Designing a Native AWS System to Classify Server Reboots in Real Time

A Technical Report submitted to the Department of Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Yashwanth Kolli

Fall 2023

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Kathryn A. Neeley, Associate Professor of STS, Department of Engineering and Society

ABSTRACT

Amazon was unable to classify EC2 server reboots in real time. As an intern, I solved this problem by creating a fully AWS-based solution to process server logs from end-to- end and store the classification result. I began the design by placing the logs from server reboots onto an S3 bucket, which triggered a lambda function, which stored the reboot information and directed the event to another lambda function based on the primary reboot reason. This final lambda function performed a full detailed classification based on the logs using regex expressions, and stored the result. We were able to reduce the end-to-end latency of classifying dirty server reboots from 15 minutes to a few seconds. Next steps for this project include addressing scalability, making sure that the design can handle a high volume of reboots.

1. INTRODUCTION

Amazon EC2 is a web service by AWS that provides resizable compute capacity in the cloud. In simpler terms, an EC2 server is a virtual machine that one can rent from AWS to run various types of applications and workloads. These EC2 servers need to be constantly monitored and analyzed, making sure that their health is maintained. During my internship, this maintenance was done by my team, EC2 Health Analytics.

EC2 servers can dirty reboot, which happens when a computer reboots or restarts abruptly, without proper shutdown. Prior to my arrival

on the team, dirty server reboots were being classified in 15-minute batches. Any time a server would reboot, it would accumulate in an S3 bucket, which would be processed by a Distributed Job Scheduler in 15-minute increments. Classifying these reboots in real time proved to be much more advantageous, since if a high volume of reboots occurred for a particular reason, the issue could be flagged immediately.

2. RELATED WORKS

While EC2 provides the computational power to handle big data, doing so in real- time requires a detailed architectural approach. Marz and Warren (2015) emphasized the need for systems that can manage large amounts of data while providing insights in real-time. They detail how Lambda Functions can be used to achieve this by using stream processing. The flexibility of AWS services like Lambda functions and S3 buckets make real-time log processing a real potential, as developers can create powerful data processing systems.

The official EC2 documentation (Amazon.com, n.d.?) does not just highlight its functionality, but also outlines best practices and challenges in managing applications on their servers. Marz and Warren also dive into the challenges developers might face in the real world, offering best practices to mitigate them. Examples of such best practices include Regularly backing up your EC2 instances and data to Amazon S3, testing recovery procedures, implementing consistent tagging, and more.

This emphasis on not just the advantages but also on potential pitfalls and their solutions makes both resources valuable for this project. EC2's documentation provides insights into the infrastructure side of things, while Marz and Warren give a comprehensive view of the data side. When looked at together, they provide a holistic view of the infrastructure and teach us how to data handling using best practices.

3. PROCESS DESIGN

The workflow for this project started when an EC2 server rebooted without human interaction. The team had an internal infrastructure set up that would add the server log of an EC2 reboot to an S3 bucket. This server log would contain information such as the server id, log information, time of incident, etc.

I configured an AWS lambda function to be triggered to run whenever a new item was added to the bucket. I wrote this lambda function in GoLang, the primary language used by the team. GoLang is similar to C/C++ in which that it runs fast and accounts for efficient memory management. This lambda function would store the incident on an Amazon Aurora Database. It would store the information relating to the incident, including a unique id, the server id, and the primary reboot reason. This primary reboot reason was determined based on predefined Regex expressions defined by the team.

I used an Amazon Aurora Database because it offered many advantages. First, it offered extremely high performance compared to other options. For example, it has 2-5x faster throughput than MySQL and PostgresSQL databases. Since we needed our system to be

as real time as possible, it was important to account for this speed. Second, Amazon Aurora is also highly scalable. It supports both vertical and horizontal scaling, as the database can easily be resized to accommodate changing workloads. Read replicas can also be created from the original instance, also improving scaling functionality.

After the storage onto the database, the first lambda function would trigger another lambda function based on the primary reason. There were multiple primary reasons; therefore, the traffic influx was balanced by redirecting the classification of the secondary and tertiary reasons based on the primary reason.

This second lambda function would again use Regex expressions to classify the secondary and tertiary reasons for the reboot occurrence. The results would be stored onto the Amazon Aurora Database, ending the workflow.

This system was designed to handle a high volume of reboots at any point, as lambda functions can have up to 1000 concurrent bursts, allowing us to catch all reboots. This was beneficial for production deployment, as a there was little reason to worry about missed classifications. Specific IAM roles and policies were also put into place to ensure that only individuals on the team had admin access to these different components.

Last, I wrote extensive test cases to ensure that the written code was functional. This was done through mocking the AWS services and calls. There was a 92% code coverage with the written test cases, surpassing the original goal of 80%.

4. RESULTS

The result of this project was the reduction in the time needed to classify a reboot. Before this implementation, reboots were accumulated and classified in 15-minute batches. After the implementation of this project, the average reboot time reduced to less than five seconds.

Being able to classify these reboots in real time allowed for dynamic and up-to-date server analytics. It also allowed for our team to raise issues quickly if we noticed a high volume for a specific reboot reason.

5. CONCLUSION

The project I created holds a lot of value in the context of real-time EC2 server management within the AWS ecosystem. By designing and implementing a native AWS system for classifying server reboots in real- time, the team was able to address a critical need for timely insights into EC2 server health. This project has applications in server analytics, real-time issue identification, and overall efficiency. The benefits of reducing latency and providing dynamic server analytics allow users to make informed decisions and respond quickly to issues.

6. FUTURE WORK

To further enhance the project, several avenues can be explored. First, we need to focus on scalability to accommodate different levels of reboot activity. Additionally, we should look into performance optimization to continue to improve the system's responsiveness. Also, advanced classification methods, such as machine learning, can be used to improve accuracy. Last, the project's adaptation for potential uses beyond EC2 servers can be explored to maximize its value in different AWS scenarios.

REFERENCES

Amazon.com. (n.d.-b) What is Amazon EC2? - amazon elastic compute cloud. https:// docs.aws.amazon.com/AWSEC2/late st/UserGuide/concepts.html

Marz, N., & Warren, J. (2015). Big Data: Principles and best practices of scalable realtime data systems. Manning Publications Co.