Amazon Grocery: Automating the Retrieval Process

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

David Tran

Spring, 2022

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

Amazon Grocery: Automating the Retrieval Process

CS4991 Capstone Report, 2022
David Tran
Computer Science
University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
dqt5vt@virginia.edu

Abstract

Amazon is innovating within the grocery space, through the acquisition of companies such as Whole Foods, and rolling out grocery delivery services such as Amazon Fresh. Amazon's Grocery Automated Storage and Retrieval System (GRASRS) is commonly called Grocery ASRS. The Amazon Grocery Retrieval Team is currently working on Amazon Fresh and is developing a more efficient way to retrieve groceries from orders by using robots to automate the picking and stowing processes. They have run into many issues with debugging and system failures. Since there was no efficient way to aggregate error logs into a common repository, my project as a GRASRS intern was to create an efficient way for software engineers to access error logs to address system failures. I used AWS CloudWatch to retrieve error logs across all GRASRS development accounts and display them on a full-stack website for engineers to see. This resulted in a 50% speedup in debugging times, and greatly expedited turnaround time for system failures. In the future, the GRASRS team plans to add more functionality to the full-stack website I created, by introducing tools to visualize data and monitor the state of the system.

1. Introduction

As the largest e-commerce company in the world, Amazon has a large number of fulfillment centers across the country in order to meet online demand for products. In order to deliver products to its customers in a timely manner, Amazon attempts to streamline the end-to-end process of placing an order to delivering the order to the customer as much as possible, in order to save on costs and time. Each of Amazon's fulfillment centers serves a different purpose. Some store general goods, while others primarily store technological goods such as mobile phones, laptops, and electronics.

Recently, Amazon has been trying to break into the grocery industry and it introduced a new service called Amazon Fresh in 2007. Amazon Fresh is a grocery retailer primarily dealing with grocery delivery to Amazon Prime customers, although Amazon Fresh has a small number of physical stores throughout the United States. Several fulfillment centers across the country contain exclusively grocery products and produce. Since Amazon Fresh is a relatively new subsidiary of Amazon, much of the end-to-end process of delivering an order from the grocery fulfillment centers is manual, with humans picking the grocery products from the fulfillment centers, storing them into bags, then stowing the bags into trucks for delivery to the customer. This is a relatively expensive and slow process that could be automated and streamlined to save money and time.

The problem with grocery products, and the reasons Amazon's current automation practices cannot be applied to the grocery space are twofold. The first reason is that grocery products are perishable and have to be stored in different conditions. For example, a frozen product ~~is~~ picked and stored in a bag for too long it could expire before being delivered to the customer. The second reason is that not all products are the same. The prime example of this is fresh produce such as meat and vegetables, which often vary in freshness and ripeness. Customers typically expect their products to be of the highest quality, and current automation practices do not discriminate between products.

The GRASRS team was formed at Amazon in order to help Amazon automate the retrieval process in Amazon Fresh and grocery fulfillment centers. Since GRASRS is less than a year old, operators run into many issues with their beta system. Currently, there is no efficient way to debug these issues since debugging logs are spread across a plethora of accounts. As an intern, I was tasked with creating a full-stack website that enables team members to access all debugging logs in a common resource in the case of a system failure.

2. Related Works

An automated storage and retrieval system (ASRS) is a system used to store and retrieve products in a distribution environment, in this case, Amazon's grocery fulfillment centers. Roodbergen and Vis (2009) surveyed ASRS's, explaining their advantages over non-automated systems. ASRS's have several advantages over non-automated systems. ASRS's save money on human labor costs, since the storage and retrieval process is done by robots. They are also more accurate, since humans are inherently imperfect, and are bound to make mistakes. However, the biggest downside with ASRS's is the initial investment needed to create an ASRS in a fulfillment center. They can cost tens of millions of dollars, and must be carefully planned out to reap the benefits of an ASRS. In the case of groceries, ASRS's cannot be automatically applied, since groceries are perishable and have different freshness levels and storage requirements. This was another challenge we had to consider in the design of our solution.

Li and Xu (2014) address the process of relaying order information to the ASRS, also known as the Internet of things (IoT). The robots must be able to receive order information from another server in order to know what to retrieve and store. The robot must also be able to send information back to the servers in order to update inventory and item statuses. This is a major part of what GRASRS is doing. Converting item information into a machine-digestible message is an extremely challenging task, one that Amazon is currently developing and refining.

3. Project Design

The project that GRASRS decided upon was an operational tool to help in debugging issues in the ASRS, as well as to other tools to streamline development. Specifically, my project was aggregating error logs for system failures to streamline debugging for on-call engineers. The main service used in the project, which contained the error logs that GRASRS needed, was AWS CloudWatch, a service in AWS that collects real time metrics, data, and logs for a particular service, then aggregates them into a single repository for viewing. The GRASRS team connected their systems to CloudWatch, meaning if the system got any errors, they would be recorded in CloudWatch.

3.1. Backend Architecture

The GRASRS team decided upon the backend infrastructure of the project. For convenience purposes, and because all of Amazon Robotics was transitioning to the cloud, the decision was made for the project to be developed using AWS services. Shown in Figure 1 below is the architecture of the backend of my project.

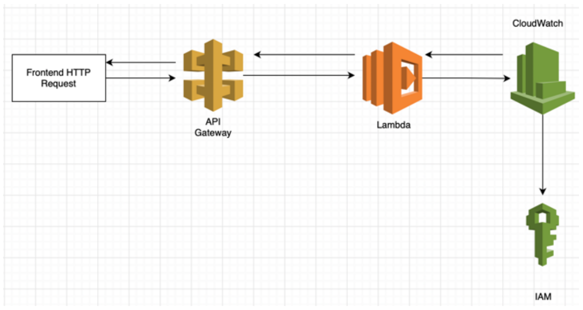

Figure 1: Backend AWS Architecture for GRASRS Operational Website

The user submits a request for certain error logs, and that HTTP request is forwarded to a REST API created through AWS API Gateway. API Gateway is a service that allows developers to create customizable APIs with support for monitoring and scaling. I used API Gateway to create an API that would accept HTTP GET requests.

Once the REST API receives a request, it will transform the request and forward it to Lambda through query parameters. Lambda is an event-driven service that allows users to run code in response to an event. Once the Lambda receives a request from the REST API, it will query the CloudWatch error logs and return the CloudWatch logs the user requested. For security purposes, CloudWatch will authenticate the user, since these error logs contain private information about the system. After the error logs are retrieved, they are sent back to the REST API to return to the frontend for the user to see.

3.2. Frontend Architecture

The frontend architecture consists of a React application that takes in request inputs, such as which GRASRS development account to query, the timestamp, the name, the log group, and the log streams. To preface, all CloudWatch error logs are contained within a log stream, which are a sequence of logs strung together. All log streams are contained within a particular log group, which is a sequence of log streams, to allow for easier querying. The application takes these query parameters and sends a GET request to the REST API. The REST API returns the result in JSON to the frontend, which is then parsed and displayed in table format for the user to see.

4. Results

The operational tool I created for GRASRS resulted in significant speedup times for accessing error logs. When accessing error logs from a single account, compared to the old system of accessing logs, log access times were about 30% faster compared to accessing CloudWatch logs from the AWS console. When accessing logs across multiple accounts, log access times were about 50% faster, a significant speedup. This is due to the fact that when switching between accounts to access CloudWatch logs, users must log out and log back into the new account, a tedious process that slows down access times.

5. Conclusion

The intentions of my project with the GRASRS team were twofold. The first reason GRASRS proposed the project was to help speed up debugging times by streamlining the process of accessing Cloudwatch logs across all of their development accounts. Doing so allows GRASRS to pinpoint system errors in the correct account, saving critical time.

The second reason was to help lay a foundational base for a multi-purpose operational tool for GRASRS to use as its scope expanded. Streamlining the debugging process through Cloudwatch would be the first tool, with more to come. My project allowed members of GRASRS to debug issues much more quickly, saving critical time.

6. Future Work

The intention of the GRASRS operational website was to be a website that contains multiple tools, not just a Cloudwatch log aggregator. With this in mind, the code for the project was kept as modular as possible, to allow for extensibility in the future. The GRASRS team has multiple pain points other than inefficient debugging. There are a plethora of other issues, such as system visualization, data aggregation, and others that are plaguing the team. GRASRS made it clear that future engineers and interns would be expanding my project by adding other features that would help the team's development process.

7. References

1. Roodbergen, J. 2009. A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, V.194,Issue2, https://www.sciencedirect.com/science/article/pii/S0377221708001598
2. Li, S., Da Xu, L. 2014. The internet of things: a survey. *Information Systems Frontiers*. 17, p.243-259, https://link.springer.com/article/10.1007/s10796-014-9492-7