ETL System: Bulk Updating Tickets by Creating a Loader Extension

A Capstone Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Brittany Sandoval-Rivera

May 9, 2023

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.

Brittany Sandoval-Rivera

Capstone advisor: Rosanne Vrugtman. Advisor, Department of Computer Science

ABSTRACT

An experience management company that allows its customers to create tickets from issues or incident reports found that its current method of updating thousands of tickets was extremely inefficient and time-consuming. The solution was to create a new loader extension into in existing ETL system, which would be able to handle thousands of API calls to the Tickets database in a single workflow. I first created a design document, outlining what I would implement in the frontend and backend. I also included flowcharts that visualized what would occur at runtime and during task configuration. When I began coding the task, I used Golang for the backend module and Typescript for the frontend extension. I tested the task end-to-end and was able to successfully update 60,000 tickets in under 24 hours. Now customers can modify or update multiple field names on thousands of tickets by creating one workflow; they no longer have to make thousands of workflows per ticket or excessively call the public API. As of now, after the workflow completes, the run history tab only shows the number of tickets that failed and the number of tickets that were updated. In the future, it would be helpful to add a downloadable CSV file that indicates which tickets updated and which tickets failed, along with an error message.

1. INTRODUCTION

Since the creation of the ticketing system, many customers were left unsatisfied due to the complexity involved in a simple ticket update. An update may include changing the status of a ticket to closed, adding a score, a comment, changing the priority to high, re-assign the ticket to a new owner, and so on. Many customers asked if there was a way to update thousands of tickets at once. The response was that they had to send thousands of individual requests to the public ticket API themselves, or have someone from the company with authorized access set up several workflows for them to trigger repeatedly.

Both of these methods were unsuitable and involved lengthy processes. This is why the loader extension, called Load into Tickets, was necessary to alleviate the operational headaches. This extension would be a part of the existing ETL (Extract, Transform, Load) system, which is useful for moving large amounts of data. The user would only need to set up a workflow once and let it run. Behind the scenes, the loader task would process these tickets and update them through the internal API at a controlled rate that would not overload the system.

2. RELATED WORKS

Several ETL systems have been built and utilized in different ways. The general idea is that heterogeneous data is extracted, transformed, and loaded into a target destination. ETL became a popular concept in the 1970s when it was introduced as a process for loading data for computation and analysis. However, Wickramasinghe (2021) reports that it has now evolved into a primary method for processing large amounts of data for data warehousing and data lake projects [1]. It first collects data from multiple sources, processes and cleanses that data to its proper format, and then loads it into a data warehouse. As Sun and Lan (2012) note, these processes must be executed in that order because they are all related to each other [2].

Harrison, et. al. (2022) identified Amazon Glue as a system that allowed users to connect over 70 data sources and load data into their data lakes [3]. Another recent work, Galici, et. al. (2020) is applying ETL to blockchain data. They extract the data,

transform, and load it into a relational database to further analyze transactions and addresses [4]. Contrary to these ETL systems, Load into Tickets will only need to have its data extracted from one source. The data in this one location specifies what ticket fields need to be updated. Furthermore, the data will only load into one database, the Tickets database, as opposed to an entire data warehouse.

3. PROJECT DESIGN

This section will describe the Load into Tickets task's key components and system architecture. Key components describe what will be included in the frontend and backend, as well as the main features of the plugin. The system architecture consists of illustrations of the task's different flows. The first diagram is the task configuration flowchart, what users must do to set up a workflow. The second flowchart shows the task at runtime, which is what will happen to the task in the backend once the workflow is executed.

3.1 Key Components

Load into Tickets task consists of creating a frontend extension and backend module. The extension is written in Typescript and is the only part that the user will interact with. The UI (Figure 1) of the extension includes a dropdown to select the data source and a section that automatically maps the fields from the extracted source to the names found in the Tickets database. For example, from the source data, the field name that the user wishes to update could be called, "TicketStatus," which will then be mapped "status," since that is the key name in the database. However, the only names that will be automatically mapped are the ones that are found on every ticket, such as ticket key, status, priority, owner, score, team, group, and comment. Upon creating tickets, users were allowed to add custom attributes. The Load to Tickets extension allows them to update these custom fields by selecting "Add Field" at the bottom of the modal. The number of fields to update is optional, though the ticket key field is required. The user will be unable to save the extension until the ticket key is mapped.

The backend component is called a module, which is essentially an interface written in Golang. The data that contains all the tickets' information, such as the field names and new values, are processed in this module. Extracting this data from the cloud and converting it to a CSV file is a different module that was created prior to this project. However, once this data is obtained, the Load to Tickets module will convert each line in the file into an API request. Tickets will be updated consecutively; requests will be sent one at a time to the Tickets API. If a request fails, then it will be retried with exponential backoff before the ticket is marked as a failure. The module keeps track of the number of tickets that were successfully updated and those that were not. If all the tickets fail, then the module will output "Failed"; otherwise it will say "Success."

3.2 System Architecture

The task configuration is shown in Figure 1, this is the Load into Tickets task that the user will see and set up. To construct an ETL process, the first extension that must be added to the workflow is an extractor task. A transformer task is typically next; however, this does not need to be added since the extractor outputs the data in the desired format required for the loader. Thus, configuring the new loader extension is the final step in the ETL flow. Upon opening the Tickets extension, the user will select the extractor data source, which then immediately auto-maps the fields.

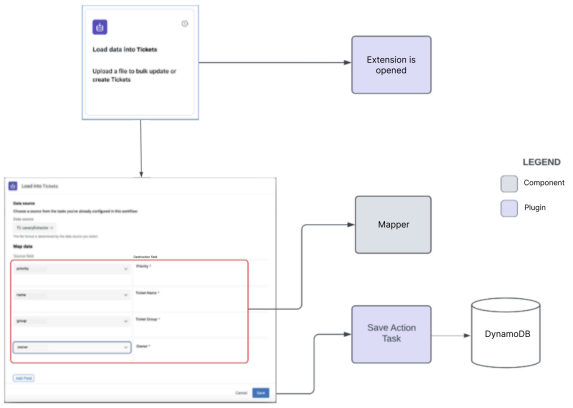

Figure 1: Task Configuration Flowchart

At least one field needs to be updated to enable the save button. Once all the mappings are complete, the user hits save, which then stores the payload containing all this information in an Amazon DynamoDB. This configuration of the workflow only needs to happen once.

The workflow could be set up to run daily, weekly, monthly, etc. The task at runtime is shown in Figure 2.

When the workflow begins to run, it will execute the tasks in the order that it was configured through a messaging system. The first module extracts the ticket data and then pushes it into an Amazon S3 bucket. The red outline displays all that is handled in the Load into tickets task. When the loader task is invoked, it will pull the ticket data from the S3 bucket and then perform all the POST requests to the database. This could take a few seconds to a few hours, depending on how many ticket updates are to be made.

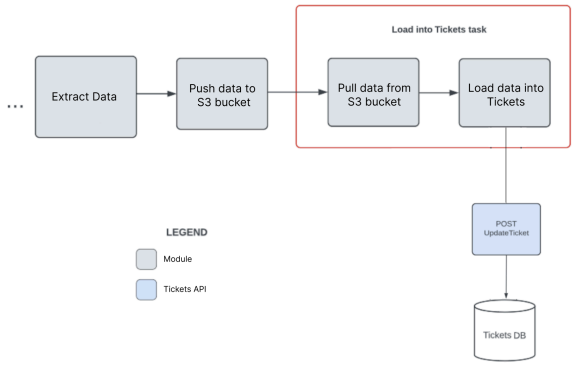

Figure 2: Task at Runtime

4. RESULTS

After thorough testing, the Load into Tickets task was added to each staging environment and production. It updated 60,000 tickets in 16 hours and 50 tickets in five seconds. However, the task is only visible for selected customers before it will be available for all brands. Customers who use the Tickets feature often will be the first to use this extension.

It used to take several days for a user to manually update thousands of tickets or to contact company support and ask them to help set up an abundance of unnecessary workflows. Now, the user can easily set up one single workflow themselves and let the loader task handle the rest in under 24 hours. What was once a tedious process, became a simple and effective automated job that boosts customer satisfaction and saves a significant amount of time and effort.

The loader task also implemented many safety measures when sending thousands of requests to the Tickets API, as opposed to customers who could accidentally overload the system during their manual updates. Not only did the task reduce customer frustration, but it also alleviated the stress placed on developers who have to constantly attend to problems when they arise.

5. CONCLUSION

The Load into Tickets task has simplified the process for bulk updating tickets by implementing an ETL system. Customers no longer have to perform manual updates themselves, such as sending thousands of requests to the public API or asking developers to create an excessive number of workflows. Instead, users now only need to set up a single workflow with the tickets task plugin and let the automated job perform the updates in under 24 hours. This will be extremely beneficial to customers that need to have tickets resolved quickly and simply. Reducing the effort required to update thousands of tickets will significantly increase customer satisfaction and attract more users to begin to use the company's ticket feature. Bulk updates will now be performed efficiently and no longer be considered a time-consuming, strenuous process.

6. FUTURE WORK

Once the workflow with the Load into Tickets task completes, the user will be presented with a modal that shows only the number of tickets that successfully updated and the number that failed to update. While this is helpful, it could be more beneficial to show the ticket keys of all those that failed so that they could be retried separately. The proposed enhancement involves attaching a downloadable CSV file to the modal screen. This file would include a line per ticket that indicates the update status. If the status is failed, then it will provide the reason for failure or an error code. This would give the user a better indication of what they need to reconfigure in the next workflow run.

The project could also be expanded if it were to include bulk creating tickets, along with bulk updating. Creating tickets in bulk has also been a request from the customers. Since the Load into Tickets task for updating tickets has already been implemented, it would be easier to add a similar feature. It would simply require adding code to the backend module that sends the create tickets requests to a different API URL. Nonetheless, the Load into Tickets task will remain extremely valuable for customers.

7. ACKNOWLEDGMENTS

I would like to thank all the members of my internship team, especially my mentor and manager for their endless support and feedback.

REFERENCES

[1] Wickramasinghe, S. (2021) *What's ETL? extract, transform & Load explained*, *BMC Blogs*. Available at: https://www.bmc.com/blogs/what-is-etl-extract-transform-load-etl-explained/ (Accessed: September 22, 2022).

[2] Sun, K. and Lan, Y. (2012) *SETL: A scalable and high performance ETL system*, *IEEE Xplore*. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6340727 (Accessed: September 22, 2022).

[3] Harrison, P., Chapman, N. and Beaton, C. (2022) *Glue*, *Amazon*. World International. Available at: https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html (Accessed: September 22, 2022).

[4] Galici, R. *et al*. (2020) *Applying the ETL process to blockchain data. Prospect and findings*, *MDPI*. Multidisciplinary Digital Publishing Institute. Available at: https://www.mdpi.com/2078-2489/11/4/204/htm (Accessed: September 22, 2022).