Serverless Tweet Analysis Implementing Cloud Patterns: TP3

Dmytro Humeniuk

April 2020

Abstract

In this assignment we develop a Flask application that performs sentiment analysis of provided tweets using two AWS lambda functions. Two cloud patterns were implemented: proxy and sharding pattern.

1 Introduction

In the last years, serverless application architecture is gaining popularity. It provides a number of advantages over traditional approaches such as ease of deployment, speed and performance. In this assignment we are using AWS lambda functions to perform tweet sentiment analysis and store the results in a PostgresSQL database using proxy and sharding patterns.

2 Design architecture

2.1 Flask application

Flask application provides a RESTful API for sentiment analysis. It performs the following actions:

- Accept an input query.
- Save the tweets from the query locally and upload the corresponding file to S3.
- Call first lamdba function.
- After lambda function finishes execution, query the database to fetch results of sentiment analysis.
- Transform the query results to a json file with assigned structure, save it locally and upload to S3.

Depending on the query to the Flask app, different cloud pattern will be used for storing the data. In our design we implement 'proxy' and 'sharding' patterns. To run the sentiment analysis the following route for the query should be used:

```
http://[AWS_EC2_PUBLIC_ADDRESS]:[PORT]/sentiment/<pattern>/
```

where 'PORT' is the number of port the application is running on (5000 in our case). By changing <pattern> to either 'proxy' or 'sharding' the corresponding pattern will be used to store data in the database.

In our application we also use S3 to store the data. For each query two folders are created: 'inputXXXXXXXXX'(for storing input request) and 'outputXXXXXXXXXX' (for storing sentiment analysis results), where 'XXXXXXXXXX' represents a unique folder ID, generated using current time in unix epochs.

Flask application runs on EC2 x4.large instance. Application was deployed on EC2 according to instructions in [1].

2.2 First lambda function

First lambda is invoked directly by Flask and it performs the following actions:

- Calculate sentiment value ('positive' or 'negative') and the corresponding score.
- Append sentiment analysis value to the json structure passed by Flask app.
- Invoke second lambda.

The input tweets as well as pattern for data storage are passed to lambda function through lambda 'event' in a json structure.

To perform sentiment analysis Textblob python library is used [2]. The Textblob sentiments analysis module contains two sentiment analysis implementations: PatternAnalyzer (based on the pattern library) and NaiveBayesAnalyzer (an NLTK classifier trained on a movie reviews corpus) [3]. NaiveBayesAnalyzer showed very slow performance comparing to PatternAnalyzer, so we chose the latter for our application. PatternAnalyzer gives a sentiment score (polarity), which shows how negative or positive a comment is. It takes values from [-1;1], but for simplicity we used the absolute value.

To get more accurate results the 'tweet' is cleaned before being classified. All special characters and links are removed. Experimentally, by comparing scores for cleaned and original text, we could observe slightly better evaluation results for the cleaned text.

2.3 Second lambda function

Second lambda function is invoked by the first lambda and it determines the logic for storing the sentiment analysis results in the database. As a database, we use a PostgreSQL relational database, which runs on db.t2.micro instance. There we created four schemas with one table in each schema. Before loading new data to the database, all its tables are flushed. If the 'proxy' pattern is specified, the functions checks if the number is even or odd and stores tweets with analysis results in the corresponding schema. For the 'sharding' pattern the schema is chosen depending on the year of tweet publication: tweets from 2020 are stored in one schema, the rest of the tweets - in the other (according to the list B). When loading data from the database in Flask application, firstly all the entries from the first schema are loaded and then merged with data loaded from second schema.

3 Results

We have evaluated the performance of our application on the dataset of 287 tweets. We calculated the time it takes to execute the query for 'sharding' and 'proxy' patterns based on 10 requests. The results are presented in the table 1. We can see that both patterns show similar performance, 4.58 s and 4.47 s to execute a query for proxy and sharding patterns respectively.

Table 1: Pattern average execution time

	Proxy pattern	Sharding pattern
Average execution time, s	4.58	4.47
Standard deviation, s	0.33	0.17

According to our observations, most of the time is taken by the first lambda function, where the sentiment analysis is performed. This time should be the same for both patterns. So, we have also measured the time it takes to load the data to the database, i.e. the time to execute second lambda function. Results are presented in the table 2. We see that results agree with the previous table: proxy pattern takes longer time to load the data. Unfortunately, the standard deviation value is comparable to the difference between loading times, so from the provided results it's difficult to evaluate which pattern has better performance. Experiment should be repeated with more samples.

Table 2: Average time of loading data to database

	Proxy pattern	Sharding pattern
Average loading time, s	1.4	1.3
Standard deviation, s	0.177	0.114

4 Running the code

Before running the scripts flask_ec.py and lambda_handler1.py you should provide your AWS and S3 credentials in the beginning of the script.

5 Conclusions

- Serverless design simplifies application development. Lambda functions are easy to use and also provide the possibility to test each function individually before deployment. Triggers can be added to launch the functions automatically.
- Texblob library PatternAnalyzer sentiment analysis implemantation shows better performance, than NaiveBayesAnalyzer.

- Sentiment analysis takes the most time of application execution time and can become the system bottleneck when larger texts are tested. The library for sentiment analysis should be chosen carefully. For better accuracy in a specific application, it's recommended to train a new model based on the data tested.
- Sharding pattern shows slightly better performance, than proxy pattern. For confirmation more tests need to be done.

References

[1] Deploy a flask app on aws ec2 — codementor. https://www.codementor.io/@jqn/deploy-a-flask-app-on-aws-ec2-13hp1ilqy2.

[2] Advanced usage: Overriding models and the blobber class — textblob 0.16.0 documentation. https://textblob.readthedocs.io/en/dev/advanced_usage.html#sentiment-analyzers.

[3] textblob documentation. https://buildmedia.readthedocs.org/media/pdf/textblob/latest/textblob.pdf.