Get Started with the LangChain Integration

NOTE

This tutorial uses LangChain's Python library. For a tutorial that uses the JavaScript library, see Get Started with the LangChain JS/TS Integration.

You can integrate Atlas Vector Search with LangChain to build LLM applications and implement retrieval-augmented generation (RAG). This tutorial demonstrates how to start using Atlas Vector Search with LangChain to perform semantic search on your data and build a RAG implementation. Specifically, you perform the following actions:

1. Set up the environment.

2. Store custom data on Atlas.
3. Create an Atlas Vector Search index on your data.
4. Run the following vector search queries:
   - Semantic search.
   - Semantic search with score.
   - Semantic search with metadata pre-filtering.
5. Implement RAG by using Atlas Vector Search to answer questions on your data.

Background

LangChain is an open-source framework that simplifies the creation of LLM applications through the use of "chains." Chains are LangChain-specific components that can be combined for a variety of AI use cases, including RAG.

By integrating Atlas Vector Search with LangChain, you can use Atlas as a vector database and use Atlas Vector Search to implement RAG by retrieving semantically similar documents from your data. To learn more about RAG, see Key Concepts.

Prerequisites

To complete this tutorial, you must have the following:

- An Atlas cluster running MongoDB version 6.0.11, 7.0.2, or later (including RCs). Ensure that your IP address is included in your Atlas project's access list.
- An OpenAI API Key. You must have a paid OpenAI account with credits available for API requests.
- An environment to run Python interactive notebooks such as Colab.

Set Up the Environment

You must first set up the environment for this tutorial. Create an interactive Python notebook by saving a file with the `.ipynb` extension, and then run the following code snippets in the notebook.

1 **Install and import dependencies.**

Run the following command:

```
%pip install --quiet langchain langchain-mc
```

Then, run the following code to import the required packages:

```
import getpass, os, pymongo, pprint
from langchain_community.document_loaders i
from langchain_core.output_parsers import S
from langchain_core.runnables import Runnab
from langchain_mongodb import MongoDBAtlasV
from langchain_openai import ChatOpenAI, Op
from langchain.prompts import PromptTemplat
from langchain.text_splitter import Recursi
from pymongo import MongoClient
```

2 Define environmental variables.

Run the following code and provide the following when prompted:

- Your OpenAI API Key.
- Your Atlas cluster's SRV connection string.

```
os.environ["OPENAI_API_KEY"] = getpass.getp
ATLAS_CONNECTION_STRING = getpass.getpass("
```

NOTE

Your connection string should use the following format:

```
mongodb+srv://<username>:<password
```

Use Atlas as a Vector Store

Then, load custom data into Atlas and instantiate Atlas as a vector database, also called a vector store. Copy and paste the following code snippets into your notebook.

1 **Connect to your Atlas cluster.**

Run the following code to establish a connection to your Atlas cluster. It specifies the following:

- `langchain_db.test` as the name of the collection for which to load the data.
- `vector_index` as the name of the Atlas Vector Search index to use for querying the data.

```
# Connect to your Atlas cluster
client = MongoClient(ATLAS_CONNECTION_STRIN

# Define collection and index name
db_name = "langchain_db"
collection_name = "test"
atlas_collection = client[db_name][collecti
vector_search_index = "vector_index"
```

2 **Load the sample data.**

For this tutorial, you use a publicly accessible PDF document titled MongoDB Atlas Best Practices as the data source for your vector store. This document describes various recommendations and core concepts for managing your Atlas deployments.

To load the sample data, run the following code snippet. It does the following:

- Retrieves the PDF from the specified URL and loads the raw text data.
- Uses a text splitter to split the data into smaller documents.
- Specifies chunk parameters, which determines the number of characters in each document and the number of characters that should overlap between two consecutive documents.

```
# Load the PDF
loader = PyPDFLoader("https://query.prod.cm
data = loader.load()

# Split PDF into documents
text_splitter = RecursiveCharacterTextSplit
docs = text_splitter.split_documents(data)

# Print the first document
docs[0]
```

HIDE OUTPUT

```
Document(page_content='Mong oDB Atlas Best P ra
```

3 Instantiate the vector store.

Run the following code to create a vector store named `vector_store` from the sample documents. This snippet uses the `MongoDBAtlasVectorSearch.from_documents` method and specifies the following parameters:

- The sample documents to store in the vector database.
- OpenAI's embedding model as the model used to convert text into vector embeddings for the `embedding` field.
- `langchain_db.test` as the Atlas collection to store the documents.
- `vector_index` as the index to use for querying the vector store.

```
# Create the vector store
vector_store = MongoDBAtlasVectorSearch.frc
    documents = docs,
    embedding = OpenAIEmbeddings(disallowec
    collection = atlas_collection,
    index_name = vector_search_index
)
```

TIP

After running the sample code, you can view your vector embeddings in the Atlas UI by navigating to the `langchain_db.test` collection in your cluster.

Create the Atlas Vector Search Index

To enable vector search queries on your vector store, create an Atlas Vector Search index on the `langchain_db.test` collection.

Required Access

To create an Atlas Vector Search index, you must have `Project Data Access Admin` or higher access to the Atlas project.

Procedure

1 **In Atlas, go to the *Clusters* page for your project.**

a. If it is not already displayed, select the organization that contains your desired project from the **Organizations** menu in the navigation bar.

b. If it is not already displayed, select your desired project from the **Projects** menu in the navigation bar.

c. If the **Clusters** page is not already displayed, click **Database** in the sidebar.

2 **Go to the Atlas Search page for your cluster.**

You can go the Atlas Search page from the sidebar, the **Data Explorer**, or your cluster details page.

Sidebar Data Explorer Cluster Details

a. In the sidebar, click **Atlas Search** under the **Services** heading.

b. From the **Select data source** dropdown, select your cluster and click **Go to Atlas Search**.

3 **Define the Atlas Vector Search index.**

a. Click **Create Search Index**.

b. Under **Atlas Vector Search**, select **JSON Editor** and then click **Next**.

c. In the **Database and Collection** section, find the `langchain_db` database, and select the `test` collection.

d. In the **Index Name** field, enter `vector_index`.

e. Replace the default definition with the following index definition and then click **Next**.

This index definition specifies indexing the following fields in an index of the vectorSearch type:

- `embedding` field as the vector type. The `embedding` field contains the embeddings created using OpenAI's

`text-embedding-ada-002` embedding model. The index definition specifies `1536` vector dimensions and measures similarity using `cosine`.
  - `page` field as the filter type for pre-filtering data by the page number in the PDF.

```
{
   "fields":[
      {
         "type": "vector",
         "path": "embedding",
         "numDimensions": 1536,
         "similarity": "cosine"
      },
      {
         "type": "filter",
         "path": "page"
      }
   ]
}
```

4 Review the index definition and then click ***Create Search Index.***

A modal window displays to let you know that your index is building.

5 **Click *Close* to close the *You're All Set!* modal window and wait for the index to finish building.**

The index should take about one minute to build. While it builds, the **Status** column reads **Initial Sync**. When it finishes building, the **Status** column reads **Active**.

Run Vector Search Queries

Once Atlas builds your index, return to your notebook and run vector search queries on your data. The following examples demonstrate various queries that you can run on your vectorized data.

Semantic Search **Semantic Search with Score** **Semant**

The following query uses the `similarity_search` method to perform a basic semantic search for the string `MongoDB Atlas security`. It returns a list of documents ranked by relevance.

```
query = "MongoDB Atlas security"
results = vector_store.similarity_search(query)

pprint.pprint(results)
```

HIDE OUTPUT

```
[Document(page_content='To ensure a secure system ri
 Document(page_content='MongoD B Atlas team are also
 Document(page_content='MongoD B.\nMongoD B Atlas ir
 Document(page_content='Atlas provides encryption of
```

Access Data Lake
Query Federated Data
Atlas Search
Atlas Vector Search
Quick Start
Create Embeddings
Create and Manage Indexes
Create and Run Queries
Review Deployment

TIP

See also:

For a full list of semantic search methods, refer to the API reference.

On this page
Background
Prerequisites
Set Up the Environment
Use Atlas as a Vector Store
Create the Atlas Vector Search Index
Run Vector Search Queries
Answer Questions on Your Data

Options

Tutorials

AI Integrations

LangChain

LangChain JS/TS

LlamaIndex

Answer Questions on Your Data

This section demonstrates how to implement RAG in your application with Atlas Vector Search and LangChain. Now that you've used Atlas Vector Search to retrieve semantically similar documents, run the following code examples to prompt the LLM to answer questions based on those documents.

Next Steps

Basic RAG RAG with Filtering

This example does the following:

- Instantiates Atlas Vector Search as a retriever to query for similar documents, including the optional `k` parameter to search for only the `10` most relevant documents.
- Defines a LangChain prompt template to instruct the LLM to use these documents as context for your query. LangChain passes these documents to the `{context}` input variable and your query to the `{question}` variable.
- Constructs a chain that specifies the following:

   - Atlas Vector Search as the retriever to search for documents that are used as context by the LLM.
   - The prompt template that you constructed.
   - OpenAI's chat model as the LLM used to generate a context-aware response.
- Prompts the chain with a sample query about Atlas security recommendations.
- Returns the LLM's response and the documents used as context. The generated response might vary.

```
# Instantiate Atlas Vector Search as a retrieve
retriever = vector_store.as_retriever(
   search_type = "similarity",
   search_kwargs = { "k": 10 }
)

# Define a prompt template
template = """
Use the following pieces of context to answer t
If you don't know the answer, just say that you
{context}
Question: {question}
"""
custom_rag_prompt = PromptTemplate.from_templat

llm = ChatOpenAI()

def format_docs(docs):
   return "\n\n".join(doc.page_content for doc

# Construct a chain to answer questions on your
rag_chain = (
   { "context": retriever | format_docs, "quest
   | custom_rag_prompt
   | llm
   | StrOutputParser()
```

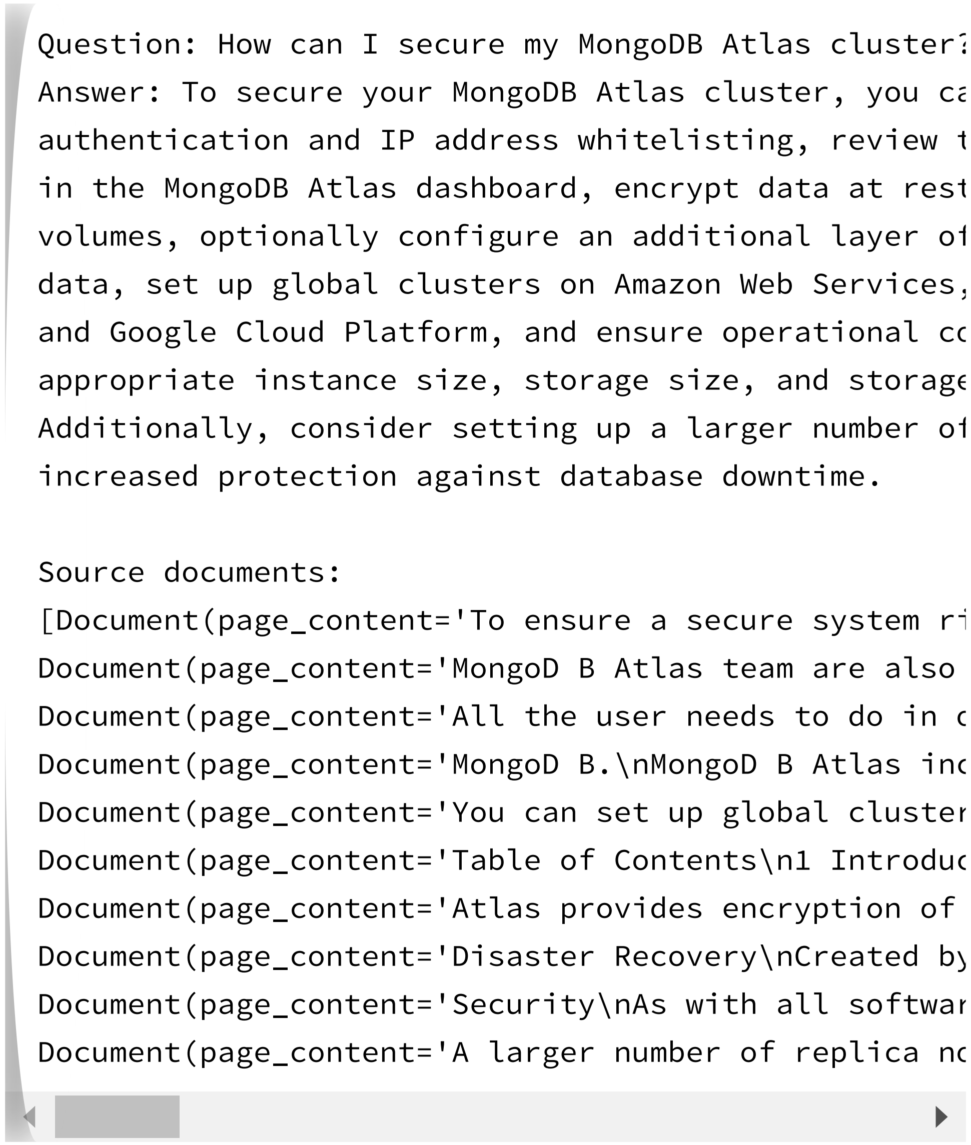
```
)

# Prompt the chain
question = "How can I secure my MongoDB Atlas c
answer = rag_chain.invoke(question)

print("Question: " + question)
print("Answer: " + answer)

# Return source documents
documents = retriever.get_relevant_documents(qu
print("\nSource documents:")
pprint.pprint(documents)
```

HIDE OUTPUT



```
Question: How can I secure my MongoDB Atlas cluster?
Answer: To secure your MongoDB Atlas cluster, you ca
authentication and IP address whitelisting, review t
in the MongoDB Atlas dashboard, encrypt data at rest
volumes, optionally configure an additional layer of
data, set up global clusters on Amazon Web Services,
and Google Cloud Platform, and ensure operational co
appropriate instance size, storage size, and storage
Additionally, consider setting up a larger number of
increased protection against database downtime.

Source documents:
[Document(page_content='To ensure a secure system ri
Document(page_content='MongoD B Atlas team are also
Document(page_content='All the user needs to do in c
Document(page_content='MongoD B.\nMongoD B Atlas inc
Document(page_content='You can set up global cluster
Document(page_content='Table of Contents\n1 Introduc
Document(page_content='Atlas provides encryption of
Document(page_content='Disaster Recovery\nCreated by
Document(page_content='Security\nAs with all softwar
Document(page_content='A larger number of replica nc
```

Next Steps

To learn about additional RAG use-cases with Atlas Vector Search, see the following templates provided by LangChain to help you build applications:

- Basic RAG with MongoDB and OpenAI
- Advanced RAG: Parent-Document Retrieval

MongoDB also provides the following developer resources:

- Introduction to LangChain and MongoDB Atlas Vector Search
- RAG with Atlas Vector Search, LangChain, and OpenAI
- Leveraging MongoDB Atlas Vector Search with LangChain
- MongoDB Developer GitHub Repository

TIP

See also:

- LangChain Documentation
- LangChain API Reference

Manage Cookies

Deployment Options

MongoDB Atlas

Enterprise Advanced

Community Edition