Friends don’t let friends build data pipelines

Abhishek Tiwari

Citation: *A. Tiwari*, ”Friends don’t let friends build data pipelines”, Abhishek Tiwari, 2018. doi:10.59350/hcf5n-nfg79

Published on: July 12, 2018

Building data pipelines can offer strategic advantages to the business. It can be used to power new analytics, insight, and product features. Often companies underestimate the necessary effort and cost involved to build and maintain data pipelines. Data pipeline initiatives are generally unfinished projects.

In this post, we will discuss why you should avoid building data pipelines in first place. Depending on the use cases, it is quite possible that you can achieve similar outcomes by using techniques such as data virtualisation or simply building microservices. And if you have a genuine data pipeline problem waiting to be solved, then we will explore how to approach this in a practical way – by confronting the 8 fallacies of the data pipeline. Lastly, we will talk about the internal platform and product divide – one key reason why data pipeline initiatives typically fail – and why it is better working backward from the product.

Data Pipeline

A data pipeline is a software that ingests data from multiple sources, transforms it and finally makes it available to internal or external products. In recent times, in order to gain valuable insights or to develop the data-driven products companies such as Netflix, Spotify, Uber, AirBnB have built internal data pipelines. These data pipelines can process data at petabytes scale and to some extent, their success can be attributed to an army of engineers devoted to build and maintain internal data pipelines.

If built correctly, data pipelines can offer strategic advantages to the business. It can be used to power new analytics, insight, and product features. Unfortunately, building data pipelines remains a daunting, time-consuming, and costly activity. Not everyone is operating at Netflix or Spotify scale data engineering function. Often companies underestimate the necessary effort and cost involved to build and maintain data pipelines. Nonetheless, there is hardly anyone talking about the failed data pipeline initiatives.

Pipeline Topology

A data pipeline is synonyms to a directed acyclic graph (DAG) which consists of a set of nodes connected by edges. Depending on frameworks, data processing units (a.k.a jobs) can be nodes of the graph and data (data stream or batch file) can be edges, and vice-versa. This includes both data pipeline developed using batch-driven frameworks such as Apache AirFlow, Luigi, etc. as well as data pipelines operating on the data stream and built using frameworks such as Apache Storm, Apache Flink, etc. These nodes and edges require a good amount of compute and storage which is typically distributed across a large number servers either running in the cloud or your own data center. In a nutshell, a data pipeline is a distributed system.

The 8 fallacies of data pipeline

Based on my own learnings and experience shared by others, a data pipeline project is never finished. One of the key reason is the false assumptions that engineering teams and companies generally make when implementing data pipelines. Here are 8 fallacies of data pipeline

- The pipeline is reliable
- Topology is stateless
- Pipeline is infinitely scalable
- Processing latency is minimum
- Everything is observable
- There is no domino effect
- Pipeline is cost-effective
- Data is homogeneous

The pipeline is reliable

The inconvenient truth is that pipeline is not reliable. A data pipeline is a software which runs on hardware. The software is error-prone and hardware failures are inevitable. A data pipeline can process data in a different order than they were received. In certain scenarios ***exactly-once*** semantics is possible but the best you can do is *at-least-once* or *at-most-once*. Generally speaking, data pipelines are leaky by nature. Pipeline reliability and performance go hand in hand. If tuned for performance, there is a good change reliability is compromised - and vice versa. If certain steps fail, your lifeline is the number of retries after that data loss is a no-brainer. If the whole pipeline fails, maybe you can replay as long as data semantic are handled properly by your application logic such as the de-duplication based on key attributes.

Topology is stateless

Another harsh reality is that most of data pipelines are not stateless. To process the data, a pipeline may need either transactional commits or reference master data - a form of state which needs to be continuously updated. Keeping state outside the pipeline can drastically slow down data processing - imagine reading reference data from external service either via API call or through data connectors.

Keeping reference localised in the pipeline is a smart option - although it requires frequent data updates. As an example, if one of the steps in your pipeline is location data enrichment by referencing the IP address attribute of the incoming data stream with the geo-IP reference data then geo-IP reference data is a state which needs to maintained and updated on a frequent basis.

Pipeline is infinitely scalable

Given the elasticity offered by public clouds, one would think that data pipelines are infinitely scalable which is entirely misleading. As mentioned before, most of data pipelines are not stateless which means scalability is not given a thing. Most likely individual components of data pipeline are scalable but data pipeline as a whole is not scalable - a phenomenon I refer as emergent behaviour where coherent pipeline-wide properties cannot be deduced directly from analysing the behaviour of individual pipeline components. Although based on the load it may be possible to infinitely scale the data pipeline but that will come with a cost which most companies other than Google or Amazon will struggle to justify. Your best shot is proactive capacity planning and better resources utilisation by removing inefficiencies.

Processing latency is minimum

If only you can infinitely scale your data pipeline, processing latency can be guaranteed to be minimum, but we know for sure that is not true. Every time data spikes - a phenomenon which will be not predictable in most cases - overall latency to process the data using data pipeline will go up. In few use cases, data pipeline latency or processing backlog combined with orderliness creates data freshness problems. For instance, if your pipeline is designed to perform the real-time streaming search on incoming Tweets to identify any Tweet with offensive words in the context of certain brand mentions, any spikes in the number of Tweets will result in processing delays.

Everything is observable

Things fail and shit happens but how can we detect the problems in data pipeline as early as possible and proactive response to those emerging issues in a timely manner? To be responsive, we need extensive instrumentation which can help in detecting anomalies, micro and macro level trends. The fact of the matter is to detect the trends or identify anomalies correctly and to keep alerts meaningful you need few baselines and/or thresholds. Unfortunately, we can not anticipate all possible failure scenarios until we have seen enough of those incidents. This means despite having extensive observability and monitoring we cannot detect new problems or issues unless we go through them. As an example, for a completely new pipeline, we can't be sure if x% drop in data processing is an anomaly or an expected behaviour. Only after running for a few days or sometime weeks we will have a reliable baseline to compare with.

There is no domino effect

Well, this is the worst part of a data pipeline. Cascading effects - either due to software errors or undetected data changes - are quite frequent. A common problem is when your data feed or API provider change data model without much notice - updates or deletion of existing attributes or non-backward compatible schema changes - it can impact the quality and accuracy of the data pipeline. Something similar Facebook did after Cambridge Analytica Fiasco, without giving enough warning to their public API users they made several big non-backward compatible schema changes in a short time which led to a domino effect.

Let take another example here, assume that your data pipeline is stitching identify of individual users using cross-device web access logs using a probabilistic method. It uses many possible geo-location attributes extracted from device fingerprint including country, state, city, zip-code, latitude, and longitude. Suddenly one day Apple releases a new user privacy feature which by default disable several geo-location attributes required by your identify stitching algorithm. Although your data pipeline can handle this exception but process data incorrectly i.e. an irreversible and incorrect stitching of user IDs. If this cross-device identity is used to show matching online adverts to individual users, you may start showing up online adverts based on the behaviour of one user to another user. By the time you detect these issues, you may have processed data for a large number of users and problem is now incomprehensible in many ways.

Pipeline is cost-effective

The investment to build a pipeline can be very high - both time and cost wise -many times unpredictable. In addition, most data pipeline initiatives don't account for the **hidden** costs of maintaining a data pipeline. Due to the emergence of new data sources, continuously changing data technologies, rapidly increasing data volumes, and unanticipated change in data landscape organisations require dedicated engineering resources for ongoing maintenance of a data pipeline. A case in point, due to increasing data volumes, engineering teams end-up continuously optimising data pipeline for the performance, sometimes it means an upgrade of the underlying data processing framework. Lack of support for older versions is a common issue with open source data pipeline frameworks which means you have to stay on top of the new releases and ensure that you are always close to the current stable release.

Data is homogeneous

It is worth the time to pay attention to the fact that data is not homogeneous on several levels - from lowest level data types (Integer, Float, Character, String, etc.) to higher level data formats (JSON, XML, Arvo, Parquet, etc). By all means, data is heterogeneous. Plus data is constantly changing - not to mention it is corruptible and mutable. When building a data pipeline we cannot foresee the changing nature of data and hence they need to adapt to cater to future emerging requirements. Moreover, the cascading effect mentioned earlier makes the situation more complicated as any data related errors can easily get propagated throughout the pipeline without being detected.

Two world views

The worldview of a data pipeline is linear. When an organisation starts processing data using the pipeline as the main activity, communication and organisation structure get shaped in that direction (***Inverse Conway Maneuver***) i.e. team start organising themselves around data pipeline and their communication structure become more liner (akin to a car assembly line). In some cases, particularly in the supply chain, ad-tech, and media intelligence environment, I have seen Spotify like squads working on individual segments of the data pipeline. This can be efficient at the beginning but counterproductive in a long run. In particular, this creates several operational challenges as the model lacks an end-to-end ownership.

An alternative worldview is a matrix based data processing model which relies on microservices which each service owning its data store. I strongly think that most of the data pipeline problems can be mapped into a matrix based microservices ecosystem. In some cases, this can be enhanced by combining data virtualization techniques with microservices architecture.

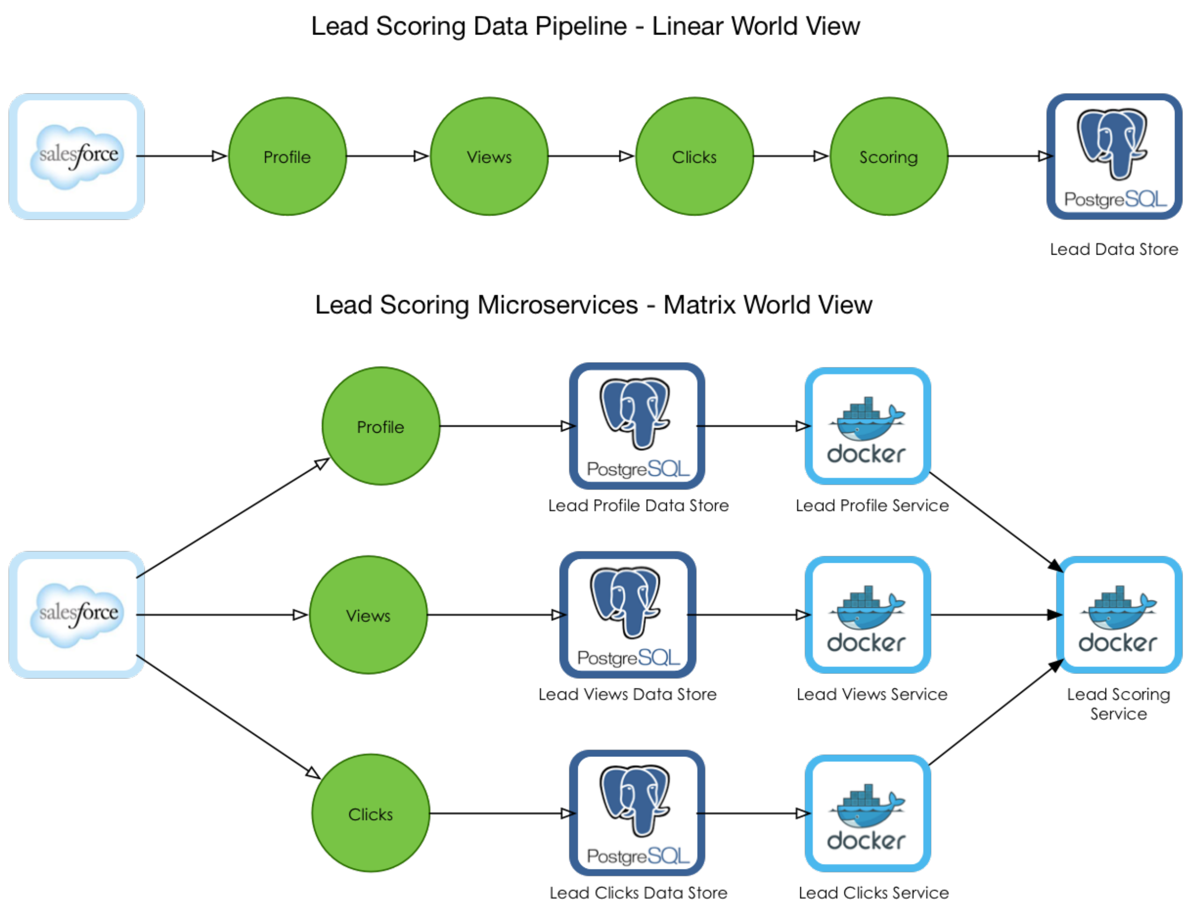

Figure 1: Two world views - Data pipeline architecture vs. microservices architecture with data virtualisation.

Testing Hell

Testing and quality assurance with data pipeline is a nightmare. Due to distributed nature testing data pipelines is a lot harder than contemporary applications. Typically, these data pipelines handle a large volume of heterogeneous data ingested at high velocity. This demands rigorous non-functional testing to characterise performance, load, fault-tolerance in conjunction with normal functional tests. Moreover, identifying the scopes and scale for functional testing of a data pipeline is another area where there is hardly many well-documented strategies or best-practices. You eventually learn how to optimise the scope and scale of your tests but not without going through a painful experience. Although, end-to-end (E2E) tests can simulate the complete data pipeline behaviour- they are in many ways a distraction. E2E tests are difficult to develop, very challenging to maintain, and they have a super slow feedback loop - it can take hours before you can recognise that your recent changes have failed the test.

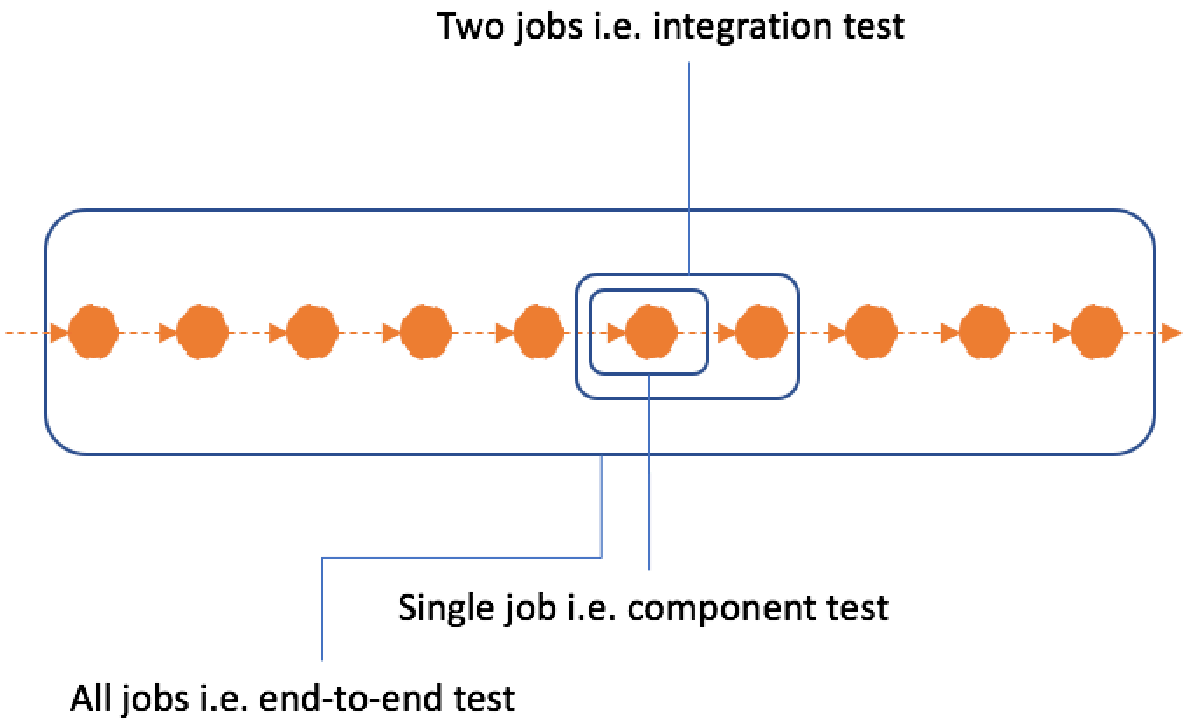
Two jobs i.e. integration test
Single job i.e. component test
All jobs i.e. end-to-end test

Figure 2: Varying level of scopes and scale for functional testing of a data pipeline.

Data pipeline and product divide

If you are not a product organisation, the sole purpose of a data pipeline is to support advanced analytics and powerful insights. Even for the most product companies, a data pipeline is generally not part of their core value proposition unless it is powering some key product features. In particular, the majority of ad-tech and digital analytics companies rely on robust data pipeline to analyse the incoming weblogs to build behavioural analytics, customer segments. Similarly, companies like Spotify and Netflix require personalisation layers based on petabyte scale data around content consumption behaviours hence why they need data pipelines. In these type of scenarios deploying dedicated engineering resources to build and maintain data pipeline is a no-brainer.

As rule of thumb only build data pipeline if it can help you to achieve your core value proposition. For a telecom company, the ability to monitor network congestion and failures is part of their core value proposition. In the same vein - for a bank, it will need to detect fraudulent transactions in real-time and block it. Always start backward from the product and create a data lineage so you know what data transformations or enrichments you need.

Lastly, do not let *Inverse Conway Maneuver* create an organisational silo i.e. data pipeline owned by a data infrastructure function or IT and product owned by digital or engineering teams. To have a successful data pipeline which positively enhances your core value proposition, your teams need an end-to-end view.