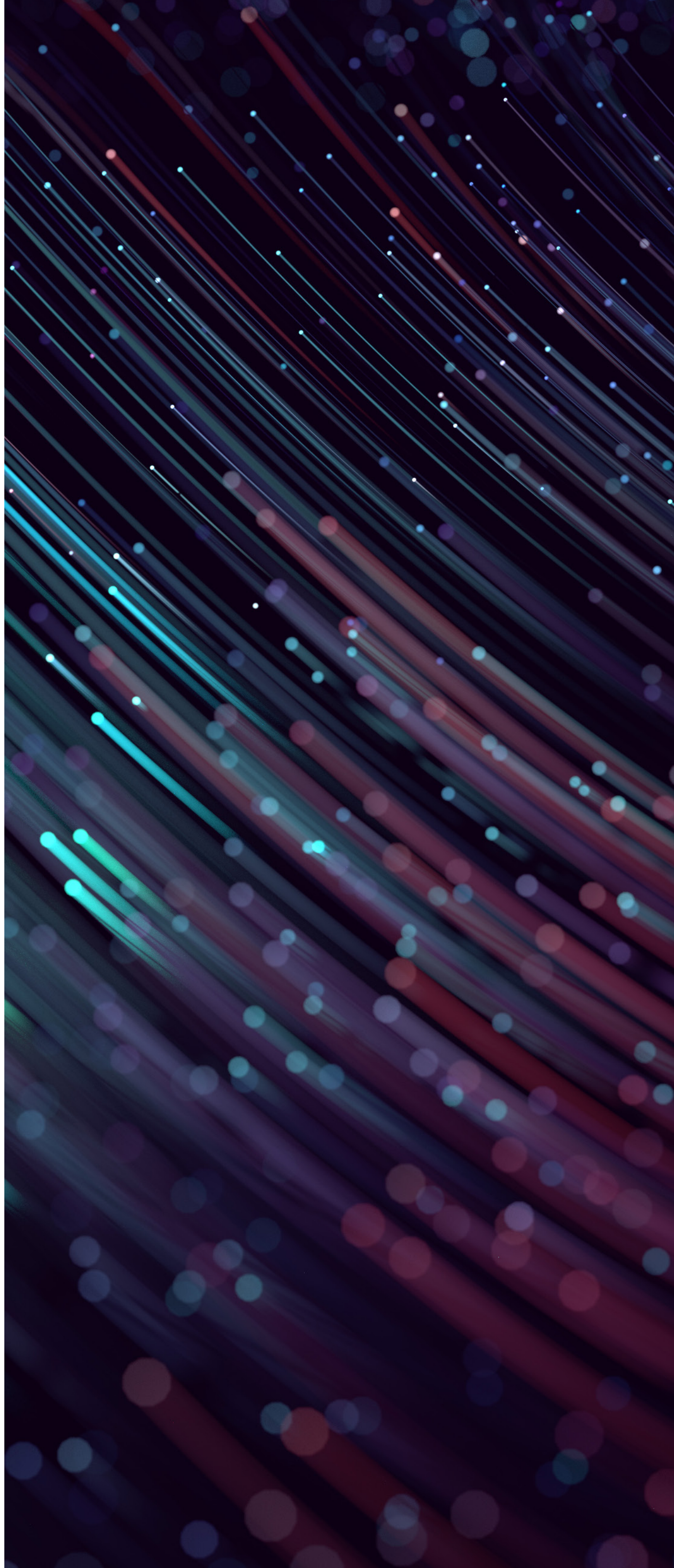
CONFLUENT

From monoliths to microservices: building event-driven systems

Table of Contents



What's inside

This eBook is designed to help IT leaders and practitioners understand how they can use Confluent to build an event-driven architecture and enable real-time business applications and automated backend operations.

You’ll see how five organizations across a wide range of industries leveraged Confluent to build a new class of event-driven microservices—to completely decouple services from one another and to serve as the data backbone for their data-streaming applications. As a result, these organizations could rapidly build and deploy business applications with greater flexibility, at scale, and be more responsive to customer demands using real-time data.

With our help, you can too.

1

Modernize Your Application Architecture with Event-Driven Microservices

Microservices have emerged as a widely discussed and adopted way to build modern and scalable applications. They are easier to build, manage and maintain than monoliths due to smaller code bases; they isolate complexity, allowing for smaller, more agile teams to create services; and they are flexible—allowing the use of a variety of platforms, programming languages, and tools, since these choices affect only an individual service and a small team at a time.

While a microservices architecture certainly makes for a very promising pattern for application modernization efforts, refactoring applications and modernizing existing applications is no small feat. Enterprises have to do this thoughtfully or risk adopting a short-sighted strategy to meet their innovation and business needs.

Over the course of the next few sections, we'll discuss the challenges with current approaches to microservices development and modernization and the evolution of a new approach that's event driven. You'll also see how Confluent empowers your developers to innovate faster while incrementally refactoring and migrating your existing applications, and get examples of five enterprises that have successfully modernized their applications.

Why You Need to Break the Monolith

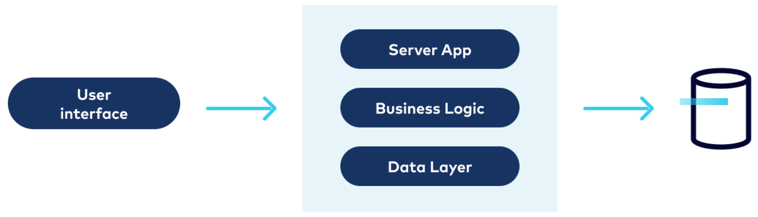
Before we dive right into the specifics of the microservices architecture, here's a quick recap on the importance of breaking the monolith.

Monolithic applications, despite having a logical modular design, were built as a large system with a single code base, sharing access to a single database and deployed as a single unit. While this approach worked a decade ago, this kind of tightly coupled architecture has many drawbacks in today's environment, where agile development to accelerate continuous innovation is critical to competitive success. As a result, the business suffers from several common issues:

1. **Slow release/update cycles:** Even the smallest change made to existing functionality in one feature requires end-to-end testing and a full redeployment of the entire application, causing slow development cycles. As more new features and functionality are added, applications become bloated and unmanageable.

2. **Higher risk of failure:** When the entire application is packaged as a single unit, a simple error in any of the modules can bring the entire application and service down.

3. **High operational costs and technical debt:** Any changes to the technology stack are expensive and increase overall technical debt.



A Typical Monolithic Application

These challenges, along with the need to innovate at speed and scale, have resulted in organizations seeking to or adopting a microservices architecture—a set of autonomous services based on specific business capabilities that can be developed, modified, tested, and scaled independently.

This type of architecture has many advantages. Developers have tremendous flexibility and autonomy to work on individual services using the framework of their choice and release new features quickly and independently. But as is so often the case, new technologies that solve problems in one area introduce new problems in others. And microservices are no exception.

Many software architects face two primary challenges in their modernization efforts:

1. How to build truly self-contained microservices that have complete development and deployment autonomy while also ensuring resiliency of the connected services. How do you completely decouple one service from another and pass data and share state with the apps they serve? If one service crashes, how do you ensure it doesn't affect another? As you scale to thousands of services, how do you not sacrifice security?

2. While using a microservices architecture is a viable strategy, it's also a radical shift. So, how do you gradually move from a single monolithic architecture to many small services without breaking mission-critical functionality? How can you look to incrementally migrate to the cloud while still maintaining data integrity across all your services?

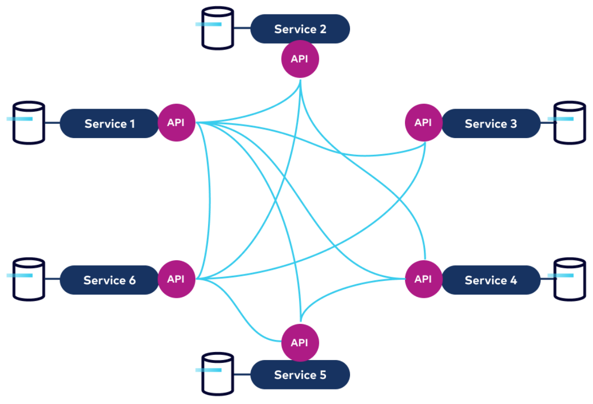
If you're starting a greenfield project with a clean slate, you can sidestep many of these transition issues and gain tons of advantages. However, you still want to avoid the challenges described in the next section on "Pitfalls of current microservices communication approaches."

If you're transitioning from a monolith to a microservices architecture, the section on "Build a modernization strategy for incrementally migrating monoliths to event-driven microservices" will be especially interesting to you.

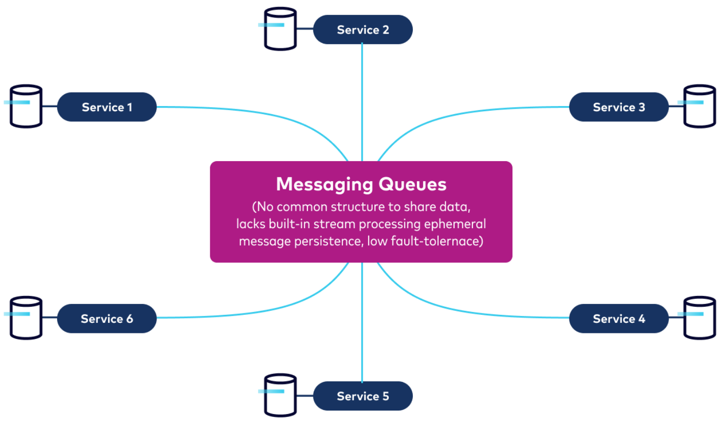
Pitfalls of current microservices communication approaches

The best approach to building microservices and orchestrating them all to effectively communicate hasn't always been crystal clear. Over the years, two main approaches have emerged for inter-service communication.

1. **REST APIs** for synchronous one-to-one communication using HTTP request/response



2. **Messaging technologies** for asynchronous communication using point-to-point (one-to-one) and pub/sub (one-to-many) messaging patterns.


Service 2
Service 1
Service 3
Messaging Queues
(No common structure to share data, lacks built-in stream processing ephemeral message persistence, low fault-tolernace)
Service 6
Service 4
Service 5

An overlooked aspect of using REST APIs is their tightly coupled communication approach, where microservices become deeply intertwined with each other. And while messaging queues (MQs) provide a loosely coupled architecture for inter-service asynchronous communication, they don't go far enough to address the challenges with REST APIs and monoliths.

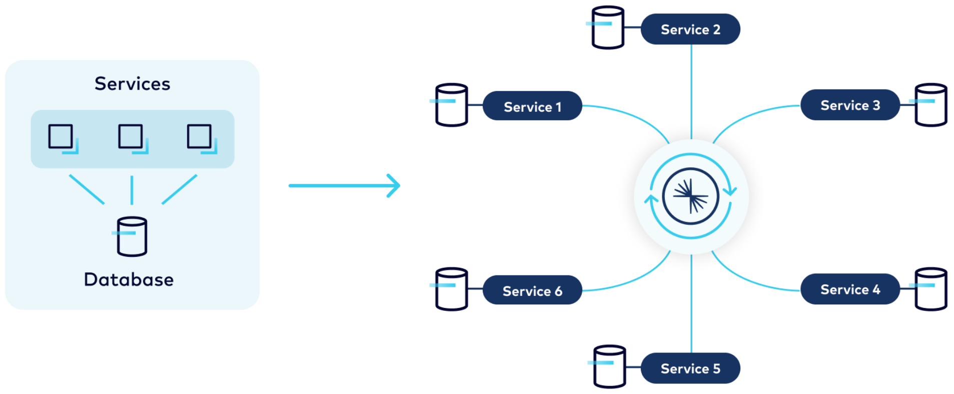
There are a few drawbacks that can threaten to turn a set of microservices back into a monolithic structure:

1. **Slow release and update cycles:** Point-to-point coupling with REST APIs requires each microservice to have built-in functionality on how to connect, handle errors, and incorporate business logic. MQs lack a common structure and governance for message sharing, which leads to inter-team dependencies and slows development cycles
2. **Poor reliability and scale:** The tight coupling of services with REST APIs requires costlier linear scaling vs. the horizontal scaling that's more common in modern cloud architectures. MQs have significant challenges dynamically scaling applications due to inherent limitations with low throughput leading to performance bottlenecks
3. **High operational costs and technical debt:** At scale, there is high operational overhead of managing hundreds or thousands of REST APIs built by tens or hundreds of developers across the enterprise. Legacy messaging queues, originally developed decades ago, are expensive to maintain, increase technical debt, and cannot easily integrate with new cloud-native services and applications

So while many organizations have started their application modernization journey using microservices, they quickly realize that using REST APIs and MQs for interservice communication at scale isn't sufficient.

Build the next generation of microservices that are event-driven

Event-driven microservices is a new paradigm for application development and modernization centered around the concept of data as events. An event-driven paradigm completely decouples services from one another, giving services the autonomy they need to evolve freely.



The benefits from taken an event-driven approach to microservices are threefold:

1. **Increase developer velocity:** You can remove inter-service bottlenecks and dependencies, meaning that your teams can connect to any system they want, while maintaining data quality, schema compatibility and version control. Your teams can also build and scale stateful services and stream processing applications, faster entirely with SQL syntax
2. **Build highly reliable and fault-tolerant microservices:** You can power disaster recovery and high availability use-cases and design a system of record for new applications that your teams build,while offloading the management to Confluent
3. **Reduce technical debt and messaging TCO:** You can begin to reduce the operational costs and technical debt you've incurred over the years Migrate your supporting data infrastructure at your own pace to a modern event-driven architecture

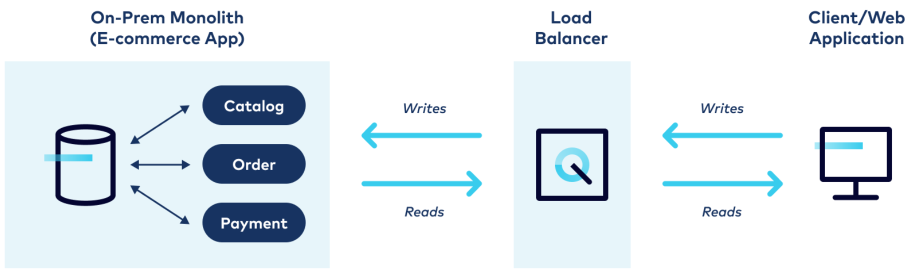
Build a modernization strategy to incrementally migrate monoliths to event-driven microservices

Now that we've established why adopting an event-driven microservices architecture is a better fit for those who are seeking to modernize their applications, how to refactor a monolith into smaller services requires careful planning. When breaking a monolithic application of strategic value, you have to determine the most logical place to start to refactor to avoid getting stuck in how your current application is built.

Therefore, most organizations adopt an incremental approach rather than a one-time complete rewrite. While doing this gradually can sometimes take an extraordinarily long time, this approach mitigates the risk of failure and reduces the need for significant budget, and personnel.

Let's walk through an example of decomposing a monolithic system into a cloud-optimized microservices-based system.

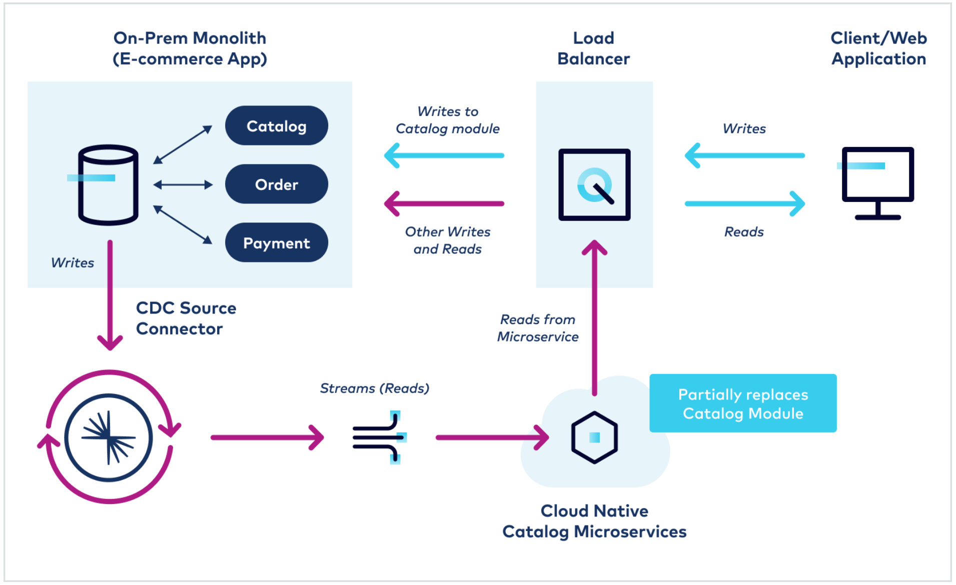
This retailer is modernizing its monolithic app.



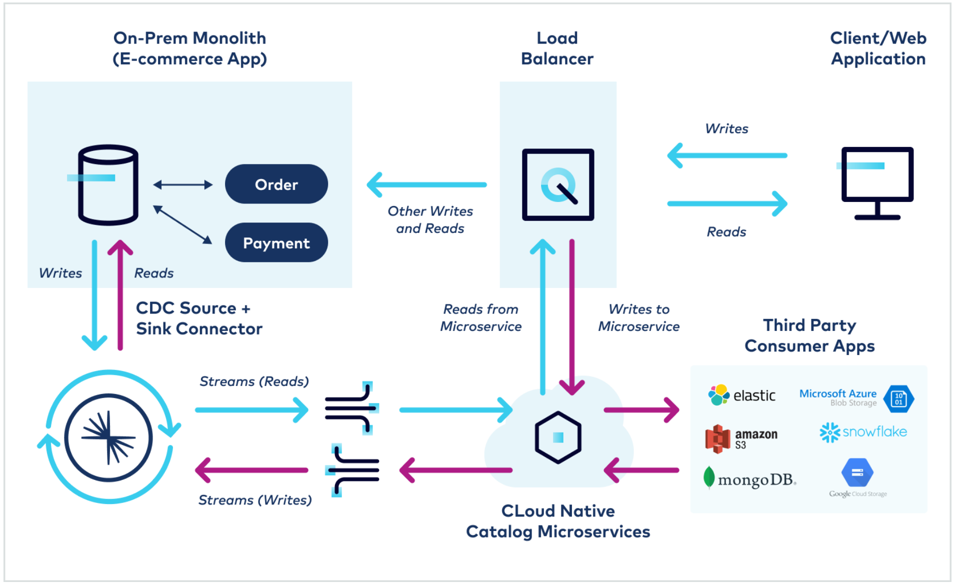
E-Commerce Application (Monolith)

The monolith consists of a Catalog service, an Order Service and a Payment Service. This app authorizes a customer, takes an order, checks products inventory, authorizes payment and ships the ordered products. There's a database (PostgreSQL, Oracle, etc) shared across multiple modules within the application.

A web client or front-end application service reads and writes data to the on-prem monolith. In most enterprises where the e-commerce app needs to scale to accommodate the number of requests they're getting, the simplest way to allow your application to handle more load is to direct traffic to a load balancer instead of the monolith and run multiple copies of the application.



Step 1: Route Reads to Catalog Microservice



Step 2: Fully Replace Catalog Module

Step 1 to start modernizing your application is to split your reads and writes to the Catalog service by reconfiguring the load balancer and/or creating a new version of the client application, which places **a read request to a different endpoint**.

The writes **continue to go to the catalog service** in the monolith and get written into the database, so other services can consume this data. Using a CDC Source connector to the PostgreSQL or Oracle database, these writes then get captured as events that get written to Confluent.

At the same time, you can design a cloud-native catalog microservice to serve up the reads from the web application.

Once the reads have been verified as being rerouted to the new catalog microservice, step 2 is where you would fully architect this catalog service to no longer be dependent on the monolith.

To achieve this, you would follow the following steps:

- Route the writes from the web application to the new microservice. This could be performed by releasing a second version of the client application, which places a write request to a different endpoint
- The data from the writes gets written to Confluent
- A CDC sink connector copies the data to the shared Oracle or PostgreSQL database to ensure the data is kept up to date for the other services, which also share access to the same database.

At the same time, you can begin building data pipelines between the new catalog microservice to other destinations on AWS, Azure, and GCP, including third-party applications such as Snowflake, MongoDB, or Databricks. You can continue taking this incremental approach to modernize the other modules within the monolith.

Let's now look at five Confluent customers who have taken this event-driven approach to build net new cloud-native applications and to modernize existing applications built as monoliths, or REST APIs or MQs for interservice communication.

2

eBay Korea cuts go-to-market time in half for its growing e-commerce platform

ebay

Challenge:

As one of the largest e-commerce companies in Korea, eBay Korea operates three online retail platforms: Gmarket, Auction, and G9. With Korea's e-commerce market more than doubling over the past five years and the uptick in online shopping due to the pandemic, eBay Korea saw a significant surge in growth.

While the increase in sales and revenues was welcome, the growth highlighted issues with the company's legacy e-commerce platforms, which had been in place for more than 15 years. Based on a monolithic architecture with a single large application and single database, the three platforms were becoming increasingly difficult to develop and maintain. Moreover, updating the code in one area often led to unexpected problems in other areas of the platform.

Solution:

To enable continued growth and increased development velocity, eBay Korea is overhauling its e-commerce platforms, replacing monolith with a microservices architecture for low latency communication between decoupled microservices.

It took only five months for eBay Korea to build an enterprise-grade event-driven architecture platform. Using a Confluent-enabled microservices architecture, eBay Korea was able to significantly reduce dependencies between systems, making them much easier to maintain and operate and increasing overall productivity.

Since the kickoff in May 2021, eBay Korea has experienced no downtime in delivering or operating services. eBay Korea can now handle 1 million transactions per second with no data loss. Multi-region clusters ensure full disaster recovery
and service stability.

"Using Confluent-designed microservices has allowed us to cut our go-to-market time in half. Confluent plays a pivotal role in delivering messages between microservices-based applications and connecting all our services as a core foundation."

Hudson Lee

Leader of Platform Engineering, eBay Korea

Read the full case study here

3

Affin Hwang modernizes and digitizes core operations on AWS and Azure to open new business opportunities

AFFIN HWANG CAPITAL
Asset Management

Challenge:

With more than RM 80 billion ($19 billion USD) under management and close to 200,000 clients, Affin Hwang Asset Management Berhad is among the leading asset management firms in Malaysia.

Over the past two decades its IT infrastructure expanded organically as it grew to meet the needs of institutions, pension funds, government-linked companies, and other clients. As a result, the back-end landscape included multiple siloed legacy systems and batch processes that, although fully functional, were beginning to strain under the load placed on them and affect the customer experience.

Solution:

To modernize its operations, Affin Hwang Asset Management launched a digitization initiative that is being powered in part by real-time data streaming with Confluent. Confluent is enabling Affin Hwang Asset Management to integrate siloed systems, reduce batch processing, and explore new product offerings. "Because we're running on AWS and Azure currently, when we adopt a technology it has to work well in a multi-cloud environment," says Woo. "So Confluent was a good fit for us."

For the PoC, the team wanted to move transaction data for investments and redemptions from multiple digital touchpoints—including its customer portal and mobile app—to a Snowflake data warehouse. They used microservices to publish the transaction data to a Kafka topic, and used Confluent's fully managed Snowflake Sink Connector to persist the data to Snowflake. The team uses stream processing to transform the data as it is in transit. Affin can therefore generate a unified 360-degree view of their customers and provide the relevant data to different parts of the organization. Stream processing has also helped reduce their overall cloud data warehouse spend.

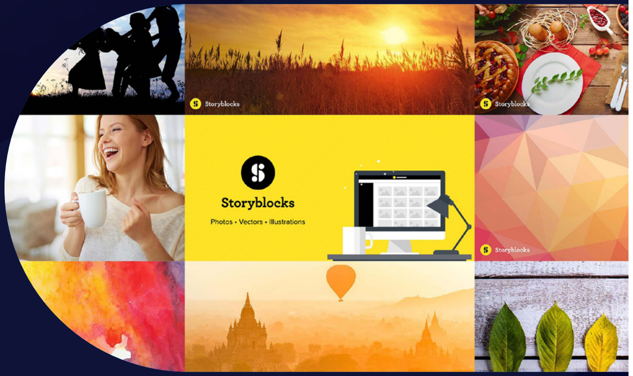
"Data that took us a whole day to get previously is now streaming via Confluent and available in seconds," says Woo. "This is central to our digitization initiative because when a new customer is onboarded or a transaction is executed, that data needs to be reflected as soon as possible, and with Confluent we can do that."

Allen Woo
Chief Innovation Officer, Affin Hwang Asset Management

Read the full case study here

4

Storyblocks reduces technical debt from REST APIs with Confluent and GCP



Storyblocks

Challenge:

Ranked the fourth-fastest growing media company in the U.S. by Inc. Magazine, Storyblocks is the first unlimited download subscription-based provider of stock video and audio. As Storyblocks transitioned from scrappy disruptor to major industry player, it first began to experience issues with a monolithic application they had built when the company was started. They also experienced other issues later with synchronous REST API calls between services, after they had split the monolith into microservices.

Namely, developers and data engineers couldn't resolve issues fast enough or iterate on their search functionality with sufficient agility, and were taking on a significant amount of technical debt as an increasing amount of data threatened to slow down the productivity and time to market for new features.

Solution:

To manage these challenges, the Storyblocks engineering team needed a new solution that could form the backbone of an entirely new data pipeline.Storyblocks began to decouple their microservices and setup an event-driven microservice for billing. Simultaneously, they began trying to spread the use of Kafka as an event bus for more streaming applications and machine learning (ML) features, and they realized Confluent Cloud very quickly made a difference.

With this Confluent-backed data pipeline in place, the Storyblocks team could begin to affect a true digital transformation both internally and externally. Instead of implementing a queue for inter-services communication, the team just puts it on the pipeline where it's stored forever. Events can be replayed on demand with powerful in-built schema validation. For the future, DeVeas envisions broader use of an events-based architecture at Storyblocks to improve the user experience and to tackle the lucrative enterprise market.

"One of the most noteworthy benefits of using Confluent for us has been for our rebranding project, which was the end result of our microservices project to consolidate all of our services into one Storyblocks.com domain where users could search videos and audio in the same place.. Confluent was a big part of this."

Chas DeVeas

Former Director of Engineering, Storyblocks

Read the full case study here

5

Judo Bank replaces point-to-point integrations to create a new CRM system and loan originator capabilities on AWS

judobank

Challenge:

Founded in 2016, Judo is Australia's first challenger bank for small-to-medium enterprises (SMEs). Judo Bank's journey is rapidly evolving as it works on its internal systems to make it easier to observe and manage an increasing amount of data as new customers sign up. As a key part of its evolution, the company first created a "service fabric" made up of point-to-point integrations that later became increasingly hard to deal with in light of the company's plans to build and launch a new customer relationship management (CRM) and loan originator system.

Solution:

With the help of Confluent Cloud, Judo Bank were able to seamlessly integrate various systems with their data platform to enable use of the new CRM., create the loan originator capabilities, and replace its core banking system with an agile foundation that will serve the company as it digitally transforms and continues to disrupt the banking world.

Judo now operates Confluent Cloud as a private cluster with managed AWS connectors. It has connections between its CRM, loan origination system, and core banking system to bidirectionally exchange data between customers' loan account creations and loan balances. "Our finance and FinOps teams have faster access to data from creation. Confluent has improved our data availability," Piekfe explains . They now have a data platform that uses event streaming for near-real-time data processing and are looking into using Confluent to enable integrations with external partners, either via APIs or sharing a public cluster directly.

"A great benefit of Confluent as an event streaming system is of course that it provides a complete publish-subscribe environment, the ability to replay, and the ability to create a data mesh and have applications tap straight into materialized views. So, it's really a great support for microservices. In the long run, we want to make Confluent the single source of truth for all of our services."

Andreas Piefke

Head of Cloud Service Architecture, Judo Bank

Read the full case study here

6

Recursion accelerates drug discovery on GCP and simplifies migration by reusing existing microservices logic

Recursion

Challenge:

The core data pipelines at Recursion start with fluorescence microscopy images captured during experiments with cells and reagents. As Recursion scaled their high-throughput screening lab, the volume of experimental data increased significantly and bottlenecks in the batch system became apparent. Processing the data from a single experiment—potentially more than 8 terabytes—did not begin until all images were available. This introduced delays and made it impossible for the laboratory to obtain real-time quality control metrics on the images. The company evaluated several open-source options, including Apache Storm and Apache Spark, but had concerns about operational complexity and migrating existing microservice logic.

Solution:

Recursion decided to use event streaming with Confluent Cloud to minimize administrative overhead, enable faster iterations on experimental results and simplify migration by reusing existing microservices.

Recursion has already made significant strides in accelerating drug discovery, with more than 30 disease models in discovery, another nine in preclinical development, and two in clinical trials. With the old batch system used for processing experiments, extracting features would take one hour for small experiments. With Confluent Cloud and the new streaming approach, the company has built a platform that makes it possible to screen much larger experiments with thousands of compounds against hundreds of disease models in minutes, and less expensively than alternative discovery approaches. "The scale and robustness of the system we built with Confluent Cloud have played a key role in accelerating our success in our mission of discovering new treatments and has helped us bring new treatments to human clinical trials", says Mabey.

"By building on top of Confluent Cloud, we created a flexible, highly available and robust pipeline that provided a clear migration path leveraging our existing microservices."

Ben Mabey

VP of Engineering, Recursion

Read the full case study here

Confluent: A modern application architecture to innovate faster in the digital age

Confluent's data streaming platform, built on top of Apache Kafka, completely decouples your microservices with asynchronous communication , and eliminates inter-service bottlenecks and dependencies so your developers and IT teams can build state-of-the-art cloud-native applications enabled by next-gen architecture.

Whether you're breaking your existing monoliths into modular components or working on a new microservices project with no legacy code or architecture, Confluent provides scalability and performance predictability, the ability to quickly add new services without additional overhead, centralized security control, and enriched data feeds with the ability to process data in flight and in real time.

If you're a software architect or developer, check out these assets on stream processing that delve into Confluent's approach to event-driven microservices:

- Stream Processing Simplified: An Inside Look at Flink for Kafka Users
- Your Guide to the Apache Flink® Table API
- Real-Time Insurance Claims Processing With Confluent
- Designing Event-Driven Microservices
- Apache Flink® 101

CONFLUENT

ABOUT CONFLUENT

Confluent is pioneering a fundamentally new category of data streaming infrastructure. Confluent's cloud-native offering is the foundational platform for data in motion—designed to be the intelligent connective tissue enabling real-time data from multiple sources to constantly stream across the organization. With Confluent, organizations can meet the new business imperative of delivering rich, digital front-end customer experiences and transitioning to real-time, software-driven backend operations.

To learn more, visit www.confluent.io