upsolver

Apache Kafka with and without a Data Lake

Whitepaper

Intro

Apache Kafka is a cornerstone of many streaming data projects. However, it is only the first step in the potentially long and arduous process of transforming streams into workable, structured data. How should you design the rest of your data architecture to build a scalable, cost effective solution for working with Kafka data? Let's look at two approaches - reading directly from Kafka vs creating a data lake - and understand when and how you should use each.

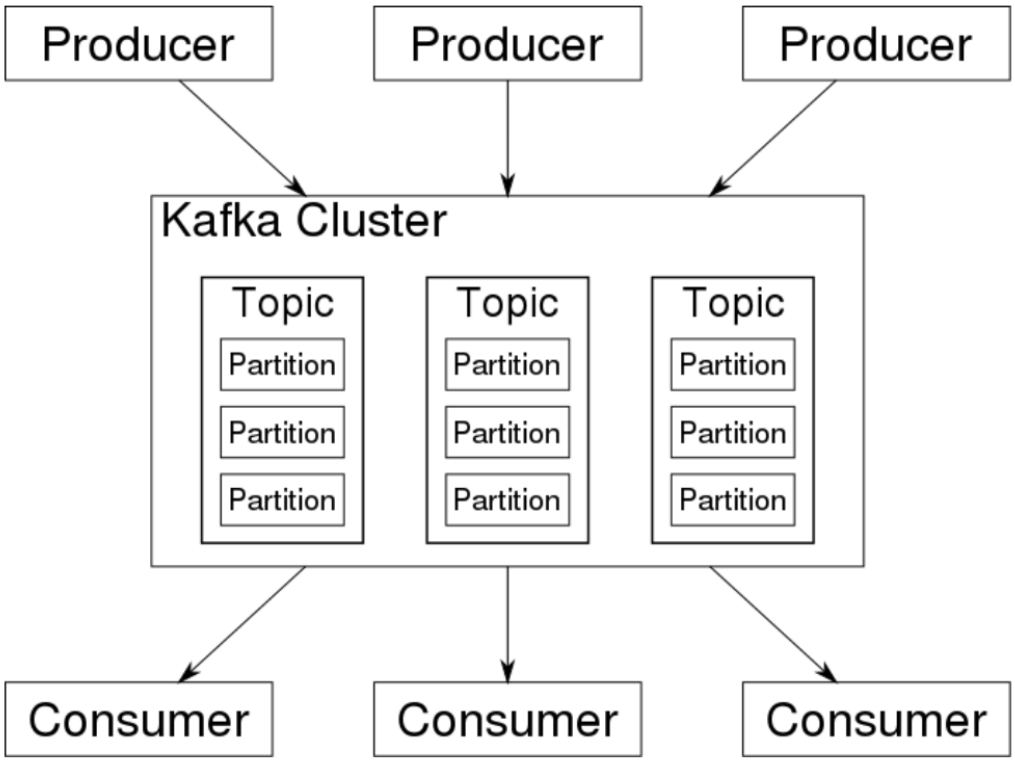
WHO SHOULD READ THIS?

- Data architects looking to build effective infrastructure for analyzing streaming data
- Engineering managers building data pipelines from Apache Kafka (or Amazon Kinesis) to analytic applications such as Apache Presto
- Anyone who wants to gain more value from analyzing Kafka data

Kafka: The Basics

Let's start with an (extremely) brief explanation of how Apache Kafka works.

Apache Kafka is open-source software used to process real-time data streams, designed as a distributed transaction log. Streams can be produced by any number of tools or processes, with each record consisting of a key, a value and a timestamp. Records are stored chronologically in partitions, which are then grouped into topics. These can be read by various consumer groups:



Kafka provides a robust, reliable and highly scalable solution for processing and queueing streaming data, and is probably the most common building block used in streaming architectures.

But what do you do once you have data in Kafka, and how do you get it into a form that developers and data analysts can actually work with? In the next part of this article, we'll explain why the answer to that question, in most cases, is to build a data lake.

Without a Data Lake: Reading Directly from Kafka

A lot of organizations look at the neatly partitioned data going into their Kafka clusters and are tempted to just read that data directly. While this might sound like an easy fix with minimal data plumbing, it also has major drawbacks that all stem from the fact that Kafka is a system that is highly optimized for writing and reading **fresh data**.

This means that:

Producing is easy…

- **Unlimited writes**: Kafka stores events on one large file on disk, and that file is appended sequentially as new events are processed. This system enables it to achieve truly incredible feats such as completing two million writes per second on three cheap machines.
- **High reliability:** Even without too much tweaking, Kafka is highly fault-tolerant and writes just work - you don't need to worry about losing relevant events.

- **Strong ordering within a partition:** Data is ordered sequentially; if two consumers read the same partition, they will both read the data in the same order. This means that multiple, unrelated consumers will see the same 'reality' - which can be extremely important in some cases (e.g. when assigning workloads).
- **Exactly-once writing:** Kafka 0.11 introduced exactly-once writing, which is very handy as a means of reducing efforts on the consumer side.

...but consuming is hard (and expensive!)

- **Retention is 10X-100X more expensive compared to using cloud storage:** Storing historical data on Kafka clusters can quickly drain your IT budget. Kafka stores multiple copies of each message on expensive hard drives connected to servers. Both of these factors contribute to at least 10x the bottom line compared to storing the data on a data lake such as Amazon S3.
- Risk to production environments: Reading from Kafka in production is not great: every additional consumer drains resources and slows performance, while reading "cold" data will likely cause cache misses.
- Waste of compute resources: Kafka consumers read an entire record to access a specific field, whereas columnar storage on a data lake (e.g. Apache Parquet) will allow you to directly access specific fields.

- **Data quality effort per consumer:** Each Kafka consumer works in a vacuum, which means data governance is an issue that needs to be multiplied by the amount of consumers - which in turn means repetitive deduplication, schema management, and monitoring processes; a data lake allows you to unify these operations on a single repository.

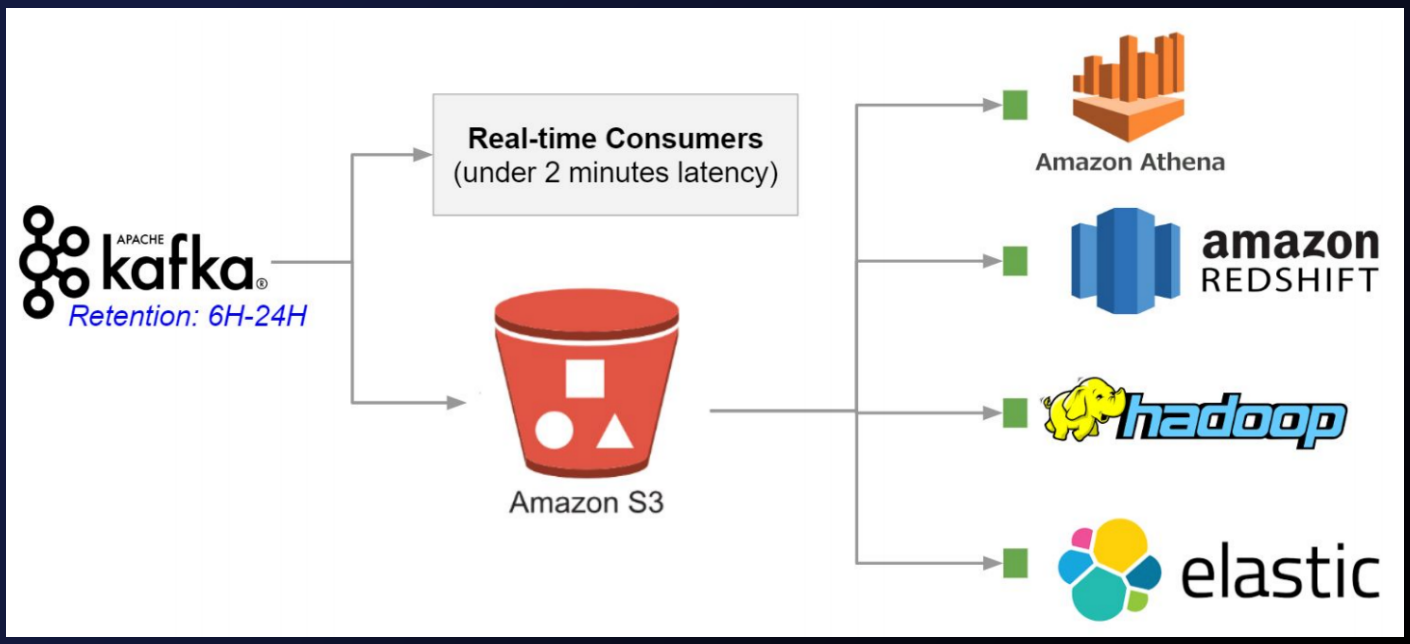
The Solution: Build a Data Lake

We've explained why reading data directly from Kafka is messy, expensive and time-consuming. Most of these problems can be solved by introducing a data lake as an intermediary stage between your Kafka and the systems you use to analyze data. This approach is advantageous as it allows you to:

- Leverage cheap storage on S3, allowing you to significantly increase retention without paying through the nose for storage.
- Introduce new use cases without worrying about Kafka performance and stability: a data lake enables flexibility to introduce new consumers and applications, without slowing down your production environment.
- Ensure data quality and governance by working on a single repository rather than individual topics.

	Data Lake	Read Directly from Kafka
Retention Cost	S3 prices	At least 10X more expensive
Performance	No #Consumers Limitation (S3)	#Consumers go up and Performance down
Data Quality Effort	Once on S3	Per Consumer

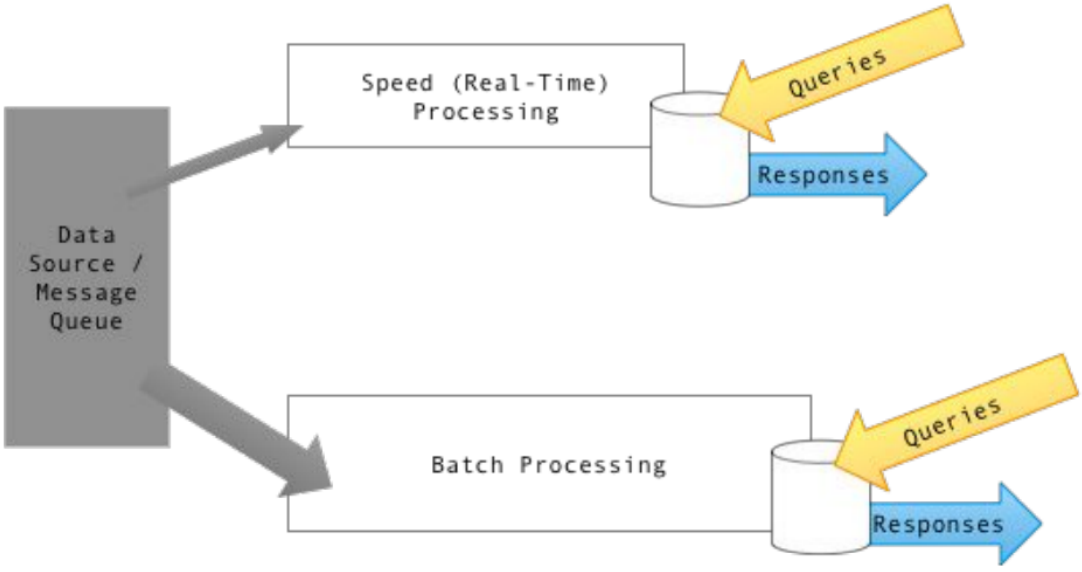
Kafka + Data Lake Reference Architecture (on AWS)



The Real-time Caveat

You might have noticed a “Real-time Consumers” block in the diagram above, although we recommended reading data from a data lake and not directly Kafka. What’s going on?! Have we been lying to you this entire time?

We haven’t but there is an important caveat, which is that the data lake architecture will not work for use cases that require an end-to-end latency of seconds. In such cases, for example as part of a fraud detection engine, you would need to implement an additional Kafka consumer that would only process real-time data (last few minutes) and merge the results with the results of a data lake process. This architectural pattern is named Lambda Architecture:



Upsolver: The Data Lake Platform

Gartner estimates that 85% of big data projects fail, often due to lack of internal resources and knowledge, and the difficulty of managing complex architectures and data pipelines.

Upsolver is here to change this reality with a complete Data Lake Platform that's powerful, agile and simple enough for any developer to launch and maintain.

Upsolver's Data Lake Platform takes the complexity out of streaming data integration, management and preparation on any data lake - whether it's HDFS on-premise or on AWS, Azure or Google Cloud.

- Schedule a Demo
- Start a Free Trial
- www.upsolver.com