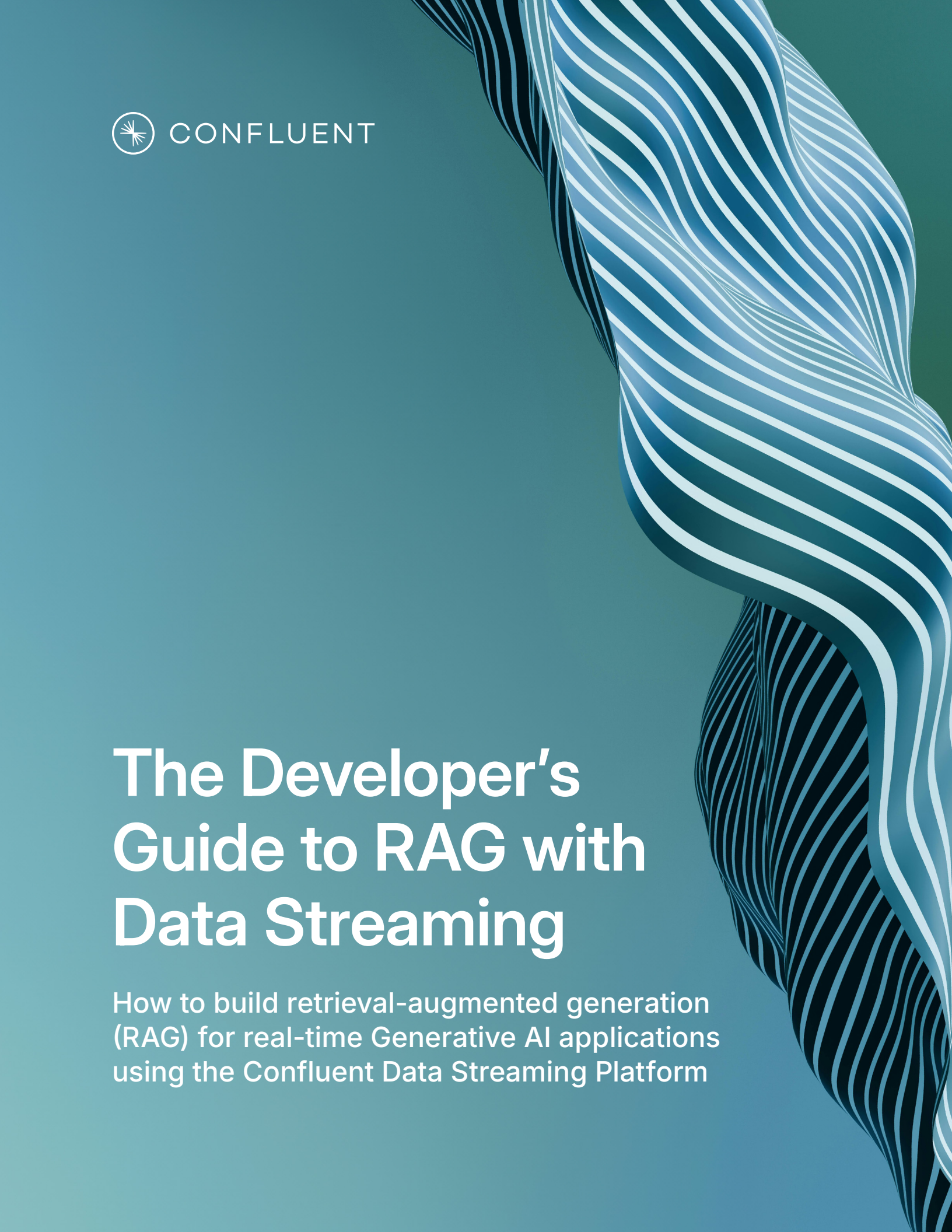
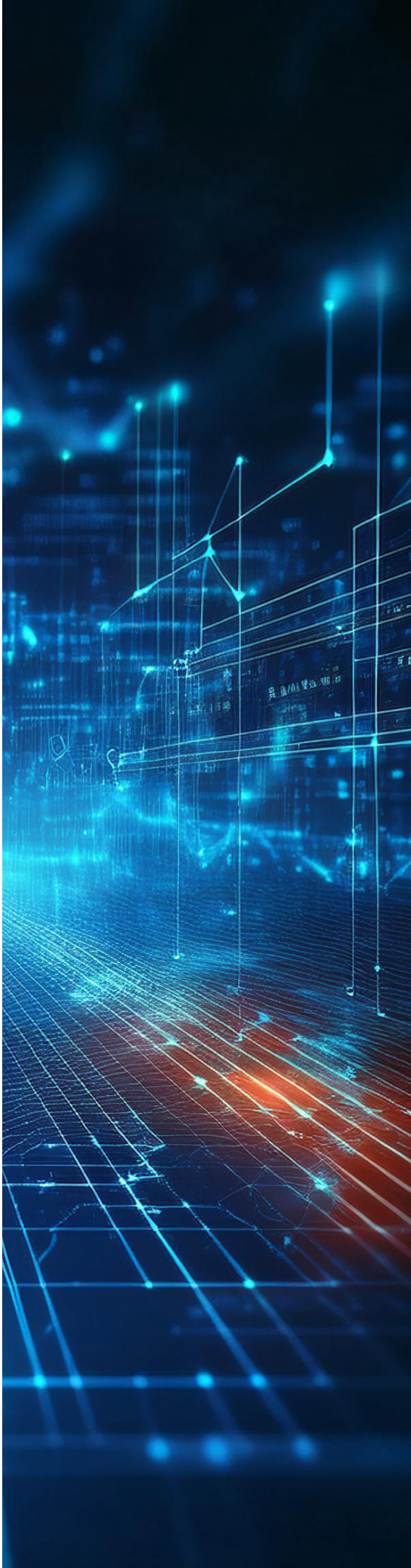


The Developer's Guide to RAG with Data Streaming

How to build retrieval-augmented generation (RAG) for real-time Generative AI applications using the Confluent Data Streaming Platform

Contents



1 Introduction to RAG

The promise of GenAI is only achievable when large language models (LLMs) have fresh, contextualized, and trustworthy data to accurately respond just in time.

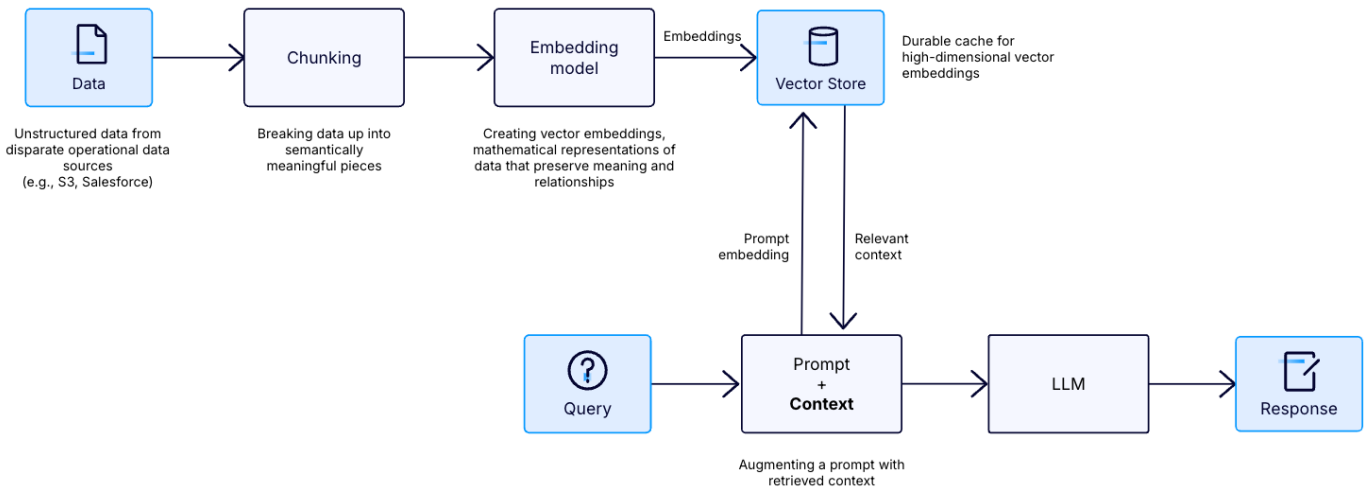
Let's consider an airline chatbot that we'll call Conflyent. Conflyent assists passengers with lost luggage. This requires augmenting publicly available data from LLMs with domain-specific data from the airline and continuously parsing and inferencing the information with context at prompt time:

RAG has emerged as a common pattern for GenAI to extend the power of LLMs to domain-specific data sets without having to retrain models. In the above chatbot example, to generate accurate responses for the passenger, RAG allows data to be retrieved from Conflyent Airlines' luggage and claims policy, customer service logs, and FAQs. Mutable structured and unstructured data is necessary real-time context for non-trivial enterprise use cases for LLM-enabled applications. This applies to countless GenAI use cases—from a product recommendation engine to a medical insights miner to AI-powered compliance agents.

LLMs are a great foundational tool for building GenAI applications and have democratized access to AI. However, they are stochastic by nature and generally trained on a large corpus of static data, without visibility into knowledge fidelity or provenance. As a result, when there is a knowledge gap, LLMs may hallucinate, generating false or misleading answers that appear convincing. As you build GenAI use cases powered by LLMs, the challenge lies in contextualizing prompts with real-time domain-specific data, necessitating patterns like RAG.

RAG is often characterized by semantic search against a vector database to find the most relevant domain-specific data for prompt contextualization. It's important to distinguish RAG from other data retrieval methods such as traditional database queries or getting data from cache. In a RAG system, the user's query is converted into a vector embedding. This embedding is then used to search the vector database for semantically similar content, instead of exact keyword matches. Relevant information is retrieved and provided to the LLM as context. The LLM generates a response based on both its training and the retrieved context.

Here's a look at a common embodiment of how RAG works:



When paired with data streaming, RAG leverages the freshest external data to bridge the gap between the vast knowledge of LLMs and the specific up-to-date information needed to improve the accuracy and relevance of generated responses. Alternate approaches to contextualizing an LLM with domain-specific data at inference time include fine-tuning or bespoke model training. However, both are inaccessible to many businesses without significant expertise and investment. In particular, fine-tuning is challenging to get right, as it can inadvertently alter the LLM to forget previously learned information or become less proficient.

RAG and data streaming unlock access to real-time data and domain-specific proprietary context while reducing hallucinations and LLM calls and token costs.

However, to successfully implement RAG, an enterprise's underlying data challenges must be addressed.

These include:

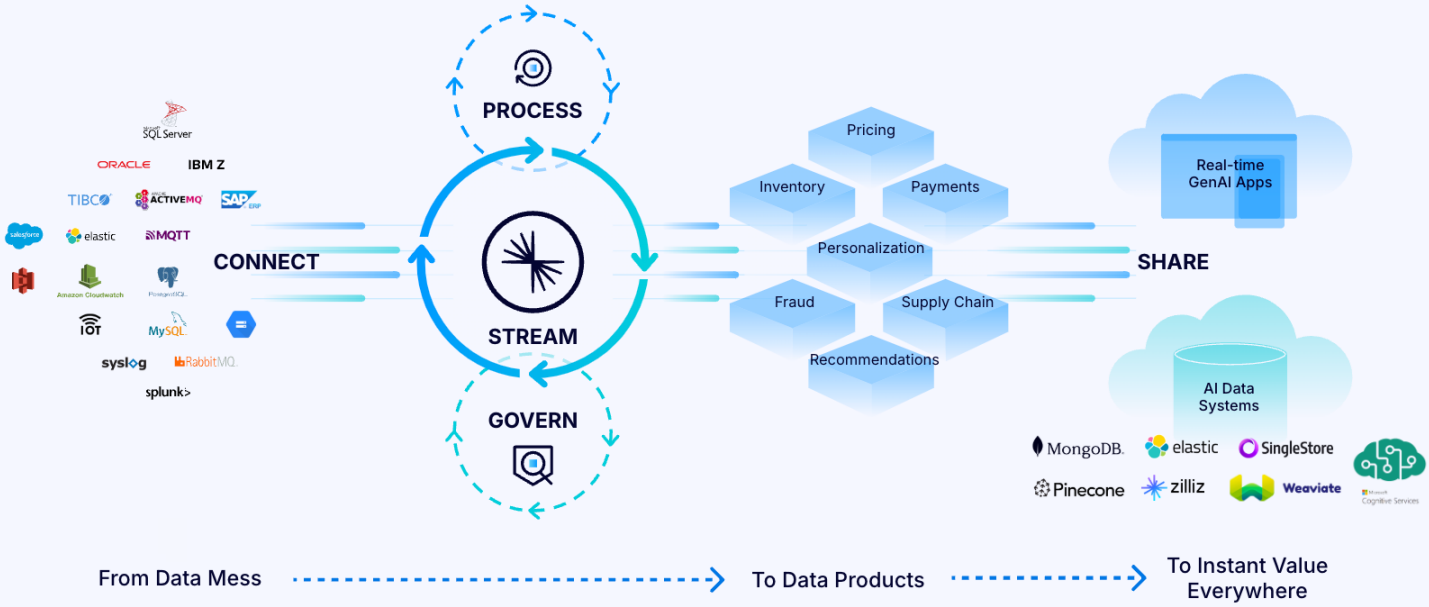
- Lack of real-time context from batch-based training and ETL pipelines.
- Siloed, heterogeneous data stores.
- Low data quality and data inconsistencies.
- Lack of data trust and fidelity from weak governance.
- Tightly coupled data architecture.

This is where the Confluent Data Streaming Platform comes in.

Watch this lightboard video for a visual explanation of RAG →

2 Confluent Data Streaming Platform and the Four Steps for Building RAG

The Confluent Data Streaming Platform supports the implementation of RAG by delivering a real-time, contextualized, and trustworthy knowledge base that can be coupled with prompts at inference time to generate better LLM responses.



Stream

Continuously capture and share real-time events with AI systems and applications.

Connect

Integrate disparate data from any environment with 120+ pre-built and custom connectors.

Process

Use stream processing frameworks like Apache Flink® to enrich data with real-time context at query execution.

Govern

Use data lineage, quality controls, and traceability to ensure data for AI is secure and verifiable.

By providing real-time data in key places—data augmentation, prompt enhancement, and response validation—data streaming eliminates unnecessary load on source databases during user queries. Confluent helps GenAI application teams accelerate time to market while providing the guardrails for data security, governance, and compliance.

The Confluent Data Streaming Platform enables development teams to:

Prevent hallucinations
Use post-processing to validate that LLM outputs are correct by having a consumer group or Flink SQL check the generated response against policy and customer data in Confluent.

Improve LLMs with real-time, domain-specific context
Integrate disparate data into a single source of truth for real-time contextualization and securely share relevant data. Additionally, safeguard data such as proprietary information or PII through Stream Governance (e.g., schema rules). Capture real-time events, enrich them, and convert them to vector embeddings for immediate use by your AI tools, models, and applications.

Lower the barrier to entry
Instead of building custom integrations or investing in more infrastructure to support RAG, leverage a fully managed Data Streaming Platform (with fully managed connectors, Flink as a cloud-native service, Stream Governance) in order to focus on building GenAI applications rather than managing data infrastructure.

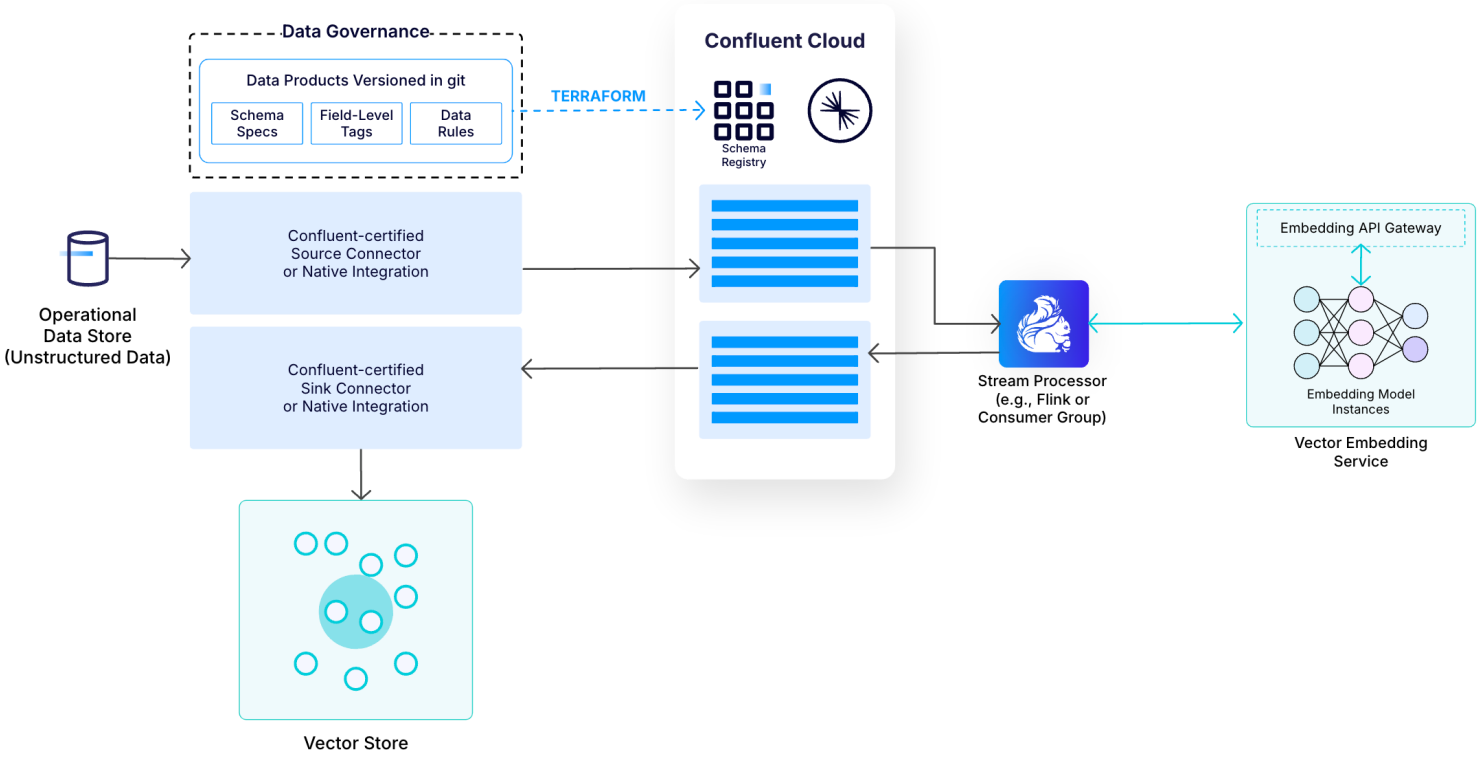
Scale independently for greater agility
Decouple teams, systems, and technologies to work more autonomously and efficiently. For example, different teams can work on different aspects of RAG applications. Keep what you have and use the LLM (e.g., GPT, Gemini, LlaMA), vector store (e.g., Pinecone, Weaviate, Milvus), and embedding model you prefer. Modular components can be substituted as technology improves, with no vendor lock-in. Easily discover, access, and reuse trusted data products.

There are **four key steps for building a RAG architecture**, and in the following sections, we'll cover how you can use Data Streaming Platform features for each step:

1. **Data Augmentation** – Preparing data for a real-time knowledge base and contextualization in LLM queries by populating a vector database.
2. **Inference** – Connecting relevant information with each prompt, contextualizing what users are asking for and ensuring GenAI applications are built to handle those responses.
3. **Workflows** – Parsing natural language, synthesizing the necessary information, and using a reasoning agent to determine what to do next to optimize performance (e.g., querying a database, getting information from cache, calling an LLM with a different prompt).
4. **Post-Processing** – Validating LLM outputs and enforcing business logic and compliance requirements to detect hallucinations and ensure the LLM has returned a trustworthy answer.

3 RAG: Data Augmentation

Let's first address data augmentation and how you can build a real-time, contextualized knowledge base for your GenAI applications.



Confluent simplifies integration of disparate data across your architecture with a large ecosystem of 120+ pre-built, zero-code, source and sink connectors (with 80+ fully managed) as well as custom connectors. These replace batch ETL processing, enabling you to ingest and integrate the latest version of your proprietary data—be it about your customers or business operations—and make it instantly accessible to power your GenAI application.

Because data streams usually contain raw information, you'll likely need to process that data into a more refined view. Flink stream processing helps you transform, filter, and aggregate individual streams into data products and views more suitable for different access patterns. For example, join a customer profile stream and flight bookings stream to create a new data product: C360 view of airline customers for loyalty rewards. Other teams can also leverage the processing that's already been done, thereby reusing data products and reducing redundant processing. Data Portal makes it easy to discover, access, and collaborate on data products across your organization.

You can then create consumer groups that take the ingested unstructured data, perform the chunking and create embeddings—mathematical representations of information that preserve meaning and relationships—using a vector embedding service and pass them on to a vector store. Confluent provides native integrations with leading vector stores such as MongoDB, Pinecone, Elasticsearch, SingleStore, Weaviate, Zilliz, Couchbase, Neo4j, Qdrant, Azure Cognitive Search, and others.

While this RAG pattern is straightforward for unstructured data, there are emerging best practices for using structured data with vector stores. To build an efficient streaming RAG pipeline, a key consideration is determining when to vectorize and run semantic search over semi-structured or structured data. Generally, data without inherent meaning absent its schema (e.g., social security numbers, credit card numbers) should not be vectorized.

When integrating structured and unstructured data, it's important not to blindly feed all data into a vector database. Sometimes, one must parse out the meaningful elements before vectorizing. For example, you can manipulate structured data to make it as close to natural language as possible so that the LLM understands what numbers and other fields mean (e.g., categorizing items with a new "price-category" field as "cheap" or "expensive" based on price thresholds can support semantification more effectively in some applications).

When to vectorize

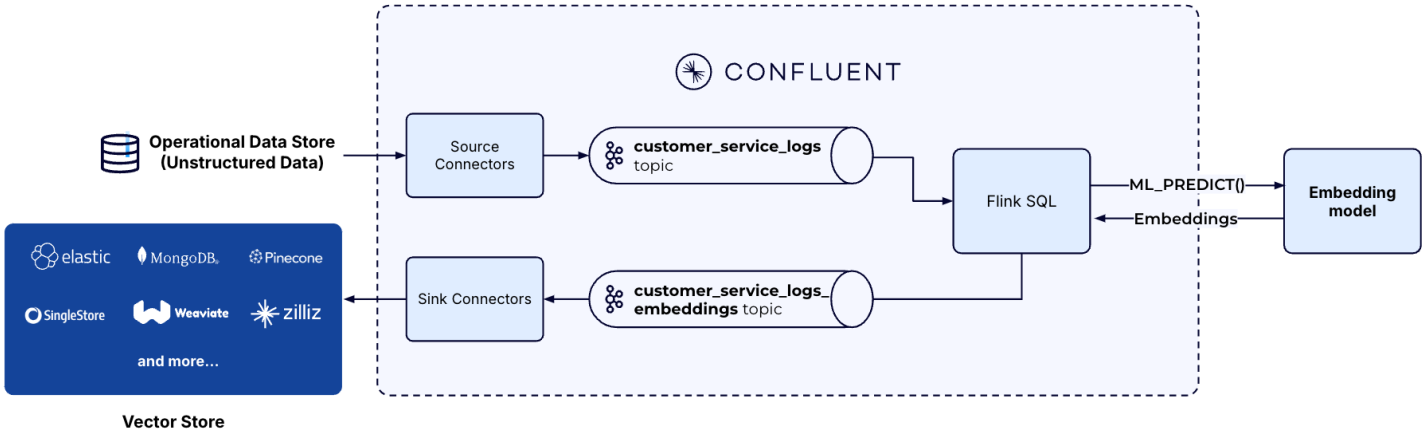
- When files are unstructured.
- When files have structured and unstructured data (e.g., research papers containing data tables).
- When structured data has unstructured values (e.g., CSV containing customer reviews).

When not to directly vectorize

- When data is solely identifiers (e.g., CSV of product IDs).
- When interested in aggregated analytics (e.g., percentage of web visitors who clicked on a banner), which has little semantic value and is better queried from relational databases.

Flink AI Model Inference allows you to integrate remote AI models into your RAG data pipeline by utilizing AI models (e.g., OpenAI, AWS Bedrock, AWS Sagemaker, Azure OpenAI, Azure ML, Google AI, Vertex AI) directly within your Flink SQL queries. In this way, Flink unifies stream processing and RAG workflows by continuously enriching data with context while enabling integration between AI workloads deployed on any cloud.

Here's an example illustrating how to create vector embeddings from customer service interactions for the airline chatbot use case, using the `ML_PREDICT` function to run the AI model in a Flink SQL query:



Here, we first create a connection via Confluent CLI that defines the endpoint and API key outside of Flink SQL, specifying the region, environment, type, endpoint, and secret key. This allows us to manage connections outside the Flink SQL statements:

```
confluent flink connection create openai-vector-connection \
--cloud aws \
--region us-west-2 \
--environment my-env-id \
--type openai \
--endpoint 'https://api.openai.com/v1/embeddings' \
--api-key 'secret_key'
```

Then, we create the model using that connection:

```
CREATE MODEL `openai_embeddings`
INPUT (input STRING)
OUTPUT (vector ARRAY<FLOAT>)
WITH(
  'TASK' = 'embedding',
  'PROVIDER' = 'OPENAI',
  'OPENAI.CONNECTION' = 'openai-vector-connection'
);

INSERT INTO customer_service_logs_embeddings
SELECT * from customer_service_logs
LATERAL TABLE (ML_PREDICT(openai_embeddings,input));
```

To see how this works, here is a RAG tutorial and corresponding GitHub repo showing how to perform real-time data augmentation—from ingesting data with connectors and vector encoding with Flink AI Model Inference, to sinking to a MongoDB Atlas vector store. →

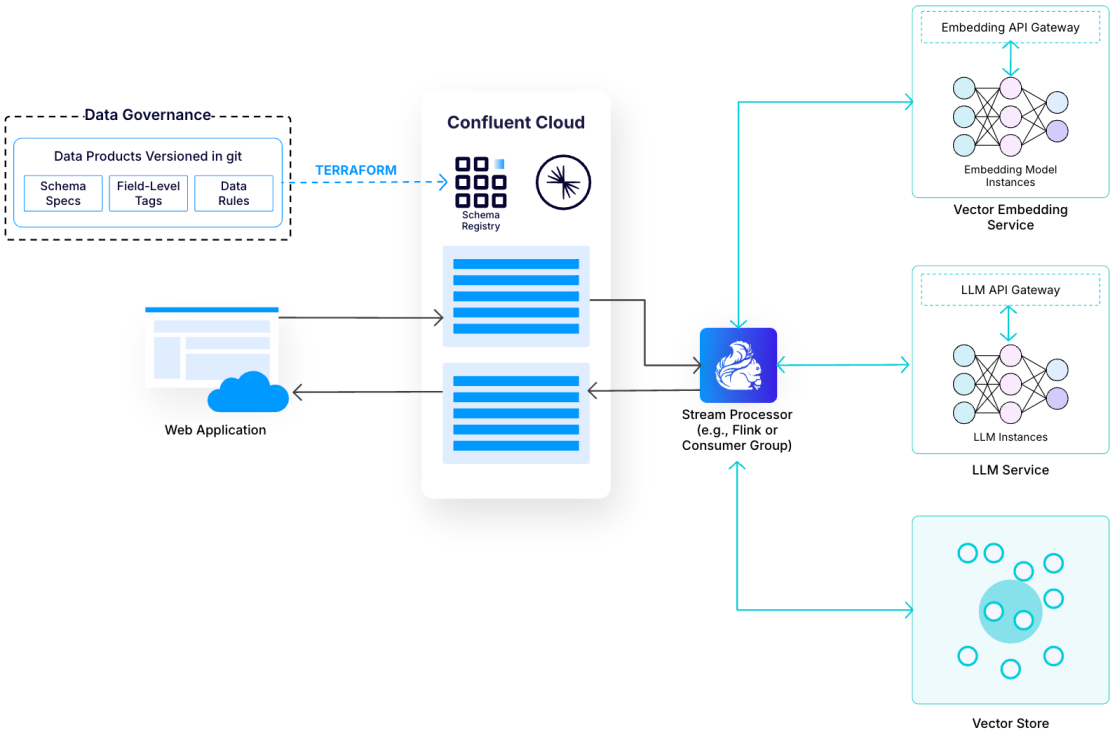
As more data connects to your GenAI applications, Stream Governance helps you establish data lineage, data quality with Schema Registry, and traceability, providing all development teams with a clear understanding of data origin, movement, transformations, and usage. You can track, tag, encrypt, classify, version, and quality control data on the move with built-in governance. Together with role-based access controls and audit logs for observability, you can secure and trust the data used for your applications.

Only clean, trustworthy, and up-to-date data enables AI chatbots to assist customers with the answers they need. Confluent enables you to process and govern data on the fly to create high-quality data products that can be reused for additional use cases such as assessing RAG performance (e.g., accuracy score based on hallucinations, load latency, recall, queries per second) and chatbot insights (e.g., customer sentiment, satisfaction ratings, session time, and number of replies to resolve an issue).

All of this is built on an event-driven architecture. What that means for RAG is that when all of your unstructured corporate knowledge is encoded in a vector store, it can be semantically related to future prompts in an accessible pattern. According to numerous studies, most enterprise data (90% per IDC) is unstructured, so this approach addresses a significant challenge. As new prompts come in, RAG retrieves the most relevant corporate knowledge for that prompt and supplies it to the LLM for real-time inference.

4 RAG: Inference

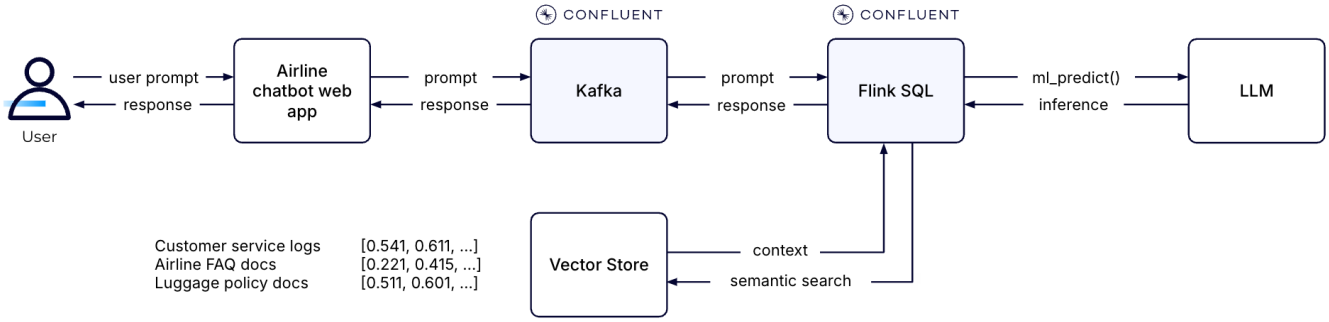
A Data Streaming Platform helps keep your vector store continuously updated with fresh proprietary information from the business. When a prompt from a user comes in from the GenAI web application, you can use Flink in Confluent to enrich and contextualize that prompt in real time with private data from your other systems, along with data retrieved from the vector store. You can then stream this information to an LLM service (e.g., OpenAPI) and pass the LLM-generated response back to the web application.



Confluent enables real-time inference by decoupling your architecture, standardizing your schemas for downstream compatibility, and enabling in-flight stream processing, combining any stream from anywhere for an enriched view of your data. If something happens in one part of your AI value stream, it can immediately trigger something to happen in another part with intelligence.

To see this in action, here is a GitHub repo showing how real-time inference enables a chatbot to understand context and provide more meaningful and accurate user interactions. →

Here's a closer look at real-time inference orchestration in Confluent Cloud:



This pattern also decouples teams for greater agility, allowing the web application team to work independently from the vector embedding team, the team building the consumer group or business logic, and so on. At the same time, Stream Governance ensures that all developers adhere to standardized schemas and data contracts with schema rules to guarantee data quality, consistency, and compatibility when sharing information across different systems or organizations. For example, a rule can set that social security numbers need to be 9 digits. By using decomposed, specialized services instead of monolithic architectures, applications can be deployed and scaled independently. This improves time-to-market, as new inference steps are simply added as consumer groups.

Flink AI Model Inference can encode unstructured data such as airline baggage policies or passenger reviews for storing in a vector database. Flink user-defined functions (UDFs) take this further with custom logic, such as applying rules to fields (e.g., hiding customer credit card numbers) or vectorizing passenger reviews.

Flink Actions are pre-packaged, turn-key stream processing workloads that handle common use cases such as deduplication or masking. Actions are easy and quick to use, allowing you to leverage the power of Flink in just a few clicks. For example, clicking on a lost baggage claims topic, then clicking to apply the deduplication Action, will generate a new topic containing only unique claims. Processing data with Flink helps reduce noise and enhance precision, ensuring the most relevant, accurate information is retrieved at inference.

5 RAG: Workflows

Consider the following query for the airline chatbot: "Can I use my frequent flyer miles or credit card points to upgrade my flight and add a second checked bag?"

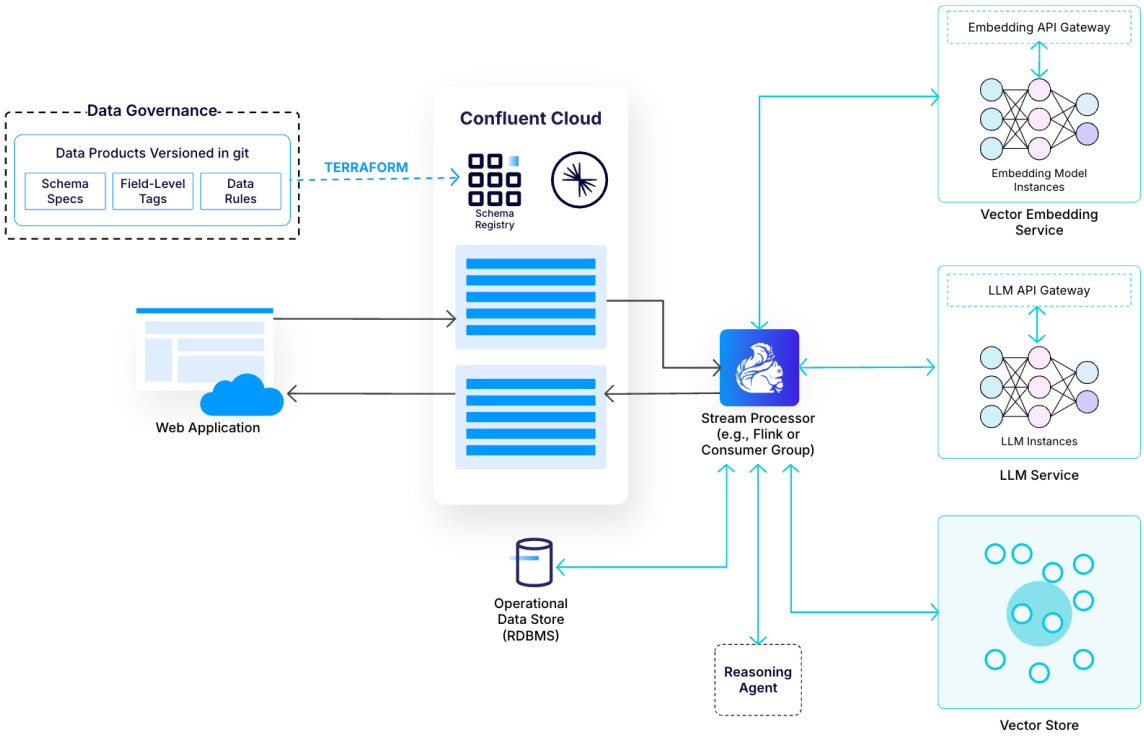
LLMs usually return better results when asked multiple simple questions rather than a longer compound question. For this reason, workflows involve breaking down a single natural language query into composable, multipart logical SQL queries to ensure that the right information is provided to the customer in real time. This is often done by using reasoning agents.

Reasoning agents in GenAI use Chain of Thought (CoT) or Tree of Thoughts (ToT) to break down complex requests into a series of steps, interacting with external tools and resources. The agent looks at tools available and determines what to do next (e.g., vector search, call an LLM several times to refine an answer, web search, query a database, access APIs). Reasoning agents can be implemented using various frameworks such as LangChain and Vertex AI or AutoGen, CrewAI, and LangGraph for multiagent systems. You can also create custom reasoning agents by directly interfacing with LLMs and implementing your own logic.

The passenger's query can be broken down into multiple steps by an agent:

1. **Flight upgrade –** RAG query against flight schedules to determine which flights have upgrade availability and how much it would cost in frequent flyer miles (e.g., Flight A has first-class seats for $1500 or 20K frequent flyer miles).
2. **Checked bag –** Check Conflyent Airline's baggage policy (unstructured data in a document or webpage) for the ticket class.
3. **Frequent flyer miles and points –** Call a microservice for the exchange rate between frequent flyer miles and credit card points. Query the frequent flyer miles system to see if the passenger has enough miles for an upgrade and query another database for their available credit card points.
4. **Finally –** Prompt the LLM with the above information to show recommended flights for the passenger.

Each of these actions requires a separate call to different systems and APIs, processed by the LLM to give a coherent response. In such a scenario, your workflow may be a chain of LLM calls, with reasoning agents making intermediate decisions on what action to take. For example, a reasoning agent can call an LLM to act as a natural language interface to an operational database (e.g., by passing a schema and using LangChain's SQLBuilderChain, you can interrogate a SQL database with natural language). Confluent acts as the real-time data highway supplying all of this contextualized, consistent data across all your AI systems.



One important optimization is checking if a question has been asked previously and pulling the response from a cache. Streaming and storing responses in a cache like Redis, or running a Flink query against a compacted Kafka topic, eliminate the need to call the LLM for every query, saving both time and cost. The decision-making process involves determining whether to call the LLM or retrieve the answer from the cache.

Throughout this process, Stream Governance in Confluent ensures security and privacy to prevent data leakage. This includes filtering out PII and PCI data, applying metadata or field-level tags, and enforcing data rules as information is processed. Monitoring, audit logs, and data lineage are all crucial for maintaining compliance.

Workflows can be written in Java and Python using Flink. Often, the way this is accomplished is through the Flink Table API. Confluent's fully managed Flink offering is an alternative to writing a one-off custom microservice—which would require hosting and need to be highly available—or a client app that reads from a topic. Flink Table API in Confluent allows for filling in gaps and making intelligent decisions about where to retrieve information. For example, if certain information is missing—such as a customer's frequent flyer number—it can be retrieved from a Flink table.

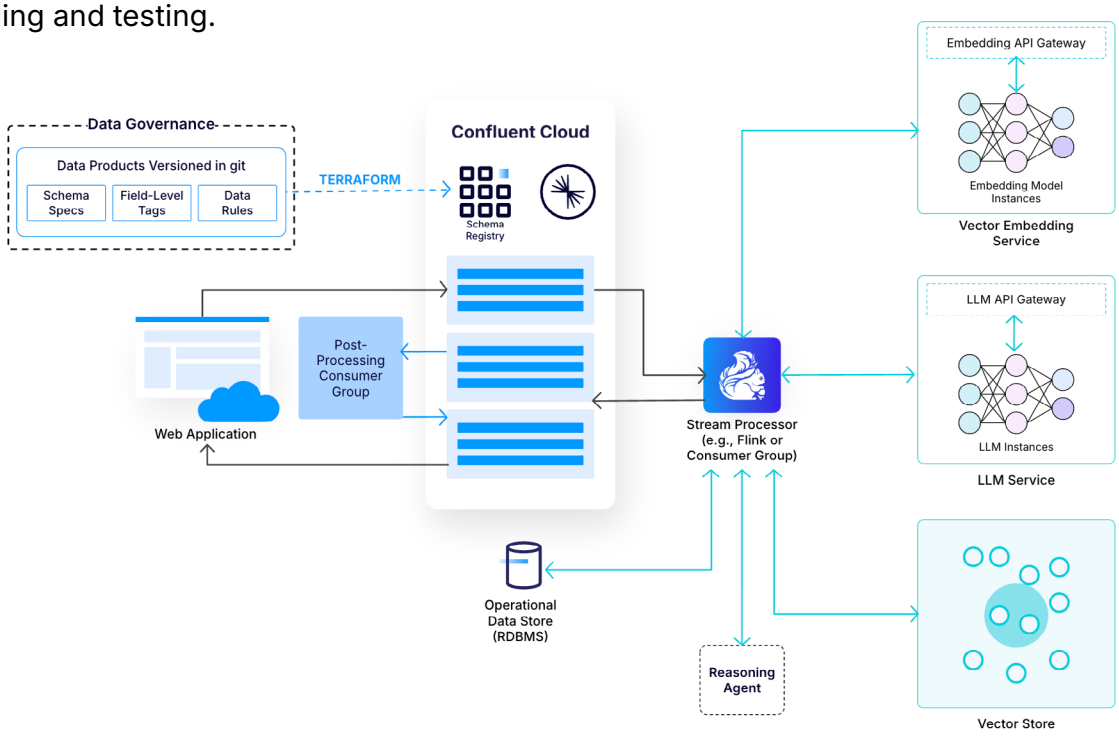
Decision-making revolves around balancing speed and accuracy. Development teams would determine whether the system should respond quickly or take additional seconds for greater accuracy. This balance can be coded and adjusted based on the specific GenAI use case.

1. See how this works in the Workflows repo →
2. Learn more about using Flink Table API in Confluent by visiting the Dev course and docs page →

6 RAG: Post-Processing

Due to hallucinations, businesses need to validate LLM outputs and enforce business logic and compliance requirements to ensure that generated answers are trustworthy. In the post-processing step, teams can use frameworks like BPML or Morphir to perform sanity checks and other safeguards to prevent hallucinations from moving downstream.

Confluent's event-driven architecture decouples the post-processing from the GenAI workflow and rest of the application, so that these facets can be developed by different teams and evolve independently. Post-processing is often developed by a more compliance-focused team, which generally has a different skill set than the team building the GenAI inference chains. By breaking up the monolith, these teams don't need to perform out-of-band coordination for deployment and are functionally independent for building and testing.



For the airline chatbot, a separate Flink job can perform airline baggage price validation or a refund policy check, for example. You can use Flink SQL or Table API for your applications.

The post-processing workflow can have multiple steps:

1. A post-processing consumer group can assess compliance with business rules to identify any discrepancies.
2. Another post-processing group can perform deeper logic checks to verify the accuracy of identifiers, such as ticket numbers or baggage tag numbers, eliminating any that are incorrect.
3. Additional post-processing steps can also be incorporated to enhance the overall system.

Technologies like Morphir can integrate with Confluent to allow for processing and managing business logic in real time. Consider fraud detection in the case of filing baggage claims. The airline would need to process a high volume of claims in real time and apply complex business rules to detect potential fraud.

Defined in Morphir, fraud detection rules might include conditions around unusual claim frequency, unsubstantiated high-value claims, delayed reporting, or patterns in claim locations.

Confluent streams all claim activities and Flink applies the fraud detection rules defined in Morphir to each claim in real time. On the right is an example of a Flink SQL query executing the business logic defined in Morphir. If a claim is flagged as potentially fraudulent, it gets published to another topic that is consumed by downstream services, such as alerting systems, dashboards, or AI-powered systems that block suspicious claims in real time.

```
SELECT
  claim_id,
  account_id,
  amount,
  location,
  claim_time
FROM
  transactions
WHERE
  amount >
morphir_defined_threshold
  OR location IN
(morphir_defined_blacklist)
  OR claim_time NOT BETWEEN
morphir_defined_normal_hours_start
AND
morphir_defined_normal_hours_end;
```

A key consideration is the trade-off between latency and accuracy. For VIP customers, for example, achieving 100% accuracy may be crucial, while internal sales and other teams might prioritize lower latency. Most users would prefer a response that is mostly accurate and delivered within ten seconds from a chatbot.

Preferences can be set within the LLM by the user or developers, allowing for options such as a one-second response time with 90% accuracy or a ten-second response for 100% accuracy. This preference will determine which workflow is used by the reasoning agent (e.g., which data topic the process reads from). This can be based on user preferences or user status (e.g., Platinum frequent flier members). Utilizing the Window function in Flink helps measure accuracy. An event-driven architecture is essential for GenAI applications to ensure they remain both accurate and quick in their responses.

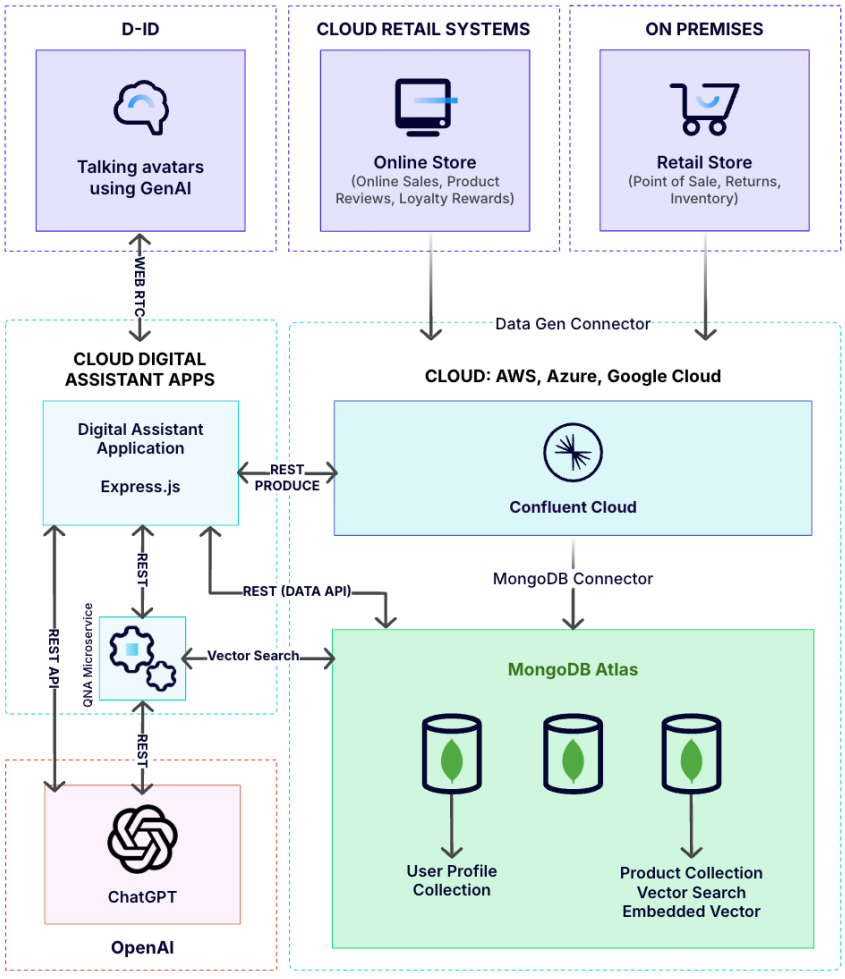
To see how it works, visit this Post-processing repo →

7 Use Cases with Reference Architectures

Here are examples of RAG use cases and their reference architectures:

Product Recommendation Engine

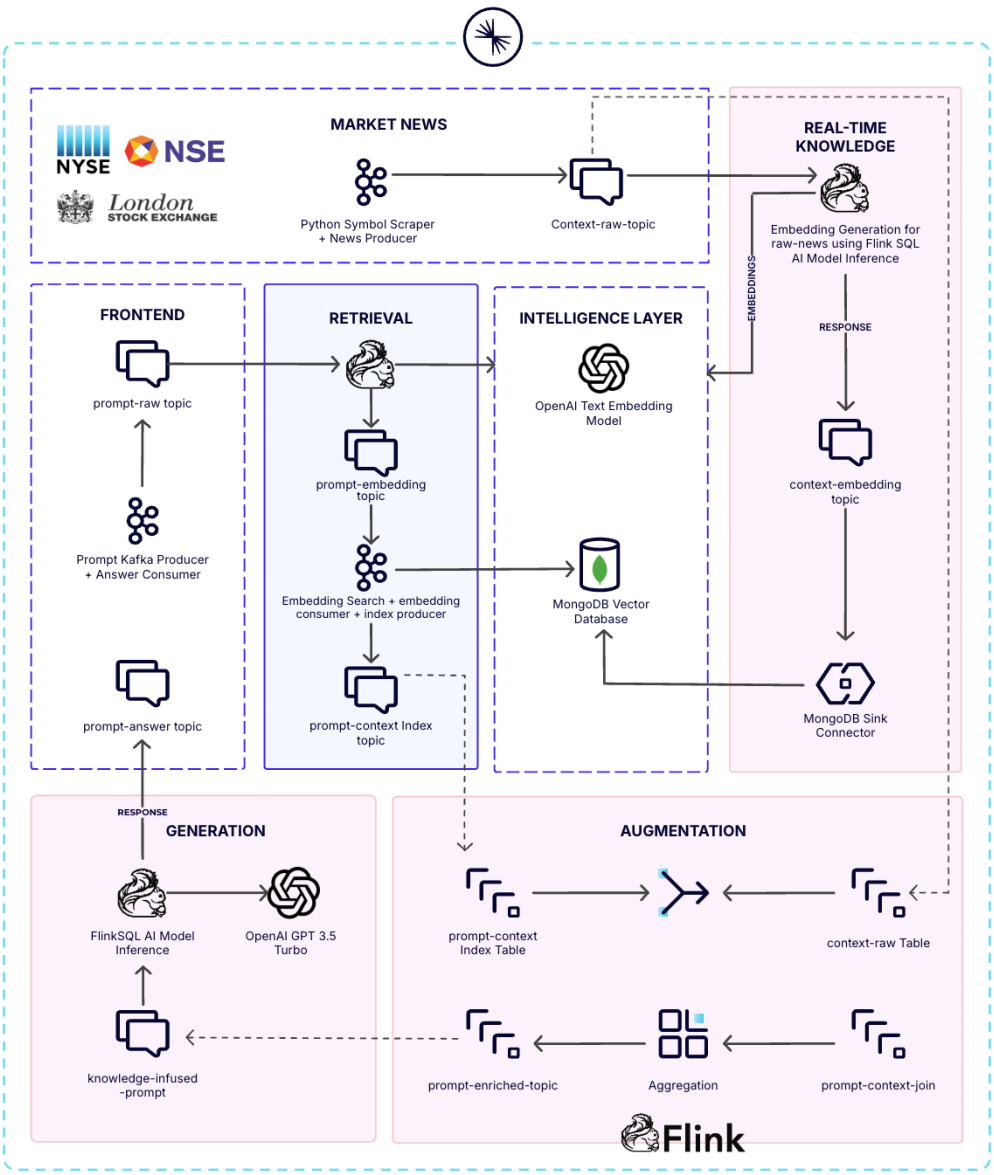
Retail data including product inventory, online sales, product reviews, and customer loyalty rewards are streamed, processed, converted to embeddings, and stored in MongoDB Atlas with vector search enabled. Leveraging relevant product information and user profile data, the GenAI application provides personalized product recommendations.



Visit the GitHub repo to try it yourself →

Real-time Sentiment Analysis for Stock Markets

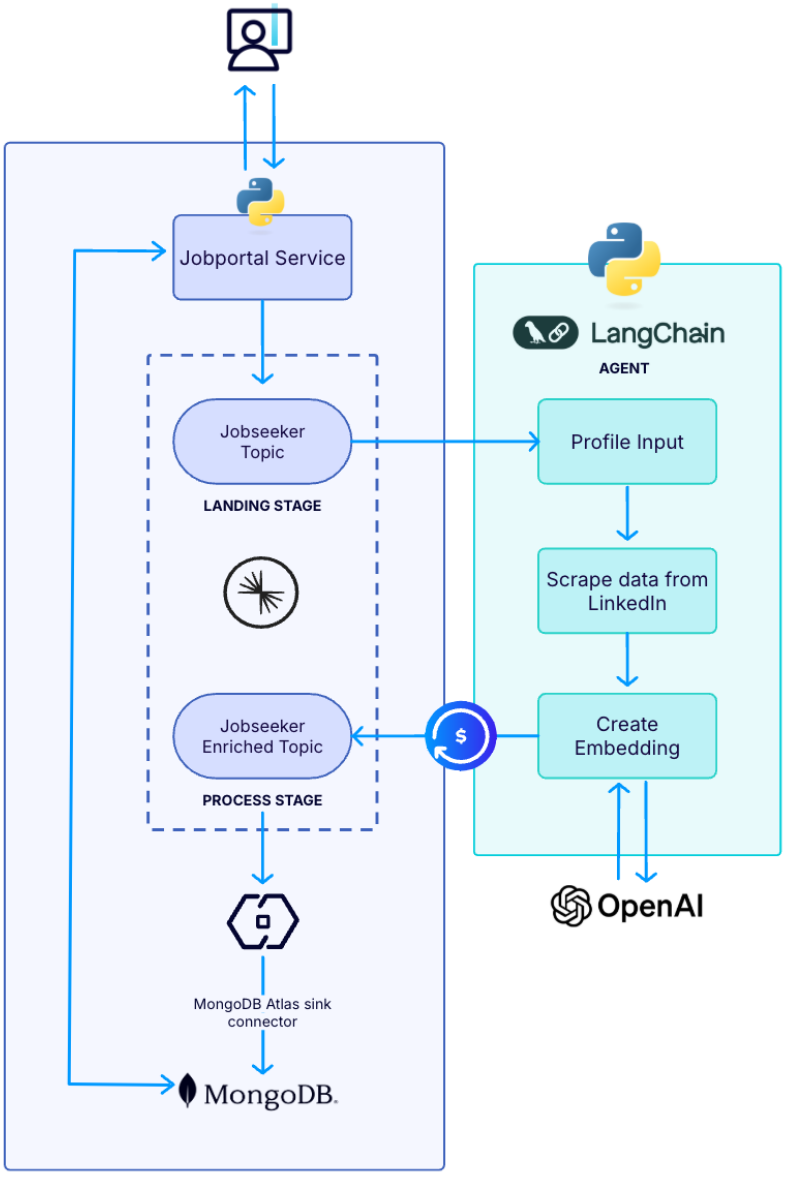
Streaming and aggregating live financial data and market news analysis, then enriching prompts with context retrieved from a vector database, allow for up-to-the-minute insights. Generated insights include how the market is reacting to a company's latest earnings report, what customers are saying about a specific product, whether there are emerging concerns, and how competitors or market trends are impacting the company's prospects.



Visit the GitHub repo to try it yourself →

Jobs Portal

A RAG-enabled jobs portal utilizes content extraction and content generation to match job seekers with employers. For jobseekers, key information is extracted from their resumes, such as a summary of experience, and GenAI recommends the latest jobs and personalizes career advice. For employers or recruiters, the system assists in generating job postings and automatically matching top candidates for listed positions.



Visit the GitHub repo to try it yourself →

8 Customer Stories

Notion

"To save our users time, write faster, and boost creativity, they need access to the latest documents at all times. We use Confluent to process and share new content and updates across all databases in real time, ensuring every system has a reliable view of the documents. Confluent lets our product and engineering teams use data products to build new RAG-based applications faster, without worrying about data infrastructure. This speeds up our GenAI use cases."

Daniel Sternberg
Head of Data and AI, Notion

Watch the video →

GEP

"We built a real-time GenAI chatbot to help enterprises identify risks and optimize their procurement and supply chain operations. Confluent allows our chatbot to retrieve the latest data to generate insights for time-sensitive situations. We use Confluent connectors for our data stores, stream processing for shaping data into various contexts, and Stream Governance to maintain trustworthy, compatible data streams so our application developers can build with real-time, reliable data faster."

Nithin Prasad
Engineering Manager, GEP Worldwide

Read the story →

Discover more use cases here →

9 Get started

To take the next step and start building your real-time, RAG-enabled GenAI applications:

1. Sign up for Confluent Cloud and receive $400 in free credits →
2. Visit the GenAI Hub for more resources and upcoming AI events →
3. Apply for Confluent's AI Accelerator Program, which provides technical and business mentorship to help AI startups fast-track innovation and growth →

About Confluent

Confluent is pioneering a fundamentally new category of data infrastructure focused on data in motion. 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 sophisticated, real-time, software-driven back-end operations.

To learn more, please visit: www.confluent.io