LOG8415:
Serverless Tweet Analysis
Implementing Cloud Patterns
Assignment 3

April 6, 2020

Abstract

In this lab assignment, you will be working on setting up a serverless data pipeline empowered by Flask framework that detects sentiments of a given list of tweets and stores the results by implementing two important Cloud patterns. You are asked to implement and deploy an architecture by adding Proxy and Sharding patterns that are used to distribute workloads in Cloud. In the first part of the assignment you will install, configure and test a Flask RESTful web service that accepts a list of tweets' IDs. Next, you will apply your newly acquired skills to implement and compare Cloud patterns in a distributed fashion and store results in a relational database service in AWS. Finally, you will report your results by producing a LaTeX article and you will submit your code.

Cloud Pattern; Serverless; AWS Lambda; Sharding; Proxy; RDS; Sentiment Analysis.

1 Objectives

The overall goals of this lab assignment are:

1. Get hands-on experience running a data pipeline to detect sentiments of tweets.
2. Learn how to implement patterns strategies to load data into AWS RDS.

In the lecture we've seen the Cloud patterns and the strategies commonly used when distributing data across VM instances, now it's time to implement and test these strategies. You will work **individually** to implement two common Cloud patterns in an application. Finally, you will do a write-up about your results using LaTeX and submit your final work.

2 Design Architecture

In below figure you can find the overall design of the architecture you are expected to deploy in AWS:
Here are the key points to consider:

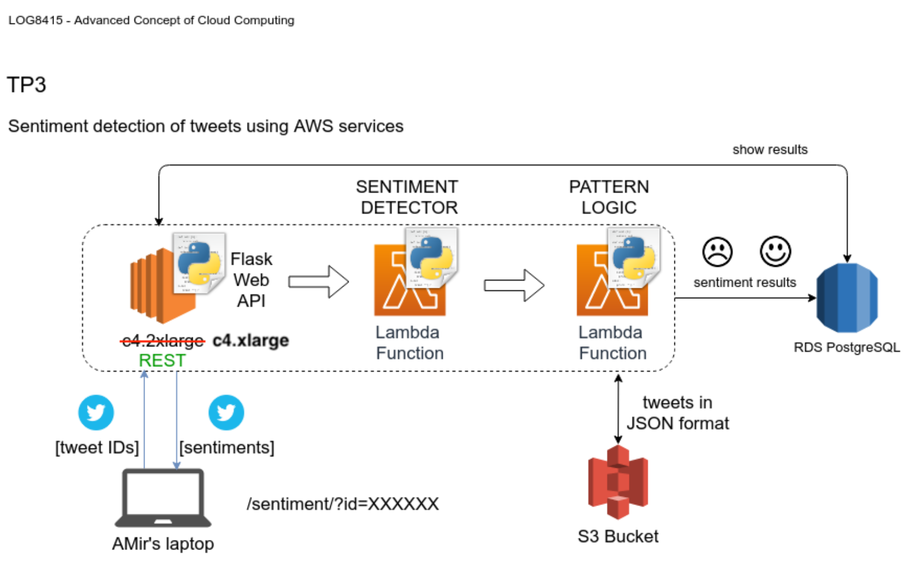

Figure 1: Design Architecture of the Sentiment Analysis Data Pipeline

- There will be a Flask application running on an EC2 instance type c4.xlarge which listens on port **5000**. This app will accept a JSON file including a list of tweets with three columns: **id**, **text** and **date**. The column *id* is the tweet ID which will be unique and column *text* is the tweet text body. Column *date* also shows the date and time this tweet has been posted on Twitter. Lab instructor will show you an example of the file being represented in this part of the assignment.

- You should open port 5000 in your EC2 security group.

- Please take a look at this page to understand how you can post JSON file to Flask:

  POST JSON TO FLASK

- The Flask app will trigger a Lambda function that will perform sentiment analysis over the list of tweets in the input JSON file. The format JSON is necessary to make sure that we can easily transfer data between our services. This Lambda function triggers another Lambda function which will determine the logic to store data in RDS instance. The Cloud patterns are implemented in the last Lambda function in our data pipeline.

- In RDS instance, we simply keep tables with corresponding schema as the input JSON file plus the extra records **sentiment** and **score** to store the overall sentiment of the tweet and its score value.

- Flask app in EC2 instance will also provide a backend to fetch and provide results of our sentiment analysis.

- Here is how the Flask app works:
  - If we provide **/sentiment/?data=...** it will perform the analysis and will return another JSON file with all the columns and result values.

- You can also implement a single sentiment analysis endpoint **/sentiment/?id=XXXXXX** to test your code, but it is OPTIONAL and not mandatory. We expect that you process a JSON file including multiple tweets at once.

- Results of your sentiment analysis (JSON files) should also be stored in S3 bucket.

- Performing the analysis and getting the results must be done within a single call.

- Feel free to choose any kind of library or code to perform sentiment analysis.

- You should test the data pipeline using the CSV file provided on Moodle. Make sure that you convert the CSV file into **JSON format** first.

- You must measure the performance of your sentiment analysis system by computing the response time in seconds. In your report, you should compare all the different implementations of Cloud patterns in terms of this indicator.

*** Please pay attention to one example of a call and input file and the expected results:

If we call your Flask application with such input (remember to convert sample CSV into JSON):

```
{
   "1":{
      "id":1246953034355298304,
      "text":"some␣sample␣tweet",
      "date": "2020-04-05␣00:12:10"
   },
   "2":{
      "id":1246490999347707905,
      "text":"this␣is␣another␣sample␣tweet",
      "date": "2020-04-06␣00:11:59"
   }
}
```

We must get the results (and also stored in S3) as such:

```
{
   "1":{
      "id":1246953034355298304,
      "sentiment":"positive",
      "score": 0.99
   },
   "2":{
      "id":1246490999347707905,
      "sentiment":"negative",
      "score": 0.87
   }
}
```

And based on the chosen pattern, we should be able to query RDS instances and see the same similar in schema-based format. Keeping the indices is optional.

3 AWS Lambda

AWS Lambda is an event-driven, server-less computing platform provided by Amazon. It is a compute service that runs code in response to events and automatically manages the compute resources required by that code.

Events that can trigger a Lambda function:

- Table updates in Amazon DynamoDB. - Modifications to objects in S3 buckets. - Notifications sent from Amazon SNS. - Messages arriving in an Amazon Kinesis stream. - AWS API call logs created by AWS CloudTrail. - Client data synchronization events in Amazon Cognito. - Custom events from web applications, or other web services.

You are free to choose either boto3 library:

https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

AWS CLI:

https://docs.aws.amazon.com/cli/latest/reference/lambda/index.html

or your preferred Python library to trigger your Lambda function. We recommend to take a look at the "invoke()" method:

```
from __future__ import print_function
import boto3
import json

lambda_client = boto3.client('lambda')

def lambda_handler(event, context):
    """This␣method␣is␣calling␣another␣Lambda␣function"""
    invoke_response = lambda_client.invoke(FunctionName="test-lambda",
                                           InvocationType='RequestResponse'
                                           )
    print(invoke_response)
    return str(invoke_response)
```

Here is a sample template code to connect to RDS instance and query it:

```
% pip3 install psycopg2

import psycopg2
from psycopg2.extras import RealDictCursor

conn = psycopg2.connect(host=_hostname,
                        user=_username,
                        password=_password,
                        dbname=_dbname)
cur = conn.cursor(cursor_factory=RealDictCursor)
print(cur)
q = "SELECT␣*␣FROM␣tbl_tweets"
cur.execute(q)
res = cur.fetchall()
return json.dumps({"result": res})
```

** Please note that all the above codes are just samples to help you understand the procedure and they might not be functional!

4 AWS Relational Database Service (RDS)

Amazon Relational Database Service (Amazon RDS) makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while automating time-consuming administration tasks such as hardware provisioning, database setup, patching and backups. It frees you to focus on your applications so you can give them the fast performance, high availability, security and compatibility they need.

For this assignment, make sure that you are using free-tier RDS:

https://aws.amazon.com/rds/free/

Please choose **db.t2.micro** instance type of **PostgreSQL** when launching an RDS instance. You do not need to stop your database since it is free and will not charge you.

This could be a good start:

Set up RDS PostgreSQL

5 Cloud Patterns

You should implement and benchmark two Cloud patterns based on what you are assigned to: [1]

| Proxy:

This pattern uses data replication between master/slave databases and a proxy to route requests and provides a read scalability on a relational database. Write requests are handled by the master and replicated on its slaves, while Read requests are processed by slaves. When applying this pattern, components must use a local proxy whenever they need to retrieve or write data. To simplify overall implementation, you will need to spin up two Relational Database Service (RDS) instances in AWS and use them as the interface to hit either of them randomly. In other words, the logic of this pattern will be:

LIST A (only if you are assigned to this list)

When inserting the results of sentiment analysis for a list of tweets, you should **randomly** select each of the two RDS instances (or schemas) per **each record** of the results and store it in the selected instance (or schema). In a very ideal case, you should end up having 50% of the results in either of the RDS databases. To perform this on different RDS instances, you need to make sure that the schema for the results is deployed in both instances similarly.

LIST B (only if you are assigned to this list)

When inserting the results of sentiment analysis for a list of tweets, you should select **even and odd** numbers of **id** column representing the tweet ID and store them in separate instance (or schema). In a very ideal case, you should end up having 50% of the results in either of the RDS databases (or schemas). To perform this on different RDS instances, you need to make sure that the schema for the results is deployed in both instances similarly.

Sharding:

Sharding is a technique that consists in splitting data between multiple databases into functional groups called shards. Requests are processed by a local router to determine the suitable databases. Data are split horizontally i.e. , on rows, and each split must be independent as much as possible to avoid joins and to benefit from the sharding. The sharding logic is applicable through multiple strategies:

[1] Please check Moodle and ask Lab instructor to make sure on which list you belong to and implement only those two patterns you are asked to.

LIST A (only if you are assigned to this list)

- A range of values
Select N first records to be inserted to the first RDS (schema) and the rest of the records to the other one.

LIST B (only if you are assigned to this list)

- A specific date
You will put tweets from **year 2020** in one instance, and the rest of the years into another one.

*** You can simplify the implementation of the patterns by using separate schemas instead of different RDS instances. Each schema can hold the results of the different pattern's logic.

6 Implementation

- The EC2 instance which will host Flask app must be running Ubuntu Linux :: latest version.

- To measure performance, you should calculate the response time in seconds for the whole process of loading data into RDS.

- Lab instructor will use a predefined class in a Python module to test your design architecture after the submission deadline. The test will be as simple as passing a list of tweets to your Flask application and getting the results back.

* Please make sure that you stop EC2 instance when you're not using it to avoid paying extra money. Lambda functions are only costly when called, there is no need to tear them down. RDS instances of free type can be running and you will not be charged for them.

7 Paper

To present your findings, you are asked to write a PDF report describing your implementation and results. In your report, please include performance evaluation, the architecture design of Cloud patterns, performance evaluation of the two Cloud patterns while loading the dataset into the RDS instances and your logic for sentiment analysis. Please remember to provide the list in which you are assigned to implement a customized version of each pattern.

8 Submission

Please follow this format to submit your code and results:

- One single PDF file named **TP3.pdf** which includes your full name and the analysis and report of your individual work.
- One file named **flask_ec2.py** which is used to deploy on EC2 instance and acts as the Flask RESTful web service.
- One file named **lambda_handler1.py** that is used to perform the sentiment analysis and it has been deployed on the first Lambda function.
- One file named **lambda_handler2.py** that is used to implement the patterns logic and it has been deployed on the second Lambda function.
- One JSON file names **results.json** which is the results of the input file provided on Moodle.
- One TXT file named **rds.txt** which includes the connection string to your RDS instance(s). It is up to you to choose either separate RDS instances per pattern or just simply use separate schemas in one RDS instance. In both cases, Lab instructor must be able to query your tables and select the results. Please provide all the necessary information to access the database here in this file.

** If you do not follow the submission format, you will be penalized.

9 Evaluation

A single final submission for this assignment is required for each individual person.

Your assignment will be graded on presentation and content as follows:
3 pts: RESTful web service up and running on EC2.
6 pts: Implementation of The Proxy and Sharding patterns;
4 pts: Comparing the performance of the patterns;
2 pts: Implementation of the sentiment analysis;
1 pts: General presentation and the quality of your report;
4 pts: Your source, scripts and results of the test by Lab instructor;

Acknowledgement

We would like to thank Amazon for supporting us through their AWS in Education Grants.