Josh Krowiorz

2.22.21

*Cloud platforms such as Azure and AWS help customers build out simple ETL pipelines (Extract, Transform, Load) that are easier to manage, and are very robust. Briefly describe your experiences with cloud technologies, and explain how you would design a data pipeline to load data from a source (such as a public API or FTP server) to some sort of cloud storage that you define.*

My experience with cloud technologies is mostly with AWS. While I was with the Brewers, we leveraged S3 and EC2 instances, and used their Elastic Transcoder extensively to process raw scouting video and prepare them for online streaming. During our initial eBis API ETL migration we began using AirFlow to handle job orchestration, which is much more flexible and efficient than the on-prem single-threaded scheduling methodologies we were used to. The Reds use Azure Databricks, but while consulting for them I was a step removed from the “E” and worked more on the “T” and “L” in on-prem MS SSIS.

My experience with ETL goes back to DTS in MS SQL2K. In fact, one of my first projects for the Brewers was to convert our eBis flat file ingestion pipeline from DTS to SSIS (before eBis API and before cloud solutions were prevalent and in fashion).

As for how I would design a data pipeline to do something like an eBis load, I don’t think there are *that* many choices to make beyond the particular tools and services. Best practices are out there for a reason. I’d envision something like Azure Data Factory to do the extracting into a raw ingestion DB of your choice (maybe Datalake). It’s important to extract only what you need, to store it exactly (I like to have a ‘book of record’ to reference prior to any transformation), and store it cleanly (avoid duplicates and unnecessary information). Implementing logging and monitoring are absolutely essential. I don’t like it when the GM finds out about a data issue before I do. Scheduling is paramount, especially when the process is intensive; avoid running during business hours and game time. Utilize staging tables and development and QA environments. Data should be pushed to production only after it’s gone to staging (which should be an exact production replica). Take backups on a schedule that makes sense, and that allows us to get back up and running quickly with the most recently accurate snapshot available in the case of a catastrophic failure.

I think once you have this basic framework in place, you can turn to optimization. Is the process taking too long end-to-end? Even if the answer is no, there are probably opportunities to improve performance. This could be anything from query optimization to distributed or parallel processing with something like MapReduce or Spark.

Optimization is usually most important during the “T” stage of ETL. And “L” is where we can customize our approach a little more. Do we want to keep a large DW of everything, basically a replica of MLB’s eBis DB? Do we want to scale data down into a reporting DB or sandbox for analysts? Do we need to roll data up so that it’s quicker to load into a web form?