Log Aggregation

Submitted April of 2023 to the

Anderson College of Business and Computing

Regis University

in partial fulfillment of the requirements for a

Master of Science in Data Science with a Specialization in Data Engineering

by

J. Jaime Rios

# Practicum Project Advisor

# Kellen SoraufTable of Contents

* Chapter 1 - Introduction
* Chapter 2 – The Data
* Chapter 3 – Technical Components
* Chapter 4 - Results
* Chapter 5 - Conclusions
* Appendix A – Apache Airflow Configuration
  + Appendix A1 – Airflow DAG Configuration
* Appendix B – PostgreSQL Configuration
* Appendix C – Metabase Configuration

# 

# Chapter 1-Introduction

**Overview**

The goal of this project was to create a Data Engineering solution for aggregating log data. It would involve gathering diverse log data for different applications and creating an ETL process to load that data to a common table within a database. ETL stands for extract, transform, load. Once loaded – I would also need to find a way to visualize the aggregated log data.

**Background:**

Log aggregation has become a hot topic. Where I work – there has been talk of utilizing Sumo Logic to aggregate data for our custom application logs. And our IT support team uses Dynatrace to monitor system and network logs.

Log aggregation involves gathering, cleansing, and collecting log data into a single source. This facilitates log monitoring and analysis. Without it, engineers and application specialists would have to locate and search for log information from multiple locations and sources (Datadog, 2021).

The true goal of this project was to create a Data Engineering pipeline. Before jumping in on something – I wanted to make sure I fully knew what that meant. According to Spati (2021) – it would entail setting up a data pipeline to ingest and cleanse data, storing it in a data warehouse and generating visualizations from the data. Seetharaman (2022) would get more specific and spell out five specific steps: picking a data source, extracting data from said source, creating an ETL to transform the data, writing/loading data from the ETL somewhere, and visualizing the data.

Research Statement:

With this project, I would be exploring the creation of an ETL pipeline. The project would involve gathering diverse log information, cleansing it and loading it to a common location.

Deliverables Statement:

The goal would be to create a data engineering pipeline to gather and cleanse data to be loaded to a single solution. I would explore different solutions for automating the ETL – including Unix cron technology and Apache Airflow. I would utilize Python to write ETL processes and to load data to a PostgreSQL database. Metabase would be used for data visualization.

# Chapter 2 – The Data

The [Loghub](https://github.com/logpai/loghub) project on github had a wide variety of different types of log files – providing the perfect data source for this project. It included logs for different applications, file systems, and operating systems. I would choose to download some 2k sample log files directly from the loghub repo – including logs for Apache, Hadoop, HDFS, Linux and Windows.

The above-mentioned log files contained a sample of 2,000 lines per log. The Loghub project also provided a separate link offering access to full log file samples – offering over 77 gigabytes of data. I would download the complete Apache log file from their separate site just to get an idea of what it would look like to run a full data set/log file through.

Here’s a look at the different types of data contained within each of the logs that I downloaded – using the Linux head command to view the top 3 records of each log file:

Text

Description automatically generated

Manually extracting the full tar.gz Apache log in Linux (after already downloaded):

Graphical user interface, text, application

Description automatically generated

**Chapter 3 – Technical Components**

The Technology

I would choose Python as the programming language of choice for this project to write the ETL. Python is a high-level interpreted programming language that is popular for its plethora of tools and libraries that can be used for several different use cases. It is particularly popular for working with data. I would often experiment with code in Python Jupyter Notebooks before adding code to my finalized scripts – it was very helpful for testing small pieces of code.

PostgreSQL would be used for my database solution. PostgreSQL is a popular open-source database management system (dbms). It is free to use, mature, stable, and SQL compliant. See appendix B for information on installing and configuring PostgreSQL on a Lubuntu Linux vm.

Metabase would be used for visualization. Metabase is also open-source and available for free. It is a business intelligence tool that can be used to visualize data. It can be used to easily import data and to generate charts for that data. It can be used to create dashboards as well – allowing the Data Engineer to easily show users what their data looks like. See appendix C for information on installing and configuring Metabase on a Lubuntu Linux vm.

For automating the ETL pipeline – I would take two different approaches. The first would be a more traditional Unix/Linux cron solution. The second would be using Apache Airflow.

Apache Airflow is a popular data pipeline/workflow tool. Its main feature is the DAG – the “Directed Acyclic Graph”. DAGs are written in Python – and once imported into Airflow – they graphically show the order of operation for a larger process – showing process tasks and their dependencies. Users can easily manage and monitor these process tasks using Airflow tools and features. Airflow makes ETL processes easy to automate. If one piece of a workflow fails – Airflow can be set up to keep trying. It will prevent a process further down the chain from kicking off until the previous process successfully completes (Kervizic, 2021). Appendix A has info on installing and configuring Airflow. Appendix A1 has information on importing DAG code files.

I got pretty far with Airflow – but did not quite get all the way there. So I resorted to something I was already familiar with – the Unix/Linux cron – the OG scheduler. Although cron technology is an old Unix standard – it is still the magic behind a lot of todays scheduling tools and utilities (Quinn, 2022) – so being familiar with it as a Data Engineer is very important. I set up a shell script that would run every 5 minutes. It called each of the Python log scripts one by one. Anything added to any of those logs since the previous cron run would be automatically added to the database. The call to my cron script in my Linux cron file looked like this:

|  |
| --- |
| # minute(0-59) hour(0-23) dayOfMonth(1-31) month(1-12) dayOfWeek (0-7) script  # A \* in any position signifies all available values  # Kick off bash script to load log entries to db every 5 minutes  0,5,10,15,20,25,30,35,40,45,50,55 \* \* \* \* /mnt/documents/Regis/Final2/bin/FinalCode/cron\_log\_runs.sh >> /mnt/documents/Regis/Final2/bin/FinalCode/cron\_log\_runs.sh.log |

The Scripts:

* config.props – property/config file with shared variable names
* create\_db.py – creates a new PostgreSQL db; takes db name as a command line argument
* create\_log\_table.py – creates log table if it does not exist; gets log name from config file
* drop\_log\_table.py – drops log table; came in very handy for deleting and rebuilding table
* db\_funcs.py – database functions that other scripts could call
* dag\_libs.py – a few functions for other scripts to call
* cron\_log\_runs.sh – called by cron; calls all the below load scripts
* load\_apache\_log.py – loads Apache log data to PostgreSQL db table
* load\_hadoop\_log.py – loads Hadoop log data to PostgreSQL db table
* load\_hdfs\_log.py – loads HDFS log data to PostgreSQL db table
* load\_linux\_log.py – loads Linux log data to PostgreSQL db table
* load\_windows\_log.py – loads Windows log data to PostgreSQL db table

I wrote the log scripts in such a way so that they would be easier to convert to DAG scripts. I may have done things a bit different otherwise. Each of the log load scripts are very similar. They are made up of functions for each step of ETL – extract, transform, load – along with supporting functions for any unique features within the log data – such as unique functions to properly format date. The extract and load functions are pretty much the same for each script. The extract function is very simple – just reading the log data to memory before passing the file handle to the transform function. The unique piece of each Python script is within the transform function – which contains different logic for parsing data within each log file.

**Chapter 4 –Results**

The end game was having the data automatically loaded to my database and being able to generate a dashboard that would graphically show what was present in the data. The below screenshots show the cron file successfully running every 5 minutes – and loading data to the db. Any lines that get added to any of the logs being monitored between runs will be added on the next cron run within 5 minutes.

|  |  |
| --- | --- |
| crontab -l (to view cron file contents)  crontab -e (to edit cron file) |  |

As can be seen – 10,000 records were loaded to the database. Metabase would make it easy to visualize that data - offering different tools for queries and exploration. First off – it would offer a standard grid style view of the actual data in a table. It would also offer different ways to ask questions of the data and query – as can be seen in the dropdown below.

|  |  |
| --- | --- |
| Graphical user interface  Description automatically generated |  |

Metabase users could use the “Question” feature to visualize data. They could choose their data source, their metrics, and the specific fields they wanted to apply those metrics to.

|  |  |
| --- | --- |
|  |  |

Another option to generate visualizations was the “SQL query” option. As can be seen above – we have some of the same values in the chart twice – once in upper case – and once in lower case. Using the SQL option would make it easy to group those together.

Chart

Description automatically generated

Metabase makes things easy to save questions and queries – and to combine them into a single dashboard. Users can save their queries and add them to a dashboard at a later time – or can create a dashboard and generate new queries from the dashboard to populate it on the fly.

|  |  |
| --- | --- |
| Selecting the “Dashboard” option from the “New” button dropdown presents the “New Dashboard” dialogue box. | Graphical user interface, text, application, email, Teams  Description automatically generated |
| Once a new dashboard is created – one can generate new questions and queries from the dashboard. | Graphical user interface, text, application, Teams  Description automatically generated |
| Users will be prompted on whether they want to save the question to the current (or another) dashboard. | Graphical user interface, text, application, email  Description automatically generated |

|  |  |
| --- | --- |
| Once users have elements on the dashboard – they can easily drag them around and resize them to change the look and feel of the dashboard. | Graphical user interface, application  Description automatically generated |
| Once a dashboard is setup – it will automatically refresh whenever new data is added to the database. | Graphical user interface, application  Description automatically generated |

Before calling my project done – I wanted to check one last thing. Being that I was working with sample files that only had 2000 lines – I wondered if my code might run into memory issues reading in full log files. I manually kicked off a full run of the Apache log file – and it did successfully complete. As shown via the time function - it took 18 seconds to complete – but did so with no issue. The result is evidenced below by the number of records shown in the below query within Metabase.

|  |  |
| --- | --- |
| Graphical user interface, text, application, chat or text message  Description automatically generated | Graphical user interface, text, application  Description automatically generated |

**Chapter 5 – Conclusion**

Although not done quite as originally planned - I accomplished my end-goal. I was able to architect a repeatable, automated process to read in log file data - adding any new log entries to a database without any manual, human intervention. I was able to take the data automatically loaded to the database and visualize it – showing information about what the data consisted of. These visualizations would be updated on the fly as new data was added to the database – allowing anyone needing the data to be able to see up to the date, current information (almost at least – could have also changed the cron to run every minute with a \*). This completes the job of the Data Engineer – who would then pass that data on to Data Scientists or Statisticians for further data analysis and/or modeling.

While I did not quite get all the way there with Airflow – I got close and was proud of that. It took a lot of configuration - and finding out how to be able to utilize outside home-made property files and custom libraries - just to get Airflow to recognize my DAG script. Though I could get it imported to Airflow and could kick it off – I just could not get a successful run out of it to populate the database. I actually plan on continuing to play with this – because it has become something pretty interesting to me that I would like to master.

|  |  |
| --- | --- |
|  |  |

**Appendix A**

**Apache Airflow Installation on Lubuntu**

Install and init Airflow:

sudo apt-get update

pip install apache-airflow

airflow db init

Create admin user:

airflow users create --username admin --firstname jaime --lastname rios --role Admin --email none@email.com

Start webserver:

airflow webserver -D

With the webserver running – one can then bring up Airflow on the browser using url: <http://localhost:8080/home>

Graphical user interface, text, application, email

Description automatically generated

I would end up stopping the service to go back and start up the scheduler beforehand. I would also install Postgres for Airflow prior to restarting – which would add a Postgres option to my connection list.

pip install apache-airflow-providers-postgres==5.4.0

airflow scheduler

airflow webserver

Graphical user interface, application

Description automatically generated

With PostgreSQL installed and now available as a connection option – we could then set up the connection to our database and test the connection. (Note: Postgre needs to already be setup on the server. See Appendix B.)

Graphical user interface, application, email

Description automatically generated

**Appendix A1**

Importing a New DAG Script

To create new DAG files – I would simply create a new Python code file – calling the appropriate Airflow libraries – and utilizing the correct syntax to declare and utilize DAGs. I would find that I would need to create a “dags” directory in my airflow home directory (/home/user/airflow/dags/). I would then need to edit the airflow.cfg file in the airflow home directory – adding a “DAG\_FOLDER” entry pointing to my dags folder:

DAG\_FOLDER = /home/jrios/airflow/dags

I would not have to restart the service after that – but I would have to completely refresh my webpage for it to recognize any new DAG files – by completely retyping the url in my browser - some times multiple times. The “dag refresh” button never seemed to work to recognize new DAG files. If there was an error with a DAG file and Airflow could not import it – it would state so:

Text

Description automatically generated with medium confidence

**Appendix B**

**PostgreSQL Installation on Lubuntu**

Install and start PostgreSQL:

sudo apt update

sudo apt install postgresql postgresql-contrib

sudo systemctl start postgresql.service

Now try login as PostgreSQL admin and start psql:

sudo -i -u postgres

psql

Create a new user:

postgres@jaime-virtualbox:~$ createuser --interactive

Enter name of role to add: psql\_user

Shall the new role be a superuser? (y/n) y

How change new users password after the fact:

postgres**=#** ALTER USER psql\_user PASSWORD 'password';

Install Python psycopg2 package

sudo apt-get install libpq-dev

sudo apt-get install python3-psycopg2

Test connection:

Text

Description automatically generated

**Appendix C**

**Metabase Installation on Lubuntu**

The first step to installing Metabase on Lubuntu is to download the jar file for it from:

<https://www.metabase.com/docs/latest/installation-and-operation/running-the-metabase-jar-file>

Then it is suggested to create a directory for it – and move the downloaded jar file into your new “metabase” directory. You will need to verify you have Java installed on your system. Once verified (or installed) – then you can kick off a Java command to start up Metabase.

jrios@jaime-virtualbox:~$ ls -l Downloads/meta\*

-rw-rw-r-- 1 jrios jrios 292130072 Apr 16 20:53 Downloads/metabase.jar

jrios@jaime-virtualbox:~$ mkdir metabase

jrios@jaime-virtualbox:~$ mv ./Downloads/metabase.jar ./metabase/

jrios@jaime-virtualbox:~$ cd metabase

jrios@jaime-virtualbox:~/metabase$ java -version

openjdk version "11.0.18" 2023-01-17

OpenJDK Runtime Environment (build 11.0.18+10-post-Ubuntu-0ubuntu120.04.1)

OpenJDK 64-Bit Server VM (build 11.0.18+10-post-Ubuntu-0ubuntu120.04.1, mixed mode, sharing)

jrios@jaime-virtualbox:~/metabase$ java -jar metabase.jar

This failed for me the first time – so I had to kick off the below command – as suggested in an error message displayed to screen:

java -jar metabase.jar migrate release-locks

The second attempt was then successful:

java -jar metabase.jar

Two key lines in the messages displayed to screen that indicate success are:

2023-04-16 21:12:03,443 INFO metabase.core :: Metabase Initialization COMPLETE

… and

2023-04-16 21:12:47,985 INFO middleware.misc :: Setting Metabase site URL to localhost:3000

During setup – one can initiate a connection to the database.

|  |  |
| --- | --- |
| Graphical user interface, text, application, email  Description automatically generated |  |

**Bibliography**

Drake, M. (2022, March 18). *How To Install PostgreSQL on Ubuntu 20.04*. DigitalOcean. <https://www.digitalocean.com/community/tutorials/how-to-install-postgresql-on-ubuntu-20-04-quickstart>

Gaenger, S. (2020, June 29). *The 11 Best Data Visualization Tools for SQL*. Chartio. <https://chartio.com/learn/business-intelligence/the-11-best-data-visualization-tools-for-sql/>

Kamat, S. (2023, March 6). *What are Airflow and Nifi and How are They Different? | Scalable Data Integration Tools for Your Business*. Data, AI & Product Engineering | Algoscale. <https://algoscale.com/blog/airflow-and-nifi-data-integration-tools>

Kervizic, J. (2021, December 10). *10 Benefits to using Airflow - Hacking Analytics*. Medium. <https://medium.com/analytics-and-data/10-benefits-to-using-airflow-33d312537bae>

Kumar, M. (2022, August 17). *Setting up Apache-Airflow in Ubuntu*. Medium. <https://medium.com/international-school-of-ai-data-science/setting-up-apache-airflow-in-ubuntu-324cfcee1427>

L. (n.d.). *GitHub - logpai/loghub: A large collection of system log datasets for log analysis research*. GitHub. <https://github.com/logpai/loghub>

*Log Aggregation: What It Is & How It Works.* (2021, August 3). Datadog. <https://www.datadoghq.com/knowledge-center/log-aggregation/#:~:text=to%20learn%20more-,What%20is%20log%20aggregation%3F,to%20facilitate%20streamlined%20log%20analysis>.

*Modules*. (n.d.). Python Documentation. <https://docs.python.org/3/tutorial/modules.html>

Quinn, Z. (2022, January 30). *How to schedule jobs using cron. Pipeline: A Data Engineering Resource*. <https://medium.com/pipeline-a-data-engineering-resource/cron-ological-order-how-to-write-cron-for-scheduling-pipelines-feb8744a1ceb>

*Running the Metabase JAR file*. (n.d.). Metabase | Business Intelligence, Dashboards, and Data Visualization. <https://www.metabase.com/docs/latest/installation-and-operation/running-the-metabase-jar-file>

Seetharaman, B. (2022, June 1). *Build a Good Data Engineering Project in Five Steps*. TopCoder.com. Retrieved from <https://www.topcoder.com/thrive/articles/build-a-good-data-engineering-project-in-five-steps>

*Set up a Database Backend — Airflow Documentation.* (n.d.). <https://airflow.apache.org/docs/apache-airflow/2.5.2/howto/set-up-database.html>

Shilin He, Jieming Zhu, Pinjia He, Michael R. Lyu. *Loghub: A Large Collection of System Log Datasets towards Automated Log Analytics*. Arxiv, 2020.

Spati, S. (2021, March 09). *Building a Data Engineering Project in 20 Minutes.* Sspaeti.com. Retrieved from <https://www.sspaeti.com/blog/data-engineering-project-in-twenty-minutes/>

Srivastava, R. (2022, March 10). *Apache NiFi Vs Kafka : What Should You Prefer?*. Ksolves Blog. <https://www.ksolves.com/blog/big-data/nifi/apache-nifi-vs-kafka-what-should-you-prefer>

Thong, A., DO. (2019, January 22). *Top 5 Free or Affordable SQL Reporting and Dashboard Tools*. Medium. <https://medium.com/holistics-software/top-5-free-or-affordable-sql-reporting-and-dashboard-tools-490df5b5f093>