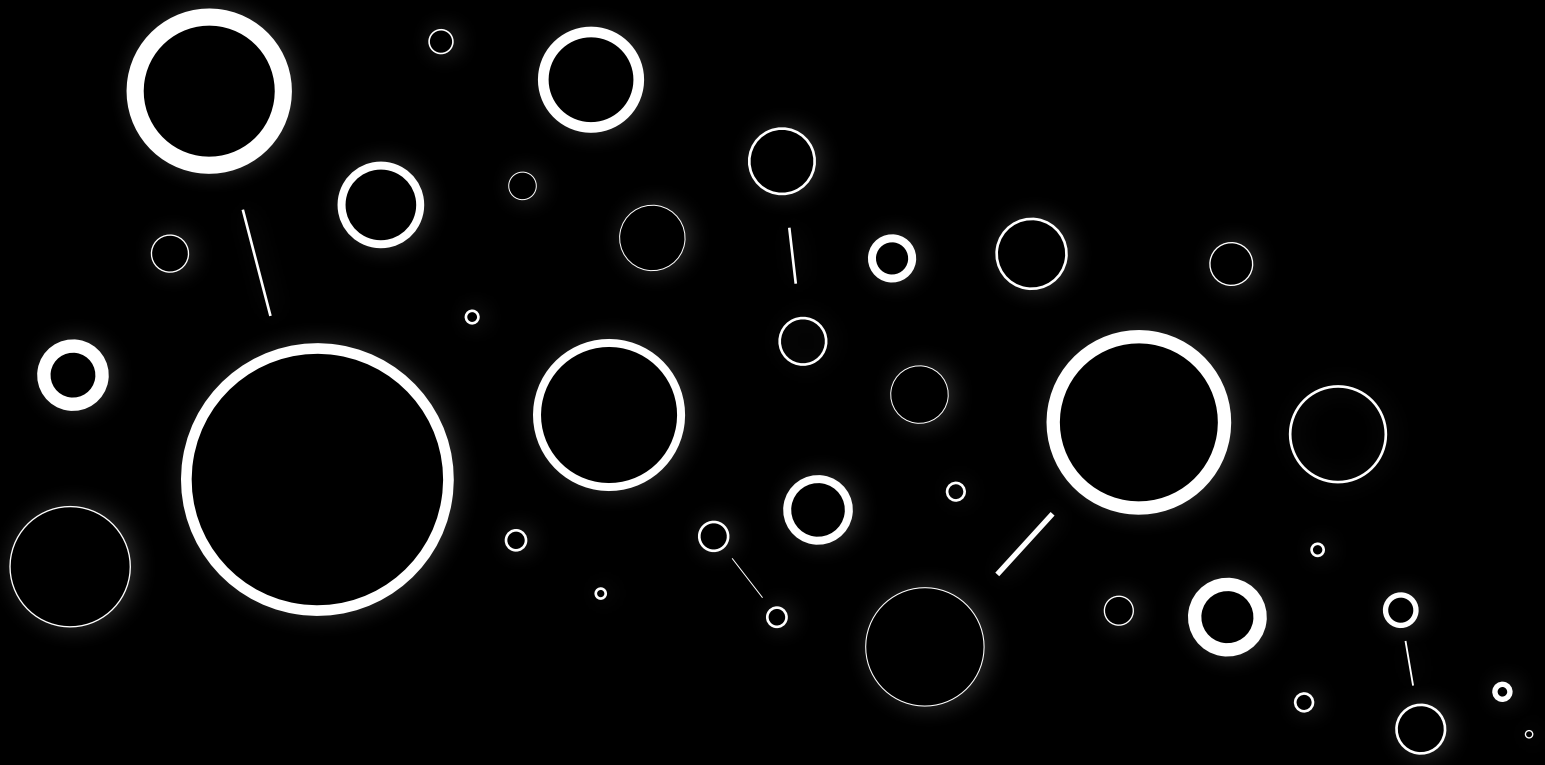
SOFTWAREMILL



Start with Apache Kafka

The essentials for implementing a production Kafka

2023 edition

Let's get familiar with the basics of Apache Kafka, one of the most popular big data tools.

Kafka is used by over 28,000 companies around the world, including over 80% of the Fortune 100, enabling them to take up the challenge of efficient real-time data streaming when broker technologies based on other standards have failed.

Kafka has various applications, ranging from simple message passing, via inter-service communication in microservices architecture, to whole-stream processing platform applications.

At SoftwareMill, we are consultants and System Integrators who leverage Confluent solutions in building scalable data platforms for our clients. Here is an introduction to Kafka and our lessons learned while consulting clients and using Apache Kafka in commercial projects.

SoftwareMill Team

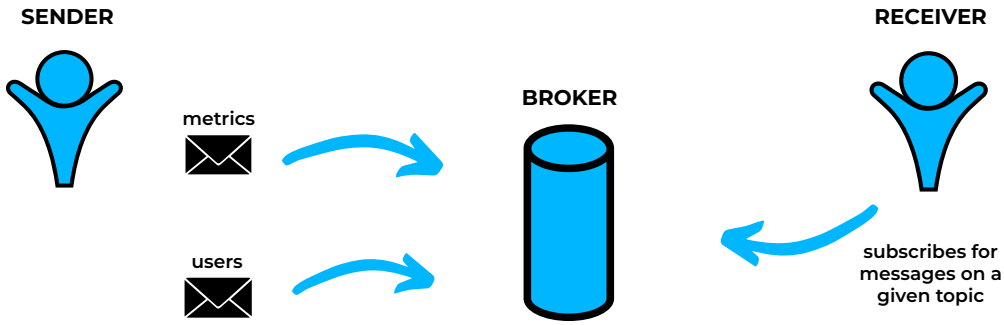
Table of Contents

Introduction to Apache Kafka

Before we dive into Kafka, let's start with a quick recap of what **publish / subscribe messaging** is.

It's a messaging pattern where the sender doesn't send data directly to a specific receiver. Instead the publisher classifies messages without knowing if there are any subscribers interested in particular types of messages and the receiver subscribes to receive a certain type of messages without knowing if there are any senders sending them. All messages are published in a broker which ensures that messages are delivered to the correct subscribers.

PUBLISH / SUBSCRIBE MESSAGING PATTERN



Apache Kafka is a type of such pub/sub messaging system. Specifically it is a distributed streaming platform for building real-time data pipelines and real time streaming applications. It can deliver in-order (under specific conditions), persistent messages in a scalable way and groups them into topics - a primary abstraction. Kafka has a distributed event log where all new records are immutable and appended to the end of the log.

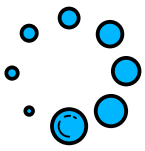
The main difference between Kafka and other messaging systems is the retention policy. It allows us to set a retention period on a per-topic basis. The message stays in the log even if it has been consumed, however after the given period, or when it reaches specific size, messages are discarded to free up space.

When is Apache Kafka best put to use

Kafka is used heavily in the big data space as a reliable way to ingest and move large amounts of data very quickly. Allows us to build a modern and scalable ETL (extract, transform, load), CDC (change data capture and Big Data Ingest systems).

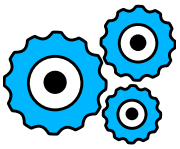
Usage of Kafka continues to grow across multiple industry segments. The technology fits perfectly into the dynamically changing data-driven markets where huge volumes of generated records need to be processed as they occur.

Like other messaging systems, Kafka facilitates the asynchronous data exchange between processes, applications and servers, and is capable of processing up to trillions events per day. Every real time big data solution can benefit from its specialized system in order to achieve the desired performance.



stream processing

log aggregation



microservices

web activity

metrics

messaging

Currently there are 28k companies using Apache Kafka. Among them Uber, Netflix, Activision, Spotify, Slack, Pinterest, Coursera and LinkedIn.

Moreover, Apache Kafka is ranked as **the highest paying, fastest growing tech skill** in the US. Jobs that demand Apache Kafka include data engineer, database architect, software developer, data scientist and validation engineer.

Data pipelines

Most of what a business does can be described as streams of events. Each software system generates events that when processed in real-time, give them **competitive advantage** and an edge of being data-driven. Apache Kafka allows users to read and write data more conveniently, therefore to build a streaming Kafka Data Pipeline. It delivers high throughput data from source to target applications to keep various business activities running.

Event-driven Microservices

Application architecture is shifting from monolithic enterprise systems to flexible, event-driven approaches. Apache Kafka is scalable, highly available, and fault-tolerant asynchronous communication backbone for microservices architecture. Services publish events to Kafka while downstream services react to those events instead of being called directly. In this fashion, event-producing services are decoupled from event-consuming services.

Stream processing

By definition Apache Kafka is a distributed streaming platform for building real-time data pipelines and real-time streaming applications. Adopting stream processing enables a **significant reduction of time** between when an event is recorded and when the system and data application reacts to it, enabling real time reaction.

Examples of Apache Kafka applications

Website activity tracking

Website activity (page views, searches, or other actions users may take) is published to central topics and becomes available for real-time processing, dashboards and offline analytics in data warehouses like Google's BigQuery.

Metrics

Kafka is often used for operational monitoring data pipelines and enables alerting and reporting on operational metrics. It aggregates statistics from distributed applications and produces centralized feeds of operational data.

Log aggregation

Kafka can be used across an organization to collect logs from multiple services and make them available in standard format to multiple consumers. It provides low-latency processing and easier support for multiple data sources and distributed data consumption.

Messaging

Kafka works well as a replacement for a more traditional message broker. In comparison to most messaging systems Kafka has better throughput, built-in partitioning, replication, and fault-tolerance which makes it a good solution for large scale message processing applications.

Apache Kafka ecosystem

Kafka, as a message queue, offers great capabilities related to message passing. However, there are a few complementary solutions that can enhance its features allowing for easier management or interoperability with other systems.

Schema Registry

Messages sent to Kafka usually have predefined schemas. Schemas can be managed and versioned using the Schema registry project. Message contains in such case binary serialised data and the reference to the specific schema version. What is more, formats such as Apache Avro allows for schema migrations, supporting changes in data format.

Kafka Connect

Integration with external systems is a very important topic. That's the role of Kafka Connect. Various connectors allow you to read data from different sources, such as databases and send them to Kafka. Other types allow you to write Kafka messages to special sinks, which similarly can be a database or other type of external system you want to integrate with.

Kafka Streams & ksqlDB

Messages retrieved from other systems often need to be transformed or aggregated to be able to achieve the designed business goals. Kafka Streams & ksqlDB can help with that. They both offer such capabilities, allowing to process messages from Kafka topics, putting results into new, different topics. Kafka Streams is a more code oriented library, where ksqlDB is oriented around SQL like language.

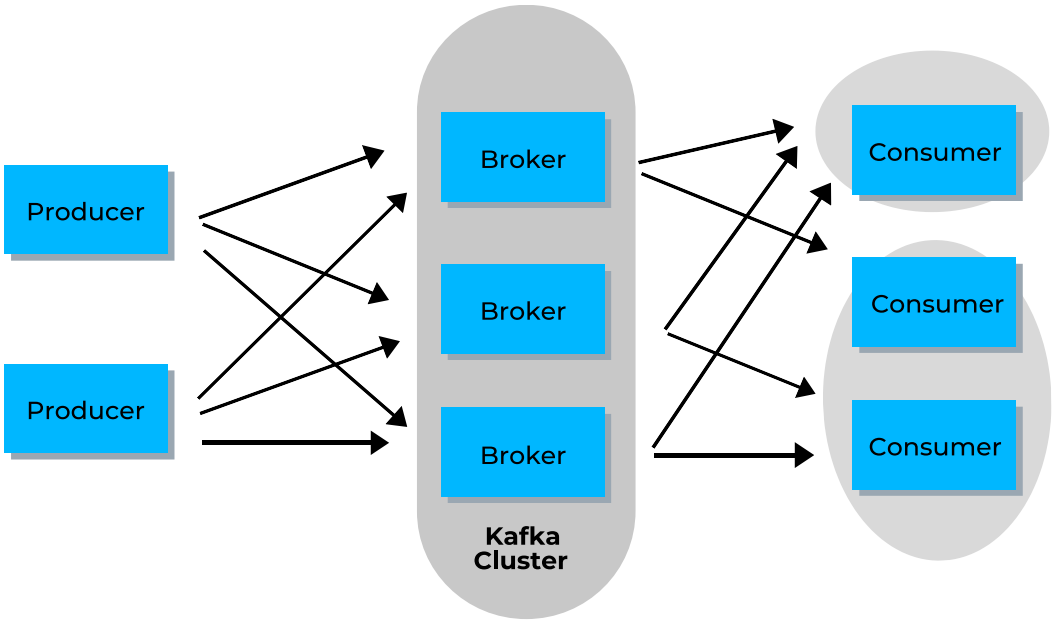
Apache Kafka architecture

High Availability and Scaling are very important. That is why Kafka is always deployed in the form of a Cluster. Cluster consists of a few Brokers (usually 3 or more). Producers send messages to Kafka topics. On the recipient side there are Consumers reading messages from those topics. In practice Consumers are grouped into Consumer Groups. Each Consumer Group receives all messages from a subscribed topic, from all of the senders. Messages are divided among all consumers belonging to the group. This way of consuming is scalable.

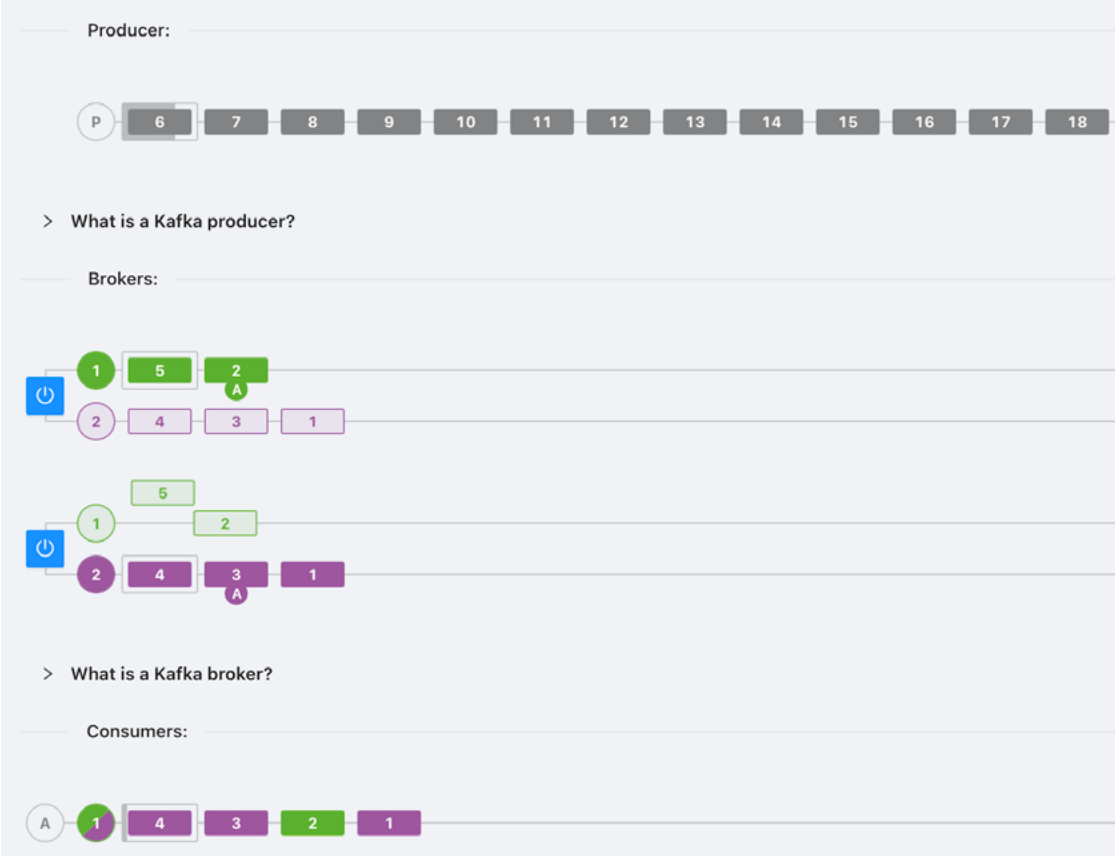
Underneath, each topic is divided into multiple partitions. Partitions are distributed among cluster brokers. When Consumer Group subscribes to a topic, specific Partitions are assigned to specific Consumers. **One partition can be assigned only to one consumer, but Consumer can have multiple partitions assigned.** This also indicates that the partition number should be greater than or equal to the number of intended consumers in the biggest Consumer Group.

Messages sent to the topic are pretty simple, but they are built of two important parts: **a key and a value**. The value is the data you are passing, but the key defines the partition to which the message will be put into. By default Kafka calculates partitions for a given message based on the message key. If the key is not specified then messages are distributed among partitions using Round Robin.

Messages inside of the single Partition are ordered. In practice this means that if the message key is a business related identifier, then all messages related to it will be also ordered.



Using the [Kafka Visualization Tool] you can simulate how data flows through a replicated Kafka topic, to gain a better understanding of the message processing model.



Further read: [What problems does Kafka solve in distributed systems?]

Questions you might have when using Kafka in your project

How to choose the number of partitions?

This is one of the most common Kafka related questions. Before going to production you should choose a number of partitions, optimally which would be sufficient for you for a few next years. Minimal number is the number of consumers you'd like to subscribe to a given topic. Maximum is not really defined. Too big a number negatively influences some Kafka internal processes, but in most cases 50 is just enough. Kafka offers a possibility to change the number of partitions, however you need to remember that after such a process, messages with the same key may no longer be put to the same partition as before the resize. See this blog post for more information.

Where to deploy Apache Kafka cluster?

As usual, there are multiple options. You may use bare-metal and provision Kafka e.g. with Ansible scripts. There are also Kafka packages for DCOS. However, if you want to choose Kubernetes, then you may leverage e.g. Strimzi. In most cases we would recommend choosing the environment you are already using for your other services. Just remember to leverage multiple Availability Zones to guarantee your cluster availability during failures. Fortunately, there is an option for you even if you don't want to maintain Kafka in your infrastructure. Multiple companies offer hosted Kafka clusters - Confluent Cloud, Instaclustr, Aiven.

Shall I use Schema Registry or not?

When working with Kafka you have multiple options related to your messages data format. To use JSON, Avro or Protobuf, that is the question. Json is the simplest choice. Easy to debug, since your messages will be plaintext. Supported by a lot of serialisation libraries etc. However, it may not be the best choice in terms of performance or data size. There comes the binary formats, like Avro or Protobuf. We have actually leveraged both of them in our various projects. Avro was chosen by Confluent and is natively supported by Kafka client libraries. What is more it is possible to leverage here the Schema Registry tool. It is a type of repository, which automatically stores schema messages, does the compatibility validations and allows clients to not attach schema with every message, just its identifier. If you have chosen Avro for your project, then you should definitely use Schema Registry.

To get know more about Schema Registry take a look at the blog post „**Dog ate my schema**".

Is Kafka really supporting exactly once delivery?

Exactly once delivery is one of most tough problems in Distributed Systems. It is quite simple to achieve at most once delivery. However in most cases you don't want to lose your data, so there comes the at least once. Kafka creators claim they managed to achieve exactly once, however in practice it is more like „effectively exactly once" - at most once with the deduplication. It works additionally with some limitations, e.g. with Kafka Streams, but not with every Kafka producer & consumer flow you can implement.

To get to know more about how it is implemented in Kafka, take a look at "**Exactly-once Semantics are Possible: Here's How Kafka Does it**".

How to run Apache Kafka in the Cloud?

When you want to leverage Kafka for event-driven microservices, brokers setup on bare VMs or on-premises can be overcomplicated, especially without proper experience. Let's see what are the simpler methods available across different cloud offerings.

Kubernetes Operator

Before diving into the typical cloud specific offerings, let's start with Kubernetes. Your company may be already leveraging it. If that is the case the reasonable option may be just to set up a Kafka Cluster on K8S using one of the available Kubernetes Operators. It may be the cheapest option, however, keep in mind that in this approach maintenance & monitoring is on your side, and this adds up to the final costs.

Strimzi

Strimzi is probably the most known Kafka operator. It not only allows running the brokers, but offers support for Kafka Connect, Mirror Maker, Exporter (for Metrics) and Kafka Bridge (HTTP API). It is even possible to manage topics automatically. We have used Strimzi in some of our projects and it worked quite well.

For an example of running Strimzi together with Kafka connect, take a look at the „Running Kafka Connect Cluster as a Native Kubernetes Application blog post".

Confluent operator

As an alternative, Confluent has released its own Kubernetes Operator. It supports only commercial Confluent Platform. However, on the other side it supports additional products from the Kafka family that Strimzi does not - such as Schema Registry and ksqlDB.

Koperator

There is one more operator, less known but promising. The Koperator was named formerly as Banzai Cloud Kafka Operator and is now part of Cisco. It allows to set up the Kafka cluster, together with Cruise Control and Prometheus. The Open Source version has a limited set of features but in the enterprise variant it is possible to use Kafka Connect and ksqlDB.

Cloud-specific

Quite often we see that business clients do not want to maintain their own cluster. It is complicated, you need to employ people with proper experience and maintain the infrastructure. Instead, they choose one of the "Kafka as a service" types of products. Let's walk through a few of the top cloud providers, and see what each of them has to offer.

Amazon Web Services (AWS)

Amazon Web Services offer a few products which can be used for data streaming or events passing. The most important ones are AWS SQS and Amazon Kinesis. When you dive into how Kinesis works underneath it looks quite similar to Kafka and some of its concepts. However, it is not a drop-in replacement. There are no compatible APIs. Fortunately, there is an alternative in the form of managed Kafka instances.

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Amazon MSK is a managed Kafka cluster. You pay for broker instances and storage (not for the ZooKeeper service). It is highly available with automatic multi-az replication. Data is encrypted at rest and in transit. SLA for this service is 99.9% uptime, but it covers only selected scenarios e.g. does not include failure caused by bugs in the Apache Kafka itself. Some people call MSK partially-managed. You need to define the sizes of Kafka server nodes, monitor performance and scale accordingly. There is a quite lengthy guide from AWS about best practices for sizing Kafka clusters on MSK.

Amazon MSK Serverless

MSK Serverless is an attempt to make MSK fully managed. You are charged hourly for the cluster, number of partitions, storage, and data transfers in and out. Like in the most serverless approaches for some use cases this will be better, for other ones more expensive. The 99.9% SLA does not apply to the MSK Serverless.

MSK Connect

Amazon MSK Connect provides managed Kafka Connect services. It is charged hourly for connector usage, depending on the number of workers. You can use it together with MSK, but not only - other Apache Kafka clusters are compatible as well.

Others

AWS offer does not include Kafka Streams and standard Schema Registry. However, it includes some alternatives. You can process MSK messages and stream data using Amazon Kinesis Data Analytics and leverage additionally AWS Glue Schema Registry. It integrates with a lot of other AWS products as well.

Google Cloud Platform (GCP)

GCP does not offer any managed Kafka. Its default products for messaging are Pub/Sub and Pub/Sub Lite. The Lite variant concept sounds similar to Kafka and what is interesting is that it offers a "Kafka API".

Pub/Sub Lite with Apache Kafka-like API

It is possible to use the Kafka client library together with Google Pub/Sub Lite Kafka Shim Client.

Unfortunately, it has some limitations:

- does not support transactions,
- messages can be produced or consumed from a single topic at a time,
- it is not possible to send messages to a specific partition.

Underneath it uses gRPC to communicate with Pub/Sub Lite services.

Pricing depends on reserved capacity - throughput, storage and egress. Uptime depending on the topic type ranges from 99.5% to 99.95%. It is HIPAA and SOC2 compliant.

Azure

Apache Kafka on HDInsight architecture

Azure HDInsight is a product that allows running Apache Hadoop, Apache Spark and other big data systems. It is possible to leverage it for Apache Kafka as well. You are billed for the provisioned cluster (pricing depends on a number of nodes and their types). Data is encrypted at rest and the service has a 99.9% uptime. You have to monitor it using Azure monitor, quite similarly to Amazon MSK.

Azure Event Hubs with Apache Kafka API

That's the approach similar to GCP Pub/Sub Lite. It is possible to use Event Hubs, leveraging Kafka Client libraries. There are some limitations, e.g. it does not support log compaction. As an alternative to Kafka Streams Azure proposes to use one of the stream processing services which can be just used with Event Hub. Pricing depends on provisioned capacity and chosen plan. Uptime is dependent on the plan and ranges from 99.95% to 99.99%.

Externally Hosted-services

Apart from cloud specific services, external companies offer running and managing Kafka and related products in the cloud of your choice, sometimes even on your own cloud account.

Confluent Cloud

Confluent Cloud is the product that offers the biggest number of known Kafka-related services. It supports Connect (including Confluent connectors), Schema Registry, but also ksqlDB. It has a few custom solutions as well - like Stream designer or Stream Governance that can be a nice base for an event-streaming platform.

Pricing depends on a few factors. There are 3 plans you can choose from:

- **Basic,**
- **Standard,**
- **Dedicated.**

Each of them has different features, prices and compliance certifications. The highest one supports ISO 27011, SOC 3, HIPAA and PCI, offers multi-az deployment with 99,99% uptime, data encryption and infinite storage. You can run it on AWS, Azure or GCP. Price is different for every plan, cloud provider, amount of data in & out, storage and others. We're not choosing node sizes here, it is a fully-managed variant, a bit similar to MSK Serverless. However, keep in mind that because of MSK limitations and exclusions in the SLA, Confluent looks much better here.

Instaclustr

Instaclustr is a company that offers various open source projects in a managed manner - e.g. Kafka, Kafka Connect and Schema Registry. It can be used with AWS, Azure and GCP but what is more with IBM Cloud and DigitalOcean. It is PCI-DSS, SOC2 and HIPAA compliant. What is interesting is that it offers 99,999% SLA and can be run on your own cloud account. Pricing? Depends on the plan (Developer or Production) and number of nodes & their sizes (both in Kafka and Connect case).

An interesting fact is that Instaclustr actually forked Schema Registry before its licence changed. Nowadays they additionally support Karapace - Schema registry replacement created by Aiven.

Aiven

Aiven is quite similar to Instaclustr. It offers Kafka, Kafka Connect and Karaspace. Can be used with AWS, Azure, GCP, DigitalOcean and UpCloud. They offer 3 pricing plans: Startup, Business and Premium. Only premium can be run under your own cloud account. Plans have different max storage and under each of them different variants of CPU & RAM per VM are available.

Summary

Apache Kafka becomes a de-facto standard for event-driven and data streaming architectures hence every major cloud provider offers Kafka-related services. They offer different pricing models, SLA terms and features. Some of the projects due to licensing limitations can't be offered as a SaaS model from anyone apart from Confluent. That is why alternatives are being created or integrations with other similar already existing services.

What to choose for your next project? That is not an easy question. **It depends!** It depends on SLA, pricing, security requirements - what certifications are needed, whether you need a 100% automatically managed environment or if a bit of maintenance is ok for you. It depends on the status of the project and what the performance requirements are. There are various factors that you need to consider before making a decision.

Good luck!

7 mistakes to avoid when using Apache Kafka

Every tool has its caveats. Small configuration mishaps may lead to a big disaster. Let's focus on 7 most common mistakes and how to avoid them.

Let's focus on 7 most common mistakes and how to avoid them.

1. **Using the default settings**
2. **Starting from setting one partition only to change this later**
3. **Using the Publisher default configs**
4. **Using basic Java Consumer**
5. **Using Kafka only because it supports exactly once semantic**
6. **Not monitoring properly**
7. **Upgrading project dependecies without looking at releases notes**

1. You use the default settings

Sending a message to a non-existing Kafka topic, by default results in its creation. Unfortunately the default settings define a single partition and a replication factor of 1. They are totally not acceptable for production usage due to possible data loss and limited scalability. What is more, those settings affect various Kafka "special" topics, e.g. intermediate topics used by Kafka Streams or ones leveraged by Kafka Connect in Distributed mode.

HOW TO AVOID THIS MISTAKE?

- Disable the auto.create.topics.enable (on broker level), this way every topic will need to be created explicitly, manually.
- Or override default.replication.factor and num.partitions (on broker level) to change the defaults for auto-created topics.

More configuration settings in the Broker Configs documentation section here >>

2. You start from setting 1 partition to change this later

The number of partitions defines the maximum number of consumers from a single consumer group. If we define 2 partitions, then 2 consumers from the same group can consume those messages. What is more, if we define too small a number, then the partitions may not get located on all possible brokers leading to nonuniform cluster utilisation. Basically this means that the maximal consuming speed and producing speed are influenced by the lower bound of the range of possible partition numbers.

What about the upper bound? When do we know that there are too many partitions? Well, a large number of partitions influences various processes. Publishers buffer batches of messages sent to Kafka per partition. Bigger number of partitions = lower probability that messages with different keys will land on the same partition, meaning lower probability of getting larger batches. Additionally more partitions means more separate buffers = more memory. This means that a too big partition number can influence producer throughput and memory usage.

HOW TO AVOID THIS MISTAKE?

- Do some performance tests. Simulate the load (let's say predicted for a few years ahead), attach the consumers and test what number of partitions will allow achieving the needed performance. Remember to use the same hardware specs as those which will be available on the production.

3. You use the Publisher default configs

In order to make Kafka Producer working you need to define only 3 configuration keys — bootstrap servers, key and value serializers. However, often it is not enough.

HOW TO AVOID THIS MISTAKE?

Kafka includes a lot of settings that may influence the messaging ordering, performance or probability of data loss, that you need to take into account, some of them are:

- **acks** — defines how many of in-sync replicas need to acknowledge the message (by default only the partition leader, what may result in data loss, because leader does not wait for data to be written to a disk, only to filesystem cache)
- **retries** — defines the number of retries during a failed send
- **max.in.flight.requests** — defines max number of not acknowledged requests the client may be processing. In practice may influence the message ordering.

What is more, underneath the Producer batches all messages it was ordered to send. It creates buffers per partition. This also means that the message was actually sent when the send callback gets called or appropriate **Future** finishes.

4. You use basic Java Consumer

Kafka Java client is quite powerful, however, it does not present the best API. Using the Java Consumer is quite painful. First, the **KafkaConsumer** class can be used only by a single thread. Then, it is required to define an "infinite" while loop, which will **poll** the broker for messages. However, the most important is how the timeouts and heartbeats work. Heartbeats are handled by an additional thread, which periodically sends a message to the broker, to show that it is working.

Kafka requires one more thing. **max.poll.interval.ms** (default 5 minutes) defines the maximum time between poll invocations. If it's not met, then the consumer will leave the consumer group. This is ultra important! Let's say that you consume messages and send them to some external HTTP API. In case of failure, you may leverage exponential backoff. If it goes over 5 minutes, then... your consumer will leave the group and the message will get delivered to another instance. Scary and difficult to maintain.

HOW TO AVOID THIS MISTAKE?

Limit the number of records fetched in a single poll invocation. What are other choices instead of the plain Java client? There are a few:

- Spring for Apache Kafka
- Alpakka Kafka — which offers both Java and Scala APIs. Did you know that it originates at our company — SoftwareMill?
- FS2 Kafka — Scala library for Kafka integration with FS2
- Micronaut Kafka
- Quarkus Kafka and others

Most of the libraries automatically manage the requirement for the poll intervals, by explicit consumption pausing. Before choosing your lib, check if it is for sure handled correctly.

5. You use Kafka because it supports exactly once semantic

Usually, when you ask your client what delivery/processing semantics are acceptable for the business you'll hear the answer: exactly once. Then, you do some Googling and you see wonderful headlines saying that Kafka supports exactly-once! That's partially true, from a theoretical perspective there is no such thing as exactly-once delivery... because, usually it's at least once with de-duplication, which effectively is exactly-once.

HOW TO AVOID THIS MISTAKE?

In Kafka's case, there are limitations around the exactly-once. You need to enable special settings on the Producer side (e.g. **enable.idempotence** which requires specific values for **max.in.flight.requests.per.connection,** retries and **acks**). On Consumer side it is more difficult. Due to failures you still can process messages more than once, then the solution is to deduplicate them, e.g. doing database upserts.
Another solution is to use Kafka Streams, which explicitly defines a setting "exactly once", but in practice this means that you get the exactly-once only by going from Kafka to Kafka.

Further read: What does Kafka's exactly-once processing really mean?

6. You don't monitor properly

Kafka Cluster is a distributed system. In such architectures a lot of things can actually go wrong. Monitoring is a necessity to know if everything works as it should. It is important to observe systems and define alerts.

Once we had a client who actually had a pretty nice dashboard with Kafka metrics. However, it was just red, because nobody fully understood the values which were there. One of the found issues was a problem with under replicated partitions, where hundreds of them were just not in sync. In the worst case scenario this could lead to service unavailability or even data loss.

HOW TO AVOID THIS MISTAKE?

After deployment observe and define alerts around most important Kafka and business metrics. There are various tools which allow you to export them quite easily (take a look at JMX or Kafka Exporters).

7. You upgrade project dependencies without looking at releases note

Since Kafka 0.11 clients are generally forward and backward compatible with Kafka brokers. When new versions are released the upgrade process is quite simple. However, a single version bump may lead to great problems. As an example let's take a look at the 2.1.0 release. That's the version that introduced the KIP-91 Intuitive Producer Timeout influencing the default value of the Producer retry parameter. It was changed from 0 to **Integer.MAX_VALUE**. Additionally, a new producer timeout was added. This change hasn't effectively introduced the infinite retries, it just enabled the retries by default, which with addition of **max.in.flight.request.per.connection** >1 (default 5) can lead to message reorder!

More on this change you can read in Kamil Charłampowicz blogpost — Does Kafka really guarantee the order of messages?

HOW TO FIX THIS MISTAKE?

Look at the release note ;)

What tools can help you when working with Kafka

Open-source Apache Kafka distribution includes a set of CLI tools allowing you to interact with the cluster and analyse its state and content. To make it easier, people created a lot of various helpful projects. Some of them offer similar features in a more user friendly manner, others extend the set of features.

- Kafka Visualization - tool visualising how Kafka Producers, Brokers and Consumers interact
- Kafka Topic Analyzer - CLI tool presenting various statistics and metrics related to Kafka topics
- Cruise Control for Apache Kafka - project helping to run Apache Kafka at large scale
- kcat (formerly known as kafkacat), kat, kafkactl, kcl, kt, plumber - various CLI tools allowing producing and consuming messages in different formats, including Avro and JSON, when interacting with Kafka

What's new in the latest Kafka versions?

Apache Kafka is actively developed and gets new releases every few weeks or months. It gets a lot of improvements and bug fixes. The most notable change done in recent years is the KRaft - the consensus protocol alternative. Its goal is to remove dependency on the Apache ZooKeeper. From Kafka 3.3 KRaft is considered as production ready (although it does not yet support some of the features). From 3.4 the ZooKeeper will be deprecated. Kafka 3.5 will contain mechanisms allowing migrating old clusters to Kraft, and from version 4.0 ZooKeeper will be removed.

Another interesting fact is Confluent invested in Immerok - a company offering managed Apache Flink service. That means Flink may become another viable Kafka messages processing method to be used natively with Confluent Cloud.

LEARN MORE ABOUT APACHE KAFKA

Apache Kafka is a mature tool for orchestrating, scaling, monitoring and tracing — all in an automated fashion. As software developers and architects, we love Apache Kafka because it comes with multiple softwares highly attractive for data integration - a clear choice to handle distributing processing.

Kafka has been on the market for almost 10 years, some of the materials describe quite old Kafka versions, however basic fundamentals for various versions are the same. If you want to dive into Kafka these are the basics you have to start from:

- **Documentation**
- **Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale**
- **Streaming Architecture: New Designs Using Apache Kafka and MapR Streams**
- **Slack Confluent**
- **SoftwareMill Tech Blog on Kafka topics**
- **Kafka Visualization Tool**

Our Apache Kafka Experts

Grzegorz Kocur

Senior DevOps with hands-on experience operating Kafka clusters - on-premise as well as in cloud computing environments, including Kubernetes

Krzysztof Atłasik

Seasoned developer with Kafka experience, certified from 2021

Krzysztof Grajek

Senior Software Engineer, 15 years of experience, working with PubSub-based production systems for 5 years

Krzysztof Ciesielski

Senior Software Engineer, 15 years of experience, working with Kafka-based production systems for 5 years

Michał Matłoka

Software Architect, certified for Apache Kafka in 2019. > 10y of software development experience, including big data technologies

Adam Warski

Scala & Distributed Systems Expert, CTO at SsoftwareMill, OSS Developer. Specialises in developing high-performance, clustered, fault-tolerant software

Maria Wąchal

CMO and Technology Evangelist driven by a passion for modern technology and a commitment to making complex concepts understandable

Data streaming platforms enable companies to connect, store, process, govern, and share data throughout an organization so businesses can build smart, real-time applications. A complete data streaming platform serves as the foundation for a modern, decentralized infrastructure.

Businesses need a data streaming platform that doesn't require spending valuable engineering resources on building foundational tooling or maintaining platform performance and availability across all your environments. Confluent provides everything you need to implement Kafka use cases quickly, reliably, and securely from inception to large scale, so you can focus on developing real-time apps that drive real business value.

Roger Illing

VP of Sales CEMEA

CONFLUENT

At SoftwareMill, we are a [Confluent Preferred Partner](#).

NEED HELP WITH APACHE KAFKA?

Need a storage solution for streaming data? We are Apache Kafka experts with a long history of developing Kafka and using Confluent Platform on production. Ready to help you leverage Confluent best practices and tools.

We offer Apache Kafka Training and Consulting. We're a Conflent Preferred Partner and we really enjoy using their toolset. There is no one way to build software, but from an engineering standpoint we find Apache Kafka very reliable when processing data in motion. Need a helping hand with implementing Kafka in your project?

LET'S TALK ABOUT YOUR PROJECT NEEDS.

WRITE US AT **HELLO@SOFTWAREMILL.COM**

SOFTWAREMILL

Stay up to date with our news

blog by SOFTWAREMILL

DATATIMES
Big Data and ML newsflash

visit our website