The Details: ETL

More on this hilariously obscure topic

Justin
Oct 27, 2020

Hello there loyal subscribers! Trying out a new format this week, based on popular request - I'm going to **dive into the details** of a previous post - where we talked about ETL - and go deeper on use cases and tools in the ecosystem. If you have thoughts about the format, reply and let me know!

Refresher: what's ETL

ETL is the process of **moving and transforming data** to get it into a more useful format.

Every company has this idealized vision of a data science and analytics team, with full visibility into how the business is doing, how the product gets used, how experiments are performing, super good looking and funny people, etc. The problem with getting there (and this is part of why data teams don't get hired until later in the company lifecycle) is that the actual, cold hard data that you need to answer important questions typically lies *all over the place*. And it needs cleaning.

I've decided to illustrate this principle with a sadly very real set of examples from my experience:

Data	Questions	Source
Product usage	Which features get used the most? Where are users dropping off in onboarding? What are our most popular integrations?	Segment, production databases
Billing	How much money are we making monthly? Who are our biggest customers?	Stripe, internal billing systems
Website engagement	What are our most popular pages? What's the conversion rate on our homepage? Is anyone reading these fucking blog posts?!?	Google Analytics, Segment
Salesforce	How many opportunities did we open this week? What's our conversion rate from leads to customers?	...Salesforce
Paid marketing	How many leads are we generating from paid marketing? What are our most effective sets of creative and copy?	Google AdWords, Facebook Ads, Twitter Ads, LinkedIn Ads, etc.

All of this data sits in different systems - and what that means in *practice* is that you can't just write a big SQL query that joins all of it together. You need to get it *out* of the source system and *into* a big fat warehouse so you can ask the questions you want. And what compounds this problem is that the **value** of this data is multiplied when it's **integrated**, e.g. you can combine these sources together:

- Website engagement + billing = which site pages drive signups who end up paying us the most money?
- Paid marketing + Salesforce = which ad channels are driving the most revenue and large customers?
- Product usage + Salesforce + billing = what features do our biggest customers use the most? What are early product signals of high engagement?

For those strategy minded citizens among us, you'll notice that each of the vendors listed in the above table are actually incentivized to help you analyze the data they're generating **in their own tools and not out of them**. So Stripe has Sigma, Salesforce has Dashboards, Google Analytics is a combined tracking tool

and analytics tool, etc. They'll let you get data out, but they want you to spend as much time in their software as possible, so they'll make it easy to analyze it in place.

So ETL - **Extract, Transform, Load** - is the process of getting siloed data **out** of source systems and **into** a central data warehouse, all together, where it can be analyzed easily and quickly.

- **Extract**: get data out of siloed source systems like Salesforce, Stripe, etc.
- **Transform**: clean dirty data, reorganize it in more useful formats, join data together
- **Load**: put the results of all of this into a data warehouse

This is a ubiquitous process - if you're a company and you want to get value out of your data, you're going to need to do some ETL sooner or later. And the canonical unit of ETL - **one ETL job** - can be as small as a few lines of SQL, or as big as a 20 step Python workflow.

Use cases / how it happens

Each job has (basically) 3 things that you need to worry about:

1. A **query** (in SQL, Python, etc.) - the actual code to get the data, transform it, etc.
2. A **schedule** -when the job should run (e.g. every day at 12PM)
3. A **dependency** list or graph - what other ETL jobs need to run first

Unfortunately no catchy acronym here.

→ A basic example

Let's walk through a very basic ETL job that aggregates the amount that we've charged each one of our customers in Stripe. Starting with the query, here's the actual mechanics of what we want to do with source Stripe data: aggregate the total number of charges, and the total amount, per user.

```
SELECT
  user_id,
  SUM(amount),
  COUNT(*)
FROM users u
LEFT JOIN stripe.charges c ON u.user_id = c.user_id
GROUP BY 1
```

After we've got the query written, we need to figure out a schedule (let's say we want this to run twice a day) and then identify if there are any dependencies for the query - since this just pulls directly from source Stripe data, there are none.

→ **A more complex example**

The query is really the hard part, in my experience - some of the queries I've built for ETL jobs can be hundreds of lines long. A good example is building **state from events**. A lot of product data exists in event formats, where every time a user does something like clicking on a button, a row gets added to the database that says that they, well, clicked a button.

At DigitalOcean, we fired an event every time a user created a new virtual machine, shut it down, made it bigger, etc. So here's what our events table might look like:

user_id	event_type	resource_id	timestamp
1	resource_created	1	'2020-01-01'
1	resource_edited	1	'2020-01-02'
1	resource_destroyed	1	'2020-01-03'
2	resource_created	2	'2020-02-01'

What if we wanted to know the **current state** of any particular resource? E.g. is resource number 1 currently active or was it shut down? It turns out that to extract that information from the above format isn't exactly a piece of cake in SQL, so the ETL job you'd build to do that can run a couple hundred lines. And it did, when I built it for our managed databases product when I was working there.

> 🔍 **Deeper Look** 🔍
>
> Earlier, we mentioned dependencies - this job had one too, and it was on another job that extracted these raw events into a cleaner, more usable format. So that "cleaning job" needed to run *first*, *before* this building state job runs. And that's where things can get pretty complicated.
>
> 🔍 **Deeper Look** 🔍

Tooling and software

The ETL tooling space is experiencing somewhat of a revolution right now, driven by an increasing standardization of ingestion formats and cheaper storage.

- **Standardization**: more and more companies are using Segment to route their customer data, so it all is starting to look the same, meaning that it's easier to work and integrate with if you're a scheduling tool (yes, I will be writing about Segment soon)
- **Cheaper storage**: with managed data warehouses like BigQuery and Snowflake, it's cheaper and easier than ever to store a lot of data, and, of course, you'll want to work with that data more

Here's the basic landscape:

→ **Cron**

The simplest and most OG ETL tool is Cron. It's a system utility that exists on most operating systems, and it lets you schedule a program to run every whatever - hour, minute, etc. If you had the code for your ETL job in a Python file called main.py, you'd run it in Cron using syntax like this:

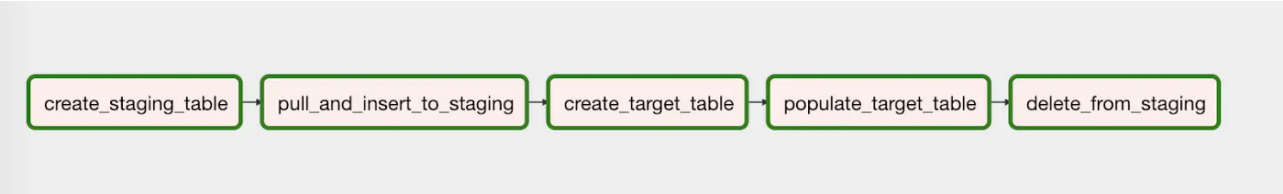
python main.py 0 * * * *

The 0 * * * * part tells your system to run the job every hour. This language isn't exactly intuitive, so there are sites like crontab.guru that help you figure out which numbers to use.

→ **Airflow**

Airflow is like Cron on steroids, purpose built for scheduling ETL jobs. It was originally developed at Airbnb, open sourced, and is now probably the most popular tool (at least open source one) for this kind of work - but it's quite complicated to use, and you need to write very specific Python code as opposed to using your normal SQL.

Airflow is centered around the concept of **DAGs**, or directional acyclic graphs. Think of it as a flowchart for how your jobs need to run - this job needs to run before that job, which needs to run at the same time as this other job, etc. Here's what a (very) basic DAG looks like:



I wrote a beginner's guide to Airflow here if you're interested in going deeper.

→ DBT

DBT (Data Build Tool) is the hot kid on the block right now. It's an open source framework for building your data warehouse (among other things) and scheduling those jobs - and, funny enough, it's built and maintained by an analytics consulting firm called Fishtown. I'm currently very deep in DBT at work, and it's quite good.

→ Other Stuff

This space is getting really popular, so there are a lot of new tools popping up. A couple of examples:

- Prefect.io – a more ergonomic and less opinionated version of Airflow, built by a former Airflow contributor (I'm an advisor 🙂)
- Dagster – an open source scheduling utility
- Luigi – an open source scheduling utility from Spotify
- Dataform – a tool for building pipelines for your data warehouse

Not all of these are new, but the basic message is that there's a good amount to choose from right now.

Terms and concepts covered

`ETL`

`Job`

`Dependency`

`Scheduling`

`Airflow`

`Cron`

`DBT`

Further reading

- DBT's explainer of what they do is a generally useful foray into the ETL ecosystem
- StitchFix wrote an opinionated post about how Data Scientists should own their ETL pipelines

---

Comments

Write a comment...