MSML 650: Cloud Computing

Fall 2023

Project Ideas

1. Compare and c[illegible]ses
   In this project, [illegible]/test different no-SQL databases:
   - Dynam[illegible] (https:[illegible]m/amazondynamodb/latest/developerguide/Introduction.html)
   - Mongc[illegible]godb.com/)
   - Cassandra (http://cassandra.apache.org/doc/latest/)

   Use the above docs (and possibly other sources, books or online) to learn about the above no-SQL database solutions, and compare/contrast them based on the following criteria:
   - System architecture: ability to scale, provide fast data access and reliability (availability)
   - Data modeling approach: the ability to represent data items and their relationships
   - Query languages/supported API operations: the ability to interact with the database and create/modify/delete tables, query tables, and add/modify/delete records to/from tables
   - Basic and complex data types: ability to support various data formats and data representation approaches

   In addition to DynamoDB, both MongoDB and Cassandra are available as managed AWS services (thus, you do not have to install them on EC2 instances, even though you can if you prefer to). Create a small database example (e.g. a music store, storing mp3 files with their associated meta data such as artist name, song title, genre, release date, price, etc.), and implement it in all three database systems. Add/query/modify/delete records in the databases. Based on the implementation and your experience, decide which database seems most appropriate for your use case.

2. Implement a photo gallery web application
   In this project, you will implement a cloud-based web application that allows users to upload photos, view photos and search photos based on their associated topics. The users should be able to upload photos, together with a provided user name (no authentication is needed) and an associated picture topic, which can be used to search the images. The application should also record a timestamp when the picture was uploaded. The application should have a list/gallery view of the uploaded photos, and this list/gallery should show a small (resized) version of the original image, together with the upload date/time, the uploading user's name and the associated topic. If the small image is clicked, a new page should show the full-sized image, together with the associated meta data. Users should be able to search the list/gallery view based on picture topics, and when uploading a new image, they should be able to a) either select and existing topic for a new image, or b) create a new topic. Users should also be able to delete pictures (for simplicity, you do not need to implement any authentication, and it is OK if any user can delete any picture).

The frontend should be implemented using html, CSS (and possibly JavaScript). The backend can be a traditional web server or a serverless (Lambda) solution. If you choose the former, I recommend using one of the web frameworks, e.g. flask (https://flask.palletsprojects.com/en/2.0.x/) for Python, as that can save you a lot of low-level coding work. If [illegible]das, you can store the static part of your website in an S3 bucket, and us[illegible] to route RESTful HTTP requests to your Lambda functions. You can use th[illegible]size-image Python libraries to resize the uploaded image. The easiest wa[illegible]nctionality is to resize the image when it is uploaded and then store the [illegible] the original (full-size) version with a different name.

First, think [illegible]em architecture. Do you want to use EC2 instances or a serverless/Lam[illegible]eb frameworks do you prefer to use? Where do you want to store the images themselves and where do you want to store the associated metadata?

Then design, implement and test your web application by trying out all of its functionality and verifying that they all work.

3. Create a data processing pipeline with AWS Kinesis

In this project, you will build a data processing pipeline that acquires and processes weather data using AWS Kinesis.

First, get familiar with AWS Kinesis (https://docs.aws.amazon.com/streams/latest/dev/introduction.html), its architecture and learn how to create producers and consumers around the Kinesis data streams using Python.

Then, create a producer (EC2 instance or Lambda) that collects weather information from the National Oceanic and Atmospheric Administration's (NOAA's) REST API. You can find information on this REST API, including the URL for the API endpoint, at https://www.programmableweb.com/api/noaa-climate-data-online-rest-api-v2. The climate data REST API documentation can be found at https://www.ncdc.noaa.gov/cdo-web/webservices/v2, and there are some data access examples and explanations of some of the data fields at https://grantwinney.com/what-is-noaa-api/. You may have to spend some time figuring out the API's data hierarchy and the meaning of various abbreviations present in the returned data. You can find some useful documentation here: https://www1.ncdc.noaa.gov/pub/data/cdo/documentation/.

Your data producer will need to issue REST API requests for weather data from all weather stations in the state of Maryland between October 1, 2021 and October 31, 2021. Please note that not all weather stations have weather data in the above date range. The producer should retrieve the daily weather parameters (dataset ID: GHCND) and extract the following values:

- the amount of precipitation (mm),
- the amount of snowfall (mm),
- the min/max temperature and/or the temperature at the time of observation (when available; in units of F or C, whichever you prefer).

Please also note that some weather stations only supply precipitation and snow measurements, and only some of them supply temperature data as well. Occasionally, some data values may be missing or not received.

The producer should add the date/time of observation, the station/location information (station ID, station name) to the extracted values, form a weather report record, and insert this record into the Kinesis data stream. You can use the requests Python library for sending/receiving REST API requests/responses via HTTP/HTTPS and you can use the json Python library to build and/or parse JSON documents.

The consumer (EC2 instance or Lambda) should remove the inserted weather records from the Kinesis dat [illegible] wo separate database items: one for snow/precipitation information, a [illegible] information. The snow/precipitation item should contain the timestamp [illegible] nformation and the snow/precipitation values, while the temperature it [illegible] timestamp, the weather station information, and the reported temp [illegible] w/precipitation item should be saved in a DynamoDB table called “Precipit [illegible] at it can be easily retrieved by the measurement’s location and should be [illegible] p value. Similarly, the temperature item should be saved in a DynamoDB table called “Temperature” in such a way that it can be easily retrieved the measurement’s location and should also be sorted by the timestamp value.

4. Create an IoT data collection system using a Raspberry Pi

In this project, you will build a custom IoT system that will collect temperature and humidity measurements from a local sensor and send the data to the cloud for processing. The cloud service you create will receive the measurements and save them in a database. This project will incur a cost of about $50-60 as the hardware components, including the Raspberry Pi, will need to be purchased.

First, read and learn about the AWS IoT Core service and the MQTT publish/subscribe IoT messaging protocol (https://docs.aws.amazon.com/iot/ latest/developerguide/what-is-aws-iot.html), the AWS IoT device SDKs using Python (https://docs.aws.amazon.com/iot/latest/developerguide/iot-sdks.html), and study the AWS IoT Python SDK documentation (https://aws.github.io/aws-iot-device-sdk-python-v2/).

You will need to purchase the following items:

a. Raspberry Pi Zero WH (with built-in WiFi) (cost: $14)
   Note: you will also need a monitor and a keyboard to work with the device.
b. Raspberry Pi Zero case (optional) (cost: $5)
c. Raspberry Pi power supply (cost: $10)
d. Micro SD card (to store the OS) (cost: $10)
e. DHT 22 temperature and humidity sensor (cost: $10)
f. Connector wires to connect the sensor and the Pi (cost: $1-2)

You will need to use the AWS IoT Core service to enable and configure the MQTT messaging service between the IoT device (Raspberry Pi) and AWS Cloud. This task will include:

- registering a new IoT device (thing) and creating and downloading a certificate that will identify the Raspberry Pi,
- copying the certificate, the public key, the private key, and the root certificate key to the Raspberry Pi,
- creating an IAM policy that allows the IoT device to publish messages to your AWS account and attach it to the certificate you previously created for the IoT device,
- creating a new identity in AWS Cognito and a policy for the IoT device and allow access to the IoT cloud service.

After purchasing the hardware items, you will need to set up and configure the Raspberry Pi. The tasks will comprise of:

- installing the Raspberry Pi OS (Debian-based Linux variant) and connecting the device to WiFi (https://www.raspberrypi.com/documentation/computers/getting-sta[illegible]nd an IDE should be included in the standard OS image),
- ins[illegible]n-DHT Python module to be able to receive measurements fro[illegible]/learn.adafruit.com/dht/overview),
- ins[illegible] IoT SDK to be able to communicate with the AWS IoT clo[illegible]i.org/project/awsiotsdk/).

You can use so[illegible]T client code from these examples (but will need some modification): [illegible]m/blogs/iot/monitor-iot-device-geolocation-with-aws-iot-events/ or https://docs.aws.amazon.com/iot/latest/developerguide/sdk-tutorials.html. You will need to include the URL for the AWS MQTT endpoint, and the access credentials (certificate and access key) in your client code. Your Python code running on the IoT device should poll the sensors regularly (e.g. once every minute) for temperature and humidity measurements and publish the values (together with a timestamp) to the AWS MQTT end point to a particular topic.

On the cloud side, you should create an AWS IoT rule that is subscribed to the IoT device's MQTT topic and triggers a Lambda function when a message is published by the Raspberry Pi. The Lambda function should receive the message from the IoT device and save it in a DynamoDB table. The saved information should include the timestamp, and the corresponding temperature and humidity values.