Three data scaling pitfalls and how to avoid them

Scaling data infrastructure is hard. But with the right strategy, you can avoid common issues.

Olivia Iannone · Follow
Published in Towards Data Science
8 min read · Oct 28, 2021

Photo by Meor Mohamad on Unsplash

While precise estimates vary, we can all agree on one thing when it comes to data: the amount of it that exists in the world is growing exponentially and will continue

to do so.

This means that most data-driven organizations have already experienced the surprising variety of challenges that stem from data scale-up. Fortunately, being prepared for data scaling challenges and staying aware of best practices can mitigate common issues.

What is data scalability?

The term "scale" is tossed around a lot when we talk about data, often in a frustratingly vague way. That's because it's an extremely broad concept that bundles together many factors, each of which can be very complex.

Data scale can encompass any combination of the following:

- The total size of a dataset
- The total number of distinct datasets that you might want to relate to one another
- The frequency at which data is updated
- The frequency at which data is queried
- The relative difficulty of processing queries and/or updates
- The distribution of data usage (ranging from a single machine to globally)

Without diving too deep into the messy details, we can define **data scalability** as a condition in which you (the human managing the system) don't have to do anything when any of the above factors change.

Data scalability could also be referred to as "data infrastructure scalability." These two concepts are intrinsically linked. After all, it is your infrastructure (the various systems you use to collect, store, and transform data) that must actually flex to handle changes in data scale.

A hallmark of data scalability is that you shouldn't actually notice it happening because your systems should continue to work normally. For example, if the frequency and complexity of queries rise significantly but performance isn't impacted, your infrastructure is scalable. If performance noticeably suffers to the

point where your team isn't able to work, or you're not meeting your goals, your data infrastructure is not truly scalable.

Why data scalability is challenging

Historically, performance issues put a hard stop to data scaling before it could reach a noteworthy size. When the only options were older, on-prem processing and storage technologies, there was a limit to how many servers or nodes we could put in place. Some on-prem architectures are more scalable than others; however, there is always a point of diminished performance.

Although "big data" has been a buzzword since the early 2010s, we only recently reached a point where it's truly attainable, as Matt Turck points out in his recent recap of the state of data in 2021 (you'll need to Ctrl-F for "Big Data"). This revolution has been powered by the rise of massively scalable cloud data warehouses.

Even with the virtually unbounded potential of cloud storage, things can still get messy (and expensive) when it comes time to actually put your data to work.

Reducing cost and increasing performance of large-scale data

To save compute resources and reduce costs when scaling your data, your main objective is to prioritize efficiency and eliminate redundancy, while still retaining as much flexibility in your workflow as possible.

How do you actually do that?

Obviously, the answers to that question vary tremendously. But in our conversations with data-driven organizations, my team has noticed some pain points that appear again and again.

It's worth noting that our company (Estuary's) focus is in the area of real-time data integration and data operations. So, we specifically investigate pain points in data pipelines and the warehouses they feed into.

However, although these examples are common to one sub-field, the underlying principles can be applied across your data infrastructure. By targeting these types of issues, you can improve the efficiency of your systems and re-align with your budget.

Pitfall 1 — Expensive queries in OLAP databases

If you store lots of data without planning for its intended use, you'll eventually run into performance and cost problems. This concept applies to pretty much any type of storage, but we see it the most with OLAP databases.

OLAP (online analytical processing) is a database or data warehouse structure designed to optimize performance in analysis-intensive applications. In contrast to OLTP, which is designed for frequent, small transactions, OLAP aggregates transactions to be less frequent but more complex. Popular cloud data warehouses used for business intelligence, like Snowflake and Bigquery, are OLAP databases.

Though they favor workflows with aggregated, high-throughput transactions, OLAP databases don't do that on their own. Their actual performance — and by extension, their cost — depends on how you use them.

One common cause of poor performance and ballooning costs in OLAP databases is a pattern of expensive querying.

There are a lot of things that can cause queries to be expensive, but a lot of them boil down to complexity.

Joining your data can be a great way to begin to broadly simplify your queries. This may seem counter-intuitive because joining doesn't necessarily reduce data *volume*; in fact, many types of joins increase data volume. What joining *can* do is make your queries way cheaper by consolidating one or more steps.

Unless all your data exists in a single table (doubtful), any query you run will invoke one or more joins. By making pre-joined data available in the database, you can skip this step and cut down the cost of each query.

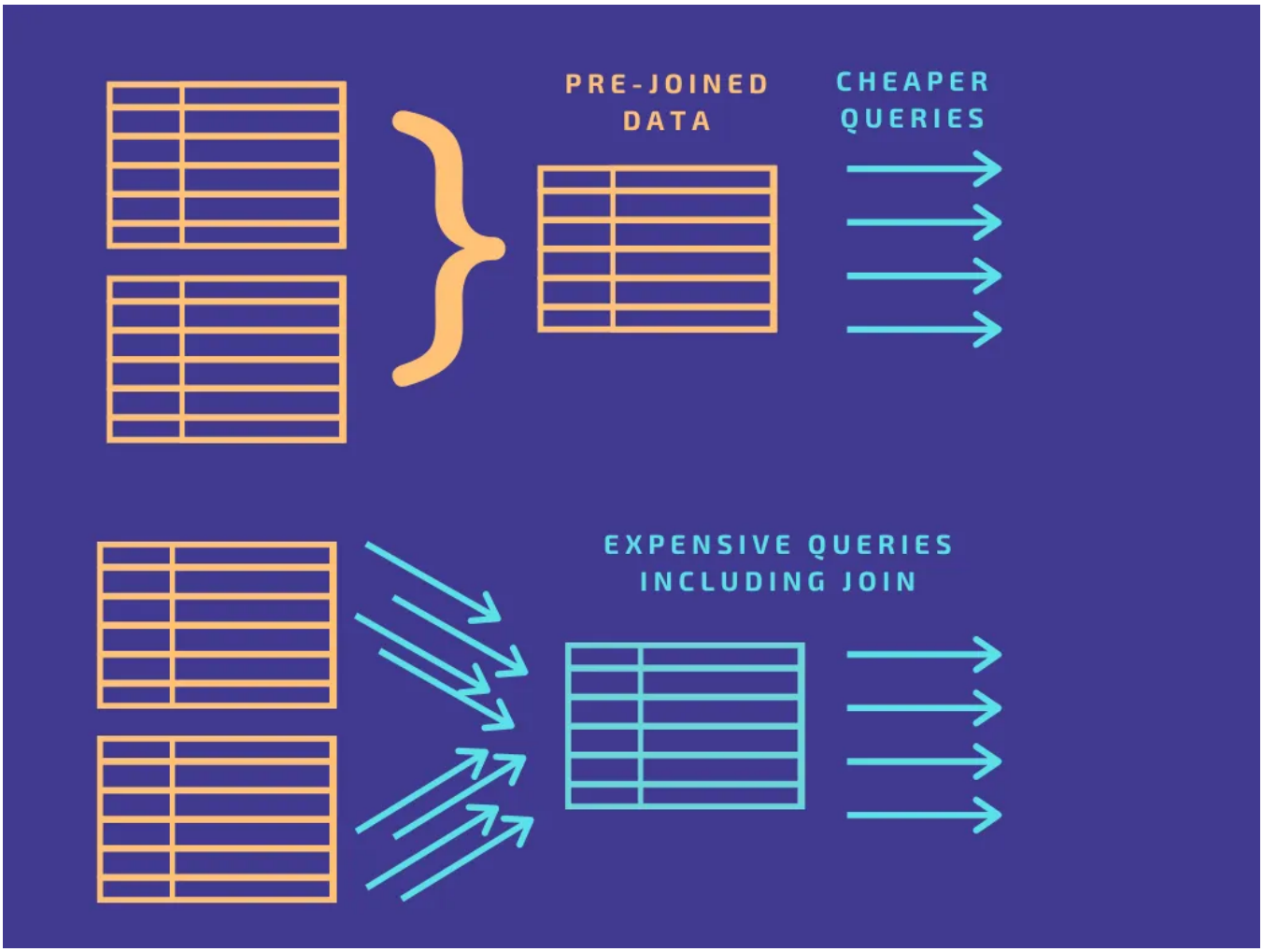


Image created by author with Canva under their Free Media License Agreement

For example, say you're tracking the performance of an online ad campaign across multiple platforms. Joining the tables on ad ID will save compute power for any query about that ad.

The key is to pre-join data *before* loading it into storage. In contrast, joining data that's already in an OLAP database is challenging. Doing so requires you to add several steps to the workflow; namely, adding and querying a materialized view in the database or warehouse (materialized views are actually extremely helpful when used well — more on that later). You'd have to use other tools, like Airflow and dbt, to manage this complex workflow, which adds work as well as latency.

Pre-joining your data is a much more straightforward process. This can be done using an operational transform system such as Spark or Flink, assuming your team includes specialized engineers who work in these systems. Certain newer tools, like Estuary's platform, Flow, can provide a simpler method for powerful stateful joins. This allows any analyst or engineer, regardless of specialty, to pre-join data, so it's a

capability you'll definitely want to keep in mind if you're choosing a data ops or ETL/ELT tool.

Pitfall 2 — Repeating the same query

As we've already discussed, queries over large datasets cost time, compute resources, money, or a combination. One way to mitigate this is to reduce query complexity; another is to reduce frequency.

It's common to find that a large percentage of your queries are repeated quite frequently. They're derived from business questions you need to ask regularly. Questions like:

Which products are selling best right now? How is our ad campaign performing? Have we seen anomalies in recent user behavior?

You may need fresh answers daily, or within minutes or seconds. This is where querying gets *really* expensive.

Enter the **materialized view.**

A materialized view is a smaller data object that contains the subset of data resulting from a specific query. Whereas a query happens after data is loaded, a materialized view is a precomputation. The heavy lifting of the computation is done once, and changes to the data are incorporated as they occur, making subsequent updates to the view much cheaper and more efficient than querying the entire database from scratch.

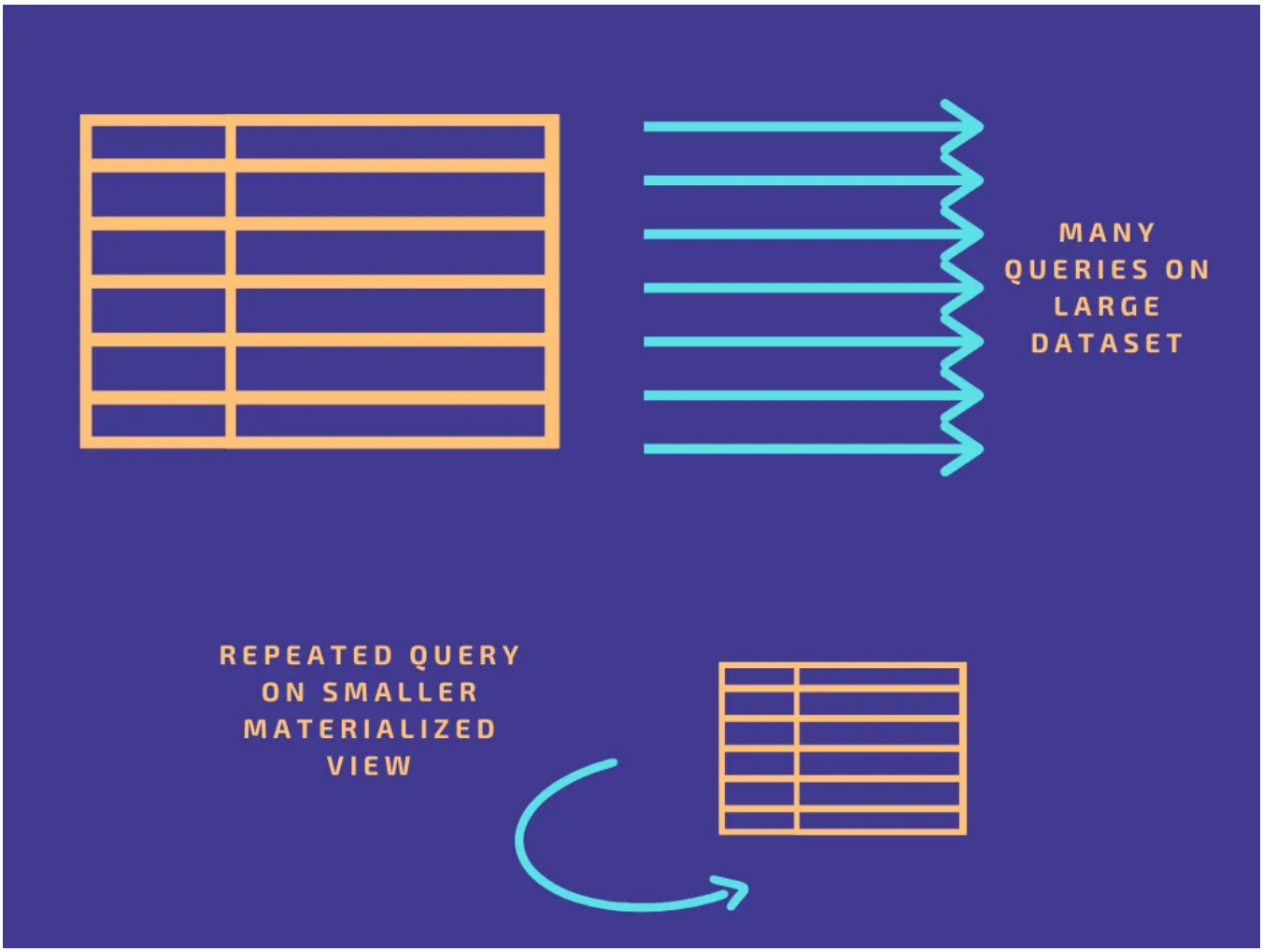


Image created by author with Canva under their Free Media License Agreement

You can create a materialized view directly within the database or data warehouse of your choice. Alternatively, you can use the tool that loads your data, such as Flow, to create the view. This is especially helpful if you want to have the same materialized view in multiple destinations, without managing the manual configuration steps for each warehouse or database.

Pitfall 3 — Rigid pipeline architecture

As you can see, healthy data scaling usually requires some form of transformation to maximize efficiency. Pre-joining and materialized views both fall under this umbrella, as do other types of aggregations and filtering you may use to reduce data volume.

These transformations can occur in different stages of the data lifecycle, so it's important not to limit where your architecture allows you to perform them. Keep in mind that your method will likely change in the future!

When you're designing a data pipeline or choosing a vendor to help you, don't get hung up on the idea of the divide between ETL and ELT. This is a limiting

framework, and modern architectures are able to be much more flexible.

For instance, it can be tempting to prioritize flexibility by choosing an ELT model: load all your data into the data warehouse to be transformed on an as-needed basis. By definition, transformations often result in a loss of detail, so you may want to store *all* data and keep your options open for future queries. However, this large volume of data will diminish performance.

A better solution could be to perform a lighter transformation before loading into the warehouse, where you'll retain an intermediate level of detail. For example, you might choose to aggregate incoming data in one dimension before storage, and have the potential to perform precise queries in the warehouse later. And if your pipeline uses a data lake as a staging area before uploading to your warehouse, as many do, you can retain the raw data there as a backup in case it's ever needed.

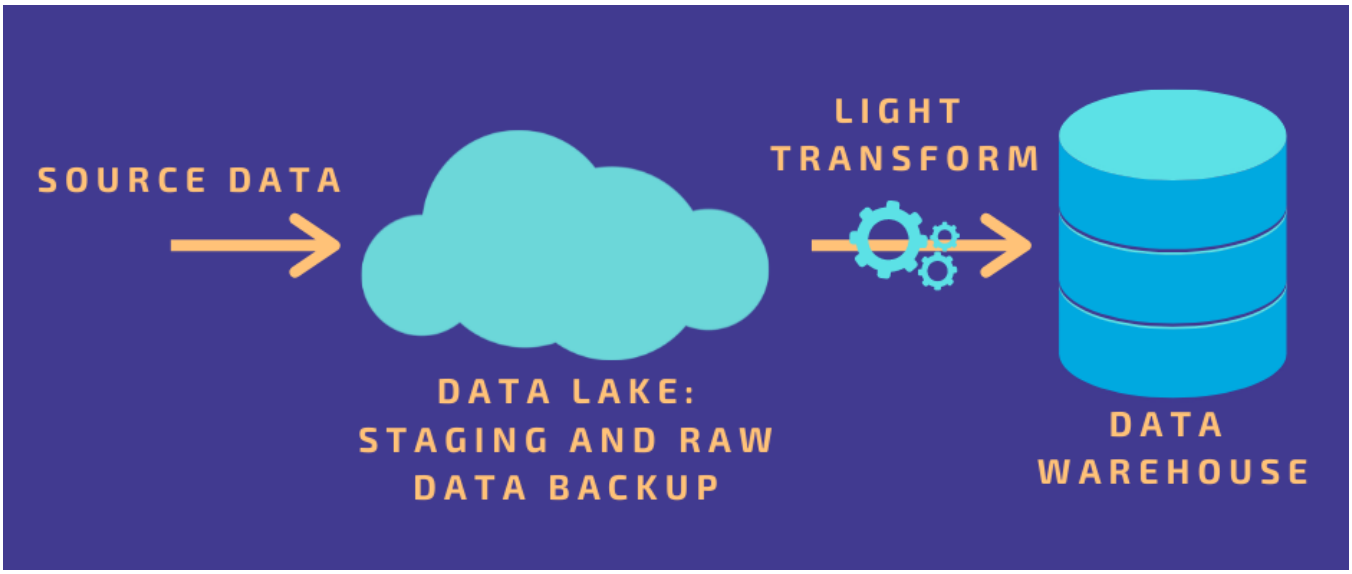


Image created by author with Canva under their Free Media License Agreement

Of course, the best approach for your organization will vary widely depending on your data structure and your business questions. So, your most important task up front is **making sure your engineering team has the ability to change the strategy over time.**

TL;DR

Data scalability is a broad topic that encompasses many aspects of your data infrastructure.

The three pitfalls we've discussed aren't all-encompassing, but they have a common theme: **you can improve your data scalability by applying transformations wisely**

and allowing yourself the flexibility for future changes.

Postscript: scaling batch and streaming data

There is one more consideration that I'd be remiss if I didn't mention. So before we wrap up this article, I'm going to leave you with a big caveat to think about.

This discussion has focused on a batch data paradigm. At this point in time, many organizations use only batch data. But if you rely on real-time data streaming, scaling can get a lot more complicated.

That's because, essentially, the compute resources that process streaming data must be re-distributed on the fly to match the amount of data flowing through. This means that scaling streaming data is a more complex problem that demands the attention of one or more specialized engineers, or a dedicated streaming solution.

As an author, I'm particularly excited about this problem because it's what my team at Estuary is solving with our real-time data pipeline platform, Flow. You can try Flow out here, no strings attached.

A version of this post was originally published on the Estuary blog.

Scalability Scaling Data Data Pipeline

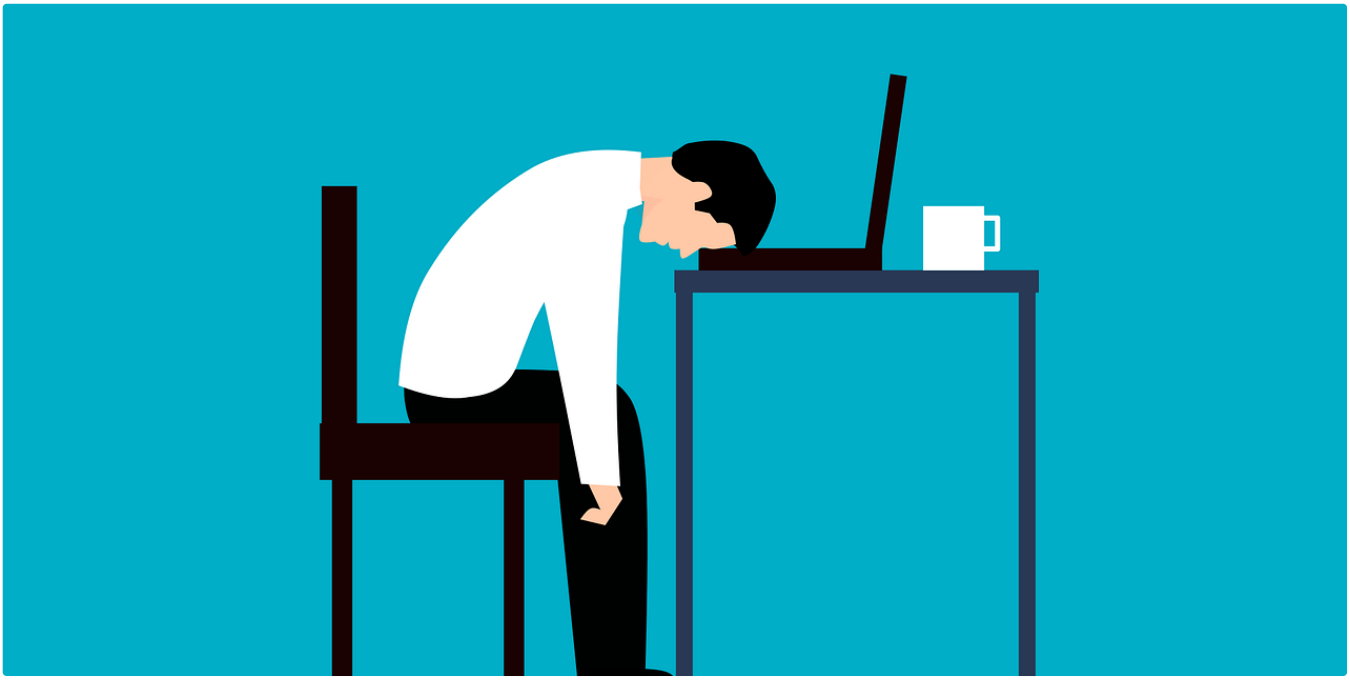
Follow

Written by Olivia Iannone

237 Followers · Writer for Towards Data Science

Writing about real-time Data Ops & more @ Estuary Technologies. Content for the full breadth of data stakeholders.

More from Olivia Iannone and Towards Data Science



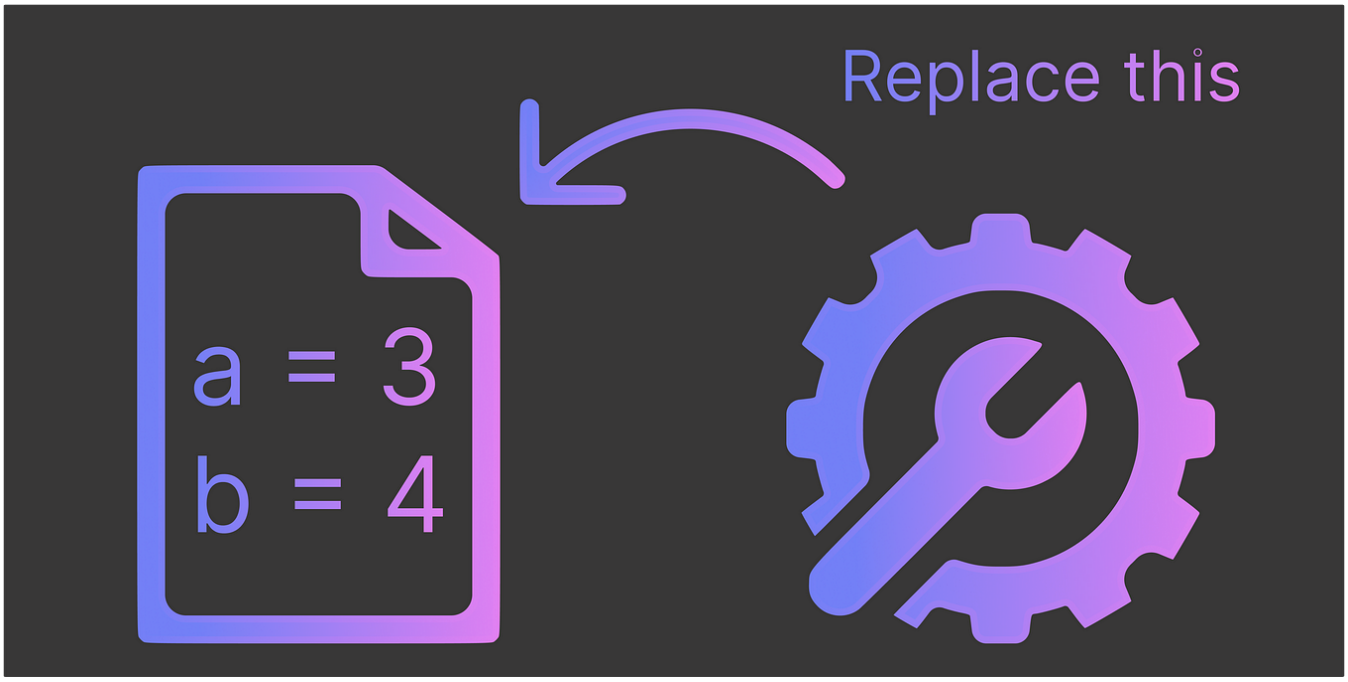
 Olivia Iannone in Towards Data Science

Re-evaluating Kafka: issues and alternatives for real-time

Kafka's challenges have exhausted many an engineer on the path to successful data streaming. What if there was an easier way?

8 min read · Sep 22, 2021

154 8



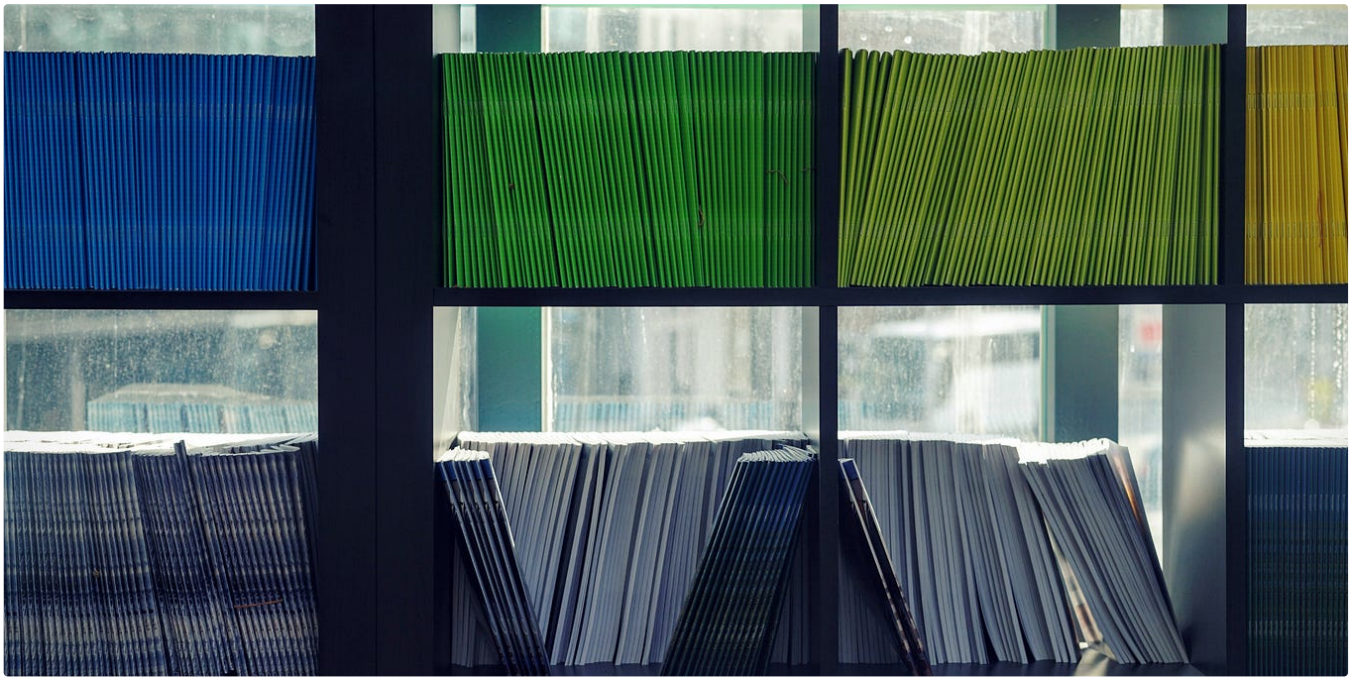
Khuyen Tran in Towards Data Science

Stop Hard Coding in a Data Science Project—Use Config Files Instead

And How to Efficiently Interact with Config Files in Python

· 6 min read · May 26

2K 21



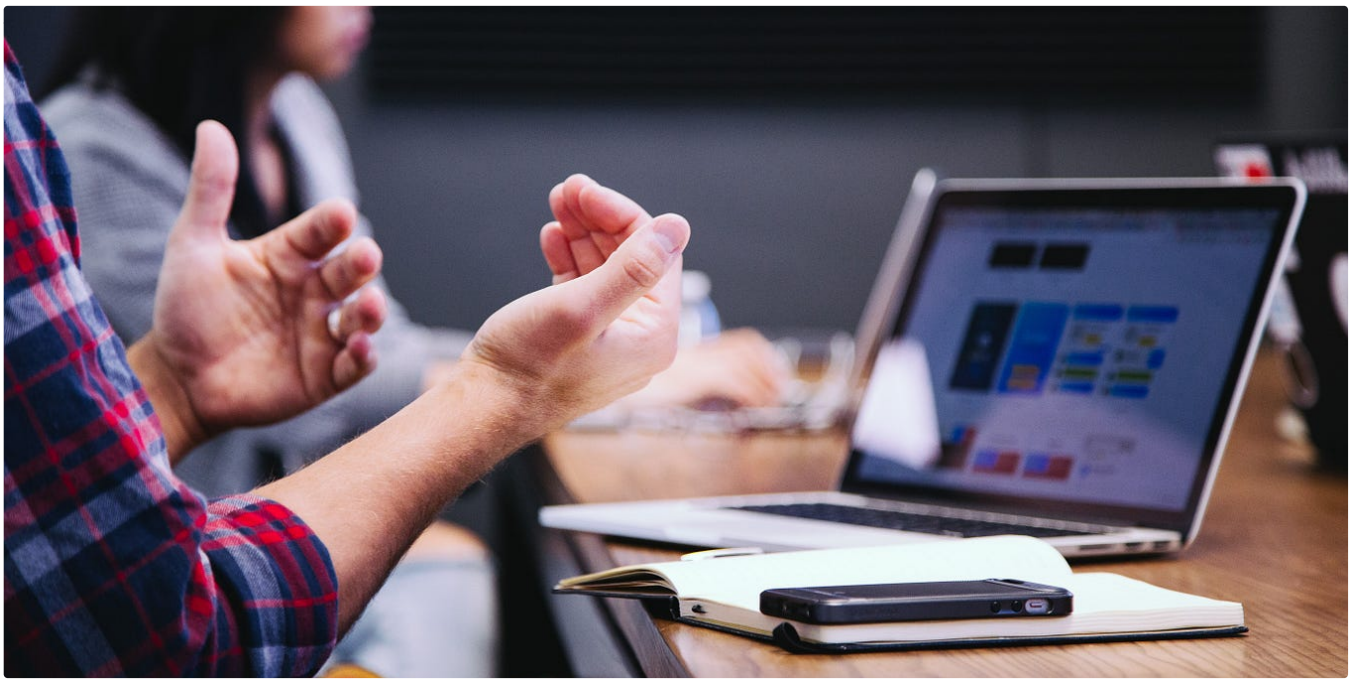
Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with OpenAI

And how you can do the same with your docs

15 min read · Apr 25

4.3K 49



Olivia Iannone in Towards Data Science

Four Software Engineering Best Practices to Improve Your Data Pipelines

From agile to abstraction, thinking about data the way we think about software can save us lots of grief.

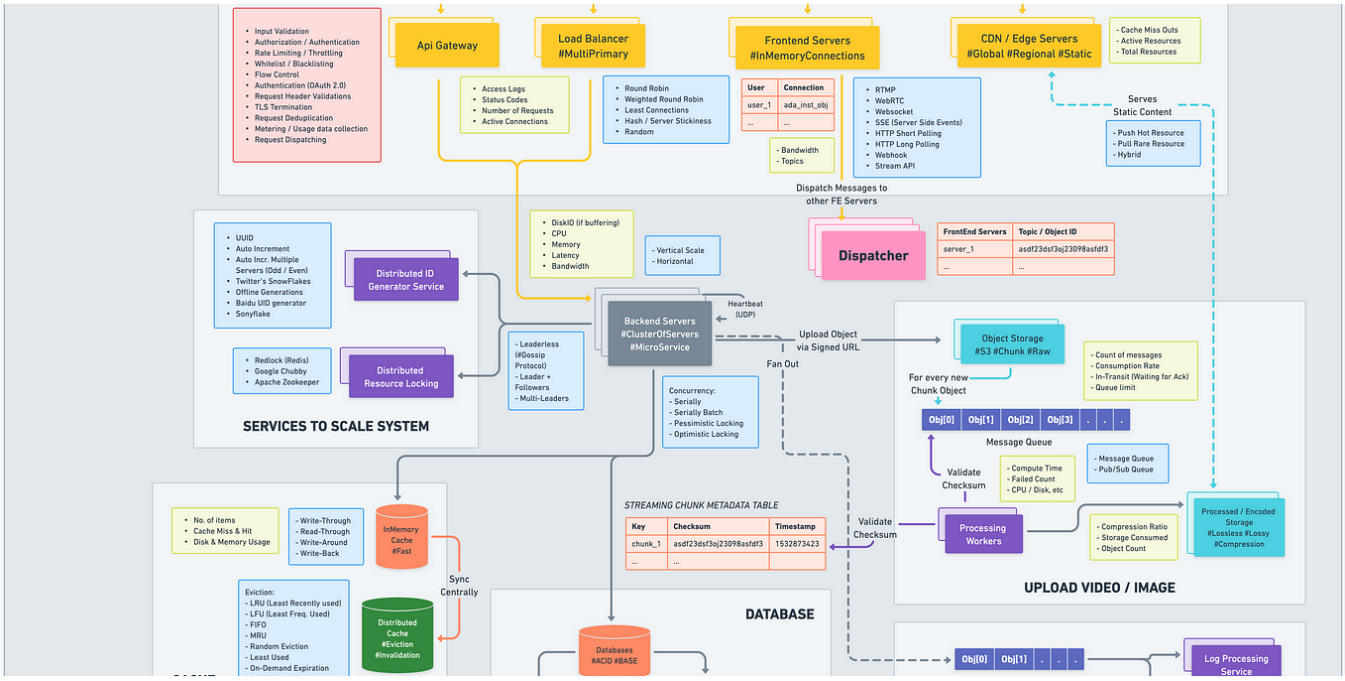
8 min read · Jul 21, 2022

151 1

See all from Olivia Iannone

See all from Towards Data Science

Recommended from Medium



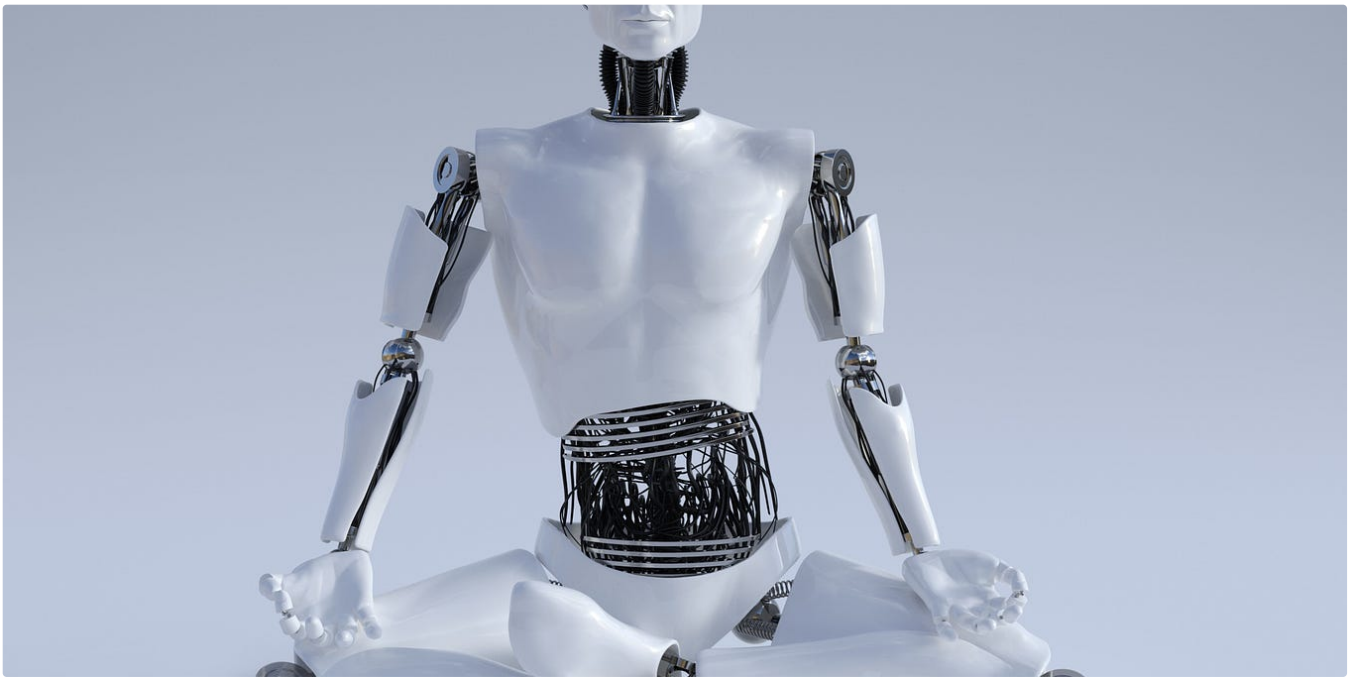
Love Sharma in Dev Genius

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

✦ · 9 min read · Apr 20

4.8K 37



The PyCoach in Artificial Corner

You're Using ChatGPT Wrong! Here's How to Be Ahead of 99% of ChatGPT Users

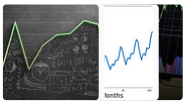
Master ChatGPT by learning prompt engineering.

· 7 min read · Mar 17

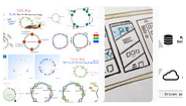
26K 460

Lists

New_Reading_List
173 stories · 8 saves



Predictive Modeling w/ Python
18 stories · 62 saves



General Coding Knowledge
20 stories · 41 saves

Now in AI: Handpicked by Better Programming
248 stories · 18 saves

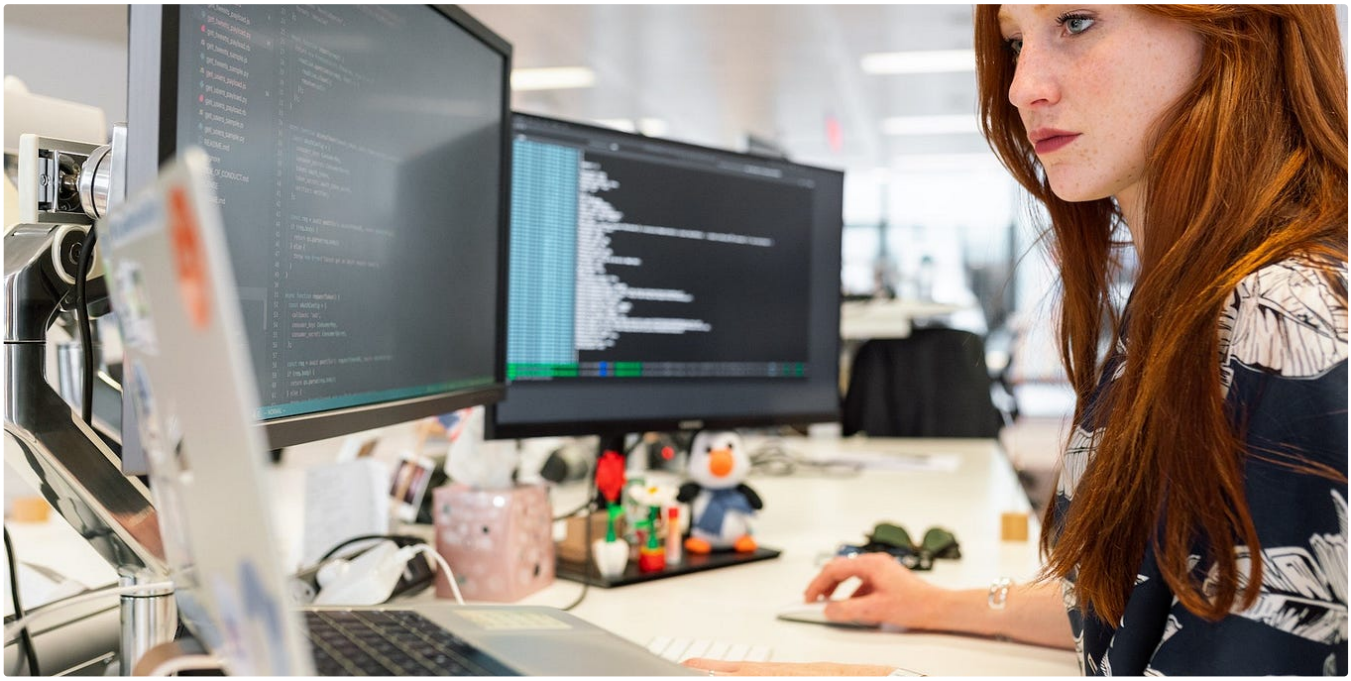
Unbecoming

10 Seconds That Ended My 20 Year Marriage

It's August in Northern Virginia, hot and humid. I still haven't showered from my morning trail run. I'm wearing my stay-at-home mom...

· 4 min read · Feb 16, 2022

51K 814



The Coding Diaries in The Coding Diaries

Why Experienced Programmers Fail Coding Interviews

A friend of mine recently joined a FAANG company as an engineering manager, and found themselves in the position of recruiting for...

✨ · 5 min read · Nov 2, 2022

4.5K 103

Mike Shakhomirov in Towards Data Science

Data pipeline design patterns

Choosing the right architecture with examples

✨ · 9 min read · Jan 2

725 7

JP Brown

What Really Happens to a Human Body at Titanic Depths

A Millisecond-by-Millisecond Explanation

✨ · 4 min read · Jun 23

28K 333

See more recommendations