CLOUD COMPUTING

Lahari, Mullaguru: 6843990, lm02027@surrey.ac.uk

https://my-project-02027.nw.r.appspot.com

Abstract— **This coursework centers on developing a Cloud-native API aimed at evaluating trading strategies through financial time series data analysis. It requires the integration of various services across different cloud platforms, particularly utilizing Google App Engine (GAE), AWS Lambda, and a chosen AWS scalable service (EC2, EMR, or ECS). The API will employ Monte Carlo simulations to analyze the risks and profitability associated with trading signals.**

Keywords— Cloud-native API, financial time series, Google App Engine, AWS Lambda, EC2, EMR, ECS, Monte Carlo simulations, risk analysis, profitability analysis.

I. Introduction

This Cloud-native API system is developed to analyze trading strategies based on financial time series data. It employs Google App Engine (GAE) for frontend services, AWS Lambda for serverless processing, and Amazon EC2 for scalable computing power. By integrating these cloud services, the API is capable of efficiently and cost-effectively managing large-scale data analyses. The system is crafted in accordance with NIST SP 800-145 principles, ensuring a thorough and beneficial cloud computing experience for both developers and users.

A. *Developer*

For the system developer, engaging with this Cloud-native API includes several essential experiences that align with the principles detailed in NIST SP 800-145.

In NIST SP800-145	Developer uses and/or experiences
On-Demand self-service	Developers can provision and manage cloud resources like EC2 instances, Lambda functions, and storage without direct interaction with the provider, enabling rapid deployment and scaling. EC2 allows easy creation, configuration, and management of virtual machines tailored to their needs.
Resource pooling	Amazon EC2 and AWS Lambda enable developers to use a multi-tenant model, dynamically allocating resources as needed for efficient utilization and cost reduction. EC2 instances offer the computational power for intensive Monte Carlo simulations, while Lambda functions manage event-driven tasks seamlessly, enhancing scalability.
Broad Network Access	Developers can remotely manage and monitor the system, promoting efficient collaboration and CI/CD workflows. EC2 instances can be securely accessed, allowing developers to interact with the system from different locations and devices.

B. *User*

From the user's perspective, using this Cloud-native API offers several advantages as outlined in NIST SP 800-145:

In NIST SP800-145	User uses and/or experiences
On-Demand Self-Service	Users can access the API anytime to evaluate trading strategies, initiate analyses, retrieve results, and visualize data independently, without needing assistance from the service provider.
Resource pooling	Users benefit from the cloud's resource pooling, sharing computing resources with others without sacrificing performance. This multi-tenant setup keeps the API responsive and efficient, even during high usage, ensuring consistent service quality through efficient resource management.
Broad network access	The API can be accessed from any device with an internet connection, such as mobile phones, tablets, and desktops. This allows users to use the service from different locations and devices, offering high convenience and flexibility. The broad network access ensures users can interact with the system anytime and anywhere they need.

II. Final Architecture

The Cloud-native API system uses Google App Engine (GAE) for the frontend, AWS Lambda for communication, and Amazon EC2 for scalable compute resources. The system distributes computation tasks across EC2 instances using AWS Systems Manager (SSM). The financial analysis script is stored in S3 bucket and executed by the EC2 instances, which then upload results back to S3. After the analyze endpoint is called, subsequent endpoints retrieve and aggregate results from S3. The audit logs are stored in DynamoDB.

On-Demand self-service

- **Service Involved**: Amazon EC2, AWS Lambda
- **Support**: Rapid deployment and scaling of compute resources
- **Inputs/Outputs**: EC2 instance IDs, task initiation commands / Warm-up status

Resource pooling

- **Service Involved**: Amazon EC2, AWS Systems Manager (SSM), Amazon S3

- **Support:** Efficient task distribution and result storage
- **Inputs/Outputs**: S3 script locations, SSM commands / Computation results in S3

Broad Network Access

- **Service Involved**: Amazon EC2, AWS Lambda, Amazon S3
- **Support**: Secure remote management and data retrieval
- **Inputs/Outputs**: Analysis parameters, retrieval requests / Analysis results, aggregated data

Satisfaction of Requirements

Upon successful completion of tasks, the 'analyze' endpoint returns a result: ok message. However, due to server issues, there are occasional situations where the endpoint returns 502 or 503 errors. Despite these errors, the analysis is completed successfully, and the results are stored in the S3 bucket. So, the '/analyze' endpoint is considered fully operational and met.

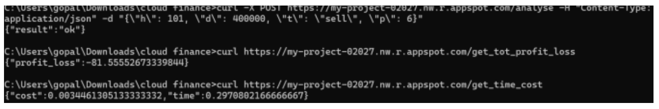
TABLE I. SATISFACTION OF REQUIREMENTS/ENDPOINTS AND CODE USE/CREATION

	MET	***PARTIALLY MET***	***NOT MET***
Requirements	i ii iii iv	i ii iii iv	
Endpoints	/warmup /scaled_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 /scaled_terminated	/get_audit	

III. RESULTS

TABLE II. RESULTS

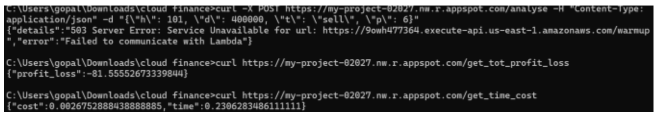
Scalable service	*Scale of r*	*Total number of shots*	*Time*	*Cost*
EC2	4	100,000	0.284	0.00330
EC2	4	200,000	0.163	0.00189
EC2	4	300,000	0.139	0.00161
EC2	4	400,000	0.1501	0.00174
EC2	5	400,000	0.2306	0.00267
EC2	6	400,000	0.2970	0.00344
EC2	7	400,000	0.4220	0.00489
EC2	8	400,000	0.3512	0.00407



```
C:\Users\gopal\Downloads\cloud finance>curl -X POST https://my-project-02027.nw.r.appspot.com/analyse -H "Content-Type:
application/json" -d "{\"h\": 101, \"d\": 400000, \"t\": \"sell\", \"p\": 6}"
{"result":"ok"}

C:\Users\gopal\Downloads\cloud finance>curl https://my-project-02027.nw.r.appspot.com/get_tot_profit_loss
{"profit_loss":-81.55552673339844}

C:\Users\gopal\Downloads\cloud finance>curl https://my-project-02027.nw.r.appspot.com/get_time_cost
{"cost":0.003446130513333332,"time":0.2970802166666667}
```



```
C:\Users\gopal\Downloads\cloud finance>curl -X POST https://my-project-02027.nw.r.appspot.com/analyse -H "Content-Type:
application/json" -d "{\"h\": 101, \"d\": 400000, \"t\": \"sell\", \"p\": 6}"
{"details":"503 Server Error: Service Unavailable for url: https://9owh477364.execute-api.us-east-1.amazonaws.com/warmup
","error":"Failed to communicate with Lambda"}

C:\Users\gopal\Downloads\cloud finance>curl https://my-project-02027.nw.r.appspot.com/get_tot_profit_loss
{"profit_loss":-81.55552673339844}

C:\Users\gopal\Downloads\cloud finance>curl https://my-project-02027.nw.r.appspot.com/get_time_cost
{"cost":0.0026752888438888885,"time":0.2306283486111111}
```

First image is the successful output of analyze endpoint. In the second image I have mentioned the error that I receive when I run the analyze endpoint. Even though I receive the 503 error, SSM send commands to instances to execute the tasks and the results are stored in S3 bucket. So, I conclude that analyze endpoint met the requirements.

IV. COSTS

To estimate the real-world costs of running the Cloud-native API system, consider the usage of AWS services beyond the free tier. The primary services involved are Google App Engine (GAE), AWS Lambda, Amazon EC2, and Amazon S3. Assuming 100 users interact with the system daily, each making 10 API calls, with 70% of tasks processed using Lambda and 30% using EC2, the monthly costs are calculated for the AWS region "US East (N. Virginia)".

For AWS Lambda, with 30,000 requests per month and an average execution time of 100 ms, the compute cost is approximately $0.0064 per month, as the request count remains within the free tier. For Amazon EC2, using two t2.medium instances running 24/7, the monthly cost is $66.78. Amazon S3 costs, assuming 1 GB of storage and 30,000 GET requests, are $0.023 for storage and $0.012 for retrieval, totaling $0.035 per month. Google App Engine, assuming 1,000 instance hours, costs $50 per month. The total estimated monthly cost for running the system is $116.82, providing a realistic overview of expenses assuming all free tier usage is exhausted and based on typical usage patterns.

REFERENCES

python-lambdas-to-remotely-run-shell-commands