Determining risks of certain trading signals

Sandnyit Pathare
6667774, sp01570@surrey.ac.uk

https://cloudcc-314612.ew.r.appspot.com

Abstract—**In this document, I have explained about my project where I have designed a cloud-based application which uses different scalable services like AWS Lambda and AWS EC2 to determine the risks of different trading signals over a certain time period. The main aim of the project was to design a pay as you go service for a user where he can have control over the types of trading signals and scalable service. The approach taken to design this application along with the cost required to run this application along with a high-level view of major system components is explained in this document.**

Keywords— Scalable Services, Lambda, EC2, GAE, Flask

I. Introduction

The NIST definition lists five essential characteristics of cloud computing: on-demand self-service, broad network access, resource pooling, rapid elasticity or expansion, and measured service. It also lists three "service models" (software, platform and infrastructure), and four "deployment models" (private, community, public and hybrid) that together categorize ways to deliver cloud services.[1]. This is sometimes referred to as 3-4-5 of Cloud.

This project explained in this document is somewhat based on SaaS service (Software as a service) with a hybrid deployment model as mentioned in NIST SP 800-145 definition. Here, I have made an attempt to make this application available on broad network, with scope for expansion wherever required. This is based on two of most popular cloud service providers: Amazon Web Services and Google Cloud Project.

Below I have explained User's perspective and Developer's perspective of this application which satisfies NIST 800-145 protocol.

A. *User*

The user does not have to download or install any software to use this application, here I have created a link using Google App Engine which makes this application accessible to a broadly available communication network and allows users to access this app using just a web browser. Because of this, it makes this application accessible with minimum dependency on the device. Users can use this link at any time. Also, the user gets an option to choose which service he wants to use (EC2 or Lambda), and he can also customise the trading signals for which he wants to determine the risk associated with the trading signals. And all this the user can do themselves without interacting with the developer and at the same time the user does not have to control the hosting environment.

B. *Developer*

From the developer's perspective, he can choose any cloud service provider (AWS, Azure, Google Cloud), and this application can be implemented by using any programming language, which is supported by the cloud service provider. Because of having a cloud service provider, the developer does not have to manage or control the network servers, operating system, or storage but at the same time, he has complete control over the application.

Even in this application, I didn't have to create servers or build a separate storage space for data, not even I had to design an operating system, everything was handled by cloud service providers (AWS and GCP in this application).

Here in this application AWS Lambda is used to run code required to calculate Risks on its own, in response to the HTTP requests through Amazon API Gateway. AWS S3 provides storage space for all calculated data. With the help of the google app engine, I was able to create a link with which the user can access this application from anywhere.

II. Final Architecture

A. *Major System Components*

a) API Gateway:

I'm using Amazon API Gateway because it's a fully managed service that helps developers create, publish, maintain, monitor, and secure APIs at any scale, making it much easier for them. APIs allow developers to access data and functionality stored on the backend by using a simple API link.[4]

Here in this application, we are using the REST API link to connect to lambda from backend and Boto3 to connect to EC2 and S3.

b) AWS Lambda

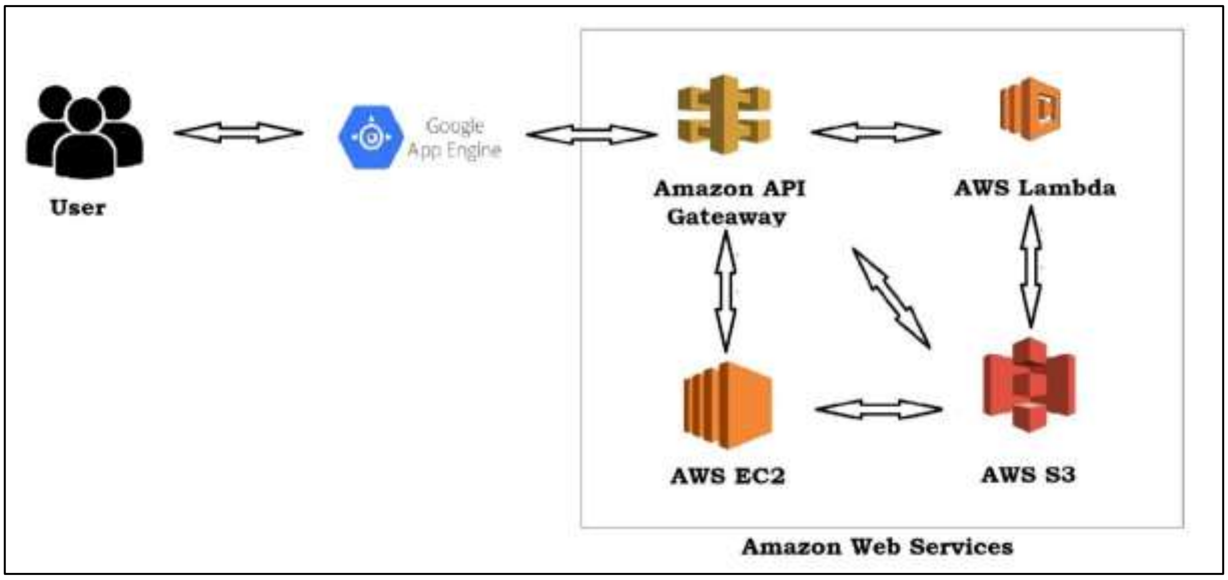
AWS Lambda supports a large number of programming languages, so I just had to send my code to it. Lambda runs code without the need of creating or maintaining servers, it automatically runs code on high-availability compute infrastructure and performs the administration of servers, code monitoring, logging, etc. [5]

For this application, Lambda is triggered using the REST API functionality of AWS. This API is triggered by using an HTTP webpage which is accessed by the user. Python 3.9 script is added to the lambda function to calculate risks associated with the trading signal. These results are again sent back using REST API to the HTTP front end page.

c) AWS EC2

Amazon EC2 provides us with a virtual computing environment termed as 'instance'. EC2 provides us with the option of scalable computing capacity where you can choose between various processors, RAM, and storage for your instances as per your requirement. [6]

Figure 1 HIGH-LEVEL OVERVIEW OF SYSTEM



For this application, I am using real-world AWS for creating EC2. A t2.micro instance is created in us-east-1 region. Key required to create this EC2 instance is stored in GAE. The communication between GAE and AWS takes place through Boto3. Boto3 is a Python API for AWS infrastructure services. [7] For this to happen I had to create an instance profile as a container for the IAM role. AWS Identity and Access Management (IAM) provides fine-grained access control across all of AWS [8]

d) AWS S3

Amazon S3 is a bucket-based storage service that stores data as objects. A file and any metadata that describes it are considered objects. A bucket is an object container. In simple words A [9]

Here I am using S3 to store results in a container. I am using Boto3 API to connect the backend to S3. I decided to store data in S3 because it is cost-efficient and easy to implement. Here I am creating a data frame with the predicted risks values and then converting it to get stored as a .csv file in a bucket named ‘cc-cw-1’.

e) Google App Engine

Google App Engine is a serverless platform for developing and hosting online applications. [10]. This service is one of the services offered by the Google Cloud Platform (GCP). GAE offers an environment which supports multiple programming languages, running small instances, configuring CPU and memory, etc.

We need to specify the programming language and entry point in an app.yaml file. And to run our application we need to specify dependencies for our python code in a separate requirements.txt file which we need to install using the pip method while deploying the application into the app engine.

Deploying this application is very easy from the developer’s perspective as he needs to have google cloud SDK installed on his system, after which the deployment can be done in a few steps.

Google App Engine acts as a backend to our application, a majority of the requirements and code are stored here. This gives the user an option to access this application just by using a link and removing dependency from hardware.

B. *System Component Interactions*

This application is built using Python programming language and HTML is being used for the front-end and user-interface. Flask is used as a framework between HTML with python.

As it was not possible to use the pandas library in Lambda, I decided to do all pandas related work in the backend and calculation of risks I did in AWS Lambda.

There was a ‘for loop’ in the code provided in the coursework description which was doing the job of calculating the risks associated. I decided to put that code in Lambda, but for that code to work, we needed a pandas dataframe. So, I converted that entire dataframe into a list and passed it on to lambda over API. I used this list to calculate the mean and std required to calculate risks. After calculating risks, the 95 and 99 percentage risks values were sent back to the backend in JSON format.

After receiving the values, I took the average of all values as mentioned in the coursework description and passed them to the graph.html page by using Flask. Flask is a small python web framework which is primarily used for making web applications in Python.

For plotting graph and table on the graph.html page, I used ready-made templates provided by google charts, with some modifications. For the audit table, I did an HTML partition for which I referred to W3School. [14]

For EC2 the backend gets input from the user, and EC2 instances are created. To allow EC2 instances to be fully started I had kept a sleep timer of 100 seconds after which the instance will execute code stored in user data. I have added basic few lines of code where pip gets installed along with all required dependencies for running the python application. EC2 instance will be terminated after running the code stored in user data.

III. SATISFACTION OF REQUIREMENTS

Below in TABLE 1, I have mentioned requirements met or not met for the coursework description.

TABLE I. SATISFACTION OF REQUIREMENTS AND CODE USE/CREATION

#	*C*	***Description***	***Code used from elsewhere, and how used***	***Code you needed to add to what you used from elsewhere***
i.	M	Created Front Page, where user can give input and select scalable service to determine risk for certain trading signals.	I referred w3school to learn how to make page using html, so some references are taken from that page [14]	Most of the code for front page was written by me according to the requirements.
ii.	M	Feed inputs given by user to backend	I referred code from labs for this, as there was a mention of taking inputs user to plot charts.	I had to modify code to take inputs from user as per coursework requirement.
iii.	M	I was able to Create Lambda function and calculate risks associated with trading signal.	To create AWS lambda, I referred to the codes given in labs and to calculate risks I used code provided in the coursework description.	I had to do some modifications in the code in coursework for sending data to Lambda, and to calculate mean and std in AWS Lambda.
iv	P	I was able to Create and terminate instances, but I was not able to calculate risks associated using them.	I referred to the code from lab 4 document to create EC2 instance, I also referred to multiple YouTube videos [15] and websites to automate creating and stopping instance, but I was notable to calculate risks associated.	I had to modify the code from Lab 4 to meet the coursework requirements, also I had to refer to YouTube videos to code for starting and terminating of instances
v	M	Created Graph and table which shows user, the determined risks for certain trading signals. I also created an Audit table which shows user inputs and last value of calculated risks.	For this most of the code was taken from "developers.google.com/charts."	I just had to edit code to add data returning back from AWS to the graph and chart table. For Audit I created a simple HTML partition
vi	M	Deployment of application into the Google App Engine was successful and I was able to generate appspot link to access my application	I followed instructions in Lab 1 to deploy my app to google app engine.	I had to edit requirements file according to the requirements of my code.

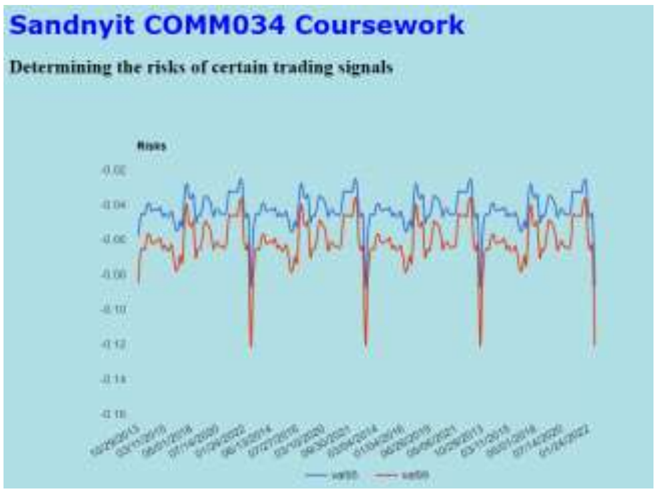
IV. RESULTS

TABLE II. RESULTS

<u>*Sr. No*</u>	<u>*Dates*</u>	<u>*Val95*</u>	<u>*Val99*</u>
1	10/29/2013	-0.059	-0.085
2	11/13/2013	-0.047	-0.068
3	01-08-2014	-0.045	-0.065
4	01/13/2014	-0.045	-0.065
5	03-04-2014	-0.04	-0.058
6	03/20/2014	-0.039	-0.057
7	30/04/2014	-0.043	-0.061
8	16/05/2014	-0.043	-0.062

Below in the FIGURE 2, I have included a screenshot of output which was calculated using AWS Lambda. For this I had specified shots as 320000, resources as 4 and 200 history and buy signal was selected.

FIGURE 2 RESULTS



Sandnyit COMM034 Coursework

Determining the risks of certain trading signals

V. Costs

What cloud costs depends on what you are paying for and the exposure of those things over different time periods. AWS billing varies according to what kind of pricing approach you are making use of e.g. (on-demand and reserved instances). On-demand (lambda) instances can prove more costly for long term use if compared to reserved instances (EC2). Costs also depend on the region (usually the USA is the cheapest).

1. Pricing calculations for lambda [11]

I have calculated the price for using an AWS lambda assuming that around 1000 requests will be done per month. I have considered 512 MB of memory to calculate the cost, I have assumed 61000 ms of time taken (which was the average time taken for this application when calculating the inputs mentioned in the coursework example), Region selected here was US-EAST-1.

1,000 requests x 61,000 ms x 0.001 ms to sec conversion factor = 61,000.00 total compute (seconds)

0.50 GB x 61,000.00 seconds = 30,500.00 total compute (GB-s)

30,500.00 GB-s x 0.0000166667 USD = 0.51 USD (monthly compute charges)

1,000 requests x 0.0000002 USD = 0.00 USD (monthly request charges)

Lambda costs - Without Free Tier (monthly): 0.51 USD

Cost for lambda will vary if the time taken per instance is changed, if we change memory or do more requests for lambda functions.

TABLE III. Cost of AWS Lambda in different scenerios

Requests (per month)	Time taken (per instance)	Memory (MBs)	Cost (USD)
1000	11000	128	0.01
1000	11000	512	0.05
1000	22000	512	0.11
10000	11000	128	0.13
10000	11000	512	0.53
10000	22000	512	1.07

These calculations are made considering there was no free-tier for lambda. If we consider the free tier the calculations would have been different.

2. Pricing calculations for EC2 [12]

I have calculated the price for EC2 assuming that the t2.micro instance will be used for up to one hour a day during weekdays and a maximum of 6 instances will be used at peak time. The region selected here was US-EAST-1.

130.3572 On-Demand instance hours per month x 0.011600 USD = 1.512144 USD

The cost almost doubled when I changed the instance type to t2.small.

130.3572 On-Demand instance hours per month x 0.023000 USD = 2.998216 USD.

This show pricing of EC2 is mostly dependent on type of instance we select making creating EC2 instances. Also, the memory used affects the overall cost of EC2.

3. Pricing Calculations for Google App Engine [13]

App Engine does not provide a free tier in a flexible environment. Apps running in the flexible environment are deployed to virtual machine types that you specify. These virtual machines are billed on a per-second basis with a one-minute minimum usage charge. [13]

Pricing is calculated on an hourly basis for the various computing resources in a flexible environment.

If we consider the Europe-west-2 region (which I used to build this application) hourly charge per core per CPU is 0.063 USD and for memory, it is 0.009 USD.

References

[1] Mell, P. and Grance, T. (2011). Special Publication 800-145 The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology.

[2] Cloud Computing Definition by NIST. [online] Available at: https://www.youtube.com/watch?v=-zrdDOLx0Kg&t=392s [Accessed 10 May 2022].

[3] A Practical View of NIST's Cloud Definition. [online] Available at: http://novacontext.com/a-practical-view-of-nists-cloud-definition/.

[4] Amazon Web Services, Inc. (2019). Amazon API Gateway. [online] Available at: https://aws.amazon.com/api-gateway/.

[5] Amazon.com. (2019). What Is AWS Lambda? - AWS Lambda. [online] Available at: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html.

[6] Amazon Web Services (2019). What Is Amazon EC2? - Amazon Elastic Compute Cloud. [online] Amazon.com. Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

[7] Amazon.com. (2022). AWS SDK for Python (Boto3) Documentation. [online] Available at: https://docs.aws.amazon.com/pythonsdk/?id=docs_gateway#:~:text=The%20AWS%20SDK%20for%20Python [Accessed 23 May 2022].

[8] Amazon Web Services, Inc. (n.d.). AWS Identity and Access Management - Amazon Web Services. [online] Available at: https://aws.amazon.com/iam/#:~:text=AWS%20Identity%20and%20Access%20Management%20(IAM)%20provides%20fine%2Dgrained

[9] docs.aws.amazon.com. (n.d.). What is Amazon S3? - Amazon Simple Storage Service. [online] Available at: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

[10] Google Cloud. (n.d.). Google App Engine Documentation. [online] Available at: https://cloud.google.com/appengine/docs.

[11] https://calculator.aws/#/createCalculator/Lambda. (n.d.). AWS Pricing Calculator. [online] Available at: https://calculator.aws.

[12] calculator.aws. (n.d.). AWS Pricing Calculator. [online] Available at: https://calculator.aws/#/createCalculator/EC2.

[13] Google Cloud. (n.d.). App Engine Pricing | App Engine Documentation. [online] Available at: https://cloud.google.com/appengine/pricing.

[14] W3Schools (2018). HTML Tutorial. [online] W3schools.com. Available at: https://www.w3schools.com/html/.

[15] www.youtube.com. (n.d.). EP-94 | How to Create EC2 instance using Python 3 with Boto3 | Automation with AWS and Python. [online] Available at: https://www.youtube.com/watch?v=ijPOevW-ZOM.