Cloud-based application that determines average value at risk (VAR) for a trading strategy based on a simple moving average

Pratyaksh Rao
6612540
pr00358@surrey.ac.uk

***Abstract*—In the current timeline, data security, scalability alongside access control and storage are cornerstones of current work cycle and requirements for any cloud services. Users can be needing various features such as cost effectiveness, secure storage, etc, but providing them to the user by the cloud service provider can be a demanding and hard task at hand. This paper proposes a basic cloud app using two architectures while exploring costs, security and storage adding up to accessibility.**

Keywords—Cloud computing; cloud storage; AWS; GCP; Lambda.

I. Introduction

Cloud computing is a architecture that enables on demand network access to many available services provided (e.g. storage, computing, applications) that can be easily managed, modified according to the demands of the user with minimal service provider interaction. My application provides a calculation and chart by accessing storage, taking input from the user relating to some parameters and showing an output by using one cloud service to do the calculation and hosting the app on another cloud service.

A. Developer

As a developer the app takes in the input for parameters from the user to alter the Data chart being printed with user varied moving average values. Taking user input once again to show the VaR based on the user settings. Making least room for developer – user interaction as most user demands will be catered through the app.

B. User

As a user the app takes into account most parameters that a user would need to adjust but not all. For example setting storage limit, No network access to broader platforms, such as mobile, resource usage is not provided unable to make it a measured service.

II. Final Architecture

A. Major System Components

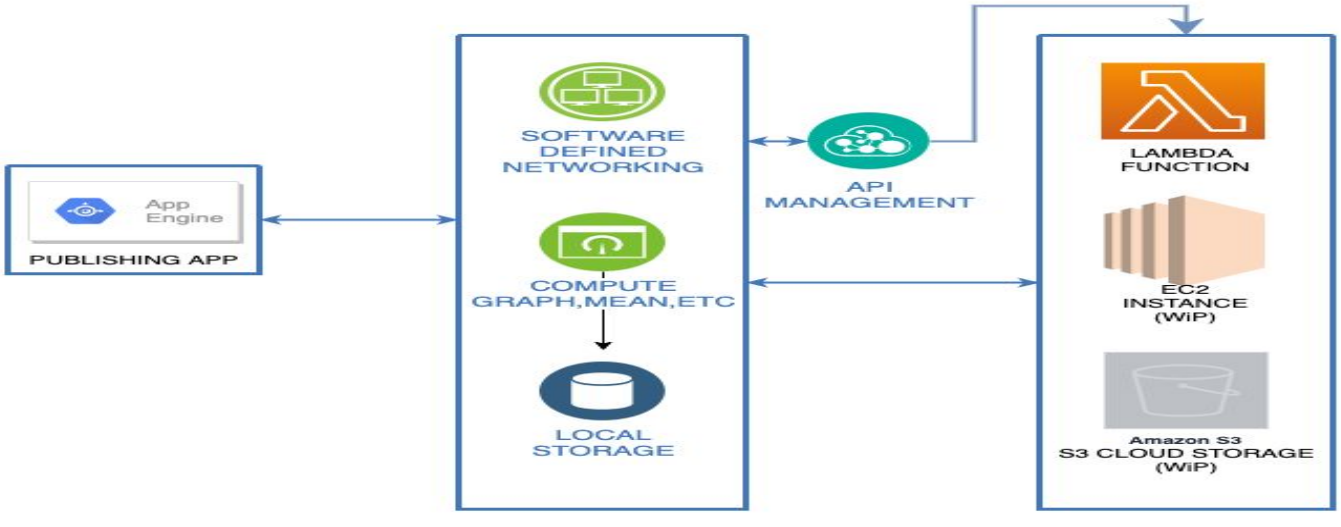
There are three major components to the application being: Local System service, Amazon web service and Google Cloud platform. These elements are interconnected with relations as shown in the diagram below.

The local storage is the destination of storage of data, application code, flask code, web page code, minor calculations code; which is then forwarded to AWS for it to compute the major calculation which is the transmitted via the local system code to the web page. This entire this is encapsulated as a web app which is then hosted using the GCP.

The Scalable service that is used is Lambda which is capable of running code in event response and makes it easier as a developer to manage and compute resources, as compared to other viable scalable service options such as EC2 which is excellent in providing resizable capacity to compute or ECS which helps in easily running and management of docker containers dealing with clusters of EC2 instances.

B. System Component Interactions

The lifecycle of the system begins with the data being stored in the local system. An HTML page is designed to create a form that takes in input from the user to print a chart and table on the second web page. The second web page is designed alongside the flask code in python which uses the data stored and does calculations such as adding Moving Average Column,etc and then that Python code is linked with the second web page to print out the chart and table.



In the second web page another input will be taken which will then print the calculation (done in AWS). This calculation's code is then written in Lambda service and then linked to the python program to print the final calculation (VaR). This app is then hosted via the GCP. In conclusion the local system communicates with AWS which then communicates the result back and then the local system communicates the whole app to GCP alongside the Data which is also communicated through the local system .

III. Implementation

Implementation of the app started which the design of the web page. The code for the webpages were taken from Lab code but was edited in order to meet the requirements of the number of user input parameters, creating charts and tables, displaying of calculated results, etc. The basic python app structure was used from the Lab code but more than 80% of the code had to be edited in order to meet the requirements for smooth flow of communication between components of the app.

Python libraries including plotly were used to plot graphs such as time series data and the moving average data. To calculate the moving average df['MA'] = df['Adj Close'].rolling(window=mavg).mean()was used alongside df['sell'] = ((df['Adj Close'] <= df['MA']) & (df['Adj Close'].shift(1) >= df['MA'].shift(1)))to calculate the sell signal from the data and df['buy'] = ((df['Adj Close'] >= df['MA']) & (df['Adj Close'].shift(1) <= df['MA'].shift(1)))to calculate the buy signal.

These were then added to the datasets 'dataframe' which was printed as a table alongside a chart on the webpage. Next step is to calculate the VaR so for that the return series was generated with df['returns'] = df['Adj Close'].pct_change(), Mean and standard deviation had to be calculated which will be sent to the Lambda function code in order to calculate the VaR using the random.gauss function and also calculate the 99th confidence value by making another function in the same code. After creating the 'POST' method of execution in the API-Gateway a invoke URL was obtained which was routed to the python code to print the results obtained from the lambda function to the web page. The lambda function code for VaR was taken from the Lab code and the 99th confidence code had to be written. By using parallel computing the execution and efficiency can be optimised if multiple Virtual Machines are used parallelly to compute the same result in comparison to serial which can cause a problem when the data is increased significantly and scaling is essential which would normally cause serial execution to lag.

The most challenging part of the coding was the web and flask development inclusing Flask, HTML, CSS

APPSPOT- https://eco-shift-268116.appspot.com/

IV.

V. Satisfaction of Requirements

TABLE I. Requirements

#	*C*	***Description***
i.	P	Function created in Lambda, Ec2 but Ec2 not yet routed to display result via app
ii.	M	Takes user input to display Chart and Table and VaR
iii.	P	The Buy/Sell signal is a candle chart instead of arrows. Can be made to be arrows
iv.	P	The table to be modified to show VaR and Profit/Loss on it instead of printing seperatly
v.	M	AWS lambda runs the code for VaR and confidence values
vi.	N	A signal trigger alarm can be made for the instance to shut ON/OFF with regard to a user query
vii.	P	Parameters such as Monte Carlo Sims and VaR period, Resourses Can be added for Better front end UI
viii.	M	The lambda function calculation and result displaying is printed after one click
ix.	N	The VaR values and profit/losses can be stored in AWS S3 cloud storage for easier access by the app
x.	P	The data currently 'lives' in the local System but can be moved to S3 for better access control from the lambda function and GCP

VI. Costs

Currently the App does not incorporate Cost management as Lambda function is Not triggered to switch OFF at any point, the data can be moved to S3 storage with the tradeoff of additional costs but higher security. With the use of Parallel Computation, the use of multiple VM's add to the costs which can be solved with single VM serial execution with the tradeoff for higher efficiency and Speed. In lambda the timeout and memory usage can be optimized and maybe user defined in order to be cost effective and just utilize the memory as needed.

VII. Security

For the Lambda Function a IAM key can be added to increase security, currently it is set on OPEN. A trigger Alarm can also be set both on Lambda and EC2 which will act as a timeout to shut down an instance if the computation stops for a set time.

VIII. Results

For Amazon (based on last two digits of URN)

MA	*VaR 99% Confidence Values*
10	0.060
50	0.045
100	0.051
150	0.046

200	0.057

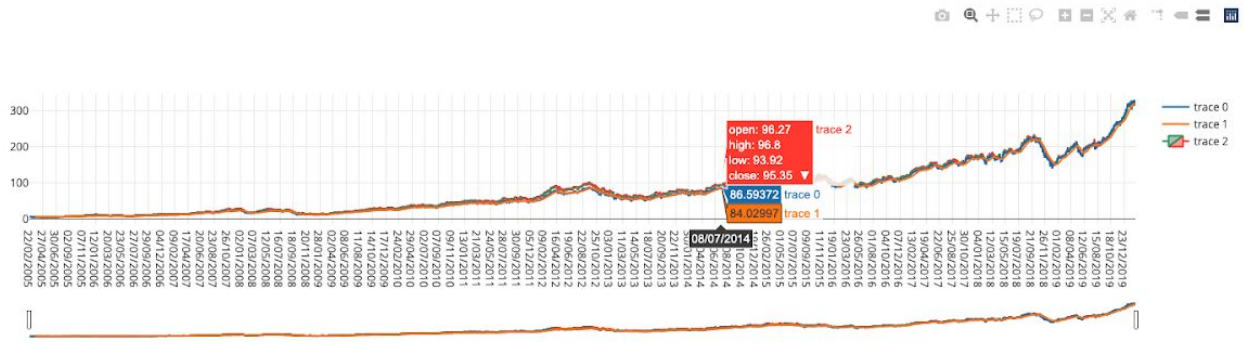
b'0.043155717017900 19'

Select company

VaR Calculator: Apple
Moving Average Period:
20
VAR Period:
10

Monte Carlo Sim:
10

Choose a service: lambda
Submit



VAR calculation

Calculate VaR 12

Submit

	Date	Open	High	Low	Close	Adj Close	Volume	MA	sell	buy	returns
0	22/02/2005	6.164286	6.307143	6.092143	6.092143	5.288228	304823400	NaN	False	False	NaN
1	23/02/2005	6.194286	6.317857	6.110714	6.302143	5.470516	336295400	NaN	False	False	0.034471
2	24/02/2005	6.320000	6.379286	6.266428	6.352143	5.513918	379757000	NaN	False	False	0.007934
3	25/02/2005	6.401429	6.422143	6.299286	6.356429	5.517640	228877600	NaN	False	False	0.000675
4	28/02/2005	6.382857	6.448571	6.280000	6.408571	5.562901	162902600	NaN	False	False	0.008203
5	01/03/2005	6.427143	6.444286	6.308571	6.357143	5.518260	117047000	NaN	False	False	-0.008025
6	02/03/2005	6.321429	6.412857	6.297143	6.302857	5.471136	114540300	NaN	False	False	-0.008540
7	03/03/2005	6.338572	6.344285	5.888571	5.970000	5.182203	352913400	NaN	False	False	-0.052810
8	04/03/2005	6.108572	6.144286	5.978571	6.115714	5.308689	189154700	NaN	False	False	0.024408
9	07/03/2005	6.114286	6.178571	6.050000	6.107143	5.301249	112658000	NaN	False	False	-0.001401
10	08/03/2005	5.985714	6.022857	5.728571	5.790000	5.025955	255362800	NaN	False	False	-0.051930
11	09/03/2005	5.662857	5.754286	5.547143	5.621428	4.879629	330616300	NaN	False	False	-0.029114
12	10/03/2005	5.647143	5.751429	5.585714	5.690000	4.939153	194277300	NaN	False	False	0.012198
13	11/03/2005	5.744286	5.798572	5.685714	5.752857	4.993715	158207700	NaN	False	False	0.011047

References

[1] https://benalexkeen.com/creating-graphs-using-flask-and-d3/
[2] https://plotly.com/python/plot-data-from-csv/
[3] https://www.w3schools.com/tags/tag_select.asp
[4] https://blog.heptanalytics.com/flask-plotly-dashboard/
[5] https://blog.quantinsti.com/calculating-value-at-risk-in-excel-python/