A Cloud-Based Solution for Efficiently Processing and Storing Highway Camera Logs

Casper Kristiansson Nicole Wijkman

October 19, 2023

1 Context

As every day passes the effectiveness and usage of data continue to grow rapidly, especially in areas of our daily life. One such domain is traffic management solutions where the efficient use of technology can improve the experience for millions of people. The United States of America has an active proposal called 511 Travel Information Telephone Services [1] that provides traffic information to its citizens. This has enabled approximately 90% of the states to offer web applications for real-time highway camera viewing [2].

PKTraffic LLC. [3] is a company that uses these cameras to capture and collect traffic data more specifically incident data. Their system is developed to efficiently and accurately collect vast amounts of data using a low-cost solution, enabling it to broaden its impact and provide a sought-after service for multiple entities.

The issue with highway cameras is that a large majority of them receive issues like 404 not found, 503 service unavailable, and a lot of other errors. Because the company collects footage from around 2,000-2,500 cameras the logs generated from all of them are quite large. In their current situation, PKTraffic doesn't have any good system in place for storing the logs in a manner so that they can easily be accessed, managed, and visualized.

2 Solution

The company PKTraffic currently is using a hybrid tech solution using both cloud products from Amazon AWS [4] and a local server. Because of this the data collection and management will be developed using AWS services like DynamoDB [5], kinesis streams [6], and Lambda [7].

2.1 Data Collection

The data collection part of the program as mentioned from the context (section 1) comes from traffic highway cameras. While the actual recording of the cameras generates a lot of logs like bytes from each request, camera information, etc. we are mostly interested in the HTTP requests made. The Goal with logging is to keep track of whether the application has successfully been capturing footage and whenever it has not been able to do it.

This means that before the application starts capturing footage we want to capture the current time and send a put log event. We then wait until the camera recording has stopped and log the reason for stopping whenever it is an expected or unexpected error.

2.2 Data Management

The data management part of the program is the actual part that manages the incoming data from the data collection. It mainly consists of two parts; a Kinesis data stream to collect and batch the data together and a Lambda script to process the data.

2.2.1 Amazon Kinesis Data Streams

The setup for a kinesis stream was extremely easy which consisted of simply creating the stream with few configurations like setting the data retention period to 24 hours and enabling on-demand mode. After completing this we can then in the python script simply use boto3 [8] client to put an event to the kinesis stream.

2.2.2 Amazon Lambda Functions

After events have been written to the kinesis data stream we need some way to analyze and manage the data. Using AWS lambda we can easily specify a trigger for the kinesis data stream like the batch size (amounts of events) or batch window. Using these settings we can easily group the logged data and perform the required analytics of the data. In our case, we want to perform two specific actions. The first one is to store the logged event in dynamoDB and the second is to update the latest status of the camera. Both of these actions are easy insertions into two different tables. This structure is also future-proof to help with any future analytics that might want to be performed on the logged data.

2.2.3 Amazon DynamoDB and MySQL

The storing of the data consists of two different database types. The first one is a NoSQL database using dynamoDB and a relational database using MySQL. The NoSQL database is used to keep track of all logged events while the NoSQL database will manage the latest status of each camera.

The reason for dividing the databases is due to performing the query to get the latest status of all of the cameras would be extremely slow and costly due to the structure of the tables.

The dynamoDB table consists of three different columns; cameraId (PK), timestamp (SK), and a status code. The status code is a simple integer that maps back to a dict which contains the specific error code. The reason for mapping the status code instead of saving the entire error in the database has to do with lowering the cost. By using this structure each item comes down to 33.7 bytes which means that storing a total of 267,000 logs results in a table size of 8 MB. Using this type of PK and SK system enables fast queries to extract logs for a given camera in a specific date range.

Lastly, the MySQL database will consist of one entry for each camera. The table will save the same information as the dynamoDB items which are the camera, last updated, and status. Using a second table for the status means that we can easily query the entire table to get the status of all cameras.

2.3 Data Visualization

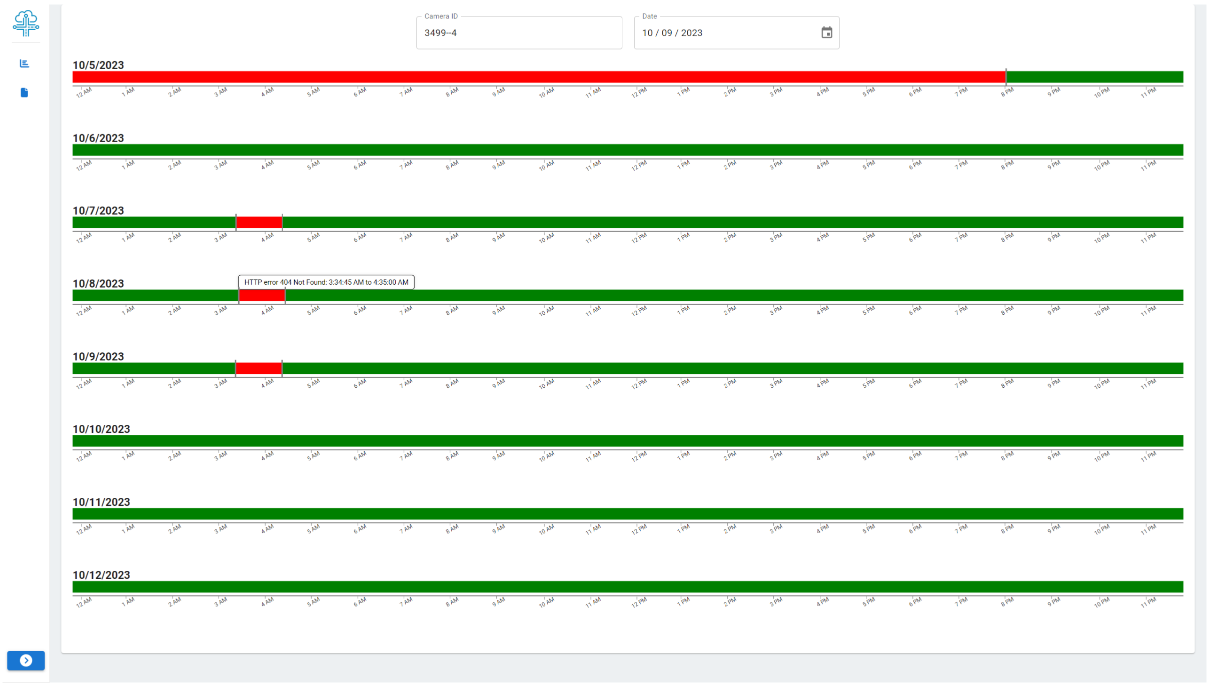
The data valuation part of the program consists of a lambda endpoint for querying the data and a React.js [9] application for visualization. The objective of this project was to create two dashboards: one showing the current status of all cameras on a map, and another for simulcasting a specific camera's uptime over a certain time span. However, due to time constraints, only the dashboard for visualizing a specific camera was completed.

2.3.1 Backend - AWS Lambda

The AWS lambda was created using Python and deployed using Serverless [10]. The Python scripts execute two queries: the first extracts all logs within a specified date range, while the second retrieves the earliest event that predates that range. The script then has a bit of logic for grouping the data before it is returned to the frontend.

2.3.2 Frontend - React.js

The frontend part of the project was developed on an existing project. This means that by simply creating a new page we accept two different inputs which are a specific camera ID and a date. We then can simply generate horizontal timelines displaying either green when a camera was recording or red whenever the camera was not recording.



Camera ID: 3499--4

Date: 10 / 09 / 2023

HTTP error 404 Not Found: 3:34:45 AM to 4:35:00 AM

Figure 1: The dashboard displaying a camera uptime

3 Result

The entire workflow from logging a specific event to visualizing it was successful. Using this kind of structure enables an efficient way to manage events that will be able to manage the required workflows of 50,000 - 100,000 highway cameras. The solution was built using AWS products that are still able to operate at a really low cost.

The Notebook.pdf contains the code and a more detailed description of everything that has been done in this project. While as discussed due to time constraints the dashboard for visualizing a map of the latest status of all cameras wasn't completed the dashboard for viewing a specific camera was. The backend structure for the map visualization is completed and therefore can easily be implemented in the future.

References

[1] *FHWA 511 Travel Information Telephone Services — ops.fhwa.dot.gov*, `https://ops.fhwa.dot.gov/511/`, (Accessed on 18/10/2023).

[2] *FL511 — fl511.com*, `https://fl511.com/`, (Accessed on 18/10/2023).

[3] P. Traffic, *Pktraffic — pktraffic.com*, `https://pktraffic.com/`, (Accessed on 18/10/2023).

[4] *Cloud services - amazon web services (aws)*, `https://aws.amazon.com/`, (Accessed on 18/10/2023).

[5] *Fast nosql key-value database – amazon dynamodb – amazon web services*, `https://aws.amazon.com/dynamodb/`, (Accessed on 18/10/2023).

[6] *Process and analyze streaming data – amazon kinesis – amazon web services*, `https://aws.amazon.com/kinesis/`, (Accessed on 18/10/2023).

[7] *Serverless computing - aws lambda - amazon web services*, `https://aws.amazon.com/lambda/`, (Accessed on 18/10/2023).

[8] *Boto3 1.28.65 documentation*, `https://boto3.amazonaws.com/v1/documentation/api/latest/index.html`, (Accessed on 18/10/2023).

[9] *React*, `https://react.dev/`, (Accessed on 18/10/2023).

[10] *Serverless: Develop & monitor apps on aws lambda*, `https://www.serverless.com/`, (Accessed on 18/10/2023).