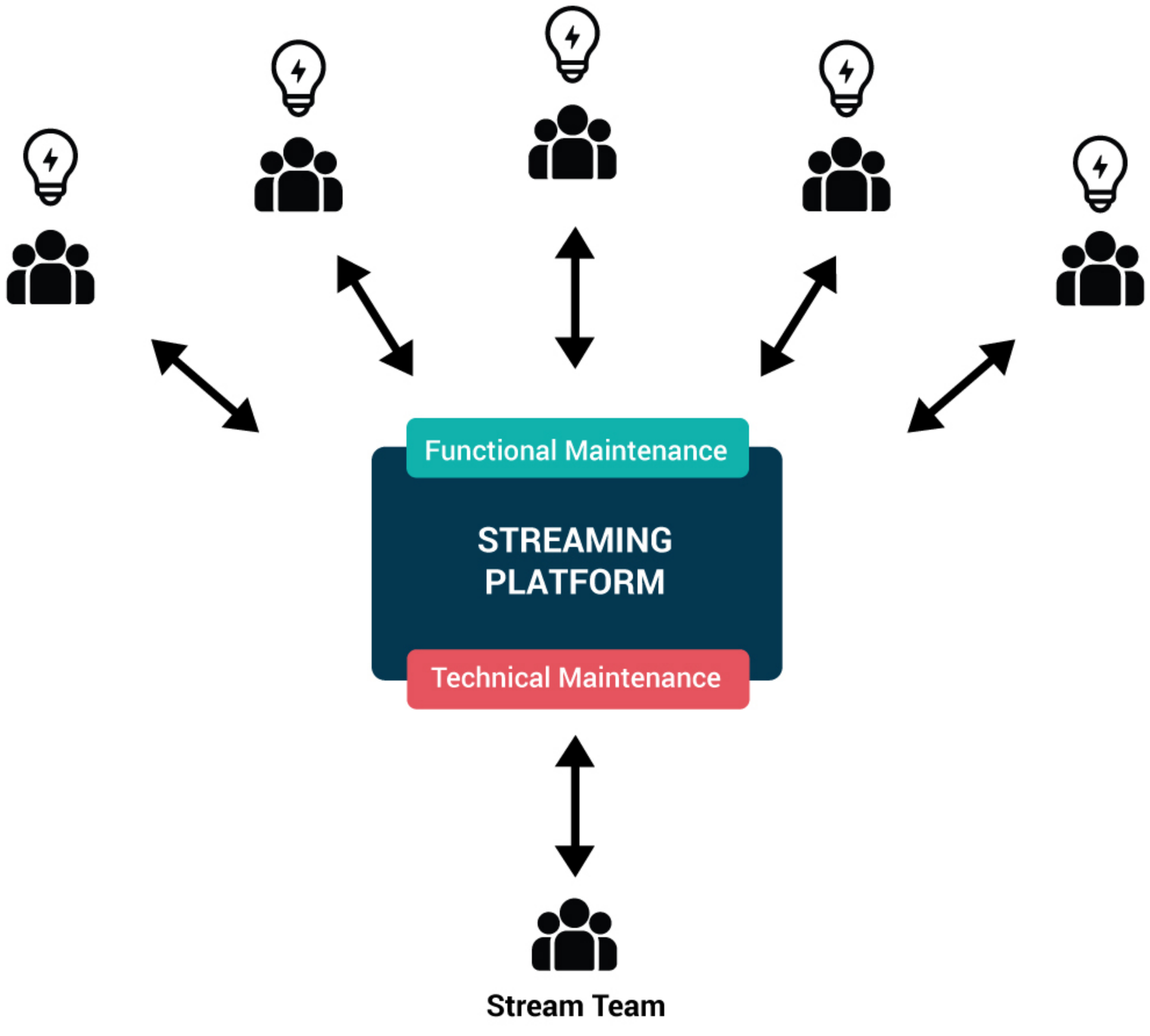
axual

The hidden costs and risks of implementing Apache Kafka for your enterprise

Situation: enterprise streaming platform

Whenever a choice is made to leverage the streaming capabilities of Apache Kafka, very often the Apache Kafka platform takes a central place in an organization, becoming the Enterprise Streaming Platform. Multiple teams from different departments all hook up their applications, and there is usually a stream team, the team whose responsibility it is to help other teams onboard, maintain the platform and even provide 24/7 support.



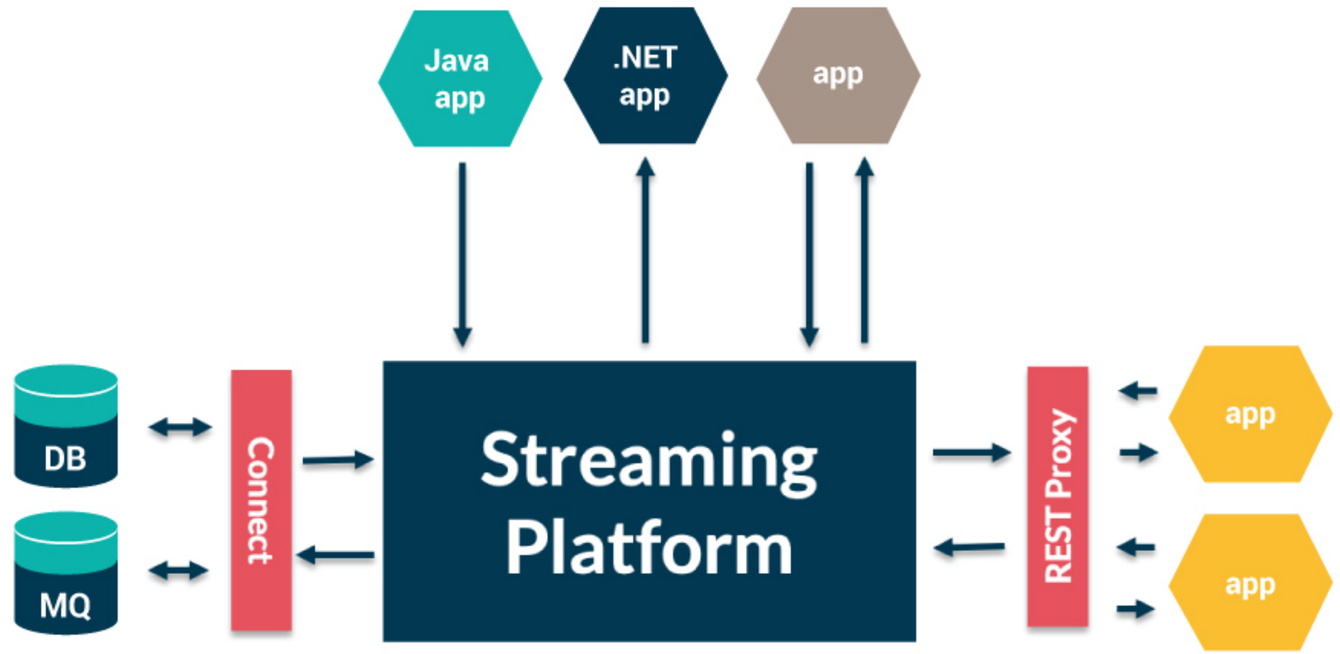
For those unfamiliar with Kafka, it is a fault-tolerant, scalable and publish-subscribe messaging system that lets enterprises develop distributed apps and run web-scale internet businesses like Twitter, LinkedIn and others. Kafka is being adopted by an increasing number of enterprises because of its tremendous capabilities; however, like every technology, it comes with its own complexities (Gartner, 2018). When it comes to Apache Kafka implementation, various considerations in terms of costs and risks come into play such as deciding which compute instance would be suitable for the brokers, determining the non-ephemeral storage, end-to-end security measures, recovery strategies, observability practices, onboarding developers, implementation, maintenance costs, etc.

In order to better understand what the hidden costs and risks of implementing Apache Kafka are, we have identified below the 4 most important factors to bear in mind before adopting Apache Kafka for your business.

FACTOR 1

The Knowledge Gap

A streaming platform based on Apache Kafka has a central place in an organization. It connects a large amount of applications streaming data, and it is used by users in different departments. The two most notable types of users are the developer and the platform operator. The developer is focused on implementing a use case powered by streaming data, whereas the platform operator is concerned with stability and availability of the platform. Apache Kafka has a steep learning curve for both personas that can make these tasks difficult.



Planning deployment of Apache Kafka

As with any application running at a scale, Kafka needs a huge level of planning and configuration in terms of OS, network, hardware, and application. The stability and performance of Kafka is heavily dependent on RAM capacity, file system tuning, disk throughput, and network latency. Kafka also depends on Apache ZooKeeper that should be deployed separately from Kafka to avoid the possibility of a single point of failure (Stephanie, 2018).

Nowadays, the use of cloud platforms is more and more widespread. This poses the question: should a streaming platform also be deployed and maintained using a platform like OpenShift, Kubernetes or DC/OS? Platforms like these are designed for stateless, ephemeral (micro)services, stateful platforms such as Apache Kafka or Apache Zookeeper are a different beast. Getting those platforms to run in a distributed way, responding to component changes while maintaining state, is hard and requires a lot of experimentation and confidence in the platform team.

Unlimited configuration options

Kafka classifies messages into topics that are then divided into partitions. Every partition is an ordered queue of the messages allocated to a particular consumer instance. In general, an increasing number of partitions results in higher throughput but at the cost of latency, availability, and memory. So, planning the amount of partitions is crucial for developers, when setting up the topic. However how do you choose this number when you really don't know what it means or what the consequence is of choosing a value which is too low or too high? At the same time, client libraries for Apache Kafka have an almost unlimited amount of configuration options for consumer, producer or streaming apps. Some of the configuration options have a large effect on delivery guarantees or durability of messages on a topic. Knowing the implication of changing certain configuration options requires a solid understanding of the platform, and thus time and capacity to get acquainted.

The bottom line is that Apache Kafka is complex to use, both from a developer and operator point of view. This requires both the connecting teams and the stream team to boost their knowledge, which is distracting from the actual value someone wants to get from the platform: business value.

FACTOR 2

A “mistake” is easily made

Security is important for any business, but at the scale of an enterprise, it becomes even more important to embed security in the system, and "turn it on" by default. It is widely known/accepted that you should not trust the network. People make mistakes, and sometimes they don't play by the rules. You want to guarantee to your customer that his/her data is safe. There are three components in Kafka Security:

Authentication via SSL or SASL:

Mutual SSL can be used for client applications to authenticate to the streaming platform. As the word "mutual" suggests, both parties trusting each other is important. This mutually trusted connection is a base for authorization, which can be done on a topic level.

Authorization via ACLs:

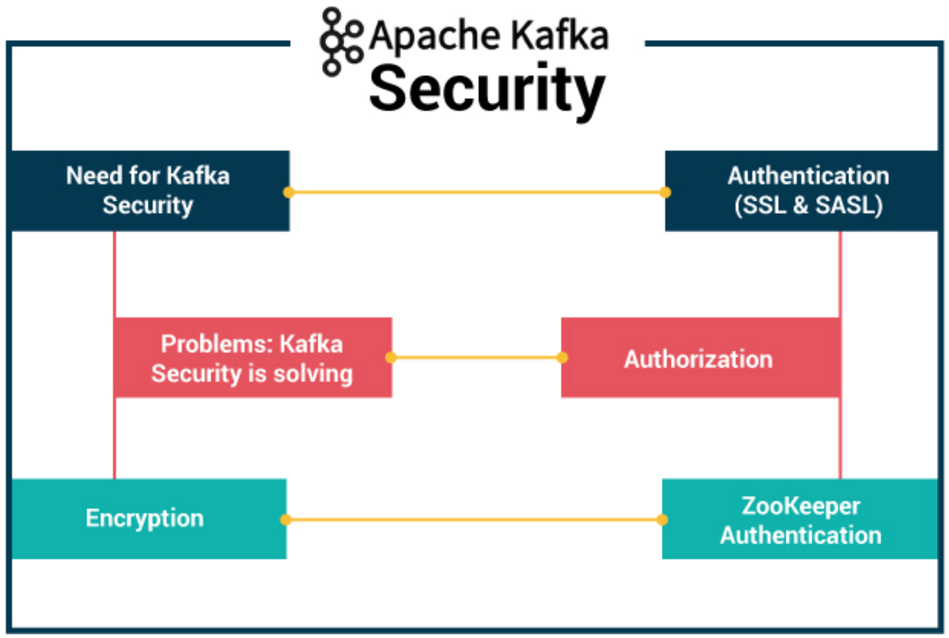
Once the clients are authenticated, now the Kafka brokers may go ahead and run them against access control lists (ACL) to know whether or not a specific client would be approved to write or read to some topic.

"User alice can view topic finance"
"User bob cannot view topic trucks"

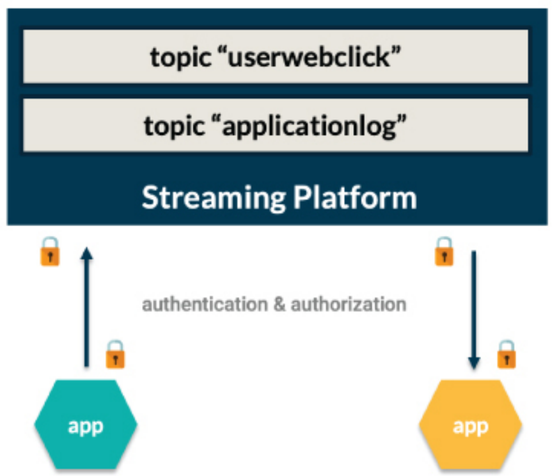
Once Kafka clients are authenticated, Kafka must decide what they can and cannot do. Authorization comes into play here, controlled by the Access Control Lists (ACL). ACLs are based on predefined rules regarding user access. For instance, User ABC can('t) do XYZ Operation on XYZ Resource from Host XYZ. Maintaining those rules for tens or hundreds of applications is cumbersome, and a mistake is easily made.

Encryption of in-flight data using SSL / TLS:

This encrypts the data between producers and Kafka and consumers and Kafka. This is a general pattern everyone uses being on the web. That's the "S" of HTTPS (that attractive green lock you notice everywhere on the web).



By default, an Apache Kafka setup is unsecured, allowing for unauthorized access to valuable customer data. You don't know what is happening with your customer's data. Also, because of misconfiguration, data from one environment could corrupt data in another. Because of the complexity of the setup in an enterprise environment, implementing security is hard and requires time to implement. Just having security support or features in a platform like Apache Kafka is great, but that does not mean that truely ***leveraging*** those features is embedded in the way of working of an organization.



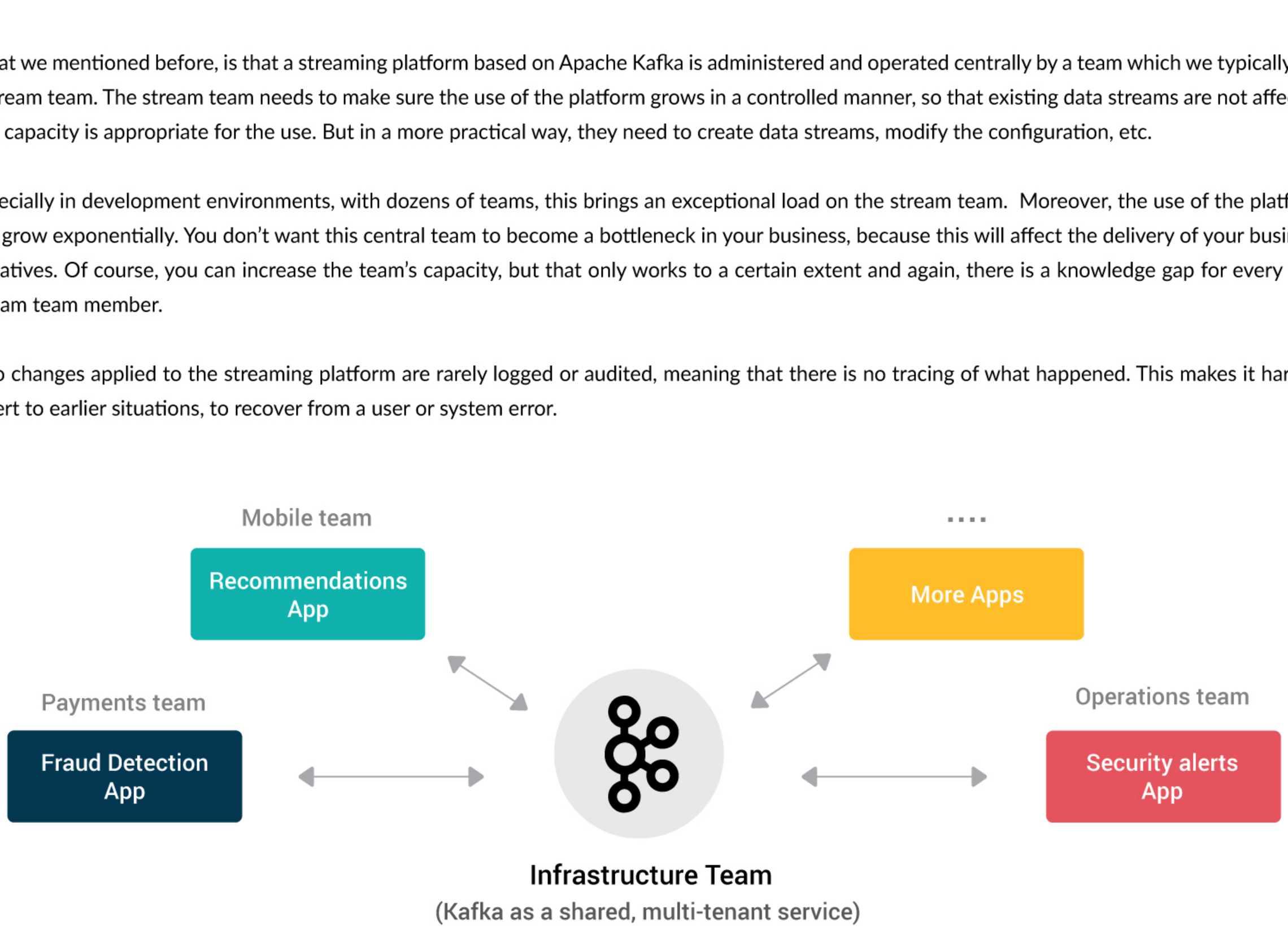
FACTOR 3

You can only move as fast as your platform team

What we mentioned before, is that a streaming platform based on Apache Kafka is administered and operated centrally by a team which we typically call a stream team. The stream team needs to make sure the use of the platform grows in a controlled manner, so that existing data streams are not affected and capacity is appropriate for the use. But in a more practical way, they need to create data streams, modify the configuration, etc.

Especially in development environments, with dozens of teams, this brings an exceptional load on the stream team. Moreover, the use of the platform will grow exponentially. You don't want this central team to become a bottleneck in your business, because this will affect the delivery of your business initiatives. Of course, you can increase the team's capacity, but that only works to a certain extent and again, there is a knowledge gap for every new stream team member.

Also changes applied to the streaming platform are rarely logged or audited, meaning that there is no tracing of what happened. This makes it hard to revert to earlier situations, to recover from a user or system error.

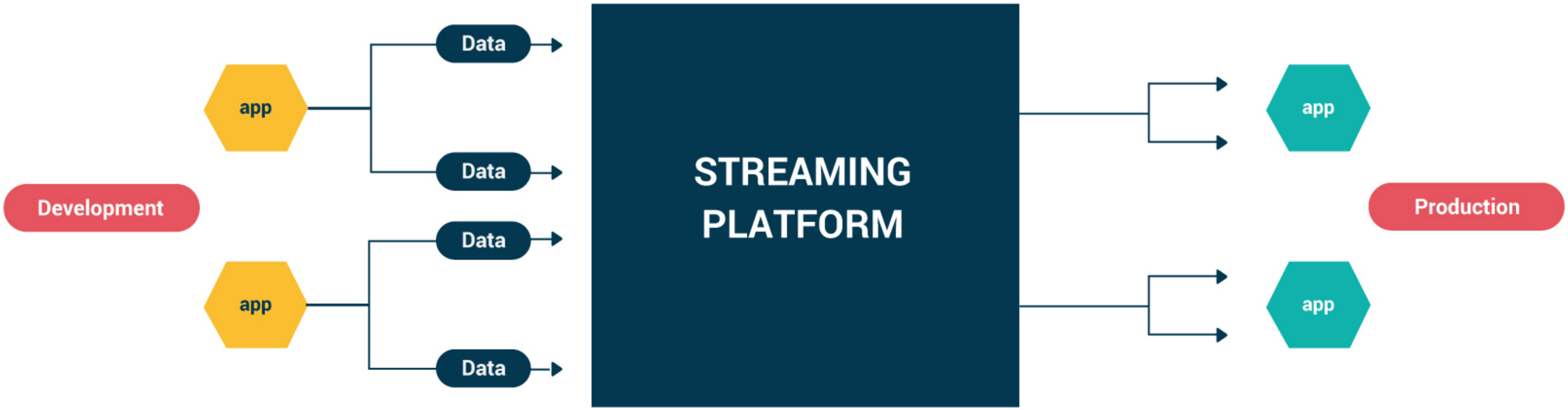


FACTOR 4

Unclear Ownership

When the platform is in use for 1-2 years, dozens of streaming use cases have been realized and put in production. In a true DevOps manner, teams maintain their applications, from DEV to PROD (Development to Production). But, their applications are powered by data. Data potentially produced by applications owned by other teams. And this is just a simple example. What if data traverses multiple stages from source to target?

When problems start to occur, how do they know where to go to resolve the issue? If the stream team doesn't have an administration of the Kafka topics and its users, this search might lead to nowhere. This leads to an increase in downtime of the application, potentially business critical or mission critical applications.



Summary

As shown in the aforementioned factors, deciding to use Apache Kafka as the enterprise streaming platform is one. Putting it to good use, while respecting all the enterprise requirements is another. There are serious costs and risks involved, and it is good for anyone to realize this. Nevertheless, Apache Kafka can be a great fault-tolerant, scalable and publish-subscribe messaging system when put to good use. We hope we gave you some insights In how to make optimal use when deciding to use Apache Kafka. If there are any questions left, don't hesitate to contact us anytime.

The Knowledge Gap

A "mistake" is easily made

You can only move as fast as your platform team

Unclear Ownership

References

Do you have questions, you want to have a demo of Axual or try it yourself?

Request a Demo