System for Telemetry Organization and Recovery

Liam Robbins

The Jet Propulsion Laboratory

UC Riverside

Mentor: Marc Pomerantz

Co-mentor: Elliott Trapp

393G | Mission Control Systems Section

Information and Data Management Group

***Abstract*—** NASA Jet Propulsion Laboratory is inundated with information in the form of engineering and telemetry data that is transmitted from its active missions and respective spacecraft. Thus, it is necessary to have a software system capable of receiving, processing, logging, and organizing this engineering data. The solution cto this problem is STOR (System for Telemetry Organization and Recovery). STOR is to be an automated engineering and telemetry data archival system that will support both current JPL missions and missions in the foreseeable future. This paper will present the system engineering and results from development of STOR.

***Keywords*—** STOR, Data archival, Telemetry Organization and Recovery,

I. Introduction

JPL operates numerous missions, each differing widely in scope and structure. Every mission produces invaluable, unique data that must be stored to be analyzed at a later time. STOR must be flexible and scalable to meet the needs of JPL's various missions, intuitive to use for both mission engineers and the data team, and autonomous to reduce human operator error in the archival pipeline.

This project focuses on the data handling aspect of big data. STOR will form the data pipeline for large amounts of mission critical data to flow through to be logged and archived in a scalable and reliable manner.

II. Project Requirements

The architecture of STOR revolves around four main requirements: STOR must have a user interface to interact with Project Users that must display archival status and analytics. STOR must be capable of receiving and archiving input from various data sources. STOR must provide a configuration interface for mission customers to input configuration data. STOR must pass quality acceptance tests in order to be approved for usage. [1]

A. User Interface Requirements

STOR's User interface must be able to assign Archival status to Engineering data into four categories:

- Non-Archival
- Pending Archive
- Active Archival
- Archival Complete

The user interface must also be capable of displaying data analytics, I.E engineering data types and storage requirements over the period of days, weeks, years.

B. Functional Requirements

STOR must be capable of archiving and logging TF, PKT, GIF, DSN monitor data, and channelized data from TDS, SLE, and DOM.

C. Project Level Requirements

The Data management team must provide a Mission Archiving Configuration Questionnaire (MACQ) to each customer. This questionnaire will specify the point of contact, as well as which data type, venue, and which data product to query from.

D. Quality Assurance Requirements

STOR must pass certain quality assurance standards in order to be able to proceed with the decommissioning of the current data archival system.

III. Project Architecture

STOR's Architecture can be deconstructed into high level and lower level portions. The higher level describes the system's input and out, as well as how it should interact with its customers. The lower level is concerned with the technologies used in the construction of STOR.

A. Higher Level Architecture

The Higher level behavior of STOR is to be able to interface directly with various mission subnets and machines and their respective Points of Contact (PoC) in order to log, archive, and display both the data and metadata required to query. STOR will provide this data to LSMD, which will then proceed with the archivation of the data.

STOR will begin with the configuration data received from the manually inputted MACQ. STOR's UI will provide an interface for both the mission engineer and the data engineer to view logging information and system diagnostics. STOR will query data from its respective flight mission machine using the parameters specified in the MACQ. STOR will then log the archival process and finally push both the data to be archived and the metadata to LSMD for the archival process.

B. Lower Level Architecture

STOR will use various technologies in order to complete its tasks. Docker will be used in order to compartmentalize each application of STOR, as well as alleviate any system requirements and compatibility issues. Ideally, each containerized application will operate completely independent of each other, and be technology and language agnostic. This modularity will expedite both the development process and the long term upkeep of the project.

STOR will utilize Apache airflow to automate and orchestrate the various applications in STOR, forming the data pipeline for the project.

STOR will use FastAPI as the interface for the data management team and its customers for the querying of data.

IV. Project Results

The first goal was implementing a dockerized mockup of a software handshake between an Apache Airflow application and a FastAPI web app. This would be done by building a lightweight FastAPI with the capability of accepting files via curl commands, and an Airflow application with the capability of curling a file to the web app.

A. Docker

Preliminary research on Docker was summarized in a presentation given to the team during weeks 3-4. The main use of Docker is in its ability to containerize applications. The use of containers is similar to the usage of Virtual Machines to isolate development environment variables and eliminate dependency issues. Virtual machines do this by creating a guest OS on top of the host OS using a hypervisor. This approach is not scalable, as the both the hypervisor and the additional OS are resource intensive. [3]

Docker isolates applications by building upon the host OS and creating a sandbox for the application to run in. System dependencies and environment variables are confined to each

application's container, eliminating the need for a virtual machine. This approach allows Docker to be extremely lightweight and comparatively easy to distribute.

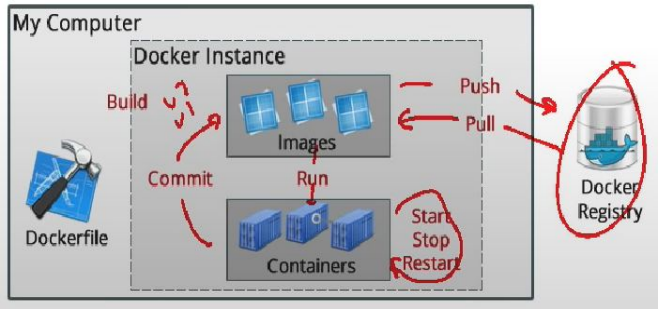

Fig. 4.1 - Docker Architecture [4]

The architecture of Docker is revealed in Figure 4.1 above. Docker's architecture is similar to that of git version control. Dockerfiles are the basis of docker, and are 'built' to create docker images. Docker images are distributed through an image registry such as docker hub. These distributed images can be further built upon by users via a docker file to create their own, custom application. Docker-compose, docker swarm, or other container orchestration tools such as kubernetes can then be used to spin up and manage multiple docker containers, possibly across multiple machines.

B. FastAPI

Week 5 focused on researching logging frameworks and implementing a FastAPI application capable of logging and accepting curl commands.

This prototype never made it into the production code, as Ned created the FastAPI server before the prototype was pushed to a repository, however

C. Apache Airflow

Weeks 6-8 focused on creating an Airflow application, dockerfile, and docker-compose deliverable in order to give a proof of concept demonstration of the data pipeline. The FastAPI web application was successfully mocked by Ned Udomkesmalee. I began with the creation of a docker-compose file that spun up Ned's image and an instance of Airflow using the official docker image on docker hub. I then built the Airflow application by creating a DAG that interfaced with the web application, completing the software handshake. [5]

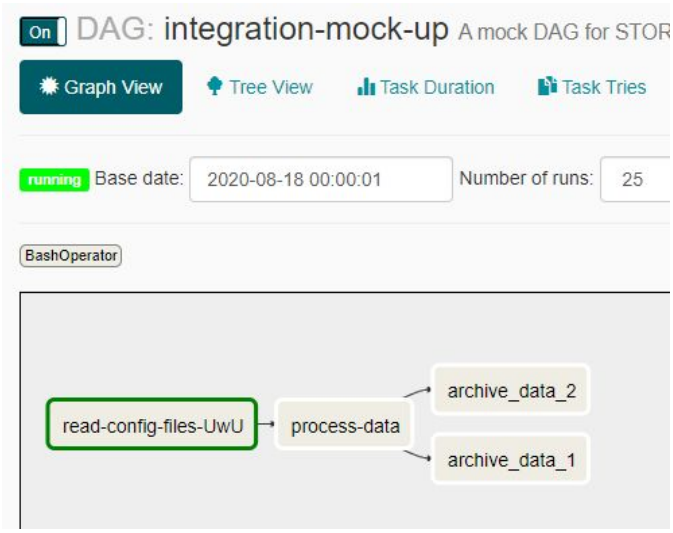

Fig. 4.2 - Directed Acyclic Graph

The Acyclic Graph shown in figure 4.2 depicts the basic skeleton by which files might be read through STOR.

D. Task automation

Week 9 focused on the automation process and the creation of scripts integral to the data pipeline. An example of such a script is the one that I worked on, which turned raw data from a json file and formats it into a processed pvl file. This pvl file will then be able to go on to be processed by STOR.

E. User Interface

Week 10 I was introduced to the task of the user interface design, and did much preliminary research into javascript and front end frameworks such as vue. However, this task was cut short by the end of the internship. I plan to continue this following a short leave of absence after the end of the internship.

V. Conclusions

The results gathered in this project will help to further develop and reinforce the STOR project plan. Development of Docker and Apache Airflow will continue to form the backbone of the project's applications and will evolve throughout the development cycle. The task automation scripts will continue to be expanded upon until the project has a diverse toolkit of scripts to disassemble various file structures to prepare them for processing. The user interface most likely will be a task to continue on in a future project.

This project will benefit future JPL missions by providing JPL with an alternative method for storing, logging, and archiving telemetry and other mission critical data. STOR will play a key part in the data pipeline for both current and future JPL operations and in its data infrastructure.

Acknowledgment

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by FIELDS (Fellowships and Internships in Extremely Large Data Sets) and the National Aeronautics and Space Administration. The program is coordinated by the University of California, Riverside and NASA JPL. Special thanks to Dr. Xinnan Du, for overseeing the program at UCR, as well as my mentor and co-mentor Marc Pomerantz and Elliott Trapp. I would also like to thank Ned Udomkesmalee and Shakeh Brys. You all made this experience very special. Thank you.

References

[1] Trapp, Elliot., *STOR requirements and Project Interface Requirements*. Jpl wiki. June 2020.

[2] Babak, Rad. Harrison, John. Bhatti, Mohammad. *An Introduction to Docker and Analysis of its Performance*. IJCSNS International Journal of Computer Science and Network Security. March 2017

[3] Hale, Jeff. *Learn Enough Docker to be Useful*. Towardsdatascience. January, 2019.

[4] Teemu, Koivisto *Efficient Data Analysis Pipelines*. University of Helsinki. Spring, 2019