Project 3 – worth 100 Points – due Monday 10/21 at 4 PM.

Part 1. AWS vs. Google Cost Estimation – 10 points

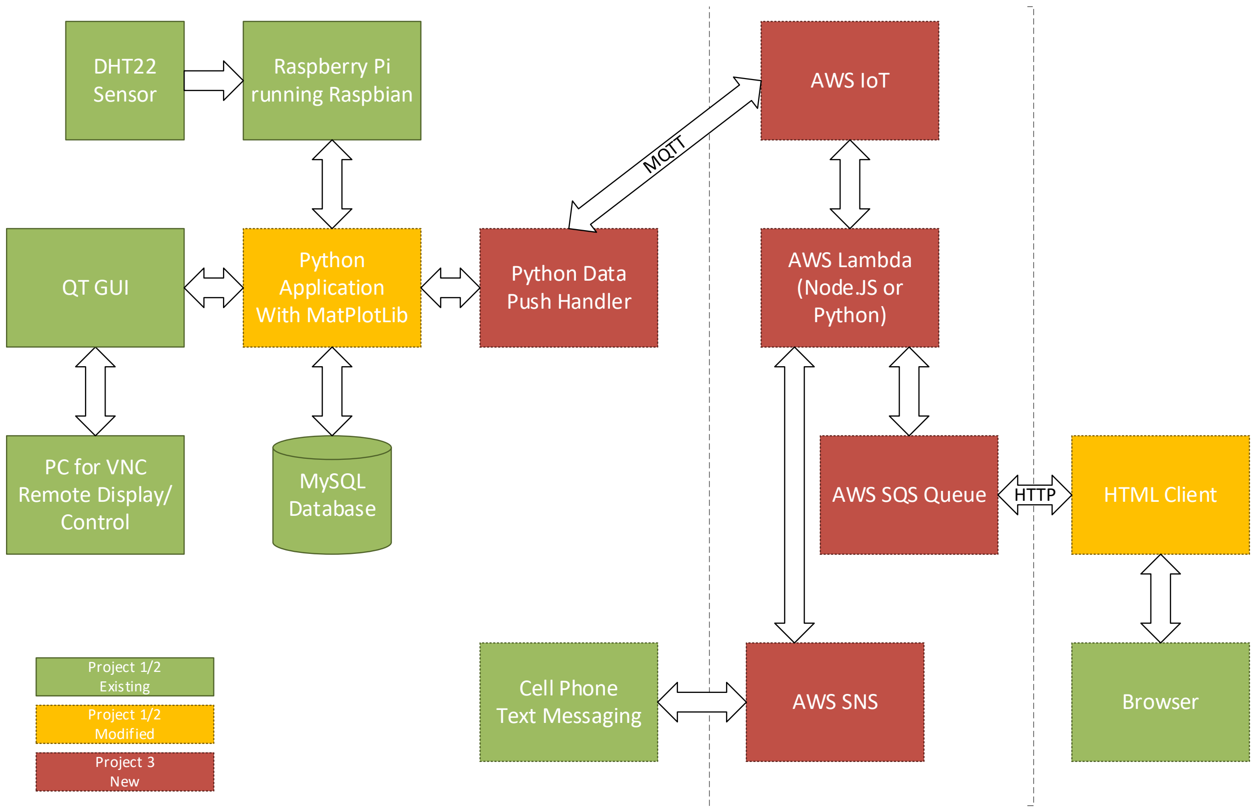
Using the AWS and Google Cloud cost estimators, determine the cost for a typical cloud-based system. Your goal is a simple monthly cost estimate to compare both Amazon Web Services and Google Cloud Platform. You may select the services and elements of your system: include at least two Linux VMs and some data storage; try to make the two systems as close to the same functionality as possible. Print the results for both calculations and include them in a PDF named CloudCost.PDF; include this PDF in your Project 3 repository

The AWS calculator is at: https://calculator.aws/#/addService

The Google calculator is: https://cloud.google.com/products/calculator/

Part 2. AWS Connectivity for Project 1 and 2 elements – 90 Points

- This project will allow you to explore further new technologies with your Project 1 and 2 parts, including MQTT and AWS elements including IoT Core, Lambda, SQS, and SNS
- The system will look something like this:



- Your Project 1 and 2 code will be used as a basis for Project 3.
- All functions of your Project 1 code and UI should continue to work after Project 3 is complete and when Project 3 is active.

- The AWS connection to AWS IoT should be made when your Project 1 code is started.
- All AWS connections must be appropriately authenticated as needed. Take care not to expose public security information.
- You will introduce a Python element in your Project 1, the Data Push Handler, for pushing MQTT messages to AWS via an IoT Thing connection. There will be two types of messages (likely JSON formatted):
    - One – a Data record with (at a minimum): Timestamp, Temperature, Humidity
    - Two – an Alert record with (at a minimum): Timestamp, Temperature Alert Level, Temperature Trigger Level, Humidity Alert Level, Humidity Trigger Level
- Note that the Data Push Handler can be a separate executable or it can be part of the Project 1 Python application.
- Your Data Push Handler will send a Data record anytime a new sensor reading comes into the Project 1 Python Application (including timed and on demand requests).
- The Data Push Handler will send an Alert record any time an incoming sensor reading of Temperature and/or Humidity exceeds alert trigger settings.
- The AWS IoT rule for your IoT Thing will send the incoming record to an AWS Lambda routine to be parsed. If the Lambda parser sees it is a Data record, the parser will send the data to an AWS SQS Queue. If the Lambda parser sees it is an Alert record, the parser will use AWS SNS to send a text message containing all the alert data to a cell phone via an SMS text message. (Note: If for some reason a cell phone is not available, you may generate an e-mail message instead.)
- Your prior Project 2 HTML client will be used to create an interface to the AWS SQS Queue. Use the AWS SQS ability to accept HTTP GET requests to pull messages from the AWS SQS Queue.
- Your HTML Client should be modified to provide two functions on button presses:
    - Get a single SQS record.
    - Get all SQS records.
- Each message pulled from the SQS queue should be displayed in a tabular form in the Project 2 web page. Duplicate messages may occur, these should be displayed as well. The client should keep the last 20 records in its tabular record display.
- The message can be dropped from the AWS SQS queue once the HTML client has received it.
- The HTML client must be able to switch from F to C display of temperature for retrieved records.
- All error conditions should be considered and allowed for. Documentation should indicate which errors are being monitored.
- For 10 points of extra credit, find a way for the HTML client to display the number of records currently in the SQS queue. This is a request that the HTML client will make on a button press, putting the count in a displayed text field. The text field should also be updated if a single or batch SQS retrieval is performed.

References:

- Google will be your friend here. Start with the dev guide for connecting a Pi to AWS IoT.
- Pi to AWS IoT, setting up IoT rules and security
    - https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html
    - https://techblog.calvinboey.com/raspberrypi-aws-iot-python/

- Using SQS with HTTP GET and POST requests:
  - https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-making-api-requests.html
- Using Lambda with SNS:
  - https://docs.aws.amazon.com/lambda/latest/dg/with-sns-example.html
- Using Lambda with SQS:
  - https://startupnextdoor.com/adding-to-sqs-queue-using-aws-lambda-and-a-serverless-api-endpoint/

Project Delivery

- The code should be yours and your team's work
  - Cite any sources for any Code from the Web; should include the URL of the resource you took it from
  - You may not directly use code from other teams, although they may give you advice or suggestions
  - If someone (students or class staff) helps you on part of your code, give credits in comments and the README and identify which code was provided
- Even though this is a prototype, I'd like to see well-structured Python and Node.JS code
- The project must run natively on an RPi3 development system
- The code must be well commented
  - A typical comment template can be copied (forked) from example.py in my GitHub repo at https://github.com/brmjr9/eid-fall2018
  - Comment/Docstring header for each file identifying the author and file description
  - Comments/Docstrings at any functions or classes including description, input, output
  - Comments for purpose of data structures or complex transactions (usually this is a why and not a how)
- Turn in a GitHub repo link (one submission per team) containing your project with
  - Any code files needed to run the project (not including standard libraries)
  - A README.md (markdown text file) including:
    - Title (i.e. EID Project 2)
    - Names of the developers/students on your team
    - A section called **Installation Instructions**
      - We should be able to follow the instructions to run your project on my RPi3 system (for Project 1,2,3,4 – not the Super-Project)
    - A section called **Project Work**
      - On a multi-person project, include who was responsible for what parts
    - A section called **Project Additions**
      - Describes any features you've added that are above the project requirements
    - (NEW) A section called **Project Issues**
      - Describes any features you had difficulty implementing and what the issues were
  - Remember to include AWS/Google cost comparison

Grading Rubric – Total 100 points (+ 10 possible extra credit)

- Project must be turned in via GitHub URL
- AWS vs. Google cost comparison in CloudCost.PDF – 10 points
- README.md structured as reviewed above – 10 points (-2 per missing element)
- Demonstration of features by executing project – 40 points
    1. AWS IoT Connectivity to Project 1 Pi via MQTT – 10 points
    2. Alert message receipt on cell phone – 5 points
    3. Single record retrieved from SQS – 5 points
    4. Batch records retrieved from SQS – 5 points
    5. Project 1 still functional (timed readings, on demand readings) – 5 points
    6. Temperature unit change on HTML – 5 points
    7. Error handling demonstration – 5 points
- Well commented and structured code – 40 points
    - Code to review (at a minimum):
        - Capture the code in your AWS Lambda function (Python or Node.JS)
        - Your Python Data Push Handler (and Project 1 changes)
        - Your modified HTML and JS elements (Project 2 changes)
        - For each above: (-2 to 4 for poor commenting, -2 to 4 for poor structure)
- 10 Points extra credit for queue message count function as described above
- 15% Grade penalty if turned in late (accepted for one week, then 0 points awarded)

RPi3 Handling Notes:

- As with all electronics, follow ESD guidelines to avoid damaging hardware
- Whenever possible – use the Linux command line or GUI to shut down or reboot the Pi, avoid just pulling power – may corrupt the system
- If you see a yellow lightning bolt on the RPi3 GUI, you are low on power (using 2 power supplies may be needed for the Pi and an attached display for instance)
- Keep a Git or other backup of your SD card and your work – they can and do corrupt