Developing a Multicloud API For Risk and Profitability Analysis in Trading Stratergies

Sruthi Lakshmi Vadakkapurathu Reghu

This report illustrates the development of a cloud-based API that integrates Google App Engine (GAE), AWS Lambda, and EC2 for analyzing trading strategies. The API employs Monte Carlo simulations to evaluate market patterns like "Three White Soldiers" and "Three Black Crows," helping users assess potential risks and profits. By processing large datasets, the system delivers critical financial insights, aiding in decision-making. Its architecture allows for seamless scaling across multiple cloud platforms, optimizing both resource use and costs. The design ensures that users can efficiently conduct comprehensive financial analyses with minimal manual intervention.

Introduction

Creating a cloud-based API that allows trading strategy analysis across different clouds aligning with the tenets of cloud computing as defined by NIST in SP 800-145. The platform is designed to enable developers and users make best use of all available cloud resources, thus meeting both the technical and operational requirements of a financial analytics system.

A. Developer

In NIST SP800-145	Developer uses and/or experiences
Resource Pooling	Backend tasks are handled effectively through resource pooling within GAE, AWS Lambda and EC2. This allows developers to concentrate on building and deploying the API without necessarily needing to control the underlying infrastructure directly, but focusing more on core functionality and logic of the system.
On-Demand Self-Service	Developers can deploy and scale cloud services as needed through the API, which interacts seamlessly with GAE, AWS Lambda, and EC2. This capability provides flexibility, enabling developers to adjust resource allocation based on the volume of requests and computational requirements.
Measured Service	The API monitors resource usage and costs, giving developers insights into the system's performance and efficiency. This measured approach is crucial for optimizing cloud resource utilization and managing expenses effectively.
Broad Network Access	Developers can easily integrate the API into other applications and systems given that it can be reached online through RESTful endpoints. This is why it has to be widely accessible in order to build a flexible cloud-native application working on different environments.
Rapid Elasticity	The system is designed to scale resources up or down in response to demand. Developers can specify the required level of parallel processing, and the system will automatically allocate resources, whether through AWS Lambda for serverless operations or EC2 for more resource-intensive tasks.

B. User

In NIST SP800-145	User uses and/or experiences
Resource Pooling	Users benefit from the system's ability to pool resources, which allows them to conduct complex trading analyses without needing to understand or control the infrastructure. The system's scalability ensures that user requests are processed smoothly, even during high demand.
On-Demand Self-Service	API endpoints are employed by users to interact with the system in order for them to obtain real time feedback on their analysis requests. The fact that the system is on-demand makes it possible for users to initiate analysis whenever they feel like and without going through the process of manual resource allocation.
Measured Service	Customers are charged depending on their usage of the resource, meaning they only pay for what they have used. This way of billing helps customers to manage their costs well, particularly in situations where there are regular analyses carried out.
Broad Network Access	The API is accessible to users from any device that is connected to the internet, thus guaranteeing its availability all the time and everywhere. This is of particular significance to financial analysts who have to act quickly based on most recent information.
Rapid Elasticity	Users enjoy the benefits of scalability as the system can handle variable workloads without affecting performance. When faced with either small or big data, resource allocations adjust to produce timely results.

II. Final Architecture

The system has an adaptable structure, using Google App Engine (GAE) as the primary frontend service which interacts with back-ends running either on AWS Lambda or EC2 according to user preference. The system is designed in such a way that at a given time, only one scalable service; either Lambda or EC2, will be used for computational tasks involved in analysis of trading strategies. The choice is determined by user input in order to enable customized resource allocation based on specific user requirements.

GAE as the API Frontend:

Role: Google App Engine (GAE) serves as the primary entry point for all user requests through the API. It handles user inputs and manages the flow of data to and from the backend services. GAE does not perform heavy computations but instead coordinates the tasks, delegating them to either AWS Lambda or EC2 based on the user's preferences and the nature of the request.

Interactions with AWS Lambda:

When the user selects AWS Lambda for execution, GAE forwards the computational tasks, such as running Monte Carlo simulations, to Lambda functions.

Inputs: The API endpoint /analyse gathers user-defined parameters, including h (history length), d (number of Monte Carlo shots), t (type of signal, e.g., buy or sell), p (days after the signal to check profit/loss) and trading signals. These inputs are transmitted to Lambda, where the actual computations occur.

Outputs: Lambda processes these inputs and returns outputs such as the 95% and 99% Value at Risk (VaR) values, profit/loss calculations, and execution details (e.g., time taken, memory used) back to GAE, which then relays this information to the user through appropriate API endpoints like /get_sig_vars9599 and /get_tot_profit_loss.AWS

Interactions with AWS EC2:

Service Involvement: Alternatively, if the user chooses EC2, GAE can initiate tasks on EC2 instances. This ideally comes handy when the user requests a higher d value, indicating the need for more extensive data processing that exceeds Lambda's capabilities.

Inputs: Similar to the Lambda interaction, GAE sends the same set of user-defined parameters (h, d, t, p, trading signals) to the EC2 instances. However, EC2 is capable of handling larger datasets and more complex simulations.

Outputs: The results, including detailed risk assessments and profit/loss calculations, are returned from EC2 to GAE. GAE then provides this data to the user via endpoints like /get_avg_vars9599 and /get_chart_url.

Handling Warm-Up and Resource Allocation:

Warm-Up: The /warmup endpoint is crucial as it prepares the selected service (Lambda or EC2) to handle the anticipated load based on the r (number of parallel executions) specified by the user. This involves allocating and initializing the necessary resources to ensure that the system can perform the required tasks without delay.

Data Fetching and Signal Processing:

Data regarding historical stock quotes are obtained from Yahoo Finance after which they are transformed into trading signals. These signals are vital for further risk assessment which guides Monte Carlo simulators accordingly and is transmitted to lambda and EC2 along with other input values.

Cost and Time Calculation:

After processing, the system gathers data on resource usage (e.g., execution time, memory consumption) and transmits this information back to GAE, which then allows users to retrieve it through the /get_warmup_cost and /get_time_cost endpoints. These outputs help users understand the implications of their resource choices in terms of both performance and cost.

Resource Termination:

Once analysis is completed active resources not required anymore such as either Lambda functions or EC2 instances get terminated so as to avoid unnecessary charges.

Summary of Input/Output Handling

Inputs: Parameters such as h, d, t, p and r define the limits of the analysis in terms of its scope and scale. The transmission is either directed to Lambda or EC2 through GAE.

Outputs: These include calculated risk metrics (VaR95, VaR99), loss/profit values, trading details and cost information which are emanating from the service chosen and sent back to GAE, which then sends them to the user via related API endpoints.

This architecture ensures that the system remains scalable, efficient and user directed enabling complex financial analysis to be done in a cloud native environment while efficiently managing resources based on users' preferences.

III. Satisfaction of Requirements

TABLE I. SATISFACTION OF REQUIREMENTS/ENDPOINTS AND CODE USE/CREATION

Endpoints	/warmup /resources_ready /get_warmup_cost /get_endpoints /analyse /get_sig_vars9599 /get_avg_vars9599 /get_sig_profit_loss /get_tot_profit_loss /get_chart_url /get_time_cost /reset /terminate /resources_terminated

IV. Results

TABLE II. Results

r	d	Warmupcost ($)	Exectioncost ($)	Exection time (ms)
3	10000	0.0000006	0.000018	9.87
3	20000	0.0000006	0.000037	19.26
3	30000	0.0000006	0.000058	28.97
5	10000	0.000001	0.000018	9.97
5	20000	0.000001	0.000038	19.70
5	30000	0.000001	0.000058	29.24

The results table illustrates that as the value of "r" (the number of parallel computations) increases, the system needs to provision more resources to handle the additional computational load, leading to higher warm-up costs. Similarly, as the value of "d") increases, the system has to perform more computations, which naturally extends the execution time.

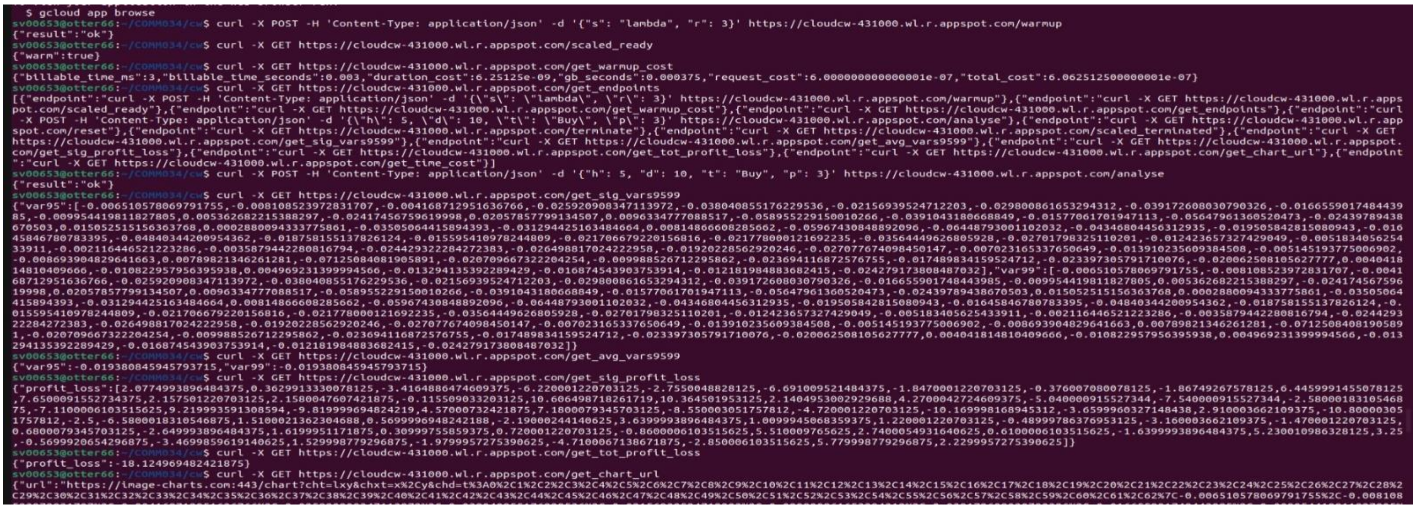

FIGURE 1 : Terminal showing the results of analysis from lambda

V. Costs

The cost analysis considers the real-world pricing for GAE, AWS Lambda, and EC2 services, assuming that the free tier limits have been exhausted. The costs are calculated based on different usage scenarios, reflecting the system's potential expenses under various load conditions.

Cost Breakdown: GAE: $0.04 per instance hour. Used primarily for managing the API and coordinating tasks between Lambda and EC2.

AWS Lambda: $0.00001667 per GB-second. Costs are based on the duration of the function execution and the amount of memory allocated.

AWS EC2: $0.0116 per instance hour (for t2.micro instances). Used for handling large datasets that require more processing power than Lambda can efficiently provide.

Example Usage Scenarios:
Moderate Usage: 10 users performing daily analyses, each using 25,000 Monte Carlo shots (d) with r = 3:
Heavy Usage: 50 users performing daily analyses, each using 50,000 Monte Carlo shots (d) with r = 5:

GAE: 2 instance hours per day per user = $120.00/month.

AWS Lambda: 1 minute of execution time per analysis = $7.50/month.

AWS EC2: 2 hours per analysis = $348.00/month.

Total Cost: $475.50/month.

Conclusion: The system is cost-effective under moderate usage scenarios, with most costs fitting within the free tier limits. However, under heavy usage, the costs can increase significantly, particularly with the use of EC2 instances. Users must consider their usage patterns and adjust the scale parameters to manage costs effectively.

References

[1] University of Surrey, "COMM034 Cloud Computing Coursework Description 2023-24," Department of Computer Science, University of Surrey, 2023.

[2] University of Surrey, "COMM034 Cloud Computing Lab materials," Department of Computer Science, University of Surrey, 2023.

[3] “Cloud Application Architectures : Building Applications and Infrastructure in the Cloud”, George Reese