Live-stream sentiment analysis of Tweets using serverless approach

LOG8415 Concepts avancés en infonuagique

Na Zhou

16th December 2021

Keywords: serverless architecture, twitter, sentiment analysis, personal project

1 Introduction

Serverless is the idea of running a server-based application without having to manage a server. The application still runs on servers, but the server management is taken care of by cloud vendors. With the serverless architecture, we don't need to plan for future capacity of the infrastructure; we save time on server management; and we only pay for what we use. However, not all the tasks are suitable for serverless computing, for example, long running tasks that can not be broken up into smaller workloads is not a good candidate.

The objective of this project is to design and implement serverless approaches to analyse live-stream of tweets. We explored two approaches on AWS[1]. The key differences between the two approaches are:

1. Deploy the Machine Learning (ML) model as a web service; Lambda sends data to the ML endpoint to get prediction.

2. Package the ML code and dependencies as a container image; upload the image to Amazon Container Registry (ECR); Lambda function is built from the image.

We will explain the strength and weakness of each implementation in section 4. We will point out the suitable usage of the two solutions in section 6.

1.1 Problem

Design and implement proof-of-concept solutions to perform live-stream sentiment analysis of Tweets. The solution shall use serverless approach. The implementation shall be on AWS.

1.2 Literature Review

When building serverless applications, we shall choose the architectural patterns that allow the design to be resilient and scale. One of the AWS whitepaper[5] provides an example of a three-tier architecture using API Gateway and Lambda. The multi-tier architecture pattern is a general framework to ensure decoupled and independently scalable application components can be separately developed, managed, and maintained.[5]

[1]AWS Education account restricts the usage of many services, e.g., CloudFormation, Redshift, QuickSight, SageMaker

J. Katzer[6] explained how serverless can help us ship amazing software while saving lots of time. He discussed the knowledge specific to serverless computing, including the concept of serverless, distributed systems, serverless architecture, patterns, tools, monitoring, security, and quality.

1.3 Proposed Solution

We could rephrase the original problem as: while streaming data from Twitter into Amazon S3, we shall enrich the data stream with sentiment analysis result before storing it to S3. The problem shall be resolved using severless approach. The proposed solution consist the following components:

A ML classifier that predicts the sentiment of each tweet. We plan to train a ML model offline, then deploy it and have an endpoint for prediction, or add it directly to Lambda.

A component that streams tweets from Twitter. We will write a python script to collect data from twitter and send them to Kinesis Firehose.

A mechanism to process the data stream, obtain sentiment result, and deliver it to S3. Kinesis Firehose will trigger Lambda to process the tweets; Lambda will get the predicted sentiment from the ML model. Firehose will delivered the processed result to S3.

A method to verify the final result. We will make a simple plot out of the result.

We implemented the solution in two different ways. We will explain them in section 2.

2 Architecture

The design is a typical implementation of the multi-tier architecture pattern[5]. We describe the two implementations in this section.

2.1 Call a model endpoint using Lambda

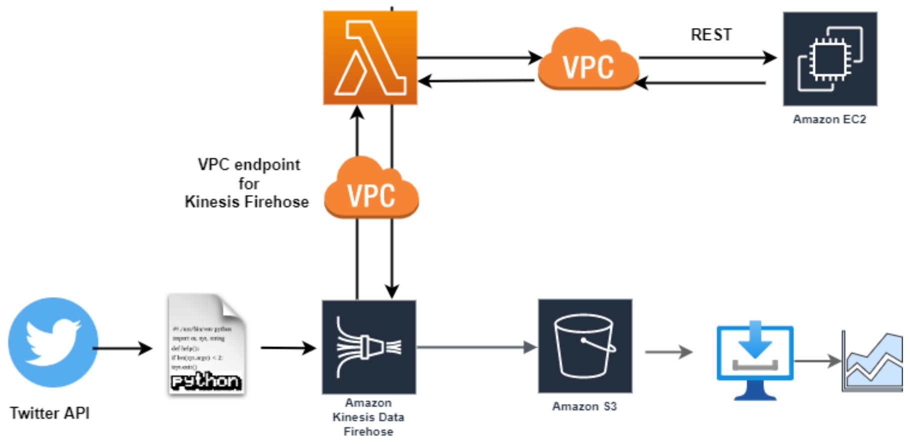

Figure 1: Architecture overview: call a model endpoint using Lambda

”Figure. 1” illustrates the overview of the solution. Besides the three-tier architecture, the implementation demonstrates how to call a ML model endpoint using Lambda.

1. Extract: Python code that streams tweets using Twitter API and Writes to Kinesis Data Firehose Using the AWS SDK. Twitter access token and AWS access token are required.

2. Transform: The core of the solution that uses AWS Kinesis Data Firehose and Lambda to process tweets and obtain the sentiment of each tweet. The EC2 instance hosts a ML model and provides an endpoint for prediction.

3. Store the processed data: The resulting data is stored in S3 bucket for further analysis.

The system is distributed as a consequence of this design pattern. In general, we need to take care of the untruthfulness of the network boundary between tiers. But, for this implementation, Firehose offers to reliably load real-time streams into its destination. The EC2 is a mean to obtain an endpoint (e.g., a simple workaround for SageMaker), but we kept the traffic among Firehose, Lambda and EC2 inside AWS network for security.

To further analyse the resulting data, we download them from S3 bucket, and then use python or other tools to get insight.

2.2 Using container image to run ML model in Lambda

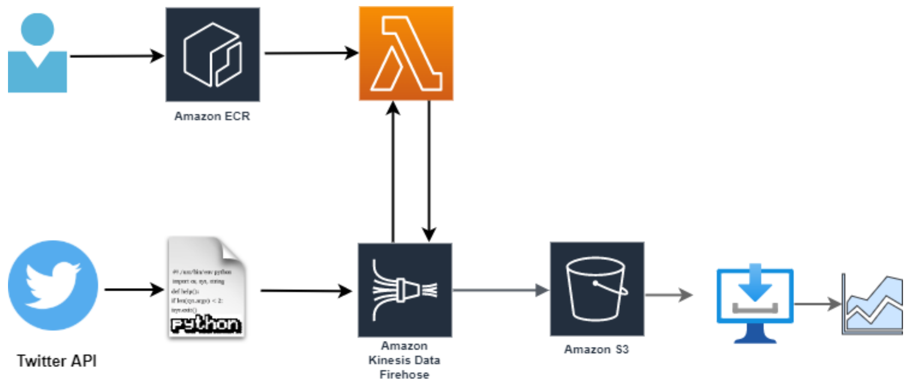

Figure 2: Architecture overview: using container image to run model in Lambda

”Figure. 2” illustrates the overview of the solution. The major difference between this implementation and the previous one is the ML model, a TensorFlow Keras model for sentiment analysis (2.9GB), lives in Lambda function. We trained and saved the model, packaged it and all code dependencies as a container image. Then, uploaded the image to ECR and created a Lambda function from the container image. The maximum container size supported by AWS is 10 GB, which overcomes the size limitation of deploying Lambda with zip.

With this implementation, we also removed the additional VPC configuration.

3 AWS Services

We briefly describe the AWS services leveraged by the solution.

- Kinesis Data Firehose: an extract, transform, and load (ETL) service that reliably captures, transforms, and delivers streaming data to data lakes, data stores, and analytical services.

- Lambda: a serverless, event-driven compute service that allows to run code for virtually any type of application or back-end service without provisioning or managing servers.

- S3: an object storage service that offers industry-leading scalability, data availability, security, and performance. To store data in S3, we create a bucket and then upload objects to the bucket. A bucket is a container for objects. An object is a file and any metadata.

- EC2: provides scalable computing capacity.
- VPC endpoints: enable connections between a VPC and supported services.
- Elastic Container Registry (ECR): a fully managed container registry to share and deploy container images and artifacts. It is integrated with other services, e.g., Lambda.

4 Data flow and methodology

We describe each component of the solution in detail in this section.

4.1 Data extract and injection

We developed a python script that uses Twitter API to retrieve Twitter data and send them to Kinesis Firehose. The script filters out tweets in English, extracts the ”text” field of the selected tweets, and sends the text to Firehose.

The final solution leverages ”tweepy Stream” to retrieve live-stream of tweets. Tweepy is an open-source python package that provides an easier way to communicate with the Twitter API since it takes care of some low-level details such as, rate limiting, authentication, etc.

In addition, in order to perform end to end test of the data processing pipeline, we wrote a script that accesses directly the Twitter API v2 ”recent search endpoint”. This script gets a defined number of tweets before sending them one by one to Firehose. This allows to verify whether all tweets are correctly processed by the pipeline. Whereas, it is difficult to do such end to end validation with streams.

4.2 Data processing

The data processing and the delivery to S3 are at the heart of the solution. Kinesis Data Firehose orchestrates the procedure. It receives tweets. Then, it triggers Lambda to process the tweets and obtains the sentiments. Finally, it delivers the result to S3.

4.2.1 Kinesis Data Firehose

Kinesis Data Firehose captures and loads data in near real time[3]. The frequency of data delivery to S3 is determined by the S3 buffer size or buffer interval value. We set the buffer size to 1 MB and buffer interval to 60 seconds.

The solution leverages its two important features. 1) Data received can be fed to a Lambda function for further processing. 2) The resulting data can be stored on S3.

Besides taking its default value, the following parameters are set in Firehose configuration:

```
- DeliveryStreamName: Tweets-PUT-S3-sF1ta
- Source: DirectPut
- Destinations: s3://projectnz  # S3 bucket name
- Transform source records with AWS Lambda: processTweets  #lambda function name
    * Buffer size: 1MB
    * Buffer interval: 60s
```

4.2.2 Lambda: calls a model endpoint

In "Figure. 1", the Lambda function sends (via REST POST) each tweet text to a ML endpoint; retrieves the predicted sentiment (Negative, Neutral, Positive) and its associated score (possibility) from the ML model. Then, the Lambda function assembles the original tweet text, the predicted sentiment, and its score to construct a data record for Firehose to deliver to S3.

Machine Learning endpoint The ML endpoint is provided by the EC2 instance, shown by "Figure. 1". The reasons of choosing EC2 are as follow:

- We can not utilize SageMaker or Comprehend due to account restrictions. AWS advises not to use AmazonML since it is obsolete.
- Having to load a model inside lambda may impact the data processing performance. Each invocation of the Lambda needs to take extra time to load the model.
- Having a ML endpoint allows to load the model into memory when starting the httpd server. Hence, when tweet text is posted to the ML endpoints, the model will make prediction immediately, which improves the performance of data processing, reduces the possibility of Lambda time out.

We trained and saved a basic model offline, using tensorflow.keras and LSTM[8]. The training dataset is available online[2]. The accuracy score of the resulting model is approximately 0.75. The text pre-processing is very basic: keep only letters and numbers in the text. However, the objective here is to obtain a model to implement the end-to-end workflow, rather than producing a sophisticated NLP model.

We then created an EC2 instance and transferred the saved model to the EC2. On the EC2 instance, we create a REST API POST endpoint using FastAPI[1] and Uvicorn server[2]. When the httpd server is started, the saved ML model is loaded to memory. Whenever data is posted to the ML endpoint, the model will do the prediction immediately and return the results.

The downside of the current implementation is that EC2 (and the endpoint) is the single point of failure and potential performance bottle neck. However, thanks to the distributed design, we can add load balancers or replace the component with other method, e.g., SageMaker endpoint, in the future.

4.2.3 Lambda: uses container image to run the TensorFlow model

In "Figure. 2", the TensorFlow model for sentiment prediction lives in Lambda function. We used the following steps to implement the solution.

- On a local ubuntu machine, create a folder; add requirements.txt, predict.py, Sentments.csv, sentiment.h5 to the folder.
  - requirements.txt contains the libraries to run the model
  - predict.py is the code of the Lambda function
  - sentiment.h5 is the saved model
  - Sentiment.csv contains some text to fit the Tokenizer
- In the same folder, create a Dockerfile for Python 3.8 using the AWS provided open-source base images.[3][7]
- Build and push the container image to Amazon ECR.

[2]https://www.kaggle.com/crowdflower/first-gop-debate-twitter-sentiment

- Create a Lambda function from the container image

The implementation takes advantage of the benefits of Lambda, such as auto scaling, reduced operational overhead. Lambda function can be configured with up to 10 GB of memory, which is far more than the need of the current model. The maximum size of the container image is 10GB and our container size is 2.9GB. To build a truly serverless system, we should avoid certain design choices that tie us to a specific private network[6]. With this implementation, VPC is not needed at all.

The problem facing the current implementation is the so-called cold start issue. Using the built-in Lambda Data-Firehose-as-source test event, the duration of the first invocation is about 570ms, whereas the duration of the subsequent invocations is about 45ms. We need to find a mitigation solution in the future.

4.3 Data delivery

The destination of Kinesis Data Firehose is S3. Data stored in S3 bucket is in JSON format, no compression, no encryption. This will definitely be an area for improvement in the future. In case of failure, Firehose stores the data in processing-failed directory in the S3 bucket.

4.4 Networking

In "Figure. 1", the traffic among Kinesis Data Firehose, Lambda and EC2 are kept inside Amazon network. The httpd server on EC2 listens at EC2 private IP:port 8000. Thus, we configure Lambda function to use the same VPC, subnet and security group as EC2. For the communication between Firehose and Lambda, we setup an interface VPC endpoint for the service kinesis-firehose.

4.5 Result validation

We first download the resulting data from S3 to local. Next, read the data and create a pandas dataframe. Then count the number of tweets by type of sentiment (Negative, Neutral, Positive). Finally, we build a simple bar chart, (e.g. Figure. 3) to show the distribution. We also leverage wordCloud to see the more frequent words.

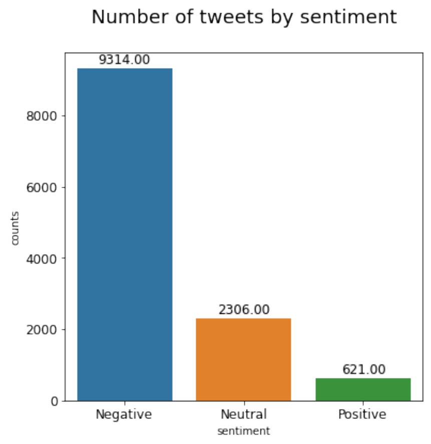

Figure 3: Result validation: an example

5 Instructions for the demo

Demo videos are located at:

`https://drive.google.com/drive/folders/1QOCC-xuOMMTLARfjxfSUciiSFXDguyay?usp=sharing`

Instructions to run the program:

- AWS access: Copy the credentials from AWS Education account to your home directory .aws/credentials
- Twitter access: Create a developer account, generate access key and token per twitter instruction. Add the keys to config.json.

```
{
"twitter_consumer_key": [value]
"twitter_consumer_secret": [value]
"twitter_access_token": [value]
"twitter_access_token_secret": [value]
}
```

- For the solution where Lambda calls a ML model endpoint, start ML model endpoint: start EC2 instance; ssh to EC2 as user ubuntu; in its home directory, run the script: start_httpd.sh, verify:

```
INFO:     Started server process [1113]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://172.31.83.21:8000 (Press CTRL+C to quit)
```

- Stream tweets: on your local host, run producer_stream.py
- Go to S3 bucket, check that new files are generated. Their content is the text of tweets and associated sentiment.
- Result validation: on your local host, run download_bucket.py to download the result and save them to a local directory. Run "plot_sentiment_result.ipynb" to see a simple bar chart.

6 Summary

The "Lambda calls a ML endpoint" design (Figure. 1) is suitable for organisations that have already had their ML models hosted somewhere. The "Lambda uses container image" design (Figure. 2) demonstrates a cost-efficient and reliable solution to run models in Lambda.

Serverless technology allows us to focus on solving the business problems instead of worrying about the infrastructure. But, it is still our responsibility to make sure the code is correct and secure.

Security is incorporated into the our implementations.
1) Implement least privilege access, grant only the required permissions to Firehose, and Lambda.
2) Use IAM roles.
3) In "Figure. 1", keep traffic inside AWS network unless public access is required.
4) S3 bucket and objects are not public.

Monitoring and logging is provided by CloudWatch. We monitor Firehose, Lambda and S3 with CloudWatch.

Performance and Scalability The single point of failure in the "Lambda calls a model endpoint" design is the ML endpoint provide by EC2. In production, this component should be a cluster behind a load balancer, or endpoints provided by other platforms. It will not be a single EC2.

The challenge of developing serverless application lies in debugging and testing of Lambda functions. Tools e.g., serverless framework, can make the develop, deploy and troubleshoot easier. We did not leverage the tool because the education account does not have permissions to use CloudFormation. We test the Lambda function locally and at Lambda console. To test the end to end data processing, we wrote a python script to send a defined number of tweets.

Another point is that knowing how to manage servers, i.e., Linux and networking, is fundamental in serverless solution development. We need to know how to manage servers before we can use a system that manages them for us.

6.1 Future work

Below, we outline several avenues for future work.

1. Mitigate the cold start issue in "Figure. 2"
2. Encrypt data at rest on S3.
3. Move the result data from S3 to Redshift to get insights faster.

References

[1] FastAPI, https://fastapi.tiangolo.com/

[2] Uvicorn ASGI server, https://www.uvicorn.org/

[3] AWS Documentation, https://docs.aws.amazon.com/

[4] Tweeter developer portal, https://developer.twitter.com

[5] AWS Serverless Multi-Tier Architectures with Amazon API Gateway and AWS Lambda, October 20, 2021, https://docs.aws.amazon.com/

[6] J. Katzer, Learning Serverless, Nov 2020, O'Reilly Media, Inc.

[7] D. Poccia, New for AWS Lambda – Container Image Support, https://aws.amazon.com/blogs/aws/new-for-aws-lambda-container-image-support/

[8] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, Sept 2019, O'Reilly Media, Inc.