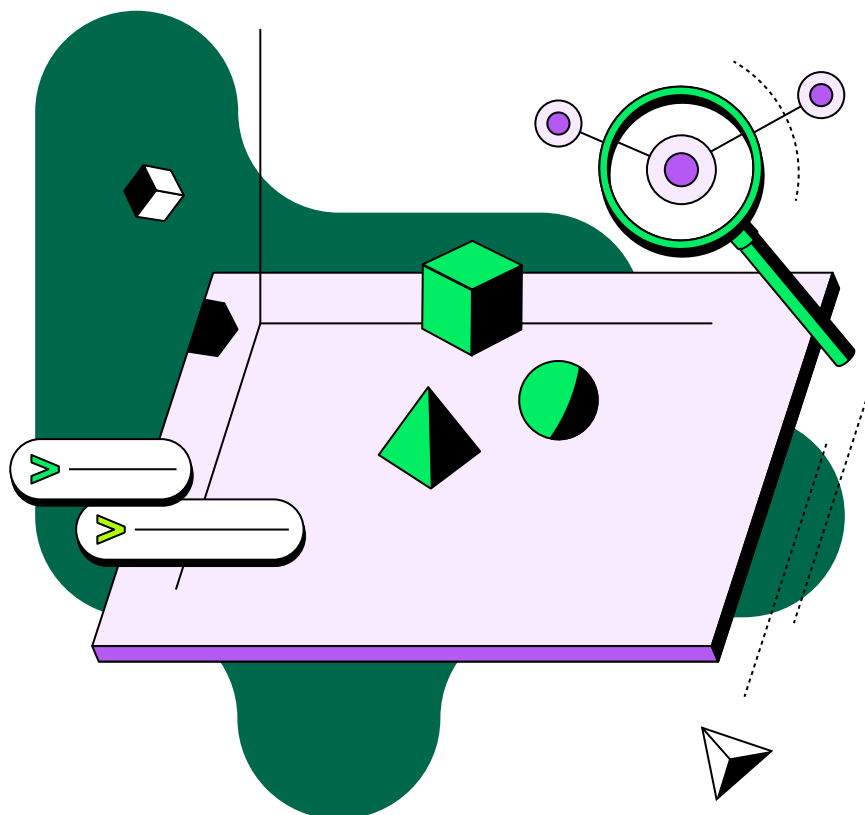




Building continuously updating RAG applications

Use native stream processing and vector search in MongoDB Atlas to continuously update, store, and search embeddings through a unified interface.

Start Free



- Use cases: [Gen AI](#)
- Industries: [Finance](#), [Healthcare](#), [Retail](#)
- Products: [Atlas](#), [Atlas Vector Search](#), [Atlas Stream Processing](#)

- Partners: [Confluent](#), [AWS](#)

Solution overview

Whether organizations leverage AI to optimize business processes or enhance customer-facing applications, providing AI models with up-to-date data is essential to delivering a differentiated experience. While retrieval-augmented generation ([RAG](#)) systems enable organizations to ground large language models ([LLMs](#)) easily and foundational models with the truth of their proprietary data, keeping that data fresh adds another level of complexity.

By continuously updating vector embeddings, the core of RAG systems, AI models have up-to-date data to provide pertinent and accurate answers. Additionally, different embedding models may offer higher levels of accuracy depending on their primary purpose. Take, for example, an embedding model trained primarily on a specific language, such as Japanese or Simplified Chinese, instead of a more popular model that might have general knowledge of several languages. The specialized model will likely create embeddings that enable the foundation model or LLM to output content more accurately.

This solution addresses the issue of continuously updating and routing the creation of vector embeddings in a RAG system. By leveraging [MongoDB Atlas Stream Processing](#) and [MongoDB Atlas Vector Search](#), both native capabilities in MongoDB Atlas, this solution walks developers through continuously updating, storing, and searching embeddings with a single interface.

While this solution demonstrates creating vector embeddings of song lyrics in different languages, the scenario is relevant to many industries and use cases, including:

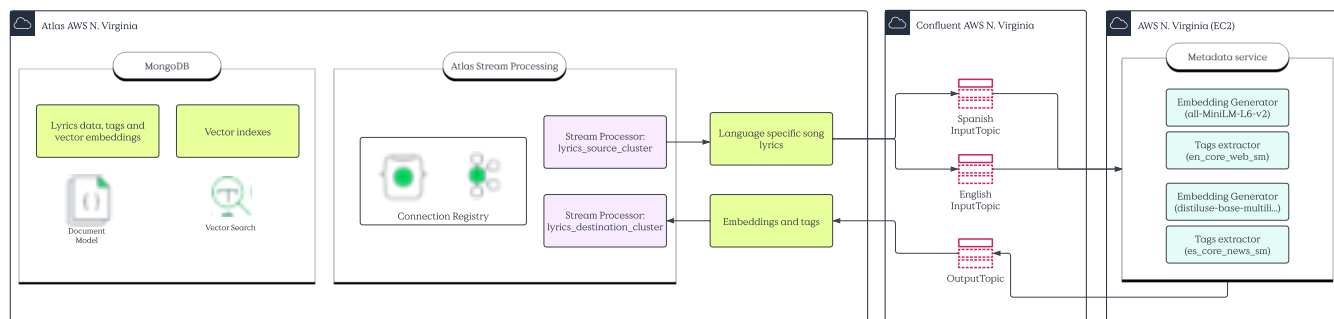
- **Financial services:** Financial documents, legal policies, and contracts often use multiple languages and differ based on country regulations. Empowering loan officers with an AI-powered interface for expediting loan creation can optimize banking workflows; however, the optimization will only benefit as much as the data is relevant and fresh.
- **Healthcare and Insurance:** From constantly updating patient records to AI-powered underwriting of insurance policies, it's important that any RAG system that optimizes these processes has access to the latest information.
- **Retail:** Personalizing retail experiences by delivering the right offer at the right time to the right customer is critical. However, consider the many languages that shoppers might use and product descriptions that have to match. Routing up-to-date, contextual data to the most accurate embedding model can improve these experiences.

Reference architectures

With MongoDB:

- **MongoDB:** With a MongoDB cluster deployed in Atlas, it allows you to store the lyrics and related information under the same document (tags, vectors, etc.). Additionally, Atlas provides a vector index to support semantic searches using the [MongoDB Aggregation Framework](#).
- **Atlas Stream Processing:** This Stream Processing Instance subscribes to the events generated by MongoDB, filters the relevant information, transforms the events, and emits them to the corresponding Kafka topic. Additionally, it will subscribe to the Kafka cluster to update the documents that change.

- **Confluent Kafka cluster:** This managed Kafka cluster will receive the new documents and updates from current documents to be processed. Additionally, the events received by the processor will be directed to Atlas Stream Processing.
- **Metadata service:**
 - **Embedding generator:** Python script that subscribes to the Kafka input topics (both Spanish and English). For each message received, it reads the lyrics and generates an embedding using a model specific for each language.
 - **Tags extractor:** Python script that extracts the tags from the lyrics (the 10 most common nouns) and adds it to the resulting event.



Scalable vector updates reference architecture with MongoDB

Data model approach

```

1  {
2    "title": "Hurricane",
3    "genre": "rock",
4    "artist": "Bob Dylan",
5    "year": 1976,

```

```
6     "views": 307418,  
7     "lyrics": "...",  
8     "language": "en",  
9     "duration": 61,  
10    "lyrics_embeddings_en": [...],  
11    "tags": ["man", "story", "night"]  
12 }
```

The data we currently have about the song consists of the following fields:

- **Title:** Name of the song
- **Genre:** a single-worded string containing a music style from a list of 6 genres
- **Artist:** Name of the artist
- **Year:** The year in which the song was written
- **Views:** Number of times the song has been listened to
- **Lyrics:** A string field containing the lyrics with each line separated by a new line delimiter
- **Language:** The language of the lyrics in ISO-369. We are only storing songs in English and Spanish.
- **Duration:** Duration of the song in seconds
- **Lyrics embedding vector:** language-specific embeddings vector
- **Tags:** A list of tags associated with the lyrics

The benefit of using the document data model is that it allows you to store all the related information of a song in a single document for easy and fast retrieval.

Building the solution

In the [GitHub repository](#) you will find detailed instructions on how to build the solution to update your embeddings asynchronously and at scale, leveraging MongoDB Atlas.

Step 1: Create a MongoDB cluster

The first step is to create a MongoDB cluster. If you don't have an Atlas account, create one following the steps in this link:

<https://www.mongodb.com/docs/guides/atlas/account/>

We will create a cluster in Atlas using AWS as our cloud provider and us-east-1 as our region. Additionally, create an Atlas Stream Processing Instance (SPI) following the instructions in the documentation:

<https://www.mongodb.com/docs/atlas/atlas-sp/manage-processing-instance/>

Step 2: Create a Kafka cluster in Confluent

To create a Kafka cluster in Confluent Cloud follow the instructions in

Rag Applications



[cluster.html#create-ak-clusters](#)

Once you have created the cluster, go to cluster settings and copy the bootstrap URL.

The screenshot shows the MongoDB Atlas interface. On the left is a sidebar with navigation icons and labels: Cluster, lyrics_demo, Cluster Overview, Networking, API Keys, Cluster Settings (highlighted), Stream Lineage, Stream Designer, Topics, ksqlDB, Connectors, Clients, Schema Registry, and Cloud Tools. The main panel is titled 'Cluster settings' and has two tabs: 'General' (selected) and 'Capacity'. Under the 'General' tab, there are three sections: 'Identification', 'Endpoints', and 'Cloud details'. The 'Identification' section shows 'Name' as 'lyrics_demo' and 'Cluster ID' as 'lkc-x66wvk'. The 'Endpoints' section shows 'Bootstrap server' as 'pkc-p11xm.us-east-1.aws.confluent.cloud:9092' and 'REST endpoint' as 'https://pkc-p11xm.us-east-1.aws.confluent.cloud:443'. A note below the endpoints says 'Use the Kafka REST API to interact with your cluster and produce records'. The 'Cloud details' section shows 'Provider' as 'AWS', 'Region' as 'us-east-1', and 'Availability' as 'Single zone'.

Cluster settings

General Capacity

Identification

Name lyrics_demo

Cluster ID lkc-x66wvk

Endpoints

Bootstrap server pkc-p11xm.us-east-1.aws.confluent.cloud:9092

REST endpoint https://pkc-p11xm.us-east-1.aws.confluent.cloud:443

Use the Kafka REST API to interact with your cluster and produce records.

Cloud details

Provider AWS

Region us-east-1

Availability Single zone

Then, create an API key to connect to your cluster.

Create an API Key

Use this API key to connect with the cluster. Store the API key and secret below somewhere safe. This is the only time you'll see the secret.

These credentials can take up to one minute to propagate.

Key	JJ036	
Secret	Mlvy6zz0svy3cyey3iz/MvPHXG1R/	

Description

atlas_stream_processing



Download and continue

The next step is to configure the topics for use in this solution: SpanishInputTopic, EnglishInputTopic, and OutputTopic.

Step 3: Configure the Stream Processing connection registry

To configure a new connection, click the configure button in the Stream Processing Instance, then click Connection Registry and add a new connection.

You will use this to connect the Atlas Stream Processing Instance with the Kafka Cluster.

Once you have created your Kafka cluster, Confluent will provide you with the bootstrap server URL, username, and password for the Connection Registry.

Next, create a connection from the Atlas Stream Processing Instance to the MongoDB Atlas cluster.

Step 4: Connect to the Stream Processing Instance

To configure the pipelines and connections in the Stream Processing Instance, you can connect to the cluster using the Mongo Shell (mongosh).

When clicking on the Connect button in the Stream Processing Instance, the Atlas UI provides instructions on connecting to the instance.

Step 5: Configuring Atlas Stream Processing

You can follow the steps to configure Atlas Stream Processing in the README file in the GitHub repo. There you will learn how to create the pipelines to subscribe to changes in MongoDB, emit to each language-specific topic, and merge the events containing the processed data with the embeddings received from the Kafka cluster into MongoDB using a MongoDB aggregation stage.

Step 6: Create the Atlas Vector Search indexes

Next, you will create language-specific vector indexes in Atlas Search.

Visit the [Atlas Vector Search Quick Start guide](#) and start building smarter searches.

The definition for the Atlas Vector Search Index for Spanish is as follows:

```
1  {
2    "fields": [
3      {
4        "type": "vector",
5        "path": "lyrics_embeddings_es",
6        "numDimensions": 768,
7        "similarity": "cosine"
8      }
9    ]
10 }
```

The definition for the Atlas Vector Search Index for English is as follows:

```
1  {
2    "fields": [
3      {
4        "type": "vector",
5        "path": "lyrics_embeddings_en",
6        "numDimensions": 384,
7        "similarity": "cosine"
8      }
9    ]
10 }
```

Step 7: Run the metadata service

The metadata service is a Python script that will subscribe to the input topics, create the tags and embeddings for the corresponding language according to the information received in the event, and write the event to the output topic.

Step 8: Run a semantic search

We created a script in Python to help you interactively run semantic queries. You can find the script in the repository under the client folder.

Key learnings

- **Maintain embedding relevancy:** Regularly update data embeddings to ensure your semantic searches remain accurate, especially if your documents change frequently.
- **Optimize language-model pairing:** To maximize semantic search accuracy, ensure your large language model (LLM) closely aligns with the language of your data to significantly enhance the relevance and precision of your search results.
- **Embrace Flexible Embeddings:** MongoDB's flexible data model eliminates the need for rigid schema definitions. This flexibility allows you to store embeddings directly alongside your data, regardless of their length or the model used to generate them.
- **Choose the right similarity function:** The effectiveness of your semantic searches depends on the chosen similarity function. Tailor your selection to your specific use case.
- **Asynchronous Embedding Generation:** Generating embeddings can be computationally expensive. Consider running this task asynchronously to avoid impacting your application's performance. Leverage the cloud's elasticity by horizontally scaling the functions responsible for embedding generation to handle bursts in workload.

Technologies and products used

MongoDB developer data platform

- [Atlas Database](#)
- [Atlas Vector Search](#)

- [Atlas Stream Processing](#)

Partner technologies

- Confluent Cloud
- AWS EC2

Author

- David Sanchez, MongoDB

Related resources



GitHub Repository: mongodb-scalable-document-embeddings

Create this demo for yourself by following the instructions and associated models in this solution's repository.

[View repository](#) >



How to Perform Semantic Search in Atlas

Learn about performing an approximate nearest neighbor search.

Learn more >



Get Started with Atlas Stream Processing


Set up Atlas Stream Processing and run your first stream processor.

Learn more >

Get started with Atlas today

Get started in seconds. Our free clusters come with 512 MB of storage so you can experiment with sample data and get familiar with our platform.

Try Free

 Illustration of hands typing on a laptop in the foreground and a superimposed desktop window and coffee cup in the background.

About

Careers

Investor Relations

Legal

Privacy Policy

GitHub

Security Information

Trust Center

Connect with Us

Deployment Options

MongoDB Atlas

Enterprise Advanced

Community Edition

Support

Contact Us

Customer Portal

Atlas Status

Customer Support

Manage Cookies

Data Basics

Vector Databases

NoSQL Databases

Document Databases

RAG Database

ACID Transactions

MERN Stack

MEAN Stack

