Simulating the Streaming of Alarm Data Using Docker and Flink – IoT Real Time Analytics TABA

Jordan O'Donovan
School of Computing
National College of Ireland
Dublin, Ireland
x19372016@student.ncirl.ie

***Abstract*—It is becoming increasingly more apparent how crucial the processing and analysing of previously unconceivable streams of data is to ensure the operation of our modern-day technologies. In this project, Docker and Flink are used to clean, process, and stream such data in a containerized environment, to investigate the effectiveness and scalability of these solutions. This data is then uploaded to a cloud solution (Amazon Web Services' S3) where it is queried through the use of Amazon Web Services' Athena, before being visualised in an interactive live dashboard created in Grafana. This project both proves and highlights how efficient Docker, Flink, S3 and Athena are when the processing, storage and querying of real-time data is required, providing both individuals and businesses with a relatively simple and cost-effective solution to handle their needs as they scale in both size and scope.**

Keywords—Apache Flink, Docker, Python, Grafana, Amazon Web Services, Pandas, Athena, S3, data streaming, real-time analytics

I. Introduction

In our modern world, the amount of data being produced is increasing exponentially. Ten years ago, in 2013, humanity generated nine zettabytes of data. This year it is expected to be about one hundred and twenty zettabytes, with this increasing to one hundred and eighty-one in 2025 [1]. There is so much data being generated that for many it has become overwhelming, proving impossible to try to contain. The age of analysing a "complete" dataset is over.

The streaming of data has become the norm through our obsession with social media such as Twitter and Facebook and entertainment streaming services such as Netflix and Spotify. As part of human nature, we feel the need to consume information as soon as it is available, instead of waiting for tomorrow's newspaper as previous generations had done.

One area in which the streaming of data is crucial is the usage of alarms. Here, one cannot wait for this data, they must be alerted to it immediately, for the sake of their property of even their life.

The objective of this project is to simulate the streaming of alarm data through the usage of Flink processing through Python and Docker, before uploading the latest data instantly to the cloud, which is where it can be accessed by other services, such as Grafana to create real-time interactive dashboards.

An additional aim of this project is to provide an example of how Docker and Flink can be used to process real-time streams of data very efficiently and rapidly in a scalable environment.

II. Data

The dataset to be used for this project was created by BachPhu, a company developing IoT solutions in Vietnam [2]. This dataset contains 462,424 rows and eight columns, "station_id" containing the ID of station an alarm was triggered at, "datapoint_id" containing the ID of the sensor, "alarm_id" containing the ID of the alarm type activated, "event_time" containing the timestamp an alarm was triggered at, "value" which houses the value of the unit being recorded e.g., temperature, voltage. "ValueThreshold" contains the minimum "value" unit needed to be seen before the alarm will be activated, "isActive" is a Boolean column, detailing whether or not an alarm is active, and the final column, "storedtime".

This unclean dataset also comes with a "README" markdown file, which explains what each of the columns

represents, but also details what each of the datapoint and alarm ID values represent, as this is not stated in the CSV file.

III. Technologies Used

To undertake this project, I will utilise several different technologies which will work in harmony to load, clean, pre-process, store and visualise this data.

For the portion of these steps which will take place on my local machine, Docker, Flink and Python will be the three key technologies. Docker will be used to house and run the data and code in a containerized environment, free from the interference of other software running on said machine. Docker will also allow for the environment to be transferred to other machines to be run in a very elementary manner, simplifying the installation and execution of all software required.

Flink, the open-source solution created by Apache, will be used to clean and process the data. I will create a separate pipeline for each of these steps. The PyFlink module will be used to create a scalable environment during the execution of the script(s) required. The scope of the scalability will have to be investigated thoroughly as the computing power of my local machine is extremely limited.

At first, I considered using Apache's Spark to process my data, but after further research it was apparent that Flink was the right solution for the task at hand. Flink was designed from the ground up to deal with the processing and streaming of real-time data, whereas Spark was initially designed to carry out batch processing of data [3].

Once the data has been cleaned and processed, it will be uploaded to Amazon Web Services' (AWS) S3 cloud storage solution in either a CSV or Parquet format. An interactive live dashboard will be created using Grafana to give key insights

into this data, in an auto-updating and easily understandable environment.

Another initial consideration I had was to use Tableau instead of Grafana, as I'm experienced with Tableau and have used it to connect to AWS solutions in the past. Upon further research it became clear that as this task requires the visualization of real-time data analytics, Grafana has more advantages than Tableau here, thus is the right product to carry out this objective.

IV. Implementation

Before my Python file could be run, I first had to create my Dockerfile and docker-compose files so that the Docker image could be configured and built. The Dockerfile ensures that Python and Flink are installed on the containerised environment, along with every other technology required such as Boto3.

Next is the Python file. In this, I first establish the Flink and Boto3 environments, before moving onto my pipelines.

A. Cleaning

In the first pipeline, I initially had to load the CSV file as a Pandas' DataFrame so that it could efficiently be converted to a Flink table using Flink's from_pandas() function. Once it had been converted to the table with its schema defined, I dropped the "storedtime" column as it didn't contain any values. I was then able to use PyFlink's sql_query() function to create two new columns, "alarm_type" and "sensor_type" using CASE WHEN statements to map the alarm_id and datapoint_id values to their string counterparts.

B. Streaming

Once the dataset had been cleaned and was ready, it was time to "stream" it to the cloud. Using more PyFlink functions, namely sql_query(), I split the data into batches and iterated through them. I used PyFlink to get each batch which was then uploaded to my AWS S3 solution. As one cannot append directly to an S3 file, I had to convert the batches to a DataFrame and append the latest to the total number of batches selected thus far, consuming an increasing amount of my local machine's limited memory. The DataFrame was then converted to a CSV file and uploaded to the S3 database, overwriting the file already stored. I then used the time module to wait several seconds before selecting the next batch, to further simulate this streaming of data.

C. Cloud solutions

As one cannot connect Grafana directly to an S3 database, I first had to use AWS's Athena to act as a middleman between the two services. Here, I used SQL to create a table which could be used to query the CSV file stored in S3. The usage of Athena allows for a very expansive number of functions available to use in Grafana, while also giving complete control over which and how much data is queried.

V. Results

The usage of Docker and Flink to undertake this project proved to be an incredibly efficient solution. The usage of PyFlink's sql_query() made the cleaning of the dataset very rudimentary in the initial pipeline and was crucial during the second pipeline to iterate through the data through batch processing. This project also proved that Flink is easily scalable, as proved by altering the value of my "batch_size" variable. When I had it set to process two rows per batch it performed the same as when it was processing a thousand rows per batch (though not at quite the same speed thanks to my CPU).

The usage of AWS's Athena in tandem with Grafana enables countless actions and techniques which can be taken to display this dataset.

The first visualisation I created was a bar chart displaying the type of alarm triggered in the one thousand latest rows of the dataset. At the time this screenshot was taken, issues with the AC voltage proved to be by far the most frequently seen at the time, with these two values making up almost eighty percent of the total.

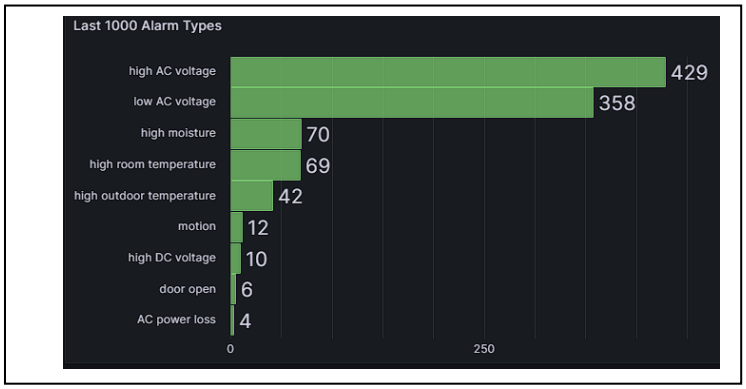

Figure 1. Bar chart displaying 1000 most recent alarm types activated.

Following this, I crated a pie chart visualising the ratio of active and inactive alarms. When a large section of the dataset is visualised here, the share of these values is virtually equal, with "False" being present in two of seventeen thousand instances more often than "True".

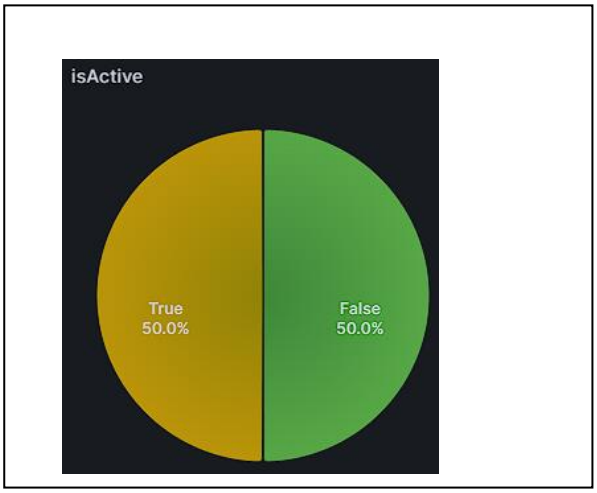

Figure 2. Pie chart showing share of True vs False values.

The next graph I created was another bar chart, this time displaying how often each station ID value was present in the dataset.

From this subsection of the dataset, two stations in particular also made up almost eighty percent of the total.

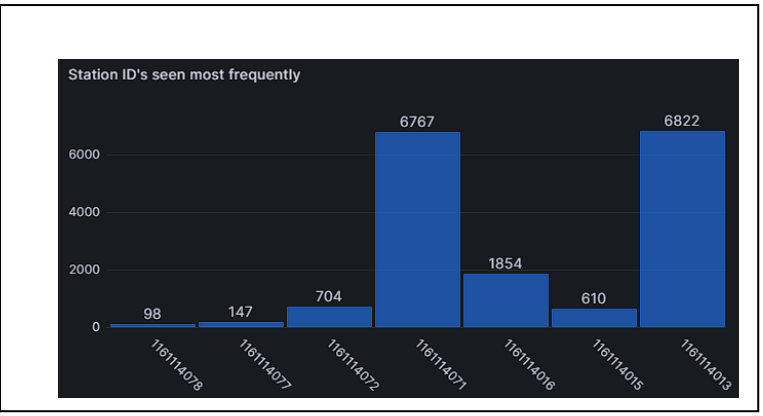

Figure 3. Bar chart displaying how often each Station ID occurs.

Figure 5. The full dashboard.

The final graph I created for my interactive dashboard was a time series graph displaying how many alarms were activated during each minute of the days.

This graph is extremely effective in communicating the trends seen with the number of alarms activated. Most minutes only had one alarm activated although on the 31st of January at 10:31am there were thirty-two instances of alarms being triggered.

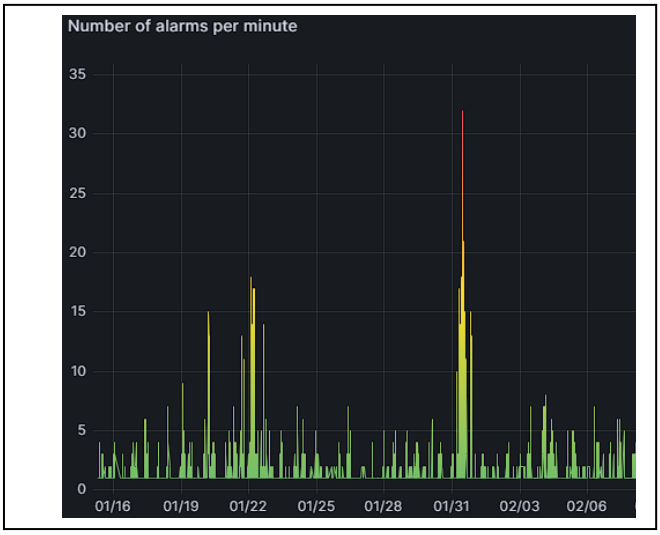

Figure 4. Time series graph displaying number of alarms per minute.

The shape of this graph has been heavily distorted to meet the requirements for this file format. The interactive dashboard can be browsed in a much more readable environment at this link.

VI. Conclusions and Future Work

To conclude, I've successfully demonstrated that this project was a resounding success. I've proved and highlighted the benefits and efficiency of using Docker and Flink to create containerised environments to run code safely and effectively in, and I've shown how useful cloud solutions can be to both store and query data, as well as creating interactive live dashboards to visualise said data in a myriad of manners.

I've also proved how easy it is to increase the capacity of Flink's workload, highlighting how scalable and useful this new technology is.

I wouldn't have been able to undertake this project without the skills and knowledge I acquired through this module. Docker and Flink were not technologies I was familiar with before this module began, and my knowledge of cloud computing solutions was extremely limited and lacking beforehand. Concepts we covered such as sliding and tumbling windows greatly helped my understanding of how technologies like Flink can be utilised to processes large streams or batches of data. These techniques helped me in appreciating how important it is to have fault tolerance and scalability solutions when creating these real-time data processing applications.

I was able to utilise what I'd learned through the lectures and labs when creating my Docker and Flink solutions, as well as when creating my S3 and Athena instances. As we had to utilise cloud computing options in the first CA (I used AWS's DynamoDB that time) I was able to transfer the knowledge acquired from that CA to create my roles and policies much more quickly than would've otherwise been possible.

If I had more time, I would experiment more with possible use cases of Flink on this dataset. I would try to see what could be done in terms of aggregating data and creating analytics with the data. I would also spend more time using Grafana, to see what else could be done with the data, whether it be with using other types of graphs or performing more complicated actions through the usage of SQL queries with AWS's Athena.

I would also measure the performance of Flink on my machine when the scalability of the Python script is altered. I would measure how long it takes to process the data with different batch sizes and how much of my local machine's

hardware is needed to perform these actions. I could also compare the performance of using these new technologies to ones I'm already familiar with, such as comparing Tableau to Grafana or Flink to Spark.

REFERENCES

[1] Duarte F. (2023) 'Amount of Data Created Daily (2023)' *Exploding Topics*, 3 April 2023. Available at: https://explodingtopics.com/blog/data-generated-per-day [Accessed April 20 2023].

[2] Truong, L. (2019) Sample of BTS monitoring data. Available at: https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/data/bts [Accessed April 7 2023].

[3] Macrometa *Apache Spark Vs Flink*. Available at: https://www.macrometa.com/event-stream-processing/spark-vs-flink [Accessed April 20 2023]..