Technische Hochschule Brandenburg
University of Applied Sciences

technical documentation

For the project

Twitter message analysis

written by :

Naoufel Frioui frioui@th-brandenburg.de

Ali Ridha Haouari *haouari@th-brandenburg.de*

Oussama Mzoughi mzoughi@th-brandenburg.de

Lecturer:

Prof. Dr. Thomas Preuss

Prof. Dr. Lars Gentsch

Technische Hochschule Brandenburg University of

Table contents

1. Introduction

In this documentation, we will discuss serverless architecture and how we used serverless tools, namely Kinesis, ApiGateway, Lambda and DyanmoDB to process Twitter data.

Many companies are now shifting towards a serverless model. There are currently three main contenders in this space, namely Amazon Web Services (AWS), Google Computing Platform (GCP) and Microsoft Azure. In this introductory post, we focus on setting up serverless architecture on AWS.

There have been many paradigms for organizations to collect, store and process large amounts of data effectively.

Before any cloud computing, organizations had to build out their own computing infrastructure and maintain severs in house. The rise of cloud infrastructure gave rise to a paradigm shift where companies could suddenly spin up hundreds of EC2 instances on AWS and instantly have a place where they could set up their infrastructure.

Open source tools like Hadoop, Cassandra, HBase, Hive, Storm etc took off. These tools were well suited for the model of spinning up N number of severs and set up their back-end platform.

To minimize the cost of having a team to manage a collection of severs to store and process data, large tech companies like Amazon and Google have developed managed tools that take the place of tools that require one to setup and maintain individual severs. While these tools will generally run on multiple severs, this fact is abstracted from the user of the tool.

2. Architecture of The Twitter Analyse System

Our application consists in the two first steps as a data translation process, as we manage to control this flux of data from TwitterStreamAPi, using our TwitterCredits as shown above, this one Direction Stream is managed through a Kinesis Amazon web service where we transform the stream to little shards, we tried to get records for every shard parsing it from a json side and before passing to the next iterator shard we try to pull hashtags in one way and locations : latitude, longitude in an other way : we used the object "entities" to reach the field Hashtags and get the "hashtag and text colomn and used the "places" object to reach the full_name field and bounding_box array to get the latitude and longitude informations .

As shown above we implemented a consumer part in a lambda service in amazon aws, we intended to put the data into a dynamo data base (aws amazon) as a NoSql data base, we filled the table hashtags with a Hashtag and HtCount (it is used to determine the number of occurrence for every Hashtag) and in the table location we filled the items with a country field, a ftCount(count the occurrence for the correspondingly to every country) and in two separate fields we filled them with a latitude and longitude.

With a GET Method the ApiGateway triggers a Lambdafunction that scans the dynamo db and get individually Hashtags and locations, parse them into json encoding , put them in an array list and send them to the APIGateway
Another developed locally application gets the data from the ApiGateway, Manage the frontend of a web application where we can see different locations on a map, and top 10 Hashtags in an other Menu , all deployed in an EC2 instance .

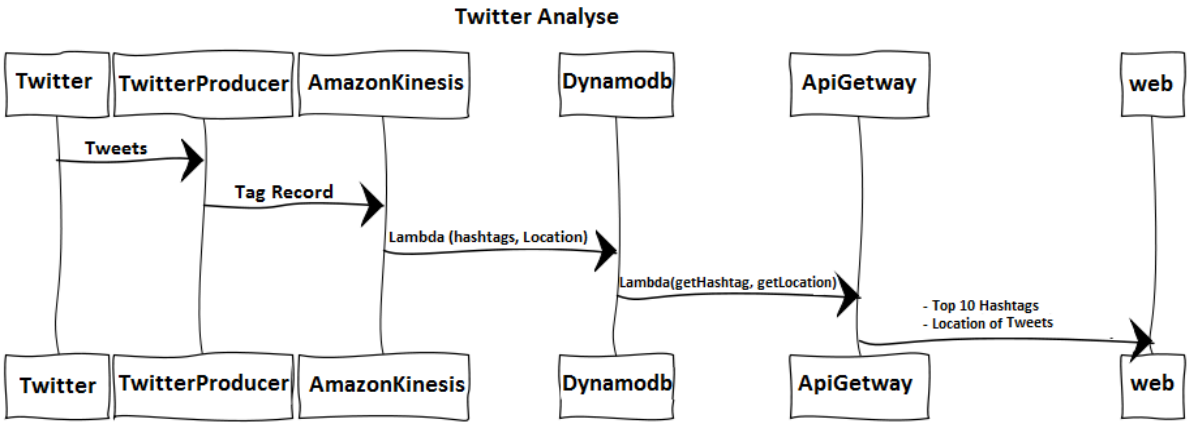

***Figure 1**: Twitter Analyse General Architecture*

3. Write Data

3.1 Twitter

3.1.1 Prerequisites: Twitter credentials

To get started, we will need credentials from Twitter to make calls to its public API. To this end, we went to apps.twitter.com and created a new app and filled out the form to get our unique credentials for making requests from Twitter for their data. Once we had our Twitter credentials we put them in a config-like file called twitterCreds.py

3.1.2 Twitter-Producer

The twitter-producer application is the part that gets the stream of tweets from Twitter. It uses the Twitter Streaming API, and in particular the Sample

Stream, where the application receives a small random sample stream, providing up to 1% of the Twitter firehose. The application provides a callback to the Streaming API, which will call the callback when there are tweets available. The callback receives parsed tweets
Here is how a map of a tweet looks like:

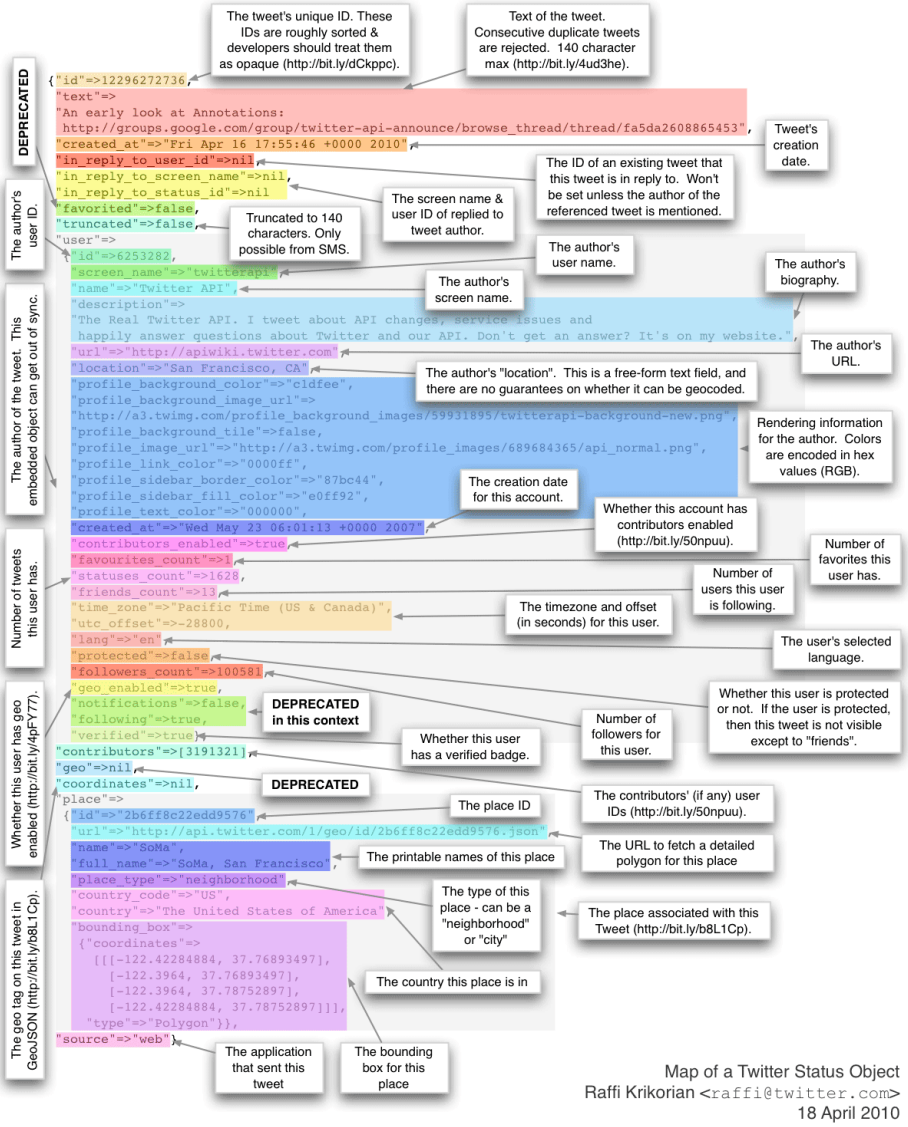

***Figure2**: map of a Tweet*

As we can see it is easy to just grab the hashtags and the created_at timestamp from each tweet, and post each tag to an Amazon Kinesis stream. The twitter-producer application will create the stream if it's not already there.

The library twitter-api abstracts the asynchronous callback handling needed in the Twitter Streaming API. However, the specifications for the Twitter Streaming API are somewhat complicated, with various rules for reconnection backoff, rate limiting, keep-alive signals, chunked tweets, and so on. Therefore, we use the twitter-streaming-client library to do the heavy lifting.

Note that twitter-producer shards on the hashtag, meaning that if several shards are being used, only one of the shards will receive a record with a given hashtag.

3.2 Intro to Kinesis Streams

Amazon Kinesis is a tool used for working with data in streams. It has a few features—Kinesis Firehose, Kinesis Analytics and Kinesis Streams and we will focus on creating and using a Kinesis Stream. Kinesis streams has standard concepts as other queueing and pub/sub systems.The figure and bullet points show the main concepts of Kinesis

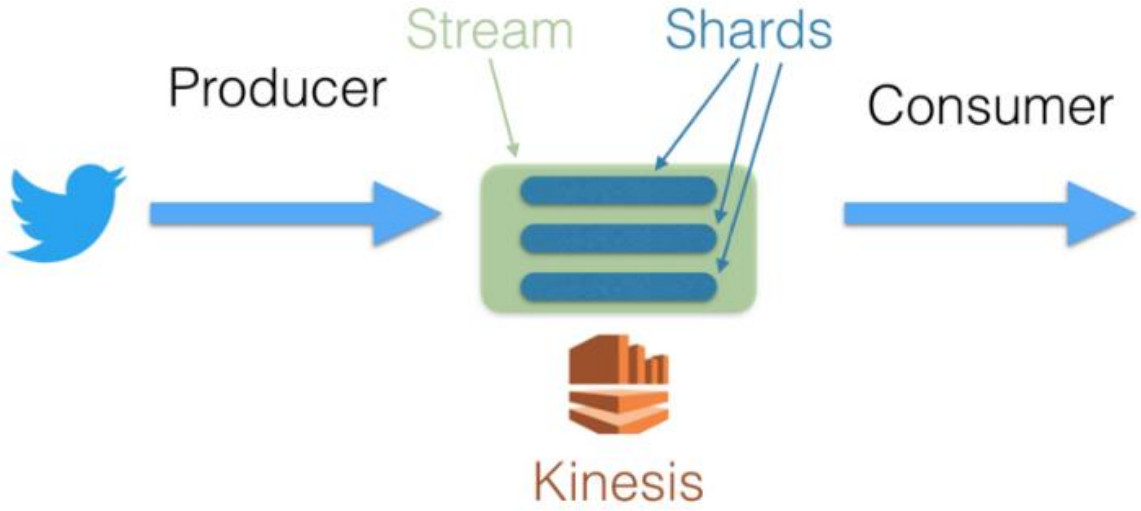

***Figure 3**: from stream to shards*

***A stream**: A queue for incoming data to reside in. Streams are labeled by a string. For example, Amazon might have an "Orders" stream, a "Customer-Review" stream, and so on.*

- ***A shard:** A stream can be composed of one or more shards. One shard can read data at a rate of up to 2 MB/sec and can write up to 1,000 records/sec up to a max of 1 MB/sec. A user should specify the number of shards that coincides with the amount of data expected to be present in their system. Pricing of Kinesis streams is done on a per/shard basis*

- ***Producer:** A producer is a source of data, typically generated external to your system in real-world applications (e.g. user click data)*

- ***Consumer:** Once the data is placed in a stream, it can be processed and stored somewhere (e.g. on HDFS or a database). Anything that reads in data from a stream is said to be a consumer.*

- ***Partition Key**: The Partition Key is a string that is hashed to determine which shard the record goes to. For instance, given record r = {name: 'Jane', city: 'New York'} one can, for example, specify the Partition Key as which will send all the records with the same city to the same shard.*

  As shown down , here is a screen shot of our kinesis stream , the monitoring per hour lets us see diverse statistics of our flux of data :

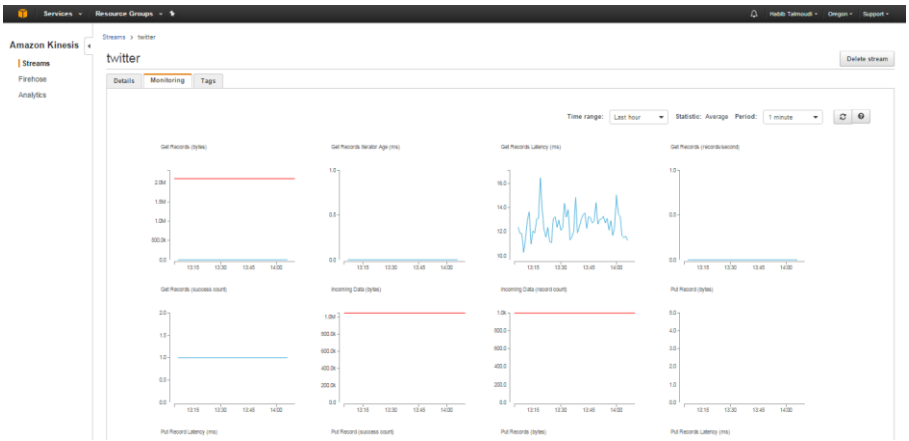

***Figure 4**: screenshot of kinesis Stream*

3.3 Amazon Lambda (from kinesis)

3.3.1 what's Lambda ?

- *AWS Lambda is a compute service where you can upload your code to AWS Lambda and the service can run the code on your behalf using AWS infrastructure. After you upload your code and create what we call a Lambda function, AWS Lambda takes care of provisioning and managing the servers that you use to run the code.*

- *So it allows you to run code in the cloud without having to worry about servers? Great! Oh and you also pay for what you use. No fixed fees. Suddenly see a spike in traffic? No problem, Amazon will spawn more instances of your code automatically to handle the load.*
- *Lambda functions can be written in Java, Node.js or Python. They are stateless pieces of code that can't run for longer than 300 seconds. This makes them perfect for building an API. A screen shot for different implemented and tested function in our lambda service presented here by , we can see the three main lambda functions :*

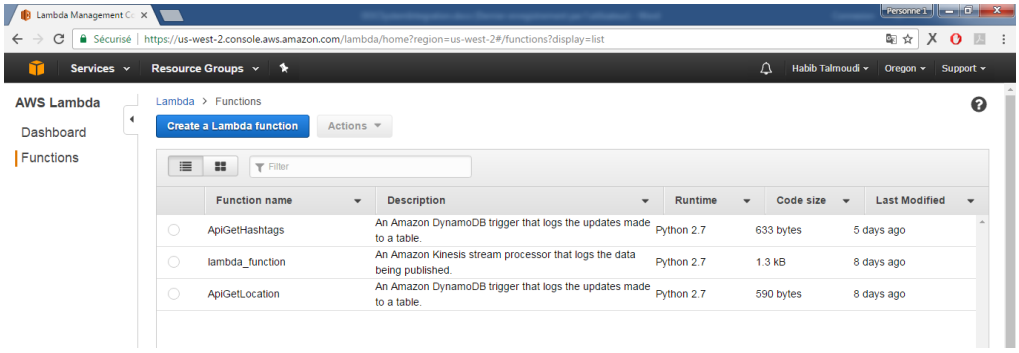

***Figure 5**: screenshot from amazon kinesis*

3.3.2 Kinesis to lambda

As said above the kinesis stream can:

"capture and store terabytes of data per hour from hundreds of thousands of sources such as website click streams, financial transactions, social media feeds, IT logs, and location-tracking events.."

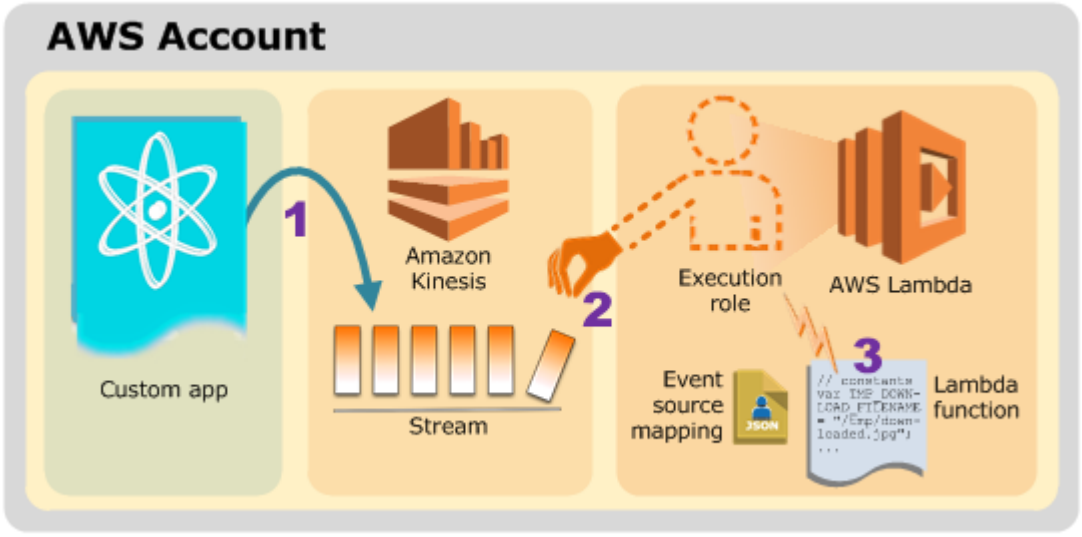

***Figure 6** : From kinesis to Lambda*

We can subscribe Lambda functions to automatically read batches of records off our Amazon Kinesis stream and process them if records are detected on the stream. AWS Lambda then polls the stream periodically (multiple times per second) for new records.

Regardless of what invokes a Lambda function, AWS Lambda always executes a Lambda function on our behalf. If our Lambda function needs to access any AWS resources, I need to grant the relevant permissions to access those resources. So we grant AWS Lambda permissions to poll our Amazon Kinesis stream. We grant all of these permissions to an IAM role (execution role) that AWS Lambda can assume to poll the stream and execute the Lambda function on our behalf.

3.3.3 what's aws IAM

IAM : identity access management

AWS Identity and Access Management (IAM) is a web service that helps you securely control access to AWS resources for your users. You use IAM to control who can use your AWS resources (authentication) and what resources they can use and in what ways (authorization).

3.4 Amazon Lambda DynamoDB

Next stop: a place to store a dictionary. Amazon has multiple cloud database solutions but it comes down to RDS and DynamoDB. Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale.

3.4.1 what's DynamoDB more precisely ?

DynamoDB *is Amazon's distributed key-value store, optimized for fast read and writes of data. Like many other distributed key-value stores, its query language does not support joins but is optimized for fast reading an writing of data allowing for a more flexible table structure than traditional relational models.*

Some key concepts include

- ***Partition key:*** *Like all key-value stores, a partition key is a unique identifier for an entry. This allows for O(1) access to a row by the partition key in DynamoDB.*
- ***Sort key:*** *Each row can be broken up and have some additional structure. Each row can be thought of as a hashmap ordered by the keys.*

- ***Provisioned Throughput*****:** *This is the amount of reads and writes you expect your table to incur****. Pricing of DynamoDB is done based on how many reads/writes you provision***

The figure below shows a mock-up of a twitter feed table. Each row contains all the tweets that should show up in a given users Twitter feed. The PrimaryKey is is the userid or Twitter Handle and the SortKey is the timestamp at which the tweet was originally created.

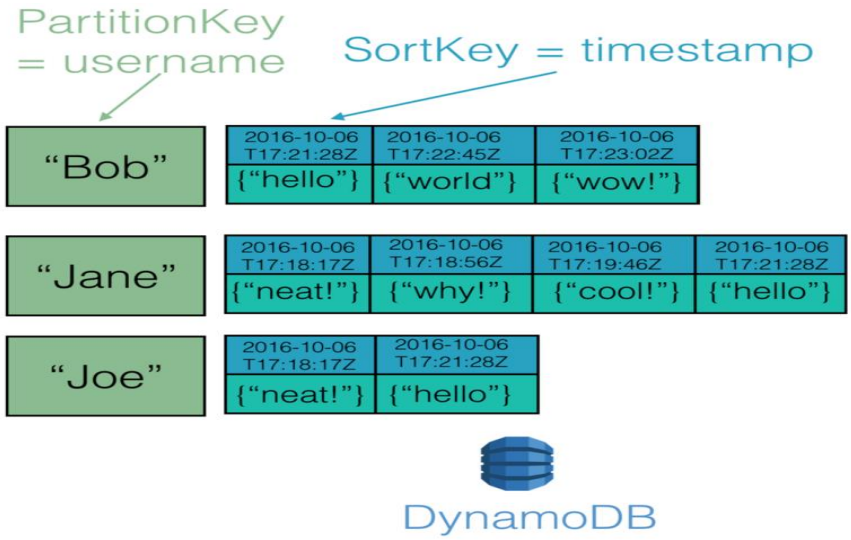

Figure 7 *: a snippet of how twitters record it in a dynamoDB*

3.4.2 What Lambda stores ?

So basicly, our lambda stores the hashtags and the occurency "htCount" in a table called "Hashtags" and in a parallel function it stores the "country" , it's occurency "ftCount" the latitude in longitude in another table called "full_name"

A screen shot from our Dynamodb data base shows the two tables that we have filled , as example here are the items in the table "hashtags"

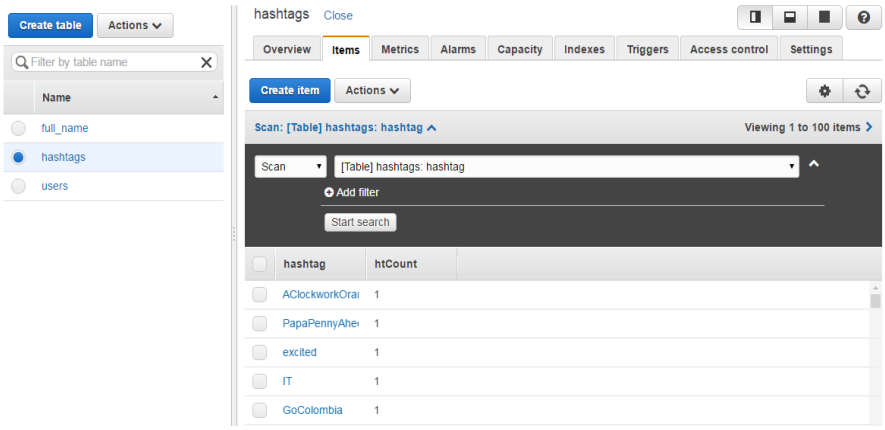

Figure 8: screenshot of Dynamodb

3.5 Write Data Summary

At This point we've taken data from a TwitterStream Api, passed through kinesis Stream to parse the data into shard and automaticly triggered by the kinesis data flow , a lambda function will store the data into a NoSql Database (Dynamo db)

4. READ DATA

The following process is how the data is fluing from dynamo db to our website through different services here the screenshot where we can see how it is all made in one way :

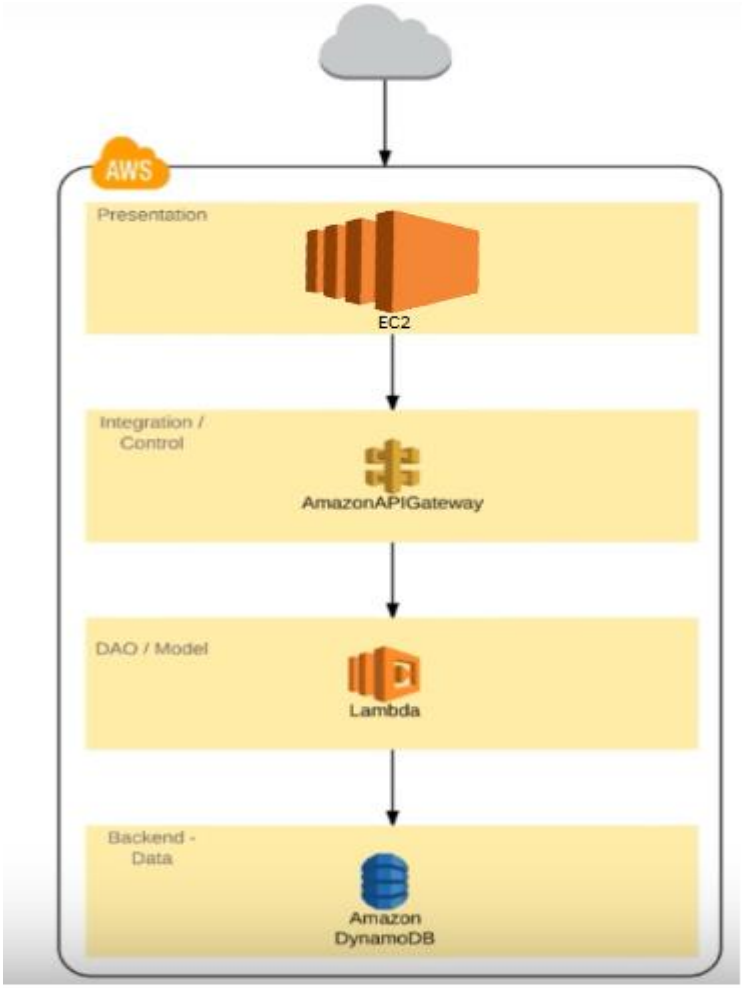

Figure 9 : *Different "read data"steps*

4.1 Amazon dynamoDB To Lambda

we used Lambda functions as triggers for our Amazon DynamoDB table. Triggers are custom actions you take in response to updates made to the DynamoDB table. We enabled Amazon DynamoDB Streams for our table. Then, we writed a Lambda function to process the updates published to the stream.

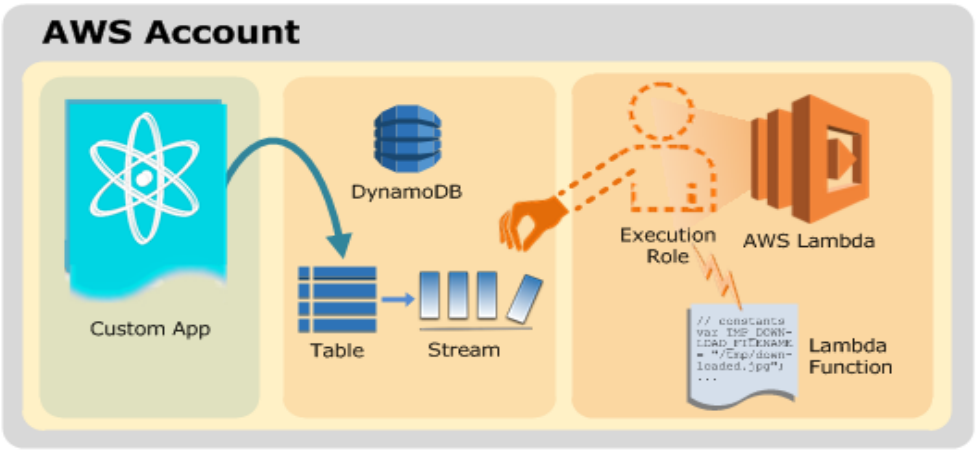

Figure 10 *: from dynamodb to lambda through IAM policies*

Regardless of what invokes a Lambda function, AWS Lambda always executes a Lambda function on our behalf. We granted AWS Lambda permissions to poll our DynamoDB stream with an IAM role (execution role) that AWS Lambda can assume to poll the stream and execute the Lambda function on our behalf.

Lambda functions written in Python are pretty fast (once they're actively being used and warmed up). The initial startup time for this function is fast.

we can see how the lambda is triggered and called by a lambda_handler function that gets an event (dynamo db) and a context :

4.2 How the Lambda Funktion is invoked

At this point we already connected the database and main handler function of our lambda as a trigger. The only thing that's left is to make the Lambda function available through an openly accessible API. The preferred way of doing this is through Amazon's API Gateway

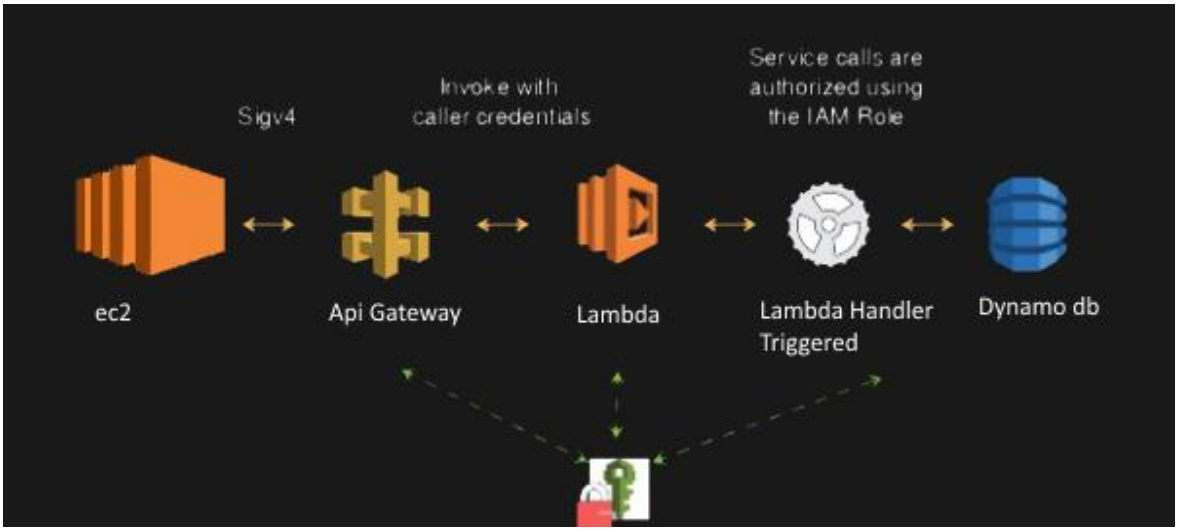

Figure 11: *integration of IAM and lambda Handler triggering*

4.3 From Lambda to API Gateway (creating API Gateway)

4.3.1 What's an API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. With a few clicks in the AWS Management Console, you can create an API that acts as a "front door" for applications to access data, business logic, or functionality from your back-end services, such as workloads running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, or any Web application. Amazon API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management

Connecting AWS Lambda with an API should be easy and straightforward. We initially linked Lambda manually to an API to do a Get Method.

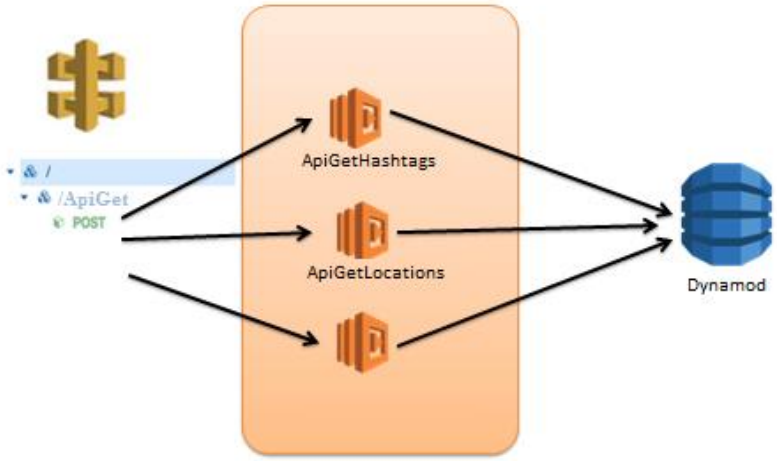

***Figure12**: different lambda functions invoked by api gateway*

With a Get method we intended to invoke two lambda function : ApuGetHashtag that brings the Data from an the dynamodb and send it as a json format in an array list and the other is an ApiGetLocation that brings the data from the full_name table and send it as a json format through every array field

As we can see in our ApiGateway , the two requests getLocation and getHashtags where we invoke a Get Method to bring the json data from the lambda , here is the screenshot :

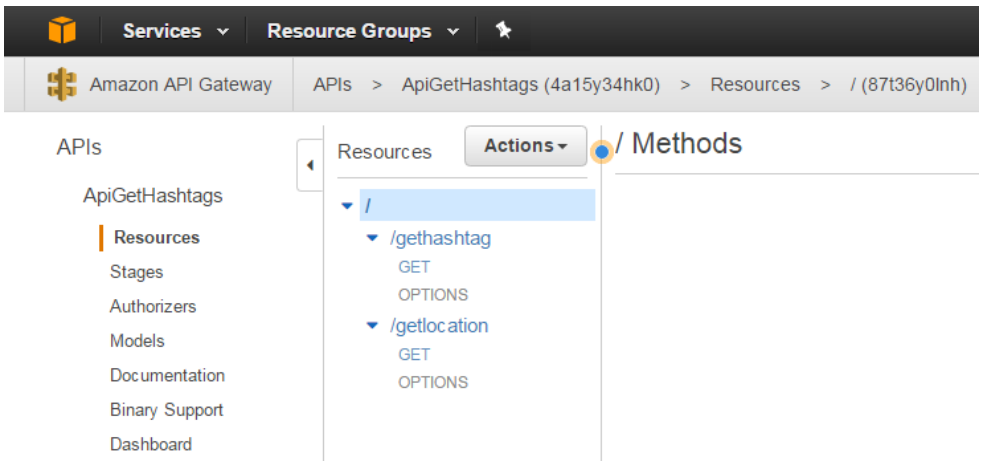

Figure13: *Screenshot of APi Gateway*

4.4 Scalability

So how scalable is this serverless application anyway? Well let's take a look at the individual AWS services and how scalable they are:

- ***DyanmoDB*** *scales very nicely and distributes your data automatically across multiple machines as needed. It can store unlimited data and can serve up to 10,000 reads/second. If that's not enough you can ask Amazon to raise that limit.*
- ***Lambda*** *functions are stateless and can scale without effort. Amazon will automatically use more EC2 instances to run your code as demand increases. There is however a safety throttle of 100 concurrent executions per region but that can be increased.*

- ***API Gateway*** *can handle "any number of requests per second", read: as much as your back-end can handle.*

4.5 Api Gateway to website

From the Api Gateway development link we managed to take the data and show it in a Format that makes it readable through the simple consumer eyes ; our goal was to extract the twitter api stream into global world locations and determine which are the currently more tweeted Hashtags .

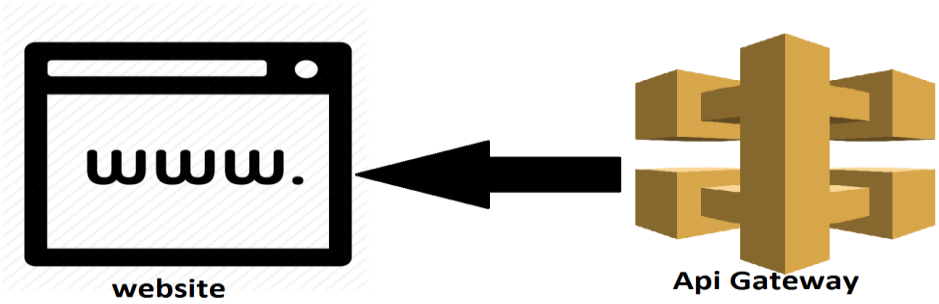

Figure 14*: from Api Gateway to The main website*

in between The tasks were as presented : bringing data from the api Gateway deployment link, reading every json file alone, creating a front-end for our Data and then uploading it into a running EC2 instance .

4.6 Reading Data Summary

So what we really did in this phase is building an automatic invocative system : which means that every part is triggered to work in real time through conditions: it only needs to open the website to trigger the ApiGateway that triggers by his side the lambda function that brings the data (parallel) in different Dynamo db tables .

A sample of how the flux of data is navigating from TwitterStream Api to the Website through our Architecture is presented in this way :

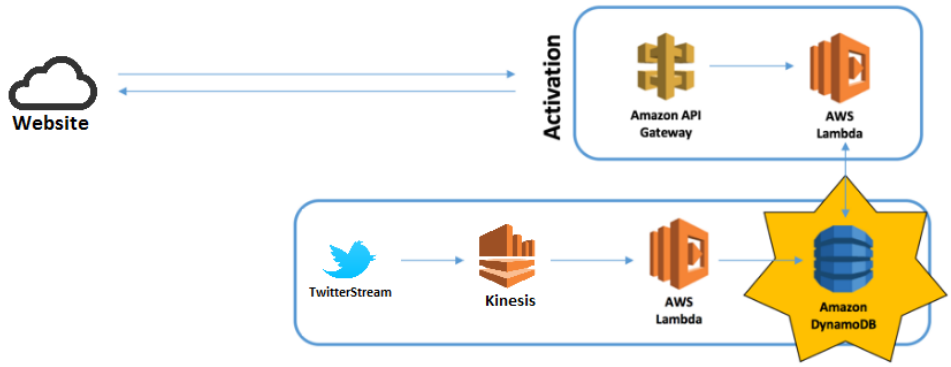

***Figure15** : Summary of the architecture flux of data*

So here is the result for our website :

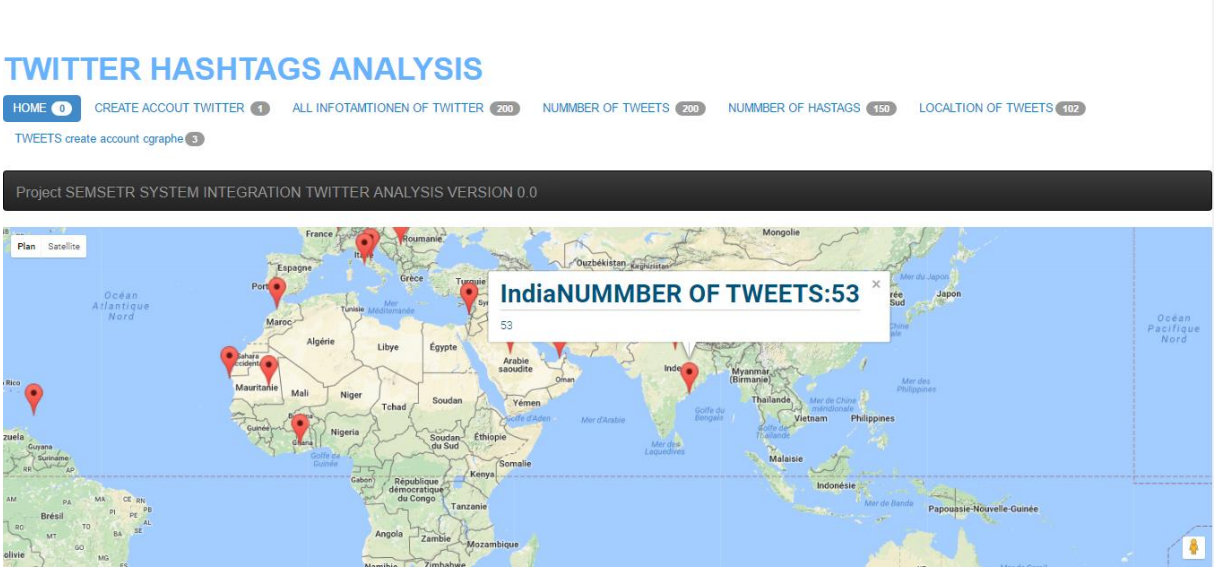

***Figure16:** screenshot of our website with a map and different menu*

In our Twitter Feed using bootstrap we made something nice in our mobile app here is a screenshot from our Mobile Phone :

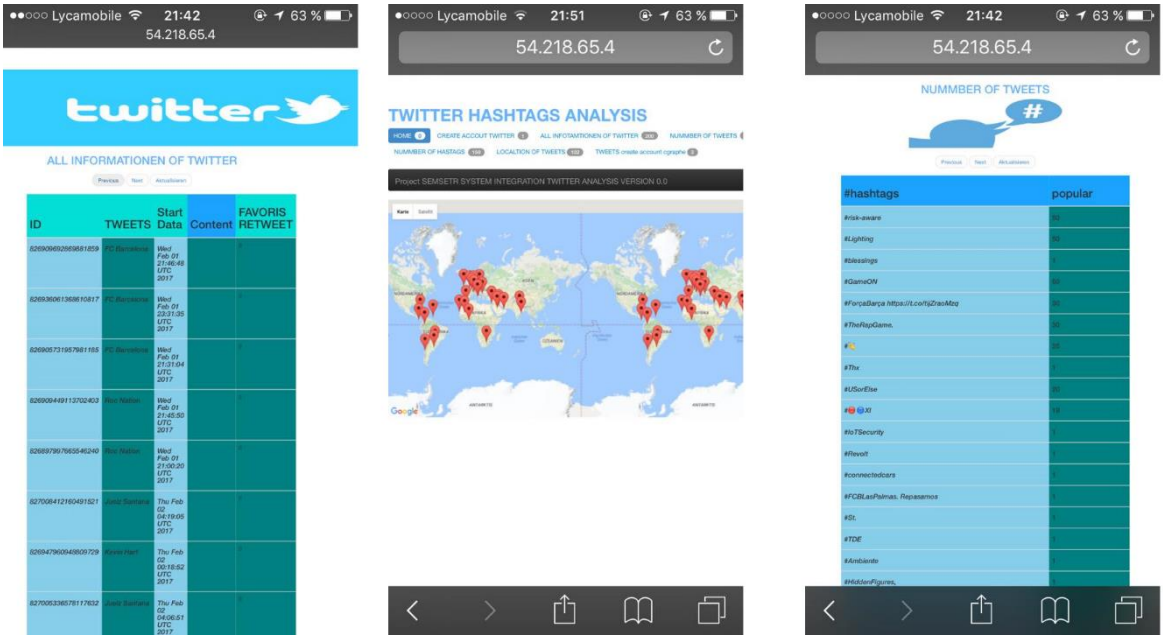

***Figure 17**: Screenshot of the website in our mobile phone*

5. Technologies

5.1 Backend Part

5.1.1 *Application Dependencies*

Application dependencies are the libraries other than your project code that are required to create and run your application. as the backend of our project was created in python , building the packages and managing dependencies it's a big part of our work

a tree of our repo. Needed this parts :
/tests (tox.ini file)

/setup.py file
/requirement.txt
/env (virtaulenv)
/pip

in the following we defined each component and his role into Packaging, Building, Deploying, Testing

5.1.2 Requirements.txt

The pip convention for specifying application dependencies is with a ***requirements.txt*** *file. When you build a Python web application you should include* ***requirements.txt*** *in the base directory of your project*

5.1.3 setup.py

There is another type of dependency specification for Python libraries known as setup.py. Setup.py is a standard for distributing and installing Python libraries. If you're building a Python library, such as requests or underwear you must include setup.py so a dependency manager can correctly install both the library as well as additional dependencies for the library. There's still quite a bit of confusion in the Python community over the difference between requirements.txt and setup.py, so read this well written post for further clarification.

5.1.4 Virtualenv

Dependencies are installed separately from system-level packages to prevent library version conflicts. The most common isolation method is virtualenv. Each virtualenv is its own copy of the Python interpreter and dependencies in the site-packages directory.The virtualenv stores dependencies in an isolated environment. The web application then relies only

on that virtualenv instance which has a separate copy of the Python interpreter and site-packages directory.

Tox: test a python package in a set of virtual environments.
*Tox is a generic **virtualenv** management and test command line tool you can **use for:***
checking your package installs correctly with different Python versions and interpreters running your tests in each of the environments, configuring your test tool of choice acting as a frontend to Continuous Integration servers, greatly reducing boilerplate and merging CI and shell-based testing.

*=> **Complex Directory structure for Programming in Python***
But as Python Programmers we automate this, right ?
No ! we lazy Python programmers , we let Someone else do it for us :

5.1.5 Cookiecutter

***Figure18** : cookiecutter logo*

Allows you to take a Template repository , it prompts it for input and generates an output directory with all this files

Features

- *Testing setup with unittest and python setup.py test or py.test*
- ***Travis-CI**: Ready for Travis Continuous Integration testing*
- ***Tox testing**: Setup to easily test for Python 2.6, 2.7, 3.3, 3.4, 3.5*
- ***Sphinx docs**: Documentation ready for generation with, for example, ReadTheDocs*
- ***Bumpversion**: Pre-configured version bumping with a single command*
- *Auto-release to PyPI when you push a new tag to master (optional)*
- *Command line interface using Click (optional)*

5.2 Continuous Delivery :

From our local to our Travis Ci we managed to continuingly putting the data through Github and from Github to Travis Ci for each commit we had a .travis.yml file and we used Visual Studio Code that makes the commit continuously for us:

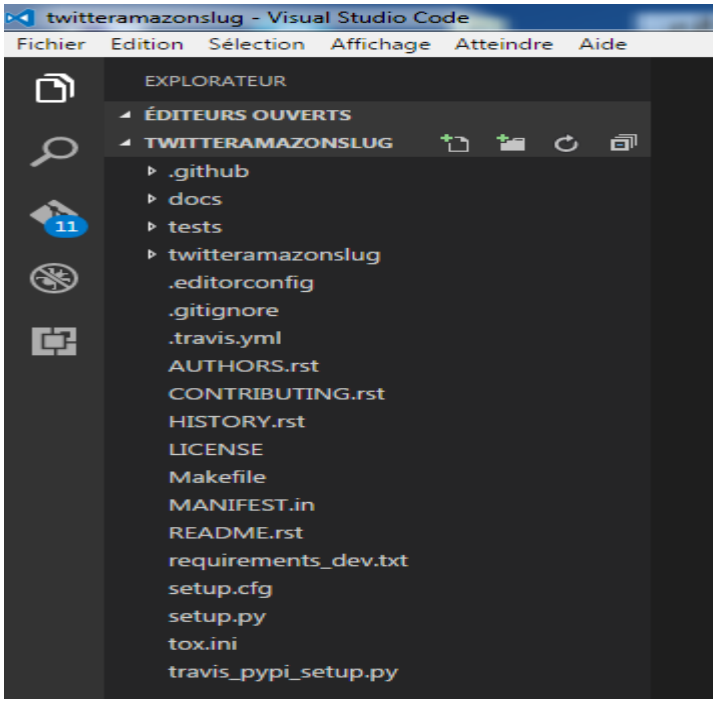

***Figure19**: screenshot of Github deployment in Visual code source*

5.3 Front end Technologies :

As usually made in the frontend we surely used CSS ,HTML5, javascript
What new was is the Angular js :
Angular js :
AngularJS is a structural Framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application's components clearly and succinctly. AngularJS's data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.
Bootstrap:
Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web

Framework Springboot :
Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.
For the dependecies we used maven :
maven : Maven is a build automation tool used primarily for Java projects. The word maven means "accumulator of knowledge" in Yiddish.[3]

Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies. Contrary to preceding tools like Apache Ant, it uses conventions for the build procedure, and only exceptions need to be written down. An XML file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging.

Rest Service :
More than a decade after its introduction, REST has become one of the most important technologies for Web applications. Its importance is likely to continue growing quickly as all technologies move towards an API orientation. Every major development language now includes frameworks for building RESTful Web services. As such, it is important for Web developers and architects to have a clear understanding of REST and RESTful services. This tutorial explains REST architecturally, then dives into the details of using it for common API-based tasks.

While REST stands for Representational State Transfer, which is an architectural style for networked hypermedia applications, it is primarily used to build Web services that are lightweight, maintainable, and scalable. A service based on REST is called a RESTful service. REST is not dependent on any protocol, but almost every RESTful service uses HTTP as its underlying protocol.

Unittest :
java unittest
SonorQube :

SonarQube is a platform for static code analysis of the technical quality of source code. SonarQube analyzes the source code with regard to different quality ranges and displays the results in a website. SonarQube is programmed in Java itself, but it also supports the analysis of Java programs with corresponding plugins

5.4 *Frontend Part :*

our frontend in reality (our client) is a whole web application in fact :
build it with maven and continuously integrated in jenkins
For the front end part we have used a REST SERVICE to get the information as a json , for this part we have used the MVVM model in angular js :
in the json there is two parts the getlocation information and the getHashtags, for the getlocation we bringed the latitude and longitude in a way to use them in a map, so we mapped the map with every index and when we click for every index in the location we see the ftcount (number of hashtags in this location) and the location name .
for the hashtags part, as received in the apiGateway GetHashtags the jsons where sorted by htCount , so we only put them in a page as a table to see the most popular hashtags in the moment
As a Web Application we need to put in the Server, for this thing we needed to upload everything in the EC2 instance
running in aws with specified attachement policies .

6. Conclusion

Because we already made a conclusion for every part we will talk here about are Brainstorming for the project and some of the perspectives :

As we were struggling through every Amazon aws services we find it really interesting to use such thing that allow you to manage data (streams) in real time and with a really comfortable manner .
we've chosen the above presented architecture essentially to easily parse data, storing it and controlling it in every way that we want (from hashtags , locations, user information etc ..)
As for the Technologies we managed to use maven or gradle in python but we find ourselves deeply into nights of studying dependencies packaging and every other details which helped us upgrade ourselves in a manner of constructing data and releases other than only installing maven in an eclipse environment and clicking a button ..
For the Perspective, due to time we were not able to deploy our code to aws deploy, we tried though, but we did find that it was much easier with travis ci and the .travis.yml file
Using the virtualenv was clearly a necessary step but using the docker container its much better , we wasn't able to use the docker because the performance of our laptops were lower than docker needs .
Perspectival speaking we can create a feed for every user of our website where he can choose which updates he needs to see in his account by us
for example : he open his session and he can see the 5 important tweets this day from his followers celebrities plus a what's going on in the world button, in which he gets redirected to a feed news where he can see the most important tweets today, top 10 hashtags, and where are they located .