A cloud-hosted web application to calculate the risks of using certain trading signals

Prakash Jha   
6659329, pj00253@surrey.ac.uk

*Abstract*—The document describes a application hosted on cloud to calculate the risks of using certain trading signals for a trading strategy. The document describes the system's major icons, their purposes, and how data is communicated between the components in order for the system to function as expected. An analysis of the costs of the system's major icons is presented. Finally, a table detailing the fulfilment of the requirements is presented.

Keywords—Scalable, on-demand, pay-as-you-go, measured-service)

# Introduction

The system discussed in this document is an application that uses the Monte Carlo method to calculate the risks of using specific trading signals for a trading strategy. It is built on the cloud computing model to provide convenient, on-demand network access to a shared pool of configurable computing resources (servers, storage, services) which can be rapidly provisioned and released with minimal management effort or interaction from service providers[1] . The system abides by 3-4-5 definition of NIST SP800-145 model . The system provides software-as-a-service model to the end users whereas platform-as-a-service and infrastructure-as-a-service to the developers. The system can be rapidly provisioned and released with minimal management effort or service provider interaction. Services of two providers , AWS and Google cloud platform, is used.

The system is scalable and is billed as per the resource usage. The system can be viewed from the developer and user perspective.

## Developer Perspective

The developer does not manage or control the underlying cloud infrastructure, which includes the network, servers, operating systems, and storage, but have control over the deployed applications and possibly the application hosting environment configurations.[citation]. The provider automatically measures all the resource usage. The developer has the option to choose from a wide range of services provided by the two providers , AWS and GCP. If capacity needs upgrading or the version of the databases,patching of the servers, such kind of concern is the providers problem because developer is only concerned about managing applications and the data[1]. The developer benefits a lot from the services provided by the provider.

## User

The user does not have to install any software and only interacts with the system through web browser. Because of this the user does not have to worry about downloading,installing, upgrading or maintaining the system. The user does not even need any particular kind of infrastructure since application is mainly accessed through browser or a thin lien All the users get the same version of the system at the same time. Since the application is on a browser, the user can access it from anywhere with a active internet connection. The user experiences the ‘software-as-a-service’ model.

With the possible exception of limited user-specific application configuration settings, the user does not manage or control the underlying cloud infrastructure, such as the network, servers, operating systems, storage, or even individual application capabilities.

# Final Architecture

## Major System Components

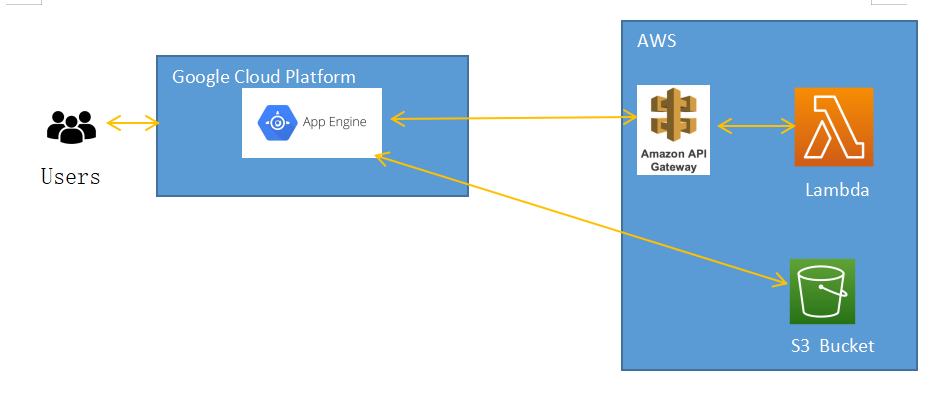
Google App Engine - The purpose of GAE in this system is to execute web application and take advantage of the large computing infrastructure of google to dynamically scale as the demand varies over time. The App Engine also eliminates some system administration and developmental tasks to make it easier to write scalable applications. GAE serves a Platform as a service in this system [2][3]. GAE provides all-in-one features to developers to run and host web applications, giving the developers the liberty to only concentrate on building the front end and their functionality[5].

AWS Lambda - The purpose of AWS lambda is to execute backend code, which is primarily important for scaling. If the user requests that more resources to be used, the application will run the lambda in parallel processing the number of times specified by the user. AWS Lambda manages all of the infrastructure required to run the code on highly available, fault-tolerant infrastructure, freeing the developer to concentrate on building backend services[4]. Lambda serves as a serverless, cost-effective backend service since the application is built only for the compute time and there is no charge when the code is not running.

S3 Bucket - The s3 bucket is used for storage in the system. It is easily accessible over the internet and provides the user the facility of going through their last run var values .

API Gateway - The API gateway acts as the middle layer between the google app engine and the AWS lambda. Amazon API Gateway enables developers to easily create, publish, maintain, monitor, and secure APIs. APIs serve as the "front door" through which applications access data, business logic, or functionality from the backend services [9]. The system uses API Gateway to create RESTful APIs that allow for real-time two-way communication AWS lambda and backend python file.

Even though EC2 is not implemented in the system, EC2 would have been selected over other EMR or ECS . EC2 could launch a virtual server in the amazon cloud with a single web service call and you can load balance in a region [10]



## System Component Interactions

The user interacts with the application through a web browser. The application provides the user three inputs i.e minimum history, number of shots and trade signal on a html file. On the click of the submit button on the html, these input values are passed on to the flask(or backend python file) through HTTP post method where they are send to the AWS lambda as json parameters via API Gateway POST method. In the AWS lambda, these values are read where they are used to run the logic to find the var95 and var99.

The values of var95 and var 99 are returned to the index.py from the aws lambda where the values are decoded and sent to the s3 bucket through a function and send to a form.htm through doRender() method through a dictionary.

# Implementation

The system's implementation can be divided into three parts, front end, backend and the storage. The front end is hosted by the google app engine. The backend is served by AWS lambda and S3 bucket is used for storage.

HTML and CSS are used to develop the front-end and partially the code for it is used from lab1 and lab3. The form.htm has three input parameters and one submit button. The 'Audit.htm' uses the jinja code similar to what is used in labs to print the data on the page, if the audit data received from s3 bucket is not empty.

The backend code is written in Python and makes use of several Python libraries such as flask,gunicorn,boto3, and others to meet the application's requirements. The code provided in the assessment is used at the backend python file to retrieve data from yfinance and mark the 'buy' and 'sell' flags. When the submit button on the form page is clicked, the code is written to calculate the var95 and var99 values based on the trade signal selected, i.e. buy or sell. There is a 'if and else' condition written to separate the calculation of values based on the signal selected, i.e. if 'buy' is selected, all the records marked as 1 in the dataframe's buy column are run in a for loop where mean and standard deviation are calculated and sent to AWS lambda along with the user entered shots to calculate the var values. If 'Sell' is selected, the same logic is followed, with the exception that the mean and standard deviation of all the records marked as 1 in the dataframe's sell column are calculated before being sent to AWS lambda. The code from the lab3 is used to pass the parameters from the backend python file to the AWS lambda. The code from the given assessment is reused in the lambda function to calculate the var95 and var99 from the mean, standard deviation, and shots, and then appended to a list before being returned to the backend file where they are decoded.

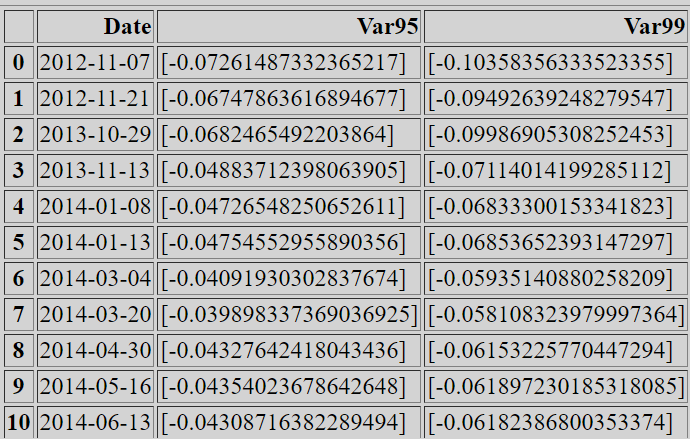
For the s3 bucket, the aws credentials is declared and with the library 'awsdatawrangler' , the code is written to write the response values into a csv file in the s3 bucket. The code is partially used from the web[reference] and partially used from the labs[]. There are two files in the bucket, one to capture the inputs of minhistory,shots and tradesignal and other to capture the values of data,var95 and var99. There is a hyperlink button made on the form page which takes it to an audit.htm page. On click of the hyperlink, the form.htm page routes it to the readtocsv() function through action attribute. The function reads the data from bucket and passes on to audit.htm with the dorender() function. In the audit.htm, jinja code is written to display the records received from the backend python file.

For the google chart, the aws lambda values var95 and var99 are assigned to two new variables. The mean of these two variables is calculated using the mean function of the statistics library and assigned to two new variables. Both variables are run through a for loop, where they are appended to a new list and then passed to the htm page via the dorender function. All of these variables are passed to the code ,used from the web, to run the line chart on the htm file.[6]

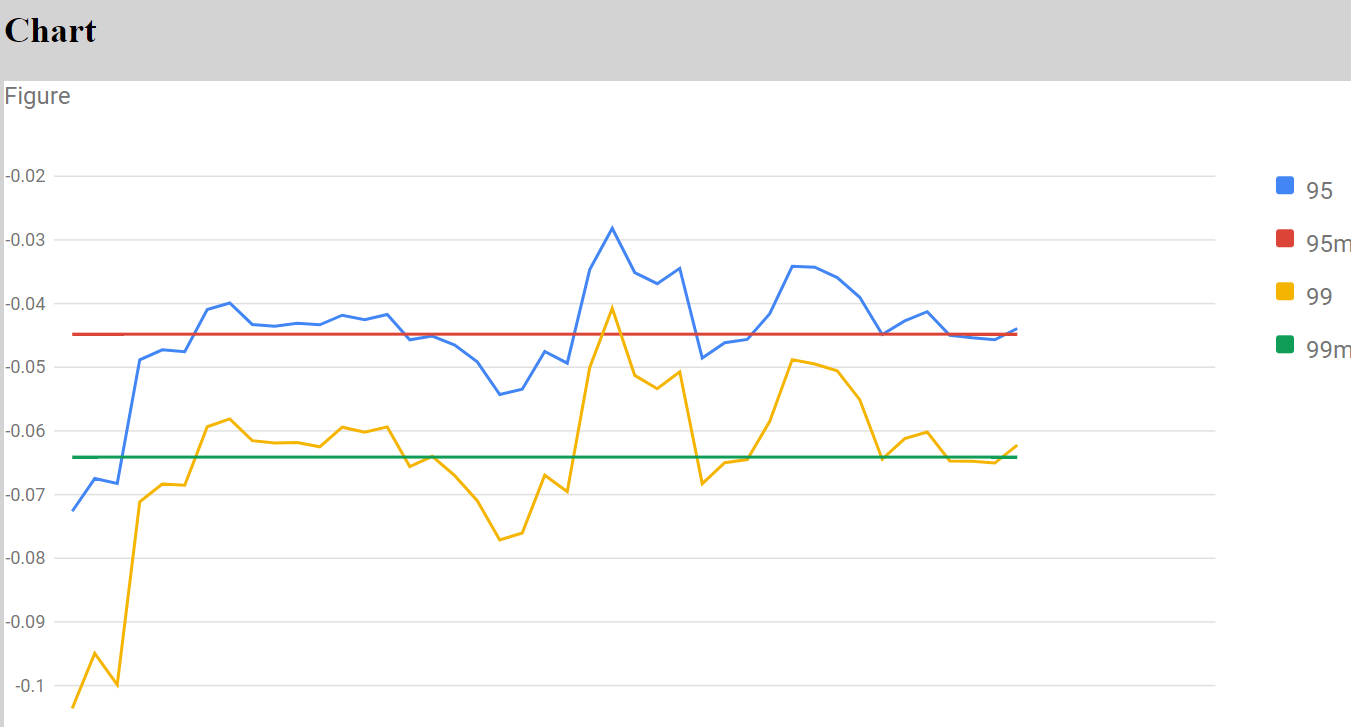
https://my-project-25051.ew.r.appspot.com/

# Results

The following is the result of inputs when the number of shots was set to 100,000 and the minimum history was set to 200 days.



The results display 42 records of var95 and var99 signals, along with their respective dates.



# Satisfaction of Requirements

1. Requirements

| # | C | Description |
| --- | --- | --- |
| i. | P | The system uses Google App Engine, AWS Lambda but not any other scalabale service. |
| ii. | P | The system offers the front end to initialize the resources but there is no provision to terminate the resources. |
| iii. | M | AWS Lambda is calculating the risk values and made available for front end. |
| iv.a.i | N | The system does not offer the provision to choose from lambda and EC2. The system could have provided a provision like radio button to choose between lambda and EC2 and on event selection of that radio button, its corresponding code would have been called and executed. |
| iv.a.ii | P | Provision to run resources input is available but it is not working. |
| iv.a.iii. | N | Could have warmed up the lambda by writing a minimal event like random number generator code |
| iv.a.iv | N | There is no provision to capture the run time and provide to the user. The system could have used the datetime library and assigned two variables such as starttime and runtime. These starttime variables would have been at the start of the code's execution and endtime variables would have been at the end of the code's execution. Both times would have been deducted and passed on the front end to provide the user with the application's exact runtime. |
| iv.b.i | M | The system has the provision to specify the value of H |
| iv.b.ii | M | The system has the provision to specify the value of D |
| iv.b.iii | M | The system has the provision to specify the value of T |
| iv.b.iv | P | The risk calculation is done and the results are stored in the audit but it is not working for each resource since there is no provision to run the resources in parallel. |
| iv.c.i | M | The system shows a table which has date ,risk values and line chart. |
| iv.c.ii | P | The audit page shows selection of D,H,T, risk average values but not compute time costs and resources. |
| iv.d | N | If EC2 would have been in the system, the session could have been terminated with the help of boto library |
| iv.e | N | Briefly justify work done  And/or explain what could/should have been done. |

# Costs

The costs involved for the application are from google app engine, S3 buckets, AWS lambda and amazon API gateway.

GAE costs - Any use of App Engine resources that exceeds the free tier incurs fees. Assuming the environment to be standard and the region to be us-central-1, then for the default instance class, the charge calculated will be 0.05$ per hour. If we compare it with the london region, the charges incurred will be 0.06$ per hour. The costs is calculated from the site[7]

S3 costs - Excluding the free tier discounts, the S3 standard storage of 1 GB per month and assuming 1000 requests to the S3 storage, the costs incurred would be

Tiered price for: 1 GB

1 GB x 0.0230000000 USD = 0.02 USD

Total tier cost = 0.0230 USD (S3 Standard storage cost)

1,000 PUT requests for S3 Storage x 0.000005 USD per request = 0.005 USD (S3 Standard PUT requests cost)

0.023 USD + 0.005 USD = 0.03 USD (Total S3 Standard Storage, data requests, S3 select cost)

S3 Standard cost (monthly): 0.03 USD.

AWS Lambda costs - Assuming the number of requests sent to the lambda is 1000000 ,duration of each request is 90 seconds and amount of memory allocated is 128 mb

Amount of memory allocated: 128 MB x 0.0009765625 GB in a MB = 0.125 GB

Pricing calculations : 1,000,000 requests x 9,000 ms x 0.001 ms to sec conversion factor = 9,000,000.00 total compute (seconds)

0.125 GB x 9,000,000.00 seconds = 1,125,000.00 total compute (GB-s)

1,125,000.00 GB-s x 0.0000166667 USD = 18.75 USD (monthly compute charges)

1,000,000 requests x 0.0000002 USD = 0.20 USD (monthly request charges)

18.75 USD + 0.20 USD = 18.95 USD

Lambda costs - Without Free Tier (monthly): 18.95 USD

API Gateway costs - For 100,000 total REST API requests,

100000 requests x 0.0000035000 USD = 0.35 USD

Total tier cost = 0.3500 USD (REST API requests)

All the AWS costs calculated from the site [8]

##### References

1. L. Gillam, Lecture Week 2, COMM034-Cloud Computing, 2020-2021, University of Surrey

[2] "Google Appengine - an overview | ScienceDirect Topics", 2021

[3] What is Google App Engine ? - Definition from WhatIs.com, 2021

[4] AWS Lambda – Product Features. (2021), from <https://aws.amazon.com/lambda/features/>

[5] Techopedia.com. 2021. What is Google App Engine (GAE)? - Definition from Techopedia. [online] Available at: <https://www.techopedia.com/definition/31267/google-app-engine-gae>

[6] Google Developers. 2021. Line Chart | Charts | Google Developers. [online] Available at: <https://developers.google.com/chart/interactive/docs/gallery/linechart>

1. <https://cloud.google.com/appengine/pricing>
2. <https://calculator.aws/#>
3. [9] https://aws.amazon.com/api-gateway/

10L. Gillam, Lecture Week 5, COMM034-Cloud Computing, 2020-2021, University of Surrey