-SPRING-2025/
   c. git checkout project-2-part-2
   d. Create a directory "submissions" in the CSE546-SPRING-2025 directory and move your zip file to the submissions directory.

3. To facilitate the testing, a standard video frame dataset and the expected recognition output of each image are provided to you at:
   a. Input: dataset/video-frames
   b. Output: dataset/Results_video_frames_100.csv

4. 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.
    c. Clean up your SQS queues.

5. Run the autograder
    - To run the autograder (without the bonus requirement): `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
        - 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 publish messages to the MQTT topic to which your face-detection component subscribes.
    - In the score of this project the workload generator will be executed on the IoT Client device (IoT-Greengrass-Client) and the autograder will take care of this.
        - You just need to make sure all the dependencies are installed on the IoT client EC2 instance.
        - Install the dependencies: `boto3, pandas, awscrt, and awsiot`. Install them using the command:
          ```
          pip3 install boto3, pandas, awscrt
          python3 -m pip install awsiotsdk
          ```
    - To test the bonus requirement, run the autograder with `--bonus_rubrics = 1` in addition to the other mandatory arguments, and use the following video frame dataset (which has images with no faces):
        i. Input: dataset/video-frames-bonus
        ii. Output: dataset/Results_video_frames_100-bonus.csv

6. 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-2, the Grading IAM requires only these permission
      i. IAMReadOnlyAccess
      ii. AmazonEC2ReadOnlyAccess
      iii. AmazonSQSFullAccess
      iv. AWSLambdaSQSQueueExecutionRole
      v. AWSLambda_ReadOnlyAccess
      vi. AWSGreengrassReadOnlyAccess
      vii. AWSIoTFullAccess

7. To facilitate testing, you MUST use only the resources from the US-East-1 region.

8. Debugging Tips:
   a. The `workload_generator.py` has a bunch of print statements commented. Like on line #42, 43, 44, 126, 137 and 222. Uncomment them to debug issues.
   b. On the IoT-Greengrass-Core `tail` the component logs to ensure the component is working as expected
   c. On the IoT-Greengrass-Client check using `ps` if the "`init_workload.py`" is running.
   d. You can comment the remote clean up stage of the autograder (`init_workload.py:#132`). Then use the test script (`<ASU-ID>_grading/CSE546-SPRING-2025/<ASU-ID>_exec.sh`) to directly debug on your IoT client.
   e. If you encounter ssh connection issues while using the AMI's mentioned for the EC2 instance.
   f. If you encounter issues with Greengrass discovery, ensure you are using the correct certificates and keys, and that they are maintained in the correct folder structure.

9. Sample Output
   <u>Please note in the below sample output some portion logs of zip and unzip are trimmed in order to save space and maintain readability. These are highlighted in red.</u>

```
++++++++++++++++++++++++++++++ CSE546 Autograder  ++++++++++++++++++++++++++++++
- 1) Extract the credentials from the credentials.txt
- 2) Execute the test cases as per the Grading Rubrics
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++ Autograder Configurations ++++++++++++++++++++++++++++
Project Path: /home/local/ASUAD/kjha9/git/GTA-CSE546-SPRING-2025/Project-2/part-2/grader
Grade Project: Project-2
Class Roster: class_roster.csv
```

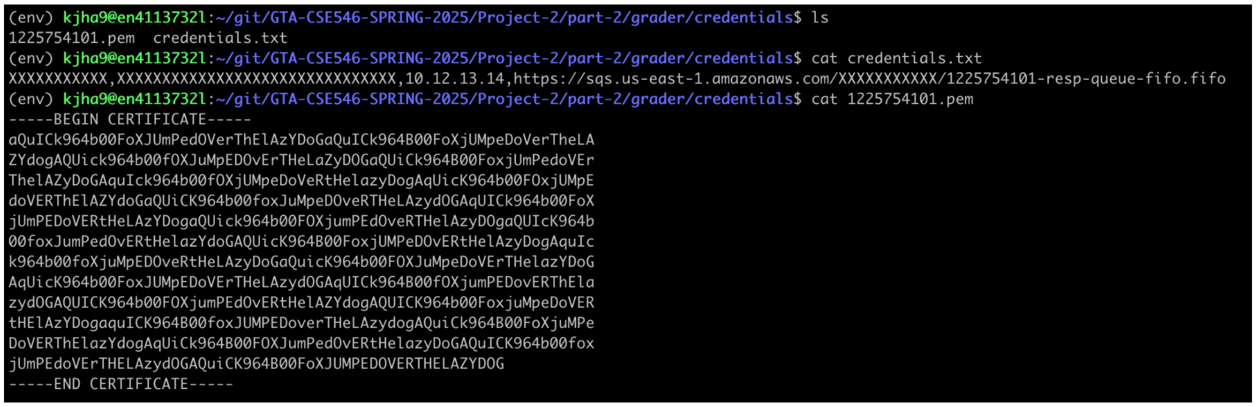
```
Zip folder path:
/home/local/ASUAD/kjha9/git/GTA-CSE546-SPRING-2025/Project-2/part-2/grader/submissions
Grading script:
/home/local/ASUAD/kjha9/git/GTA-CSE546-SPRING-2025/Project-2/part-2/grader/grade_project2_p1.py
Test Image folder path: ../../datasets/frames
Classification results file: ../../datasets/FaceRecognitionResults.csv
Autograder Results: Project-2-grades.csv
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++ Grading for Doe John ASUID: 1225754101 +++++++++++++++++++++
Extracted
/home/local/ASUAD/kjha9/git/GTA-CSE546-SPRING-2025/Project-2/part-2/grader/submissions/Project2-12257
54101.zip to extracted
File: extracted/credentials/credentials.txt has values ('XXXXXXXXXXXXXXXX', 'XXXXXXXXXXXXXXXX',
'X.XX.XX.XX', 'https://sqs.us-east-1.amazonaws.com/XXXXXXXXX/1225754101-resp-queue-fifo.fifo')
Credentials parsing complete.
-----------------------------------------------------------------
IAM ACCESS KEY ID: XXXXXXXXXXXXXXXX
IAM SECRET ACCESS KEY: XXXXXXXXXXXXXXXX
-----------------------------------------------------------------
Following policies are attached with IAM user:cse546-AutoGrader: ['AWSGreengrassReadOnlyAccess',
'AWSIoTFullAccess', 'AmazonEC2ReadOnlyAccess', 'IAMReadOnlyAccess', 'AmazonSQSFullAccess',
'AWSLambda_ReadOnlyAccess']
[IAM-log] AmazonEC2ReadOnlyAccess policy attached with grading IAM
[IAM-log] AmazonSQSFullAccess policy attached with grading IAM
[IAM-log] AWSLambda_ReadOnlyAccess policy attached with grading IAM
[IAM-log] AWSGreengrassReadOnlyAccess policy attached with grading IAM
[Cloudwatch-log] Alarm:Billing-alarm-5$ with
ARN:arn:aws:cloudwatch:us-east-1:XXXXXXXXX:alarm:Billing-alarm-5$ found in state:ALARM. It is
configued with statistic:Maximum, threshold:5.0 and Comparison Operator:GreaterThanOrEqualToThreshold
[Cloudwatch-log] CAUTION !!! Billing
alarm:arn:aws:cloudwatch:us-east-1:XXXXXXXXX:alarm:Billing-alarm-5$ is triggered. Release the
unwanted resources
[Cloudwatch-log] CAUTION !! You do not have a Cloudwatch alarm set. Kindly refer to the Project-0
document and learn how to set a billing alarm
-------------- CSE546 Cloud Computing Grading Console -----------
IAM ACCESS KEY ID: XXXXXXXXXXXXXXXX
IAM SECRET ACCESS KEY: XXXXXXXXXXXXXXXX
Elastic IPv4 address of IoT-Greengrass-Client :X.XX.XX.XX
SQS Response Queue URL: https://sqs.us-east-1.amazonaws.com/XXXXXXXXX/1225754101-resp-queue-fifo.fifo
-----------------------------------------------------------------
----------------- Executing Test-Case:1 ----------------
[EC2-log] IoT Greengrass Core and client validation Pass. Found 1 IoT-Greengrass-Core instance in
running state. Found 1 IoT-Greengrass-Client instance in running state.Points deducted: 0
[IoT-log] Found a IoT Core Device: GreengrassQuickStartCore-195585ade54 with state HEALTHY.
[IoT-log] Found a component com.clientdevices.FaceDetection in RUNNING state.
[IoT-log] 1225754101-IoTThing found. Points deducted: 0
[Lambda-log] The function: face-recognition-part-1 exists.
```

```
[Lambda-log] --------------------------------------------------------
[SQS-log] The expectation is that both the Request and Response SQS should exist and be EMPTY
[SQS-log] - WARN: This will purge any messages available in the SQS
[SQS-log] --------------------------------------------------------
[SQS-log] SQS Request Queue:1225754101-req-queue has 0 pending messages.
[SQS-log] SQS Response Queue:1225754101-resp-queue has 0 pending messages.
[SQS-log] Points deducted:0
----------------- Executing Test-Case:2 ----------------
  adding: 1225754101_grading/ (stored 0%)
  adding: 1225754101_grading/CSE546-SPRING-2025/ (stored 0%)
  adding: 1225754101_grading/CSE546-SPRING-2025/.git/ (stored 0%)
  ....
  ....
  adding: 1225754101_grading/CSE546-SPRING-2025/FaceRecognitionResults.csv (deflated 82%)
  adding: 1225754101_grading/CSE546-SPRING-2025/workload_generator.py (deflated 66%)
  adding: 1225754101_grading/CSE546-SPRING-2025/1225754101_exec.sh (deflated 44%)
  adding: 1225754101_grading/CSE546-SPRING-2025/command_line_utils.py (deflated 89%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/ (stored 0%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_67.jpg (deflated 6%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_90.jpg (deflated 1%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_36.jpg (deflated 7%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_51.jpg (deflated 6%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_27.jpg (deflated 6%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_73.jpg (deflated 1%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_17.jpg (deflated 1%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_24.jpg (deflated 1%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_71.jpg (deflated 8%)
  adding: 1225754101_grading/CSE546-SPRING-2025/frames/test_96.jpg (deflated 1%)
  ....
  ....
  (Last message repeated 90 times)
 Creating the testing package ...
 -- Git clone Project-2 Part-2
 -- Generate the test script file
Shell script '1225754101_exec.sh' generated successfully!
 -- Move the test package to the IoT-Greengrass-Client ...
 -- ssh connect to the IoT-Greengrass-Client : X.XX.XX.XX
 -- Execute the workload generator on the IoT-Greengrass-Client : X.XX.XX.XX
Archive:  1225754101_grading.zip
   creating: 1225754101_grading/
   creating: 1225754101_grading/CSE546-SPRING-2025/
   creating: 1225754101_grading/CSE546-SPRING-2025/.git/
   ...
   ...
  inflating: 1225754101_grading/CSE546-SPRING-2025/FaceRecognitionResults.csv
  inflating: 1225754101_grading/CSE546-SPRING-2025/workload_generator.py
  inflating: 1225754101_grading/CSE546-SPRING-2025/1225754101_exec.sh
  inflating: 1225754101_grading/CSE546-SPRING-2025/command_line_utils.py
   creating: 1225754101_grading/CSE546-SPRING-2025/frames/
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_67.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_90.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_36.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_96.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_84.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_01.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_87.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_09.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_47.jpg
  inflating: 1225754101_grading/CSE546-SPRING-2025/frames/test_21.jpg
  ....
  ....
  (Last message repeated 90 times)
  inflating: 1225754101_grading/CSE546-SPRING-2025/README.md
Performing greengrass discovery... on thing IOT-client-workload-gen
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup(gg_group_
id='greengrassV2-coreDevice-GreengrassQuickStartCore-195585ade54',
cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-east-1:XXXXXXXXX:thing/Greengrass
QuickStartCore-195585ade54',
connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='172.31.82.115',
host_address='172.31.82.115', metadata='', port=8883)])], certificate_authorities=['-----BEGIN
CERTIFICATE-----\nabcdefghijklmnopqrstuvwxyzAquickBrownFoxJumpedOverThelazyDog\abcdefghijklmnopqrstuv
wxyzAquickBrownFoxJumpedOverThelazyDog\abcdefghijklmnopqrstuvwxyzAquickBrownFoxJumpedOverThelazyDog\a
bcdefghijklmnopqrstuvwxyzAquickBrownFoxJumpedOverThelazyDog\abcdefghijklmnopqrstuvwxyzAquickBrownFoxJ
umpedOverThelazyDog\abcdefghijklmnopqrstuvwxyzAquickBrownFoxJumpedOverThelazyDog\abcdefghijklmnopqrst
uvwxyzAquickBrownFoxJumpedOverThelazyDog\abcdefghijklmnopqrstuvwxyzAquickBrownFoxJumpedOverThelazyDog
\abcdefghijklmnopqrstuvwxyzAquickBrownFoxJumpedOverThelazyDog\abcdefghijklmnopqrstuvwxyzAquickBrownFo
xJumpedOverThelazyDog\abcdefghijklmnopqrstuvwxyzAquickBrownFoxJumpedOverThelazyDog\abcdefghijklmnopqr
stuvwxyzAquickBrownFoxJumpedOverThelazyDog\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:us-east-1:XXXXXXXXX:thing/GreengrassQuickStartCore-195585ade54 at host
172.31.82.115 port 8883
Connected!
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
Publish received on topic clients/1225754101-IoTThing
....
....
(Last message repeated 90 times)
```

```
MQTT publishing completed ...
Polling thread stopping ...
Workload complete! Dumping statistics...
[Workload-gen] ----- Workload Generator Statistics -----
[Workload-gen] Total number of requests: 100
[Workload-gen] Total number of requests completed successfully: 100
[Workload-gen] Total number of failed requests: 0
[Workload-gen] Total number of correct predictions : 100
[Workload-gen] Total number of wrong predictions: 0
[Workload-gen] Total test duration: 77.5672435760498 (seconds)
[Workload-gen] -----------------------------------------
 -- starting remote cleanup ..
 -- starting local cleanup ..
[Test-Case-3-log] 100/100 completed successfully.Points:[20.0/20]
[Test-Case-3-log] 100/100 correct predictions.Points:[20.0/20]
[Test-Case-3-log] Test Average Latency: 0.775672435760498 sec. `avg latency<2.5s`.Points:[60/60]
[Test-Case-3-log] ---------------------------------------------------------
Total Grade Points: 100.0
Removed extracted folder: extracted
Total time taken to grade for Doe John ASUID: 1225754101: 331.48881125450134 seconds
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Grading complete for Project-2. Check the Project-2-grades.csv file.
```

Submission

1. Submit the following in a zip file to Canvas:
   a. Credentials: Make a directory named “credentials”,
      i. Include your txt file named ‘credentials.txt’
         1. Add the following parameter values separated by commas in the order as mentioned below
            a. ACCESS KEY ID of the grading IAM user
            b. SECRET ACCESS KEY of the grading IAM user
            c. Elastic IPv4 address of your IoT-Greengrass-Client EC2 instance
            d. SQS response queue URL
      ii. Include the .pem key file. This is the pem file you use to ssh to an EC2 instance. You MUST name your pem file as <ASU-ID>.pem



```
(env) kjha9@en4113732l:~/git/GTA-CSE546-SPRING-2025/Project-2/part-2/grader/credentials$ ls
1225754101.pem  credentials.txt
(env) kjha9@en4113732l:~/git/GTA-CSE546-SPRING-2025/Project-2/part-2/grader/credentials$ cat credentials.txt
XXXXXXXXXXX,XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX,10.12.13.14,https://sqs.us-east-1.amazonaws.com/XXXXXXXXXXX/1225754101-resp-queue-fifo.fifo
(env) kjha9@en4113732l:~/git/GTA-CSE546-SPRING-2025/Project-2/part-2/grader/credentials$ cat 1225754101.pem
-----BEGIN CERTIFICATE-----
aQuICk964b00FoXJUmPedOVerThElAzYDoGaQuICk964B00FoXjUMpeDoVerTheLA
ZYdogAQUick964b00fOXJuMpEDOvErTHeLaZyDOGaQUiCk964B00FoxjUmPedoVEr
ThelAZyDoGAquIck964b00fOXjUMpeDoVeRtHelazyDogAqUicK964b00FOxjUMpE
doVERThElAZYdoGaQUiCK964b00foxJuMpeDOveRTHeLAzydOGAqUICk964b00FoX
jUmPEDoVERtHeLAzYDogaQUick964b00FOXjumPEdOveRTHelAzyDOgaQUIcK964b
00foxJumPedOvERtHelazYdoGAQUicK964B00FoxjUMPeDOvERtHelAzyDogAquIc
k964b00foXjuMpEDOveRtHeLAzyDoGaQuicK964b00FOXJuMpeDoVErTHelazYDoG
AqUicK964b00FoxJUMpEDoVErTHeLAzydOGAqUICk964B00fOXjumPEDovERThEla
zydOGAQUICK964b00FOXjumPEdOvERtHelAZYdogAQUICK964b00FoxjuMpeDoVER
tHElAzYDogaquICK964B00foxJUMPEDoverTHeLAzydogAQuiCk964B00FoXjuMPe
DoVERThElazYdogAqUiCk964B00FOXJumPedOvERtHelazyDoGAQuICK964b00fox
jUmPEdoVErTHELAzydOGAQuiCK964B00FoXJUMPEDOVERTHELAZYDOG
-----END CERTIFICATE-----
```

b. Face detection code: Make a directory named 'face-detection' and include your code file named 'fd_component.py'
c. Face recognition code: Make a directory named 'face-recognition' and include your code file named 'fr_lambda.py'
d. Note: DO NOT submit any additional Python files

2. You MUST name the zip file following the below naming convention:
    a. If you are not opting for bonus points: "Project2-<Your ASU ID>.zip", e.g., Project2-1225754101.zip
    b. If you are attempting the bonus points: "Project2-<Your ASU ID>-Bonus.zip", e.g., Project2-1225754101-Bonus.zip
    This is **IMPORTANT**. Otherwise, you will not be graded for the bonus points.

    You can use the following command to create your zip file:
    zip -r Project2-1225754101.zip credentials/ face-detection/ face-recognition/
    Note: Extensions automatically generated by Canvas due to multiple submission attempts will be managed by the grading script.

3. Do not submit any other file.

4. Do not change your code after the submission deadline. We 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 an academic integrity violation.

Rubrics

#	Test Objective	Test Criteria		Points

1	Validate the initial state of the resources (EC2, Lambda and SQS)	To check if 1) There is 1 instance named "IoT-Greengrass-Core" and in "running" state. 2) There is 1 instance named "IoT-Greengrass-Client" and in "running" state. 3) There exists only one Lambda function with name "face-recognition". 4) The SQS request and response queues must exist and be empty. 5) There exists a core device in HEALTHY state. 6) There exists a face detection component on the IoT core and in "running" state. 7) A IoT Thing named <ASU-ID>-IoTThing exists. 8) The face-recognition Lambda has a SQS trigger enabled.	1) Deduct 100 points if any of the below is true - IoT-Greengrass-Core is not accessible. - IoT-Greengrass-Client is not accessible. - A greengrass core device is not in HEALTHY state. - com.clientdevices.FaceDetection component does not exist in any core device or is not in running state. - <ASU-ID>-IoTThing does not exist. - face-recognition lambda does not exist. - SQS queues do not exist. - face-recognition Lambda does not have a SQS trigger. 2) Deduct 10 points each if any SQS queue is not empty.	
2	Validate the completeness, correctness and the latency of the requests	Run workload generator script to publish messages on provided MQTT topic for the face-detection component with 100 messages: 1) All requests must be successfully published and consumed. 2) All the classification results must be correct. 3) Total runtime must be within the limits suggested in the project document.	For each of the 100 requests 1) +0.2 correct classification result (20) 2) +0.2 for consumption of every request. (20) 3) (without bonus) Average latency for 100 requests (60) < 2.5 secs (60) >= 2.5 secs and < 3.5 secs (40) >=3.5 secs and < 4.5 secs (20) >= 4.5 secs (0) 4) (with bonus) Average latency for 100 requests (80) < 1.5 secs (80) >= 1.5 secs and < 2.5 secs (60) >=2.5 secs and < 3.5 secs (40) >= 3.5 secs (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.