MongoDB

MONGODB DEVELOPER CENTER > DEVELOPER TOPICS > PRODUCTS > ATLAS > TUTORIALS

How to Do Semantic Search in MongoDB Using Atlas Vector Search

Benjamin Flast

8 min read • Published Jul 19, 2023 • Updated Jan 12, 2024

Node.js Serverless Search JavaScript Atlas

Rate this tutorial ☆ ☆ ☆ ☆ ☆

Have you ever been looking for something but don't quite have the words? Do you remember some characteristics of a movie but can't remember the name? Have you ever been trying to get another sweatshirt just like the one you had back in the day but don't know how to search for it? Are you using large language models, but they only know information up until 2021? Do you want it to get with the times?! Well then, vector search may be just what you're looking for.

What is vector search?

Vector search is a capability that allows you to do semantic search where you are searching data based on meaning. This technique employs machine learning models, often called encoders, to transform text, audio, images, or other types of data into high-dimensional vectors. These vectors capture the semantic meaning of the data, which can then be searched through to find similar content based on vectors being "near" one another in a high-dimensional space. This can be a great compliment to traditional keyword-based search techniques but is also seeing an explosion of excitement because of its relevance to augment the capabilities of large language models (LLMs) by providing ground truth outside of what the LLMs "know." In search use cases, this allows you to find relevant results even when the exact wording isn't known. This technique can be useful in a variety of contexts, such as natural language processing and recommendation systems.

Note: As you probably already know, MongoDB Atlas has supported full-text search since 2020, allowing you to do rich text search on your MongoDB data.

The core difference between vector search and text search is that vector search queries on meaning instead of explicit text and therefore can also search data beyond just text.

Benefits of vector search

- Semantic understanding: Rather than searching for exact matches, vector search enables semantic searching. This means that even if the query words aren't present in the index, but the meanings of the phrases are similar, they will still be considered a match.
- Scalable: Vector search can be done on large datasets, making it perfect for use cases where you have a lot of data.
- Flexible: Different types of data, including text but also unstructured data like audio and images, can be semantically searched.

Benefits of vector search with MongoDB

- Efficiency: By storing the vectors together with the original data, you avoid the need to sync data between your application database and your vector store at both query and write time.
- Consistency: Storing the vectors with the data ensures that the vectors are always associated with the correct data. This can be important in situations where the vector generation process might change over time. By storing the vectors, you can be sure that you always have the correct vector for a given piece of data.
- Simplicity: Storing vectors with the data simplifies the overall architecture

of your application. You don't need to maintain a separate service or database for the vectors, reducing the complexity and potential points of failure in your system.

- Scalability: With the power of MongoDB Atlas, vector search on MongoDB scales horizontally and vertically, allowing you to power the most demanding workloads.

Want to experience Vector Search with MongoDB quick and easy? Check out this automated demo on GitHub as you walk through the tutorial.

Set up a MongoDB Atlas cluster

Now, let's get into setting up a MongoDB Atlas cluster, which we will use to store our embeddings.

Step 1: Create an account

To create a MongoDB Atlas cluster, first, you need to create a MongoDB Atlas account if you don't already have one. Visit the MongoDB Atlas website and click on "Register."

Step 2: Build a new cluster

After creating an account, you'll be directed to the MongoDB Atlas dashboard. You can create a cluster in the dashboard, or using our public API, CLI, or Terraform provider. To do this in the dashboard, click on "Create Cluster," and then choose the shared clusters option. We suggest creating an M0 tier cluster.

If you need help, check out our tutorial demonstrating the deployment of Atlas using various strategies.

Step 3: Create your collections

Now, we're going to create your collections in the cluster so that we can insert our data. They need to be created now so that you can create an Atlas trigger that will target them.

For this tutorial, you can create your own collection if you have data to use. If you'd like to use our sample data, you need to first create an empty collection in the cluster so that we can set up the trigger to embed them as they are inserted. Go ahead and create a "sample_mflix" database and "movies" collection now using the UI, if you'd like to use our sample data.

Setting up an Atlas trigger

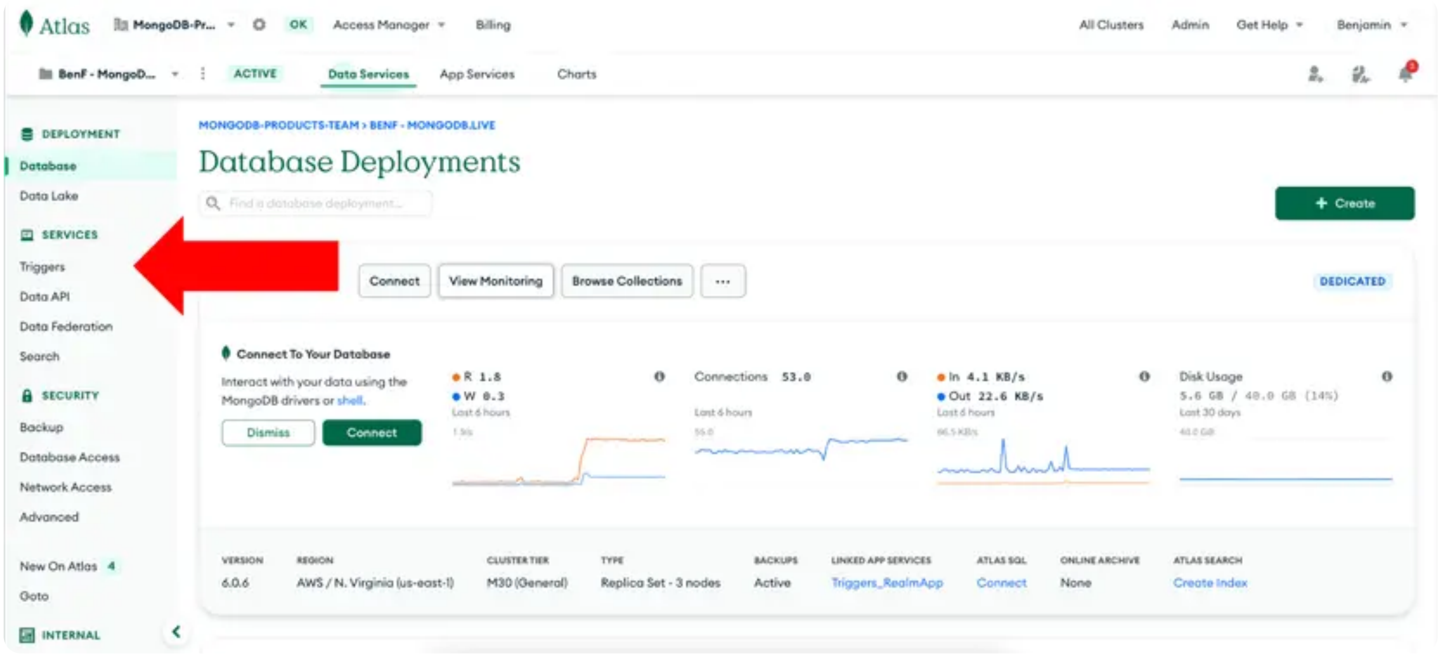
We will create an Atlas trigger to call the OpenAI API whenever a new document is inserted into the cluster.

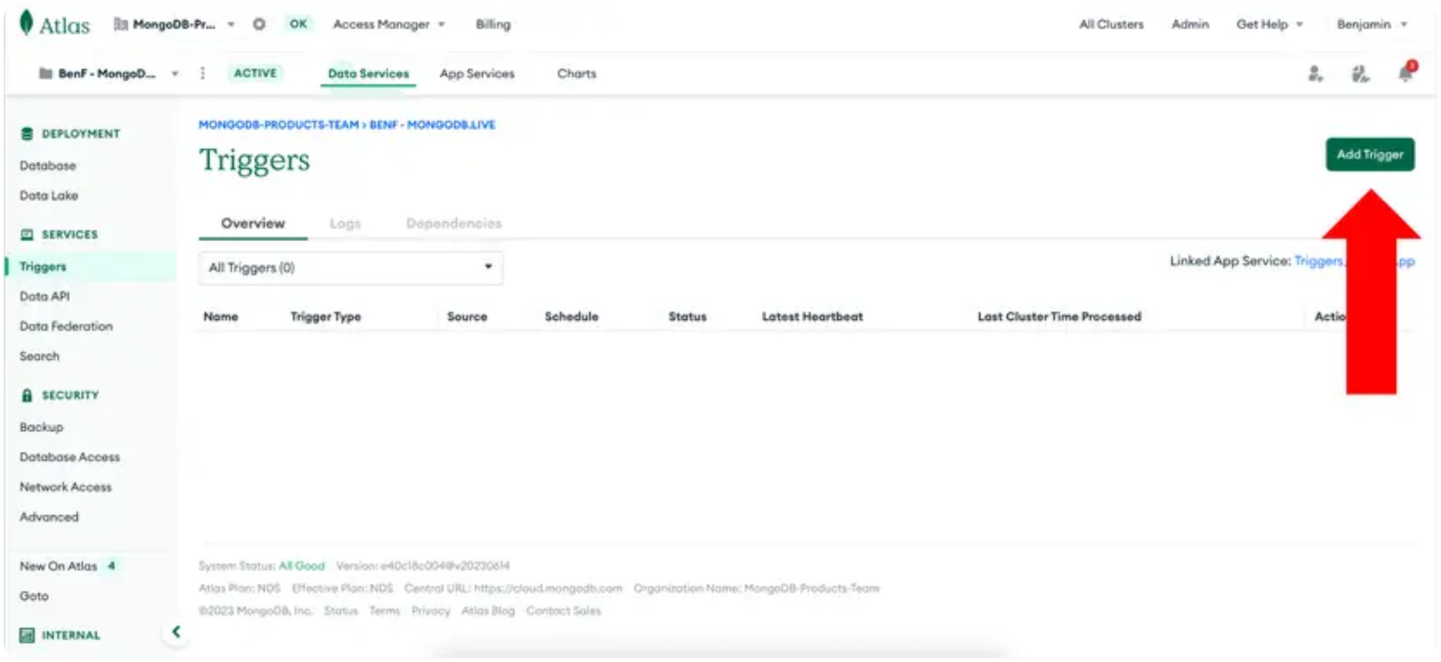
To proceed to the next step using OpenAI, you need to have set up an account on OpenAI and created an API key.

If you don't want to embed all the data in the collection you can use the "sample_mflix.embedded_movies" collection for this which already has embeddings generated by Open AI, and just create an index and run Vector Search queries.

Step 1: Create a trigger

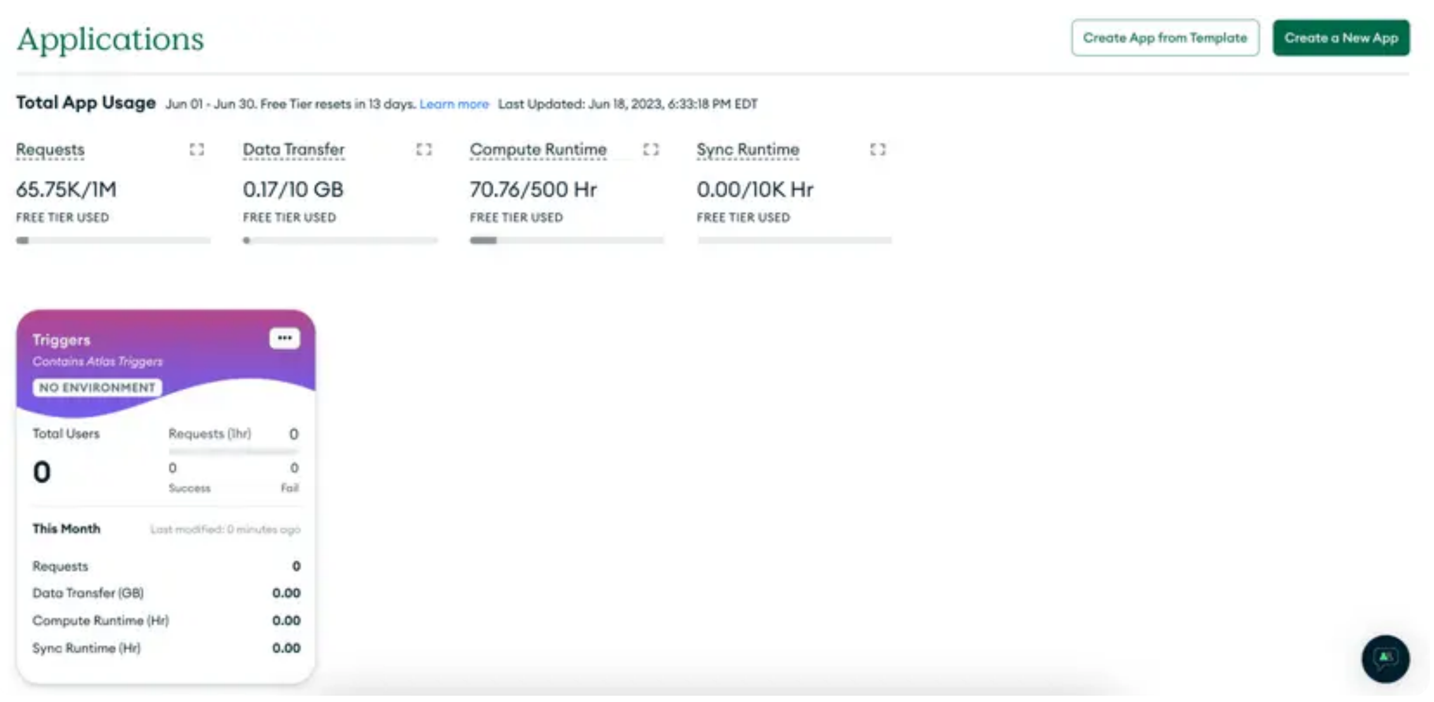
To create a trigger, navigate to the “Triggers” section in the MongoDB Atlas dashboard, and click on “Add Trigger.”



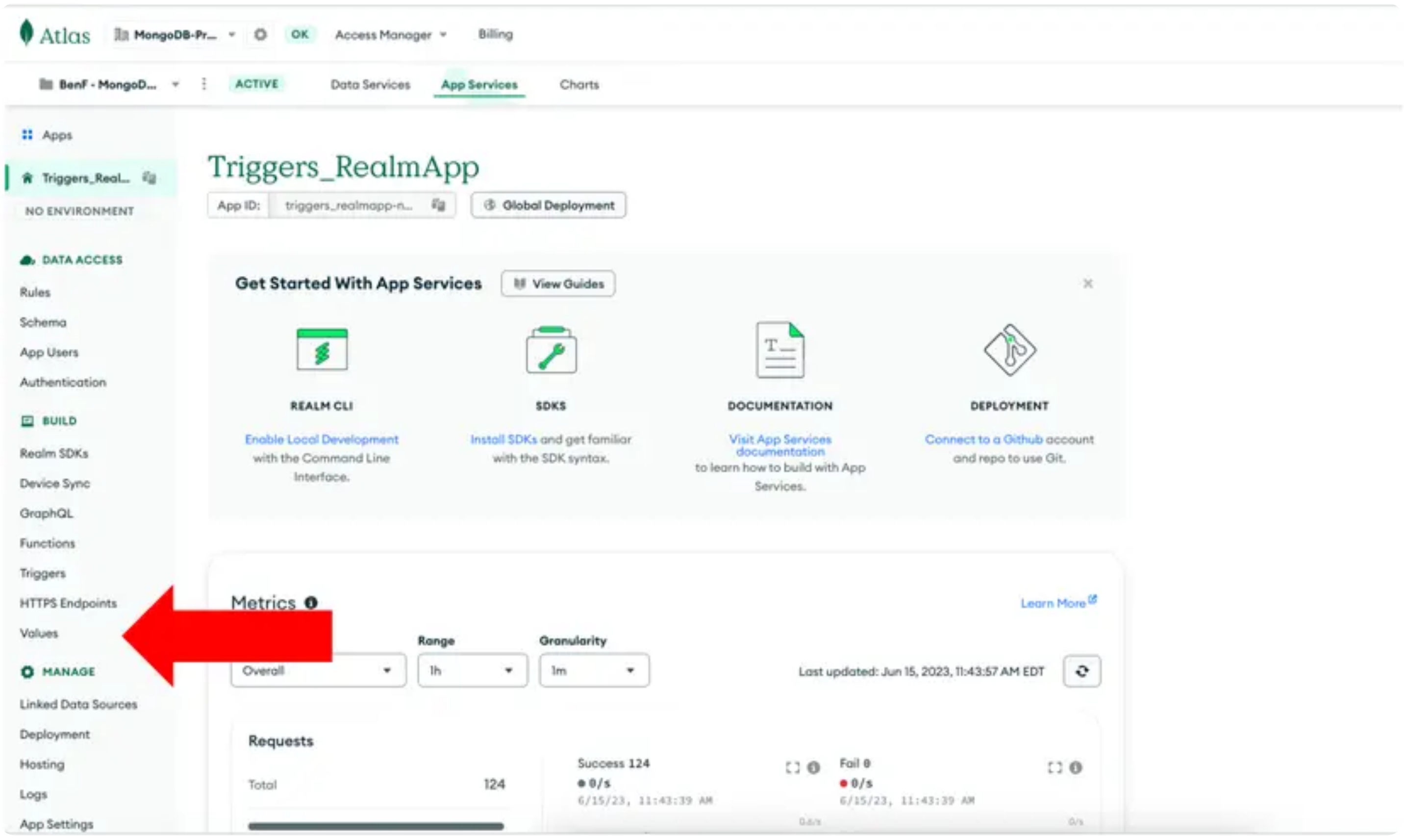


Step 2: Set up secrets and values for your OpenAI credentials

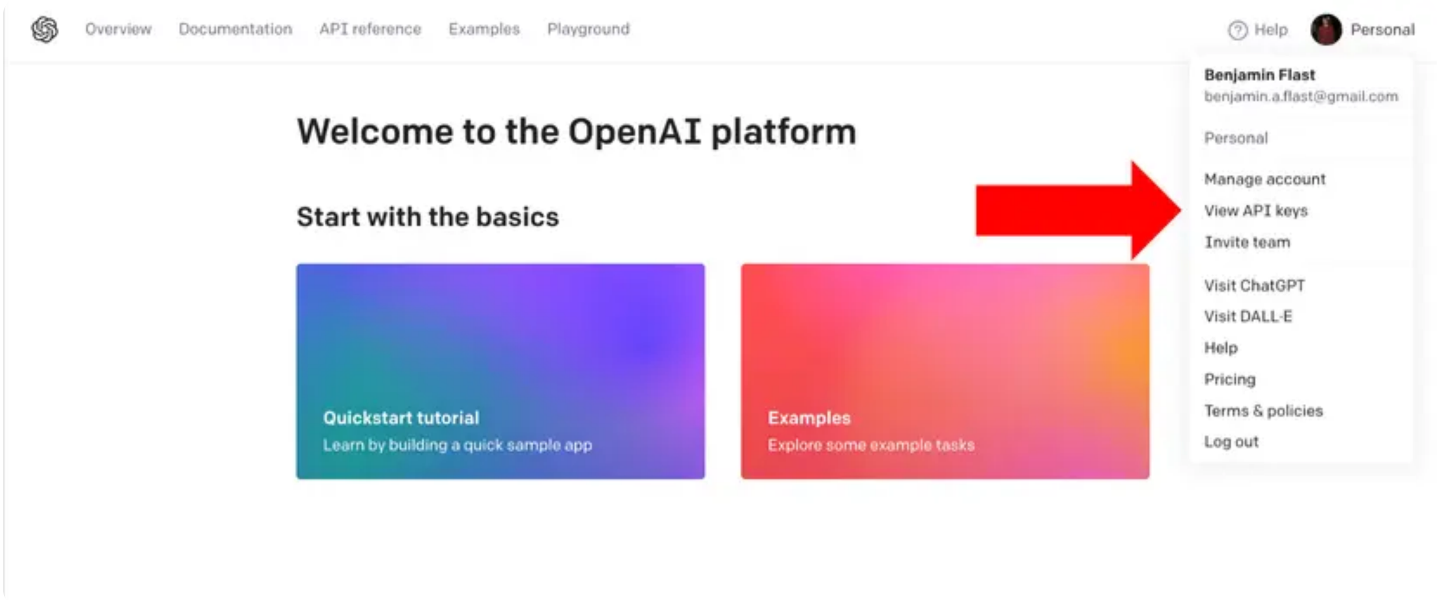
Go over to “App Services” and select your “Triggers” application.

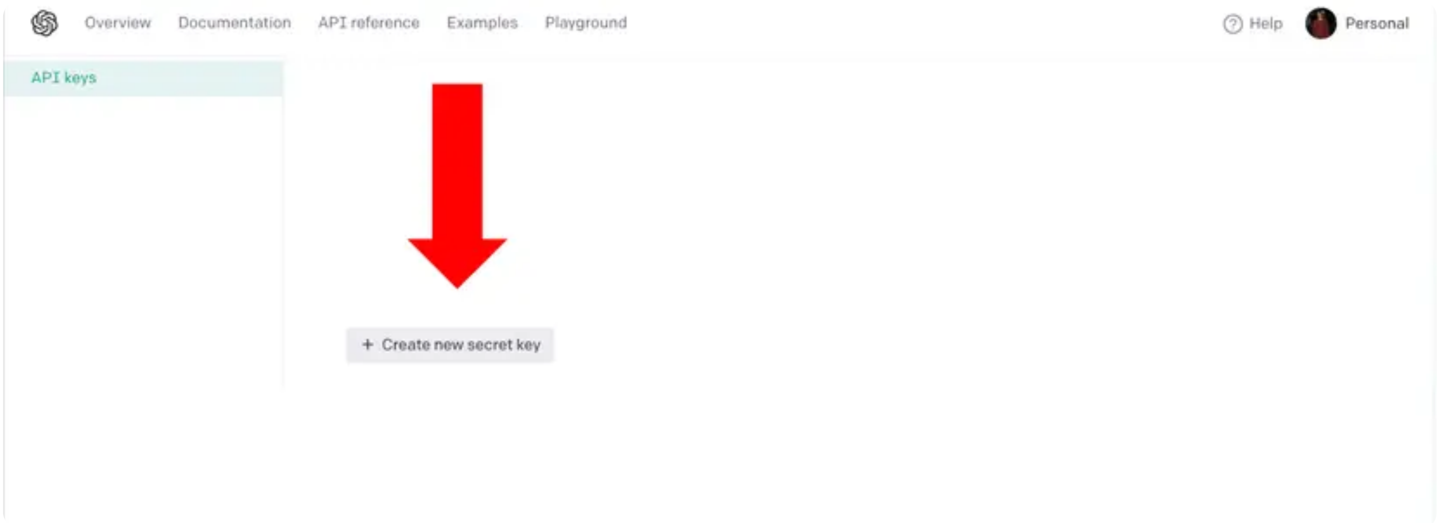

Applications
Create App from Template
Create a New App
Total App Usage Jun 01 - Jun 30. Free Tier resets in 13 days. Learn more Last Updated: Jun 18, 2023, 6:33:18 PM EDT
Requests 65.75K/1M FREE TIER USED
Data Transfer 0.17/10 GB FREE TIER USED
Compute Runtime 70.76/500 Hr FREE TIER USED
Sync Runtime 0.00/10K Hr FREE TIER USED
Triggers
Contains Atlas Triggers
NO ENVIRONMENT
Total Users 0
Requests (1hr) 0
0 Success 0 Fail
This Month
Requests 0
Data Transfer (GB) 0.00
Compute Runtime (Hr) 0.00
Sync Runtime (Hr) 0.00

Click “Values.”

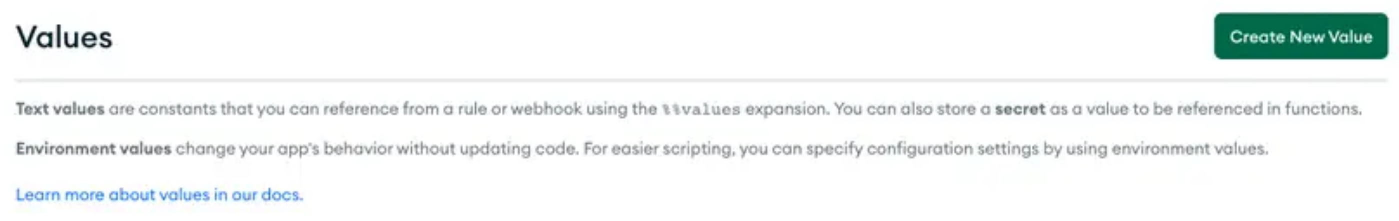


You’ll need your OpenAI API key, which you can create on their website:





Create a new Value

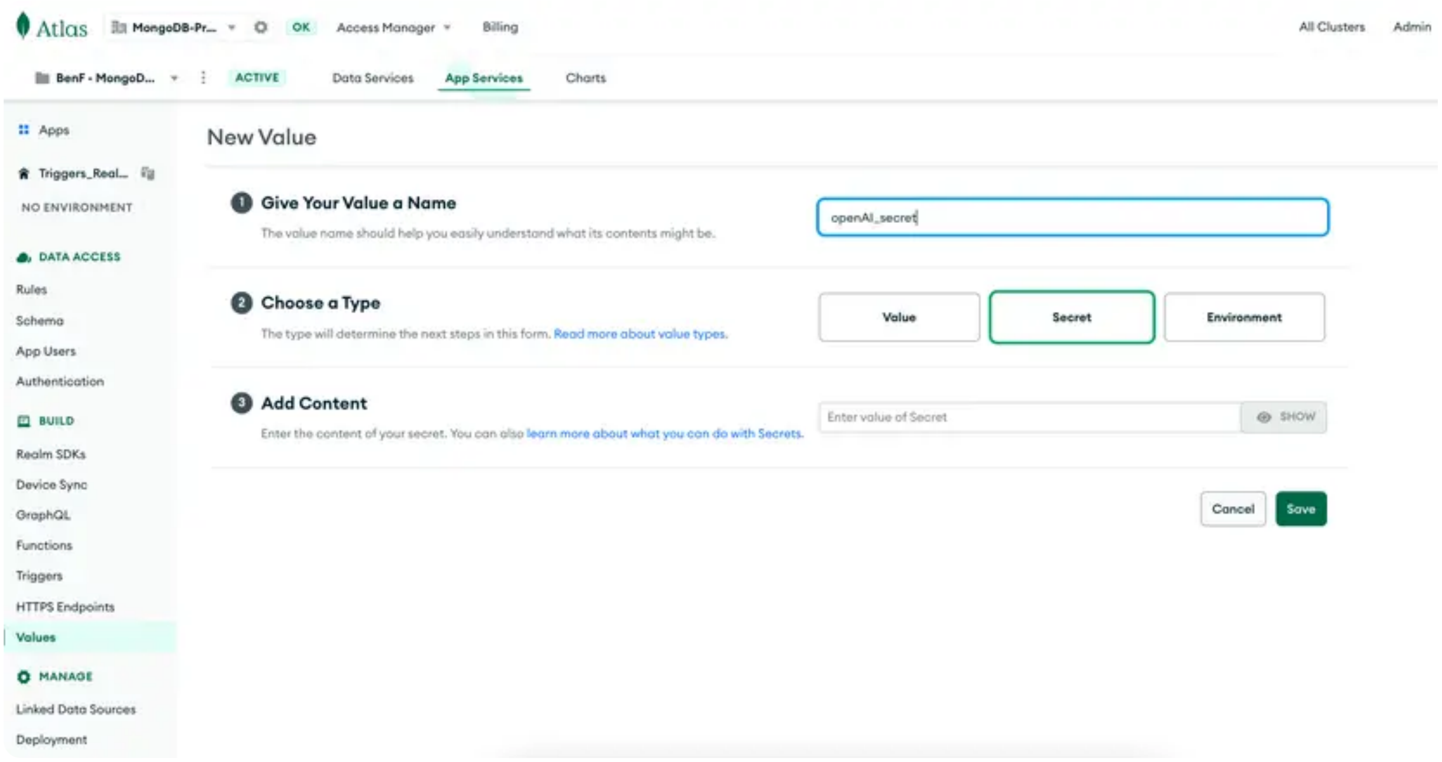


Text values are constants that you can reference from a rule or webhook using the %%values expansion. You can also store a secret as a value to be referenced in functions.

Environment values change your app's behavior without updating code. For easier scripting, you can specify configuration settings by using environment values.

Learn more about values in our docs.

Select “Secret” and then paste in your OpenAI API key.



New Value

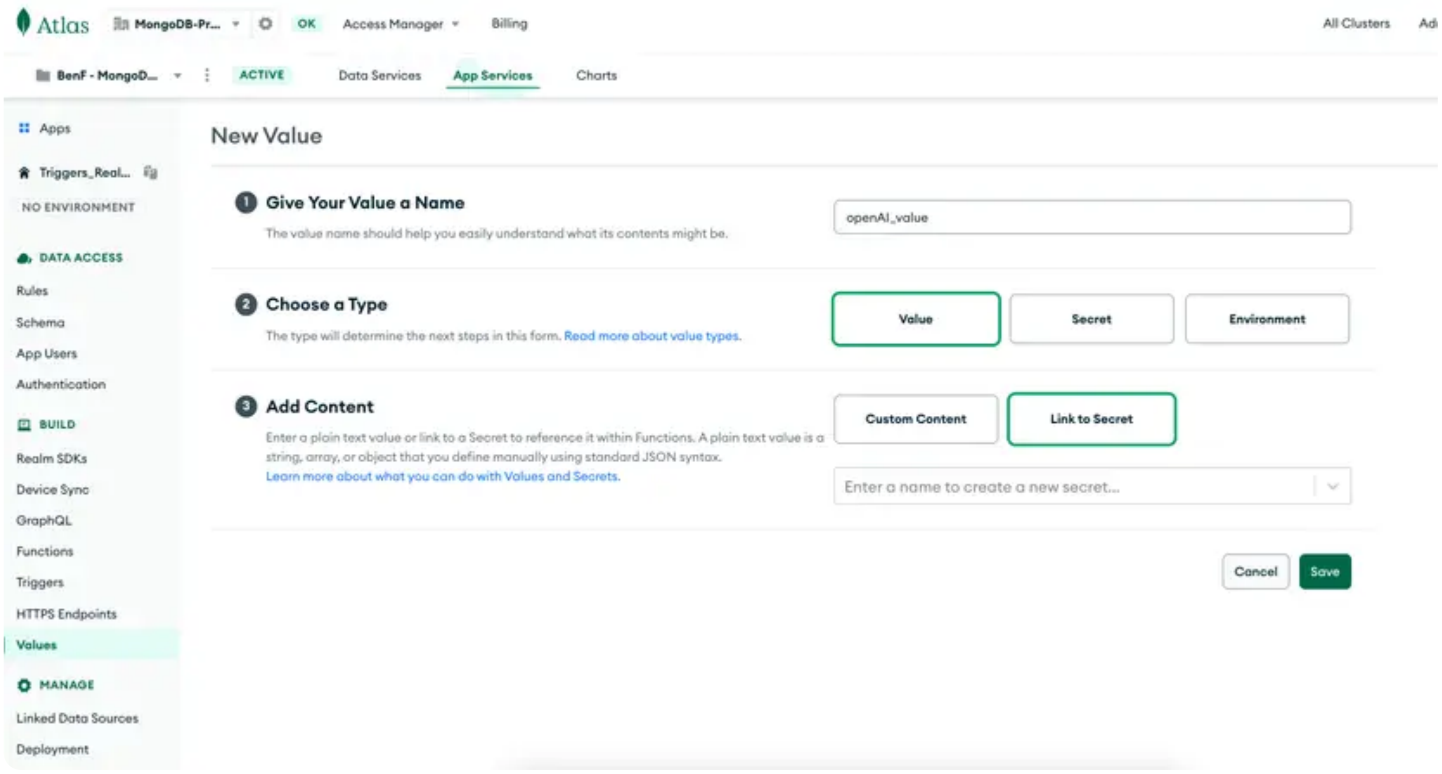
1 Give Your Value a Name
The value name should help you easily understand what its contents might be.
openAI_secret

2 Choose a Type
The type will determine the next steps in this form. Read more about value types.
Value [✓ Secret] Environment

3 Add Content
Enter the content of your secret. You can also learn more about what you can do with Secrets.
Enter value of Secret SHOW

Cancel Save

Then, create another value – this time, a "Value" – and link it to your secret. This is how you will securely reference this API key in your trigger.



New Value

1 Give Your Value a Name
The value name should help you easily understand what its contents might be.
openAI_value

2 Choose a Type
The type will determine the next steps in this form. Read more about value types.
[✓ Value] Secret Environment

3 Add Content
Enter a plain text value or link to a Secret to reference it within Functions. A plain text value is a string, array, or object that you define manually using standard JSON syntax.
Learn more about what you can do with Values and Secrets.
Custom Content [✓ Link to Secret]
Enter a name to create a new secret...

Cancel Save

Now, you can go back to the “Data Services” tab and into the triggers menu. If the trigger you created earlier does not show up, just add a new trigger. It will be able to utilize the values you set up in App Services earlier.

Step 3: Configure the trigger

Select the “Database” type for your trigger. Then, link the source cluster and set the “Trigger Source Details” to be the Database and Collection to watch for changes. For this tutorial, we are using the “sample_mflix” database and the “movies” collection. Set the Operation Type to 'Insert' ‘Update’ ‘Replace’ operation. Check the “Full Document” flag and in the Event Type, choose “Function.”

In the Function Editor, use the code snippet below, replacing DB Name and Collection Name with the database and collection names you’d like to use, respectively.

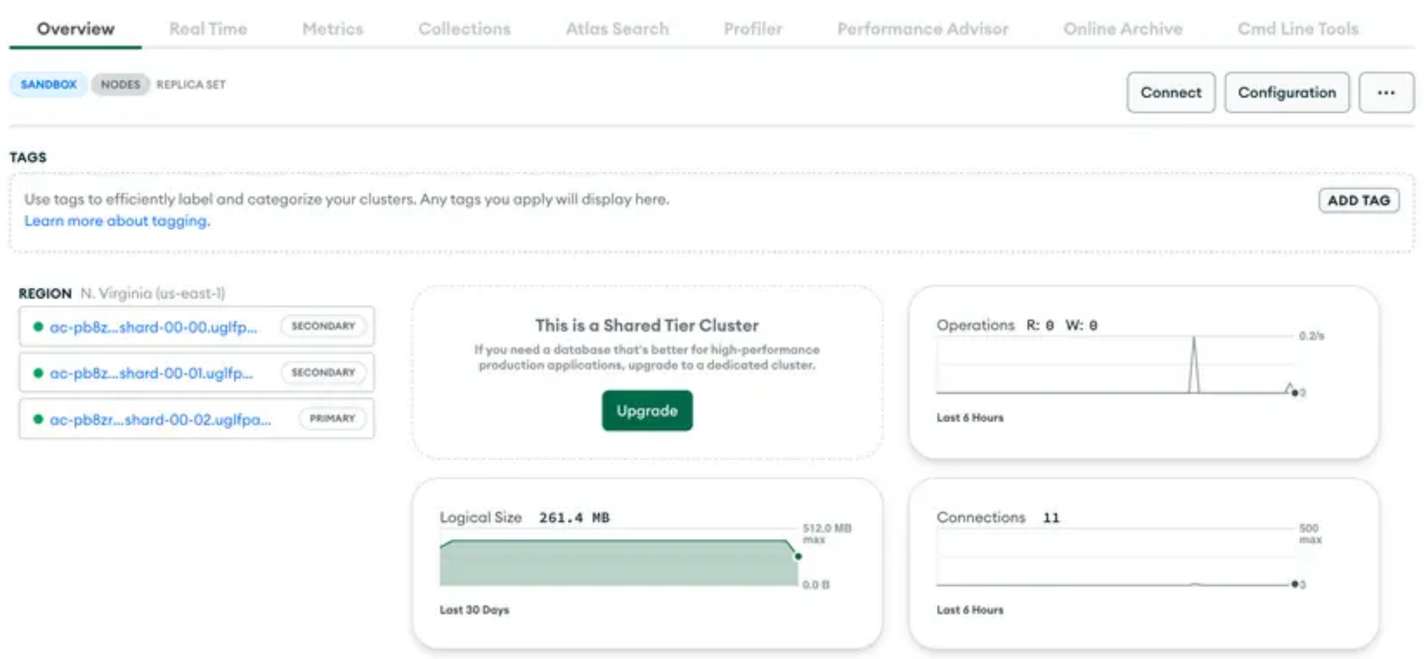
This trigger will see when a new document is created or updated in this collection. Once that happens, it will make a call to the OpenAI API to create an embedding of the desired field, and then it will insert that vector embedding into the document with a new field name.

```
exports = async function(changeEvent) {
    // Get the full document from the change event.
    const doc = changeEvent.fullDocument;

    // Define the OpenAI API url and key.
    const url = 'https://api.openai.com/v1/embeddings';
    // Use the name you gave the value of your API key in the '
    const openai_key = context.values.get("openAI_value");
    try {
        console.log(`Processing document with id: ${doc._id}`);

        // Call OpenAI API to get the embeddings.
        let response = await context.http.post({
            url: url,
             headers: {
                'Authorization': [`Bearer ${openai_key}`],
                'Content-Type': ['application/json']
            },
            body: JSON.stringify({
                // The field inside your document that contains
                input: doc.plot,
                model: "text-embedding-ada-002"
            })
        });

        // Parse the JSON response
        let responseData = EJSON.parse(response.body.text());

        // Check the response status.
        if(response.statusCode === 200) {
            console.log("Successfully received embedding.");

            const embedding = responseData.data[0].embedding;

            // Use the name of your MongoDB Atlas Cluster
            const collection = context.services.get("<CLUSTER_N

            // Update the document in MongoDB.
            const result = await collection.updateOne(
                { _id: doc._id },
                // The name of the new field you'd like to cont
                { $set: { plot_embedding: embedding }}
            );

            if(result.modifiedCount === 1) {
                console.log("Successfully updated the document.
            } else {
                console.log("Failed to update the document.");
            }
        } else {
            console.log(`Failed to receive embedding. Status cc
        }

    } catch(err) {
        console.error(err);
    }
};
```

Configure index

Now, head over to Atlas Search and create an index. Use the JSON index definition and insert the following, replacing the embedding field name with the field of your choice. If you are using the sample_mflix database, it should be "plot_embedding", and give it a name. I've used "moviesPlotIndex" for my setup with the sample data.

First, click the "atlas search" tab on your cluster

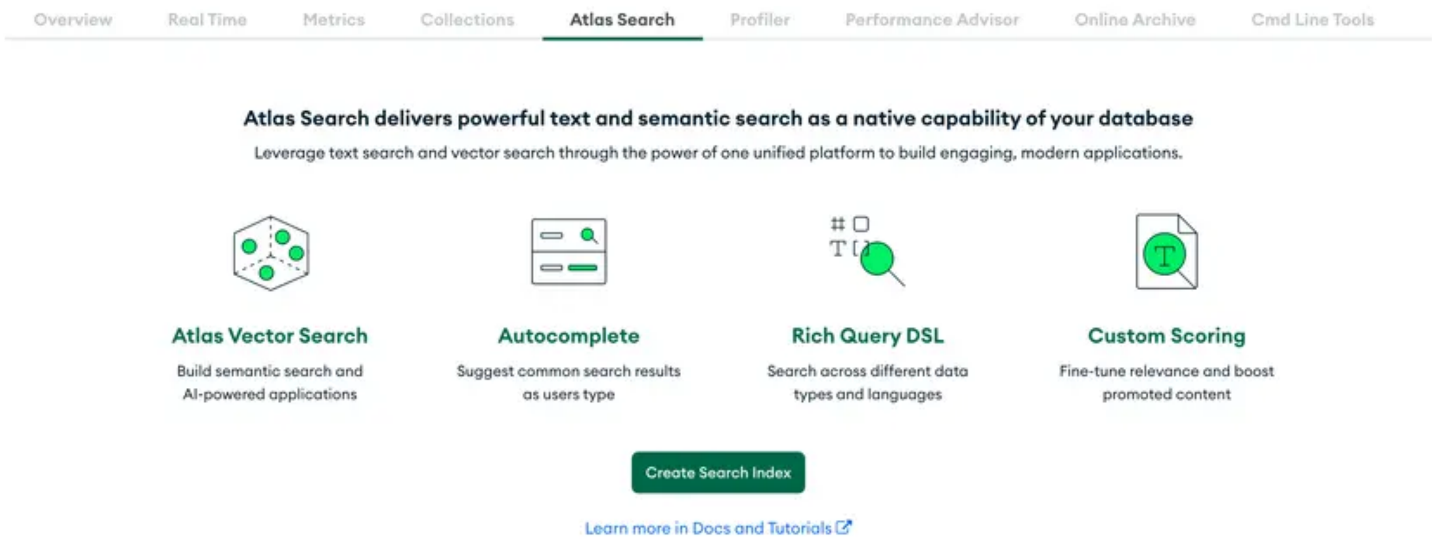


Use tags to efficiently label and categorize your clusters. Any tags you apply will display here.
Learn more about tagging.

This is a Shared Tier Cluster

If you need a database that's better for high-performance production applications, upgrade to a dedicated cluster.

Then, click "Create Search Index."



Atlas Search delivers powerful text and semantic search as a native capability of your database

Leverage text search and vector search through the power of one unified platform to build engaging, modern applications.

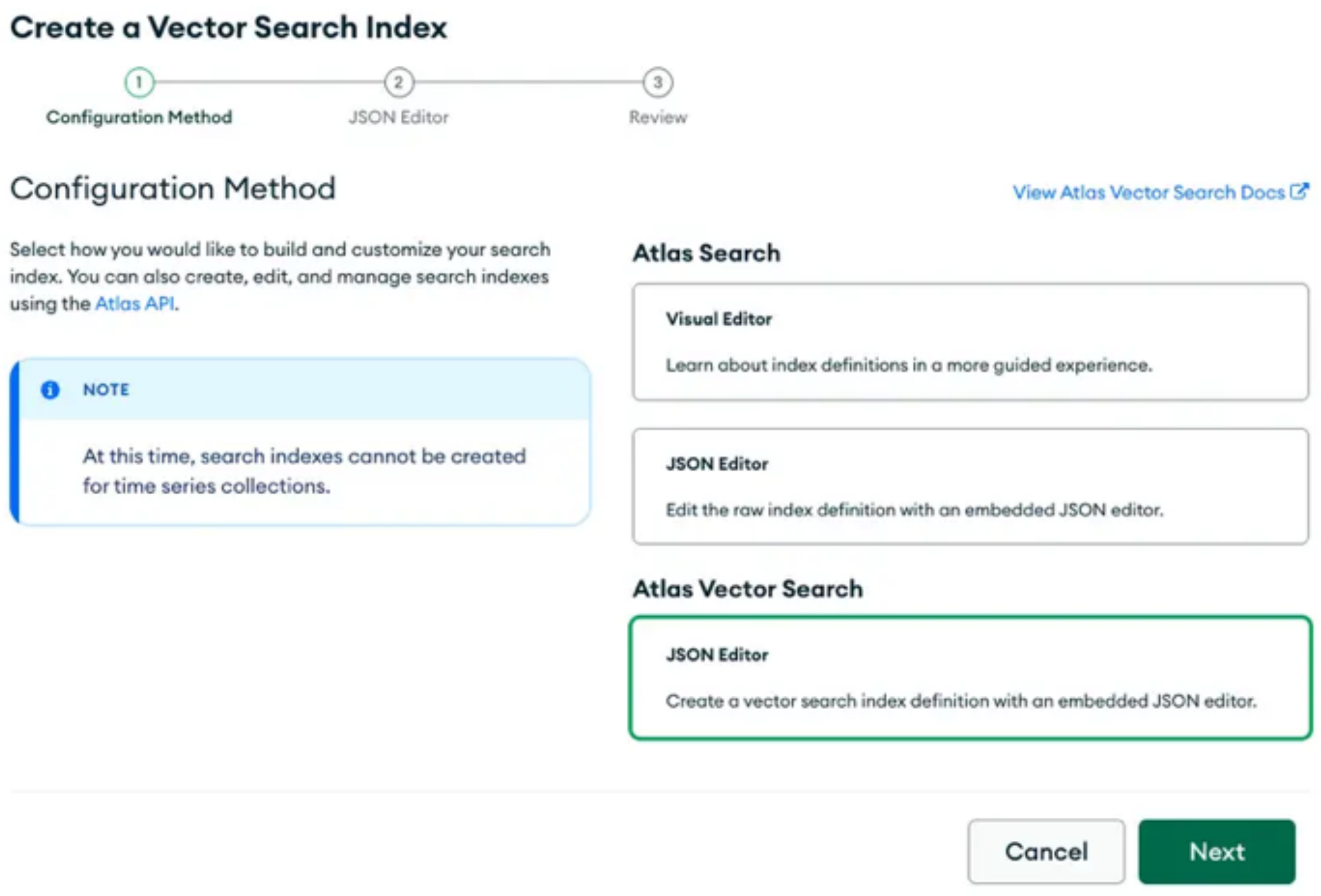
Atlas Vector Search
Build semantic search and AI-powered applications

Autocomplete
Suggest common search results as users type

Rich Query DSL
Search across different data types and languages

Custom Scoring
Fine-tune relevance and boost promoted content

Create "JSON Editor."



Create a Vector Search Index

Configuration Method

Select how you would like to build and customize your search index. You can also create, edit, and manage search indexes using the Atlas API.

NOTE

At this time, search indexes cannot be created for time series collections.

Atlas Search

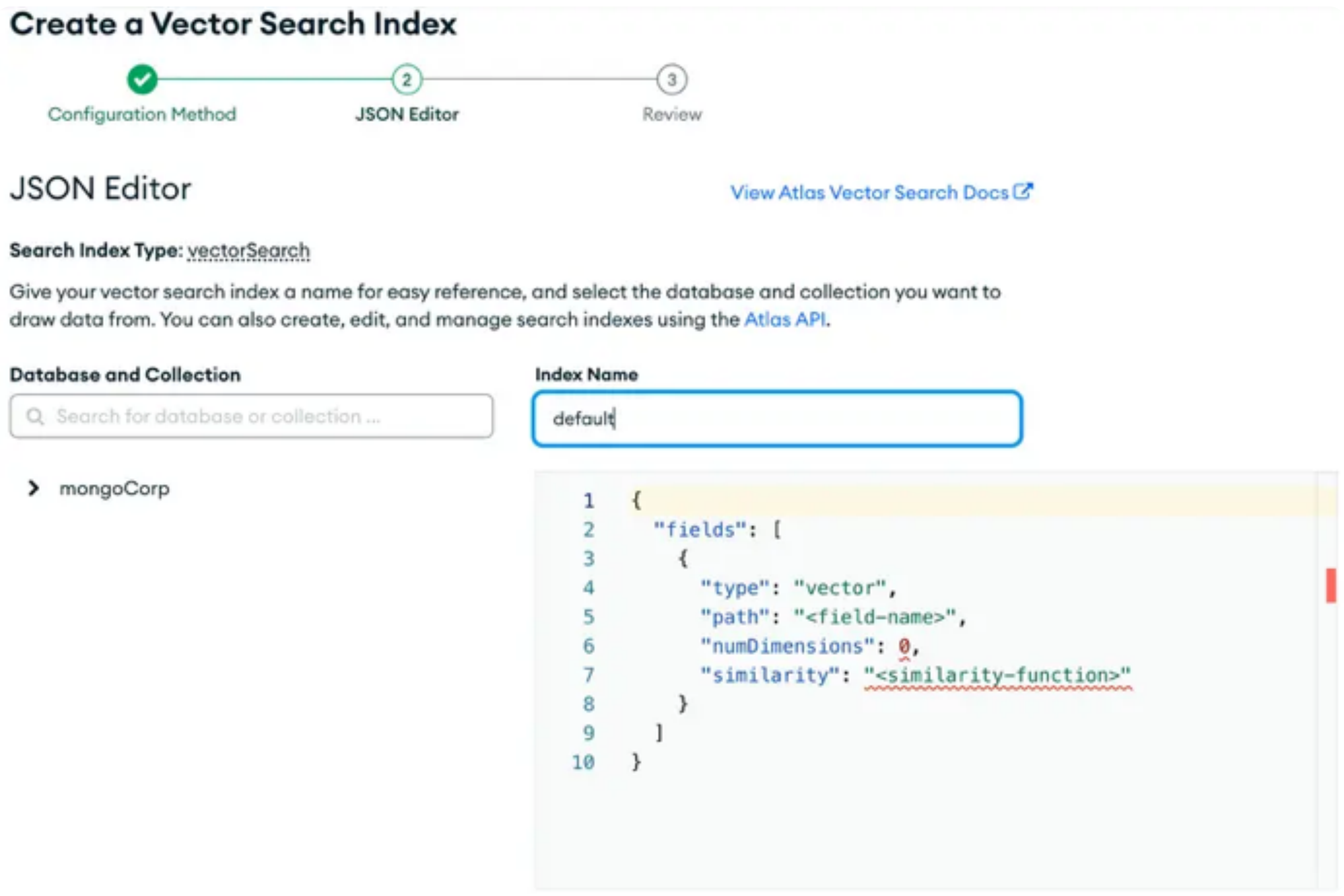
Visual Editor
Learn about index definitions in a more guided experience.

JSON Editor
Edit the raw index definition with an embedded JSON editor.

Atlas Vector Search

JSON Editor
Create a vector search index definition with an embedded JSON editor.

Then, select your Database and Collection on the left and a drop in the code snippet below for your index definition.



Search Index Type: vectorSearch

Give your vector search index a name for easy reference, and select the database and collection you want to draw data from. You can also create, edit, and manage search indexes using the Atlas API.

Database and Collection: Search for database or collection ...

mongoCorp

Index Name: default

```
{
  "fields": [
    {
      "type": "vector",
      "path": "<field-name>",
      "numDimensions": 0,
      "similarity": "<similarity-function>"
    }
  ]
}
```

```
{
  "type": "vectorSearch",
  "fields": [{
    "path": "plot_embedding",
    "dimensions": 1536,
    "similarity": "cosine",
    "type": "vector"
  }]
}
```

Insert your data

Now, you need to insert your data. As your data is inserted, it will be embedded using the script and then indexed using the KNN index we just set.

If you have your own data, you can insert it now using something like MongoImports.

If you're going to use the sample movie data, you can just go to the cluster, click the ... menu, and load the sample data. If everything has been set up correctly, the sample_mflix database and movies collections will have the plot embeddings created on the "plot" field and added to a new "plot_embeddings" field.

Now, to query your data with JavaScript

Once the documents in your collection have their embeddings generated, you can perform a query. But because this is using vector search, your query needs

MongoDB Developer

use that embedding inside of your application.

```
const axios = require('axios');
const MongoClient = require('mongodb').MongoClient;

async function getEmbedding(query) {
    // Define the OpenAI API url and key.
    const url = 'https://api.openai.com/v1/embeddings';
    const openai_key = 'your_openai_key'; // Replace with your

    // Call OpenAI API to get the embeddings.
    let response = await axios.post(url, {
        input: query,
        model: "text-embedding-ada-002"
    }, {
        headers: {
            'Authorization': `Bearer ${openai_key}`,
            'Content-Type': 'application/json'
        }
    });

    if(response.status === 200) {
        return response.data.data[0].embedding;
    } else {
        throw new Error(`Failed to get embedding. Status code:
    }
}

async function findSimilarDocuments(embedding) {
    const url = 'your_mongodb_url'; // Replace with your MongoD
    const client = new MongoClient(url);

    try {
        await client.connect();

        const db = client.db('<DB_NAME>'); // Replace with your
        const collection = db.collection('<COLLECTION_NAME>');

        // Query for similar documents.
        const documents = await collection.aggregate([
  {"$vectorSearch": {
    "queryVector": embedding,
    "path": "plot_embedding",
    "numCandidates": 100,
    "limit": 5,
    "index": "moviesPlotIndex",
      }}
]).toArray();

```

```
        return documents;
    } finally {
        await client.close();
    }
}

async function main() {
    const query = 'your_query'; // Replace with your query.

    try {
        const embedding = await getEmbedding(query);
        const documents = await findSimilarDocuments(embedding)

        console.log(documents);
    } catch(err) {
        console.error(err);
    }
}

main();
```

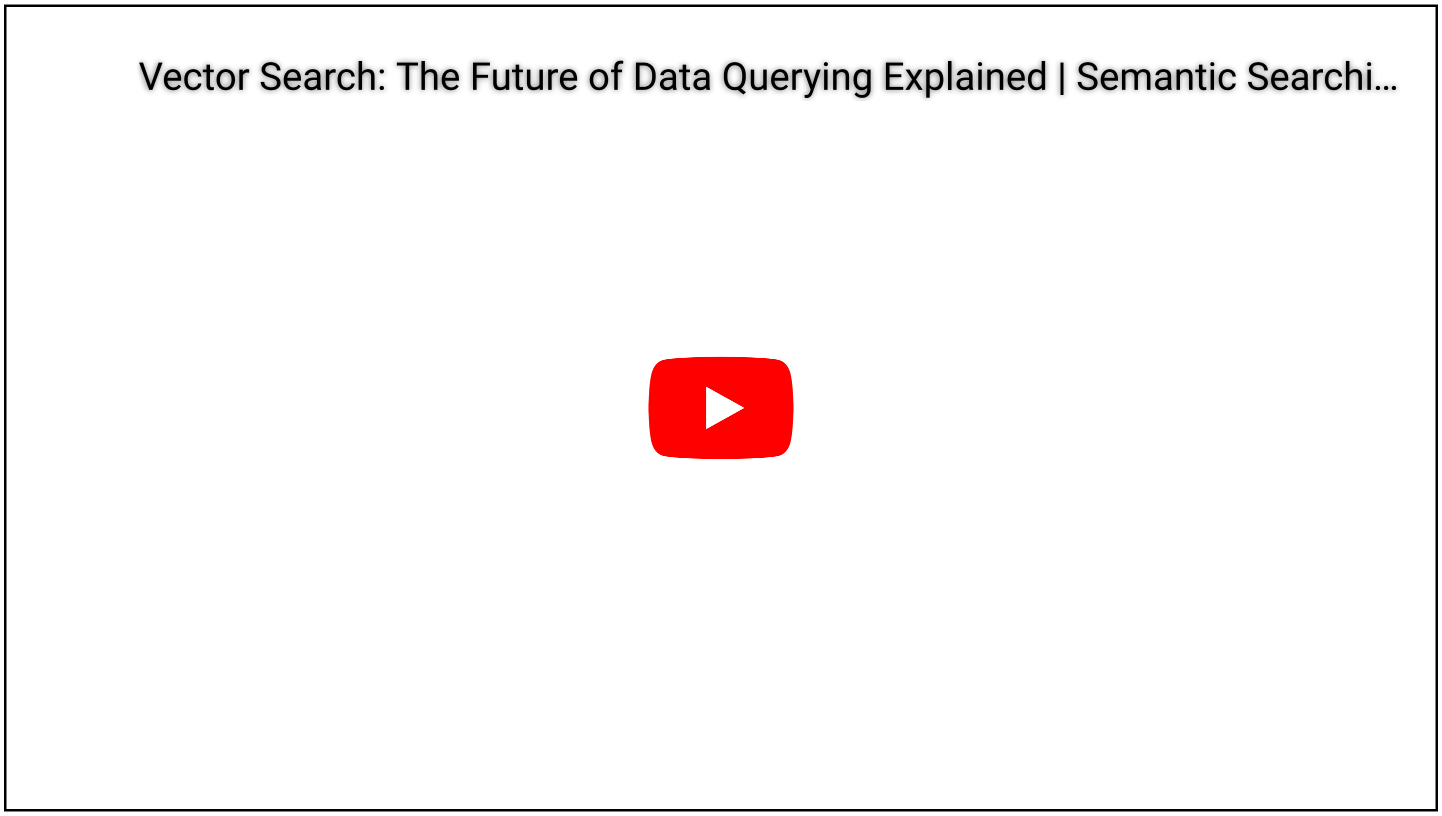
This script first transforms your query into an embedding using the OpenAI API, and then queries your MongoDB cluster for documents with similar embeddings.

> Support for the '$vectorSearch' aggregation pipeline stage is available with MongoDB Atlas 6.0.11 and 7.0.2.

Remember to replace 'your_openai_key', 'your_mongodb_url', 'your_query', '<DB_NAME>', and '<COLLECTION_NAME>' with your actual OpenAI key, MongoDB URL, query, database name, and collection name, respectively.

And that's it! You've successfully set up a MongoDB Atlas cluster and Atlas trigger which calls the OpenAI API to embed documents when they get inserted into the cluster, and you've performed a vector search query.

> If you prefer learning by watching, check out the video version of this article!


Vector Search: The Future of Data Querying Explained | Semantic Searchi...

Rate this tutorial ☆ ☆ ☆ ☆ ☆

Related

TUTORIAL

Visualize MongoDB Atlas Database Audit Logs

Jun 12, 2023 | 11 min read

TUTORIAL

RAG with Atlas Vector Search, LangChain, and OpenAI

May 09, 2024 | 10 min read

TUTORIAL

Build a Cocktail API with Beanie and MongoDB

Apr 02, 2024 | 6 min read

TUTORIAL

Atlas Cluster Automation Using Scheduled Triggers

Jun 10, 2024 | 11 min read

Request a Tutorial

English

About

Careers

Investor Relations

Legal Notices

Privacy Notices

Security Information

Trust Center

Support

Contact Us

Customer Portal

Atlas Status

Customer Support

Manage Cookies

Social

GitHub

Stack Overflow

LinkedIn

YouTube

X

Twitch

Facebook