The case for API Management-powered Kafka

Driving more Kafka ROI with Event-Native API Management

GRAVITEE

Introduction

If you're a business that's invested heavily in Apache Kafka, you are likely hoping to get a return on investment via lower time to value with streaming data. You're also hoping to streamline and standardize your data pipelines and centralize around one operating model for processing data with real time applications.

But to get to the next level of unlocking real business value from your data in Kafka, you need a consistent way for application developers across your organization to publish and subscribe to data in Kafka. Even further, to truly unlock value from your data streams, you need to make that data accessible to customers outside your business. And to reach the final level of value, you need to be able to monetize your data streams in Kafka.

The easy path – that doesn't scale

Of course, the easiest way to give application developers access to Kafka data is to teach them all how to use the Kafka client and have them write their own applications. This requires onboarding each developer into an integrated development environment, ensuring that they follow best practices, and teaching them the nuances of the Kafka client. That's not too difficult, but trying to do this across the entire organization can be a significant challenge.

What is **certainly** not scalable is trying to give access to **external** users this way. As an analogy, consider your production databases. Would you give external applications direct SQL access to your database? Probably not. What about telling your customers to write applications using a database wrapper like SQLAlchemy? Again, not likely.

These tools can occasionally suffice for internal use, but opening them up as the standard method for external developers to access your data – and bring you business value via time to value and monetization – will open the door for performance bottlenecks and security vulnerabilities everywhere you look.

The story is the same for Kafka. Fortunately, for traditional relational databases, the way to scalably expose data is well known - **put APIs in front of them**. The benefits of putting an API in front of a database include, to name just a few:

What about putting a REST interface in front of Kafka and using that as the preferred method allowing developers, internal and external, to access your data streams? This works! It is a real, scalable solution that can become a standard for your organization.

BENEFITS OF APIS IN FRONT OF A DATABASE

- Utilizing a standardized, secure protocol (e.g. HTTP) for connecting to the backend.
- Authenticating each request via standard internet security protocols (e.g. OAuth).
- Separation between the client and the server via a proxy that can moderate and optimize requests to the backend (via connection pooling, caching, etc).

Great, so I can just use the Kafka HTTP client?

You certainly can try using the native Kafka HTTP client, and it could work just fine for you. But digging a bit deeper, some additional questions immediately rise to the surface:

SHOULD I USE THE KAFKA HTTP CLIENT?

- How can you scalably provide some kind of governance on top of HTTP requests that can be easily translated to the Kafka backend?
- How can you ensure the stability of your service, protect against too many requests and DDoS attacks, and encrypt communication between client and server?
- How can you provide documentation, monetization, and customization for clients to access the data in a self-service manner?
- What if you store my data in Kafka in a binary serialization format like Avro? Should you send the data back to clients in Avro format, or should you first convert it to a human-readable format like JSON?

These are tough questions that don't have an easy answer when using a native HTTP client of any type. Fortunately, this is why API management solutions exist. A full-lifecycle API management platform is a tool specifically designed to provide governance, rate limiting, authentication, authorization, documentation, and monetization on top of vital backend services.

This story doesn't end with Kafka, either. What if you have other backend systems that you need to expose like RabbitMQ, MQTT, Solace, GraphQL servers, etc - not to mention traditional REST and SOAP APIs. Wouldn't it be great if there was an API management platform with a gateway purposely built to handle data streams alongside traditional REST backends and a platform that provides value across the entire API lifecycle, that can provide access to these types of systems in a standardized manner to unlock self-service and monetization?

Good news – Gravitee APIM is that tool!

Fortunately, Gravitee can do all those things. Here's how:

- Gravitee APIM has a reactive gateway that can handle both synchronous and asynchronous APIs using the same platform, via the same interface and tools.
- Gravitee has native support for multiple entrypoints to backend streaming systems. Clients can interact with Kafka via regular HTTP GET or POST requests, Websocket connections, and server-sent events. Gravitee can also push data to clients via Webhooks whenever an event happens in Kafka.
- Gravitee has policies for data transformation between JSON, XML, Avro, and, soon, Protobuf. Gravitee can load schemas from ConfluentSchema Registry and compatible systems, and apply the schema on publish and subscribe to Kafka. This way, clients can consume messages from Kafka in whatever format they feel comfortable with while also storing the data in Kafka in whatever format is best optimized for your use case (likely a binary format that would be hard to work with without a serializer in between).
- Gravitee's gateway can run anywhere – on-prem, in the cloud, on VMs, and on Kubernetes. Gravitee's gateway has industry-leading performance and is open-source. Gravitee also has a Kubernetes operator purpose-built for the container ecosystem.

Lastly, Gravitee has the proof-points. We are used as an API gateway in front of Kafka by leading companies in manufacturing, banking, and entertainment across the globe. And with our newest APIM release, we are providing a revamped user interface on top of these value-adding features that will make unlocking value from your streaming data easier than it has ever been.

Consistent, secure, and flexible Kafka exposure

Make Gravitee work for you

If what you've read has resonated so far, please reach out. We will be happy to show you a product demo, chat about your use cases, and work together to help you unlock value from your streaming data. Happy streaming! Book a demo

How to contact us

Contact Us

If you're interested and want to reach out, you can contact us.

Book a Demo

If you'd like to skip (some of) the sales pitch and see a demo, you can book one of those.

Community Forum

If you want to gvie OSS a go, check out our community forum where you can find links to our GitHub repo and connect with the folks who have driven over 350,000 Docker pulls / month.

The Easy Path – That Doesn't Scale

Of course, the easiest way to give application developers access to Kafka data is to teach them all how to use the Kafka client and have them write their own applications. This requires onboarding each developer into an integrated development environment, ensuring that they follow best practices, and teaching them the nuances of the Kafka client. That's not too difficult, but trying to do this across the entire organization can be a significant challenge.

What is **certainly** not scalable is trying to give access to **external** users this way. As an analogy, consider your production databases. Would you give external applications direct SQL access to your database? Probably not. What about telling your customers to write applications using a database wrapper like SQLAlchemy? Again, not likely.

These tools can occasionally suffice for internal use, but opening them up as the standard method for external developers to access your data – and bring you business value via time to value and monetization – will open the door for performance bottlenecks and security vulnerabilities everywhere you look.

The story is the same for Kafka. Fortunately, for traditional relational databases, the way to scalably expose data is well known - **put APIs in front of them**. The benefits of putting an API in front of a database include, to name just a few:

> **BENEFITS OF APIS IN FRONT OF A DATABASE**
>
> - Utilizing a standardized, secure protocol (e.g. HTTP) for connecting to the backend.
> - Authenticating each request via standard internet security protocols (e.g. OAuth).
> - Separation between the client and the server via a proxy that can moderate and optimize requests to the backend (via connection pooling, caching, etc).

What about putting a REST interface in front of Kafka and using that as the preferred method allowing developers, internal and external, to access your data streams? This works! It is a real, scalable solution that can become a standard for your organization.