Visualizing AWS EC2 Unsellable Server Instances Using the Elastic Stack

A Technical Report submitted to the Department of Computer Science

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

Ryan Robinson

Spring, 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

Rosanne Vrugtman, Department of Computer Science

Visualizing AWS EC2 Unsellable Server Instances Using the Elastic Stack

CS4991 Capstone Report, 2023

Ryan Robinson
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
rpr6at@virginia.edu

Abstract

AWS EC2 was built iteratively and relatively quickly to get ahead of other competing cloud providers. Because of this, it is possible for servers to go "missing," becoming unsellable, which cuts into a significant minority (billions of dollars) of Amazon AWS profits per year. In order to hunt down and recover missing servers, the data on these servers needs to be compiled in an easily digestible format. To do this, my team and I compiled the data from all EC2 instances during their lifecycle. We then processed and displayed the data using the Elastic Stack hosted on EC2 Instances. Because the generated data was easily shareable and displayable, our team was able to demonstrate the problem more clearly to higher AWS executives who provided us with necessary resources to recover more servers. During the last two weeks of my internship, The tool helped recover tens of thousands of dollars worth of servers. To continue the development of this tool, I would create deployment scripts so that the tool could be more easily maintained without having to have any knowledge of the Elastic Stack.

1. Introduction

To customers, a service as widely used as AWS may appear to function flawlessly. Behind the scenes, however, there are a variety of issues and challenges at every layer. One of these core issues is the "unsellable servers" problem. The AWS cloud is made up of millions of physical servers in hundreds of data centers around the world. With this many servers, it is inevitable that a certain percentage of them will fail due to any number of issues. AWS has, of course, accounted for this.

When a server goes down, the data is transferred to another server, and the failed hardware is put through a maintenance process that in most cases will restore the server to its original state. The server space can then be resold to customers. The "unsellable" problem arises when the maintenance process fails to recover the server. When this happens, the server becomes "unsellable," and is now taking up resources without generating any profit. There are tens of thousands of reasons why this could occur, and that is what makes the problem so difficult to diagnose and solve. It is like a bunch of tiny holes leaking server space from the cloud. In addition, these points of failure are quite varied, and each category of failure will require a specific team to manually recover the lost server space.

My team's responsibility was to provide an overhead view of all Unsellable Servers in the EC2 fleet, and work towards categorizing all failures to their respective teams so the servers could be recovered. To do this, we built software that classified

every tracked unsellable server once a day and published the data in a CSV. We then had to manually go through all our classifications and develop patterns to help us determine which classifications belonged to which teams. Unfortunately, this raw data was difficult to sort through, and even more difficult to present to other teams or higher executives. Our team desperately needed a more robust interface that would allow us to complete these tasks more easily.

2. Related Works

AWS was originally conceived in 2006 when executives at Amazon realized they could sell the server infrastructure they had created to service their website (Mille, 2016). While this allowed them to be the first to market, it also meant that some of their core infrastructure is older and not as waterproof as other server providers who built from a clean slate. This initial style of development] is a contributor to the Unsellable problem.

Elasticsearch, as described by Namaug (2022), has many advantages over other data storage options, allowing for the search and retrieval of large amounts of data in real time. This allows for dynamic graphing and display of very large data sets, which was exactly what I needed to do with the Unsellable data. This is one of the reasons I chose the Elastic stack to implement my solution.

3. Process Design

To solve this problem, I followed a strict design plan.

3.1 Review of Current Architecture

Before the start of my project, my team had already built a working classification engine. Every day, the engine would take as input all the data on known unsellable servers, and output a CSV file giving each server a variety of different classifications. This spreadsheet was then published on a data storage service. These spreadsheets were large, containing hundreds of thousands of rows. In order to extract aggregate data from these spreadsheets so we could identify trends, our team needed to construct advanced SQL queries, run them over the dataset, and manually graph the results of the query. Oftentimes the query outputs were quite large, so creating these graphs was time consuming and tedious. In addition, if any other team wanted to analyze our data, they were required to either ask us to do it for them or construct the advanced SQL queries themselves. Although our data was very useful for teams trying to identify unsellable servers in their domain, it often went unused because it was so difficult to interface with.

3.2 Requirements

The direction of the project was dictated by the client needs and limitations of the system.

3.2.1 Client Needs

My project had a variety of different clients, each with different priorities, so it was important to balance all of them to create the most effective solution. The first client was my own team. First and foremost, it was our job to analyze the data. We had to both ensure we were classifying the unsellable servers as accurately as possible, as well as identify new trends in rising unsellable servers and report them. To do this more efficiently, my team needed an interface that:

A. Automatically grabbed the unsellable data from the CSV daily snapshots
B. Processed the data so it could be queried
C. Provided a simple, not cluttered interface over the data that allowed easy filtering of the based on all defined attributes in the dataset
D. Displayed the query results as a trend line over time

By accomplishing these goals, I would be able to satisfy my first client. My second clients were the upper managers with leadership roles in EC2 and the engineers that needed to present the data to them. These clients were not as likely to analyze the data themselves. They needed an interface that:

A. Wasn't over cluttered or distracting. Easy to understand just by looking at it.
B. Could be easily shared between devices and tweaked. During a meeting, if someone wants to view the presented data on their device, they should not need to waste time setting up specific filter presets.

My final clients were the other teams that had actual ownership of the unsellable servers we were tracking down. They had the same requirements as before, but in addition they needed:

A. A familiar interface, This would increase the likelihood of widespread adoption.
B. Easy to access, not locked behind a bunch of security walls specific to my team.

My project was built with all these requirements in mind.

3.2.2 System Limitations

Given that I had access to all Amazon's resources to build my project, technically I could have approached it in any way I wanted. My one major restriction was that any solution I proposed basically needed to use AWS and Amazon infrastructure. Even if I preferred another service, there was no funding allocated for any other service.

3.3 Final Solution

The solution I eventually implemented addressed the requirements by using elastic stack services hosted on AWS EC2 instances. The elastic stack consists of a database, called elasticsearch, a pipelining tool, called Logstash, and a dashboarding tool, called Kibana. I chose this approach over a custom interface because it would be easier for my team to maintain. While all these tools were useful, the main reason for landing on this solution was because of Kibana.

Many people in the organization were familiar with Kibana and enjoyed using it. This meant that it was much more likely a tool based on it would be widely adopted. In addition to fulfilling all the desired functionality, the interface also allowed for much customizability from the user, as they could add or subtract filters depending on their specific use case. For example, if they only needed to use two filters, they could add those to the interface and not have the clutter of 33 other useless filters. Most importantly, Kibana addressed the main issue other dashboarding tools had with sharing. Once a user had added the desired filters to the interface and set their values, all they needed to do to share their results was copy the URL and send it. In addition to being useful in meetings, this allowed for the tool to be spread quickly throughout the organization.

3.4 Key Components

After getting my solution approved and implementing it, the final architecture was as follows:

A. When the daily CSV data was uploaded to the data bucket, a switch was triggered
B. This signal was sent to a Logstash data pipeline hosted on an EC2 server instance. When it received the signal, it began importing the data and processing it into a format digestible by the Elasticsearch database.
C. All data was then imported into the Elasticsearch database
D. Once the data had been successfully cataloged, it could then be accessed by a Kibana dashboard
E. The built Kibana dashboard consisted of several graphs graphing trend different Unsellable server trend lines over time. To filter the data based on server attributes, users could add and subtract filters based on their need. A few very commonly used filters were always included in the dashboard for the sake of convenience
F. To access the dashboard, users needed to have the valid AWS certifications for our organization. This was required for security reasons.

3.5 Challenges

While in the end this interface was functional and used throughout the organization, there were some major challenges getting it to work. For one, gathering the requirements from all the stakeholders was difficult, as many people had different ideas about where the project should go. Compiling them all into one concise vision that everyone was happy with was a difficult task.

A more technical challenge was that the daily CSV data was not originally accessible by Logstash. I had to change the classification engine to publish the data to a data bucket that the pipeline could retrieve data from, which required me to make changes to our teams' main project (the classification engine). This comes with a lot more yellow tape than when a project is built from scratch.

4. Results

After the product was released, our team saw immediate impact. Within a week of release, the tool was used to demonstrate problems related to unsellables in meetings with higher management. The interface was circulated among stakeholders very quickly and many teams began to use it.

During one meeting that I was a part of, participants were able to identify a problem where a subset of the unsellable servers had been rapidly increasing since a specific change was pushed. Several thousand total servers were affected, and when I left my internship the team that owned them had already begun to make headway in recovering all of them.

While it is difficult for me to know now how many servers the tool has helped recover, given the upward trend I saw in the final two weeks in my internship and an email I received in November from my boss telling me that the tool was still in use, It would be safe to assume it has aided in the recovery of hundreds of thousands if not millions of dollars worth of AWS hardware.

5. Conclusion

Data is only as useful as it is accessible. My project allowed other teams and managers a very clear window into the work our team was doing, and facilitated the sharing of our data. The interface greatly increased the productivity of our team and our usefulness to other teams. Due to the shareability of the interface, our teams' visibility among the organization has also greatly increased, giving us access to more resources to continue our work finding and driving down the number of unsellable servers. Overall, the interface was the capstone of a major project for our team. While I had worked with many of the tools used before, I had never developed a project with the reach and user base of this interface. I learned how to balance the needs of many stakeholders and deliver results in a timely and efficient manner.

6. Future Work

The next phase of this project would be to build automated and general deployment scripts. While the project is completely functional for its use case, if a similar interface was required, our team would be forced to manually recreate the architecture that I built for their use case. While this is not particularly time consuming or difficult, there is room for error and confusion, and it is a waste of resources if it has already been done by another engineer. For this reason, if I was given more time or I was to build this project from the ground up again, I would prioritize using automated deployment scripts to reduce the complexity of implementation for future engineers.

References

Miller, R. (2016, July 2). How AWS came to be. TechCrunch. Retrieved February 24, 2023, from https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/

Namuag, P. (2022, August 2). What is Elasticsearch and why use it? Severalnines. Retrieved February 24, 2023, from https://severalnines.com/blog/what-is-elasticsearch-and-why-use-it/