What's a Data Lake?

And why is everything water related?

Justin
May 25, 2022

83 2

Hello lovely Technically subscribers. This week's post is about Data Lakes, but before then:

If you work in data, you've probably heard of Coalesce, the analytics engineering conference. I just submitted my proposal, I'm hoping to talk about how data teams can write better and more interesting blog posts. Highly recommend applying to speak! You can do so here, only a few days left.

(this is not sponsored, I just want good talks at Coalesce this year)

Now, on to our regularly scheduled content.

The TL;DR

A **Data Lake** is an **unstructured place to put data**. It's usually meant for long term storage and infrequent querying.

- Companies are collecting more data than ever before (yada yada), but not all of it gets used immediately
- A data lake is a place to **drop data**, in an unstructured format, usually for long term storage
- Data lakes are easiest to understand **in contrast to data warehouses**, where schemas are defined in advance
- A new trend is seeing data lakes **act as quasi data warehouses**, and it's possible the categories might…merge?

You'll often see data lakes as part of those infamous infrastructure diagrams that nobody (myself included) understands. So it's worth understanding why companies use them, and where they fit into the modern data stack.

Use cases and background

To get into why a company might use a data lake – and how it's different than a data warehouse – we need a better sense of what kinds of data companies are working with. *Generally*, you can think of companies as generating and working with two broad categories of data:

1. **Structured operational data**

Most companies generate relatively simple data from their day to day operations. A few examples:

- Data about your users (for authentication and beyond) and organizations
- Data about products you're selling and orders your customers make
- Every SaaS app has a production database with useful information to analyze
- Invoicing and payment data for your products

This kind of data is generally **structured**, generated at a **reliable cadence**, and not massive. It's likely already being stored in a system that's either your own (a production database) or a 3rd party's (e.g. if you're using Stripe for payments).

1. **Unstructured data**

Beyond the basic structured operational data you're capturing, you might have other data that gets generated through the course of business that's interesting to analyze. A few examples:

- If your product involves any sort of text generation, you might want to store that text (e.g. Twitter stores the content of tweets)
- Log data for any server or application performance monitoring
- Website click and pageview data
- Granular product usage data

These types of data are generally unstructured and don't necessarily fit well into a relational data structure like a SQL database. And even if it *can* fit into a structured schema, the data might be either (a) too voluminous, or (b) being generated too quickly to go in your data warehouse (more on this later).

Once you've stored all this data, what do you do with it? Analytical use cases generally split into two categories (which, conveniently, can map to the categories of data above):

1. **Operational, regular cadence, reporting**

Companies have day to day questions they need to answer on a regular cadence:

- How many users do we have?
- How much revenue did we make this month?
- What's our churn rate?

The answers to these questions make their way to a graph on a dashboard somewhere, and get checked regularly by the data team, management team, and whoever cares about the answers.

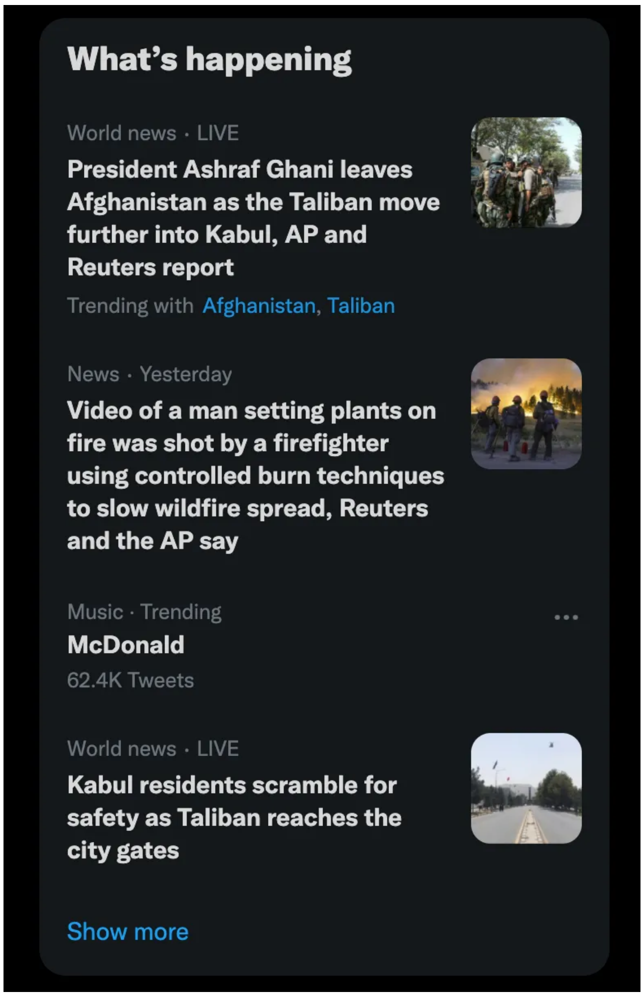
The *data* that answers these questions usually comes from our first category of data in the Technically Taxonomy™, **structured data**. Knowing how many users you have will probably rely on your users table, revenue will come from a revenue table, etc. It's unlikely that unstructured data has a bearing on your company's core metrics [1].

1. **Project basis, ML, real-time**

Beyond regular reporting, there's a lot more that companies do with their data! Teams might want to do a deeper analysis on a particular metric, build a Machine Learning model to better understand the user base, or funnel real-time data that personalizes a piece of the product experience.

Sometimes, the data that teams need to do this kind of deeper work is structured. When I was at my last job, I was tasked with building a model for understanding the churn profile of our customers, and it turned out that most of the useful data was the basic operational stuff that we already had stored.

But a lot of times, it relies on that deeper, more detailed, unstructured data that companies collect. Twitter does something like this to surface trending topics:



What's happening

World news · LIVE
President Ashraf Ghani leaves Afghanistan as the Taliban move further into Kabul, AP and Reuters report
Trending with Afghanistan, Taliban

News · Yesterday
Video of a man setting plants on fire was shot by a firefighter using controlled burn techniques to slow wildfire spread, Reuters and the AP say

Music · Trending
McDonald
62.4K Tweets

World news · LIVE
Kabul residents scramble for safety as Taliban reaches the city gates

Show more

To do this (I'd guess), they're looking through tweet content in real time.

Data warehouses vs. data lakes

Now that we've got a better grounding around what data teams are doing to generate and use company data, the whole "data warehouse vs. data lake" discussion is going to make a lot more sense. The TL;DR is that:

- **Data warehouses** are good fits for structured, regularly occurring data
- **Data lakes** are good fits for unstructured, special use case data

> 🧠 **Jog your memory** 🧠
>
> Data warehouses are special purpose databases built for storing and querying analytical data.

> 🧠 **Jog your memory** 🧠

Let's start with the technical definition that differentiates these two products, and then dive into why each is a good fit for its respective use case.

1) Schema on write vs. schema on read

When you adopt a data warehouse like Snowflake, BigQuery, etc. you need to define your schema up front (this is a general feature of any relational database). For example, if you want a table in your data warehouse that tracks information about your users, you need to decide in advance what the columns are, what data types they contain, etc. This is **schema on write**, e.g when you write (put) the data into the warehouse.

In a data lake, on the other hand, you can just throw data in there without any specific structure. It's when you get data *out* of the data lake that you need to define its schema so you can actually use the data. This is **schema on read**.

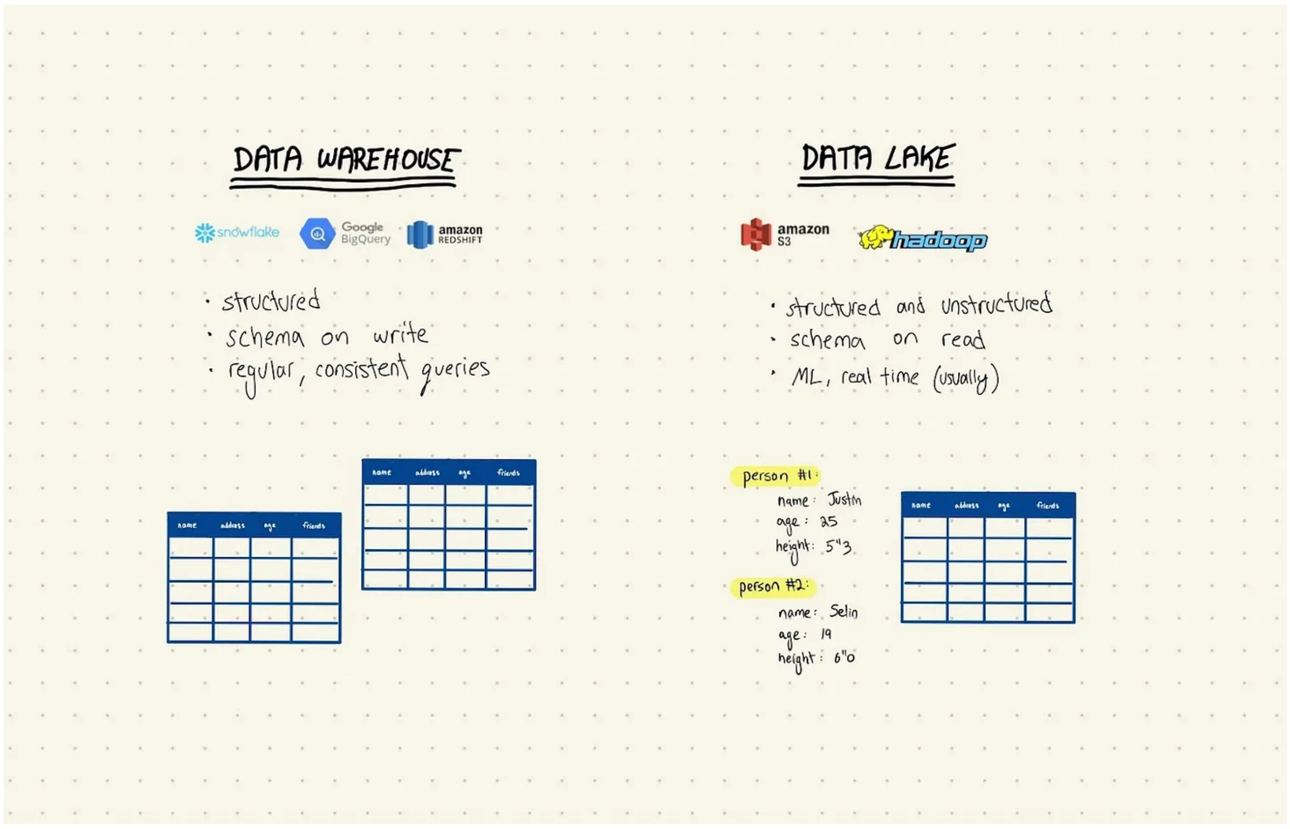
> 🔍 **Deeper Look**🔍
>
> Under the hood, most data lakes use **object storage** as the mechanism for storing data. It treats each "unit" of data as an independent object, with associated metadata and an identifier. That less structured mechanism makes it easier to scale and replicate across different data centers.
>
> 🔍 **Deeper Look**🔍

By the way, schema on read doesn't refer to any specific product feature – it just means that you need to deal with your "messy" data when you pull it out of the data lake, as opposed to before you put it in.

2) Regular reporting vs. special projects

The tradeoff between schema on write and schema on read is about **reliability vs. flexibility**. When you're pulling data for your daily reports – and the query you're writing today looks awfully similar to the one you wrote yesterday, and the day before – the structure that the data warehouse provides is nice. But if you're gathering and working with data for a one-off ML project, that overhead might not be worth your time.

Schema on read is a tradeoff like any other. It means it's easier to store data – because you don't need to clean or structure it when you drop it in – but it leaves you with more work to do when analysis time comes. Having unstructured data sitting around without any schema can also lead to longer term data hygiene and governance issues.



This is why – like most technology choices – the decision between warehouses and lakes comes down to use cases. If there's one piece of wisdom I can leave you with, dear readers, it is this.

This is why in general, the question for most companies isn't "data warehouse *or* data lake" – instead, it's about **how to use the two in tandem most effectively**. In SaaS at least, most startups will start with a data warehouse, and then potentially invest in a data lake once they're generating more unstructured data and the company grows.

> 🚨 **Confusion Alert** 🚨
>
> If you're seeing that "unstructured" keyword and wondering…isn't this just NoSQL? NoSQL is still a database – it's *specifically* for unstructured data, and

> usually supports production use cases. Data lakes are really just giant buckets for storing analytical stuff – they're not made for running apps.
>
> 🚨 **Confusion Alert** 🚨

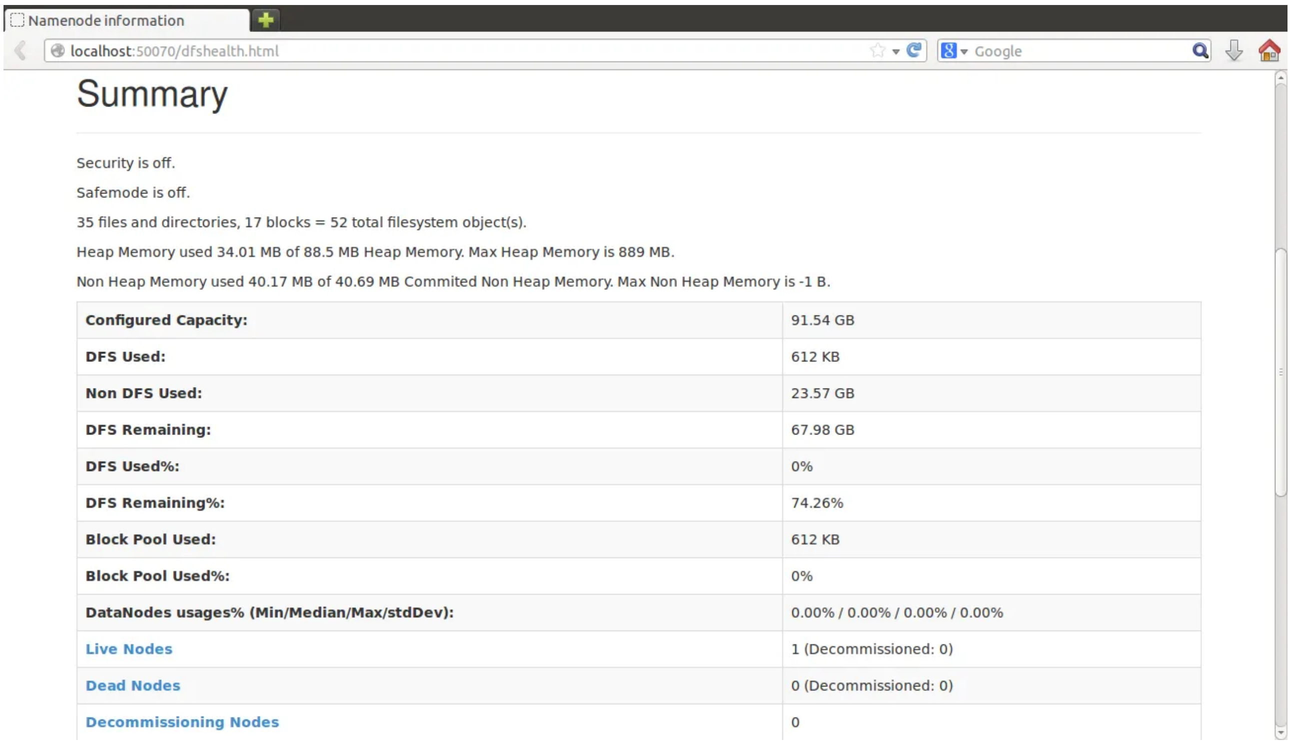
Popular data lakes + DeltaLake

What data lakes do people actually use?

The most popular data lakes are just the native object storage products that each major cloud provider offers. AWS's object storage solution – S3 (Simple Storage Service) – is probably the most widely used data lake right now. S3 is a general purpose product – your favorite apps probably use it to store profile pictures – but AWS has some ancillary products and services that format it for use as a data lake, e.g. LakeFormation lets you set data access policies for your data in S3.

In the open source world, HDFS – the Hadoop Filesystem – was one of the first data lakes to hit mainstream popularity. It's a framework for setting up distributed file systems across multiple servers, and teams used it for storing unstructured data too.

> 🧷 **Workplace Example** 🧷
>
> At DigitalOcean, we used HDFS as our data lake before eventually migrating to Snowflake. To manage the schema part, we used another piece of the Hadoop ecosystem, Hive, to organize the raw data into partitions and write SQL against it. The end result was chunks of data stored as objects in HDFS, and a system on top to query them as if they were a series of database tables.
>
> 🧷 **Workplace Example** 🧷



Summary

Security is off.

Safemode is off.

35 files and directories, 17 blocks = 52 total filesystem object(s).

Heap Memory used 34.01 MB of 88.5 MB Heap Memory. Max Heap Memory is 889 MB.

Non Heap Memory used 40.17 MB of 40.69 MB Commited Non Heap Memory. Max Non Heap Memory is -1 B.

Configured Capacity:	91.54 GB
DFS Used:	612 KB
Non DFS Used:	23.57 GB
DFS Remaining:	67.98 GB
DFS Used%:	0%
DFS Remaining%:	74.26%
Block Pool Used:	612 KB
Block Pool Used%:	0%
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	1 (Decommissioned: 0)
Dead Nodes	0 (Decommissioned: 0)
Decommissioning Nodes	0

Recently, some open source tech from Databricks has been making waves. They call it DeltaLake, and it purports to give you the best of data warehouses (structure) *and* data lakes (unstructured, flexible). To quote myself:

> *"The new Delta Lake product purports to give you data warehouse speeds when querying your data lake, so you can keep storage costs really low. Logistically, Delta Lake is an open source layer that sits on top of your typical data lakes, like S3 or HDFS. I think the open source version is from Databricks originally, but they seem to be deliberately obscuring that fact under the guise of a shell company called LF Projects. If only I ran a true crime podcast…"*

Whether this will last remains to be seen (see this post about migrating from DeltaLake to Snowflake), but it's something to keep an eye on.

Terms and concepts covered

```
Data warehouse
```

```
Data lake
```

`Schema on write`

`Schema on read`

`Partitions`

---

1 As companies get larger and teams get more specialized, this can change. For example, there might be a team at Twitter whose main KPI is increasing the number of words in each tweet, and as such, a core metric of theirs *will* rely on unstructured data (tweet text). But I can't be worrying about that nonsense in this newsletter.

---

2 Comments

Write a comment...

Andreas Freund May 25, 2022

Loved the explainer Justin! The twitter example reminds me of a really good excerpt about describing "load" from Designing Data-Intensive Applications by Martin Kleppman. More related to the topic of structured vs "unstructured" DBs in production-environment than Data Lakes, but I think it does a great job of driving home why the differences matter and the consequences of the trade-offs. Found the excerpt here: https://ebrary.net/64604/computer_science/scalability

Huge fan btw! Thanks for doing this.

♡ 1 Reply Collapse •••

Dickson Pau Writes Desilorance Aug 26, 2022

Would you say the data lake is the central part of the modern cloud infrastructure?

♡ Reply Gift a subscription Collapse •••