Tutorial: To Find It Fast, Look No Further

ATLAS SEARCH WORKSHOP

Overview

Giving your users the ability to find exactly what they are looking for in your application is critical for a fantastic user experience. Atlas Search makes that easier than ever. In this hands-on coding lab, we will use the $search operator in the MongoDB aggregation pipeline in an application to build fine-grained searches across text, numerics, and geospatial data. Among the many topics we'll cover are:

- Fuzzy Matching
- Autocomplete
- Highlighting
- Facets
- Scoring

Along the way, we will also discuss performance tips and caveats that you should keep in mind as you build out your Search queries. From finding your favorite movie to finding your favorite restaurant, look no further!

Session Outline

Part 1 - Concepts and Setup

- Workshop Overview & What we will learn
- Sneak-Peak of Final Application
- Hands-on Tutorial
    - Stand up infrastructure (MongoDB Atlas, MongoDB Compass)
- Hands-on Tutorial
    - Spinning up an Atlas Cluster and Loading Data
- Introduction to Aggregations in MongoDB

Part 2- Atlas Search Basics

- Hands-on Tutorial
    - Create Search Index
    - Create your first Search Query
    - Create the API

Part 3- Building the App and Advanced Search Queries

- Hook the API into our Front End application

- Host App Services application (optional)
- Playing with Search Functionalities (Hands-On):
    - Text
    - Fuzzy
    - Phrase
    - Compound
    - Boost score modifier
- Demo - What's Cooking (Advanced Search)
    - GeoJSON (near & geoWithin)
    - Range
    - Filter
    - Compound
    - Function Score
    - Score Modifier

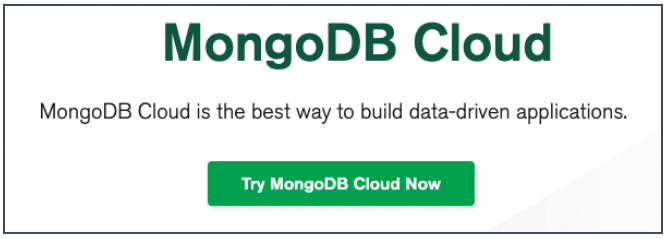
Step 1: Stand up the Infrastructure

Because the infrastructure takes a few minutes to deploy, let's kick off that process and we'll use the time while the deployment is running to introduce the lab. So let's begin by getting our database-as-a-service (Atlas) setup.

MongoDB Atlas (DBaaS)

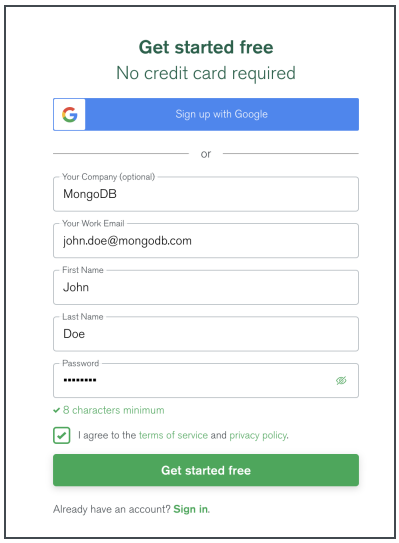
Atlas is a service that provides fully managed MongoDB deployments. If you have an existing MongoDB Cloud account, feel free to use it for this tutorial. The instructions in this section are meant to walk you through the process, starting from scratch.

Navigate to MongoDB Atlas and select 'Try MongoDB Cloud Now', or click on' Login' if you already have an account.

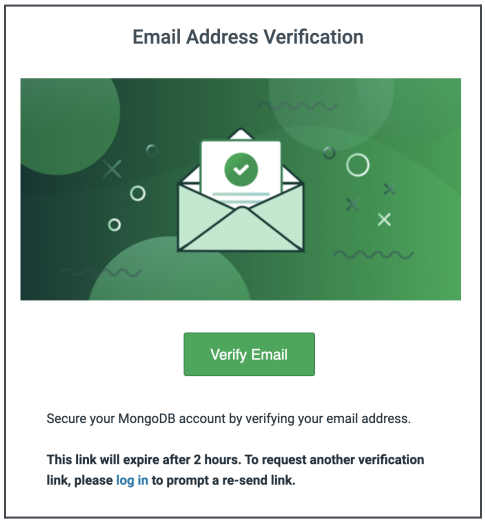


Sign-Up Instructions

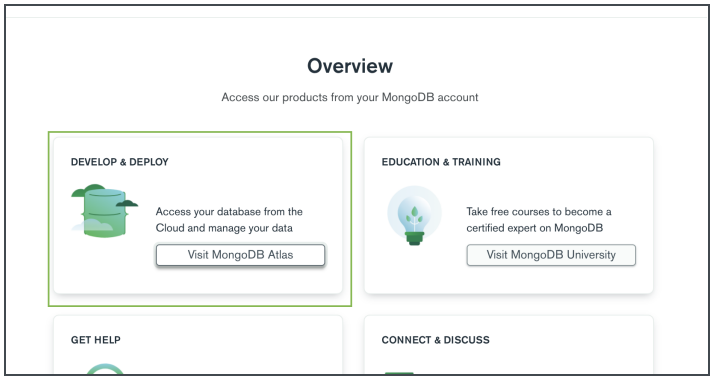
If you already have an Atlas account, skip this part and go to "Creating an Organization and Project". In order to sign-up, fill in your details as shown below and click on 'Get started free'. Alternatively, you can click on 'Sign up with Google' and use your Google account.



You will then receive an email in your inbox. Navigate to the email and click on 'Verify Email'.



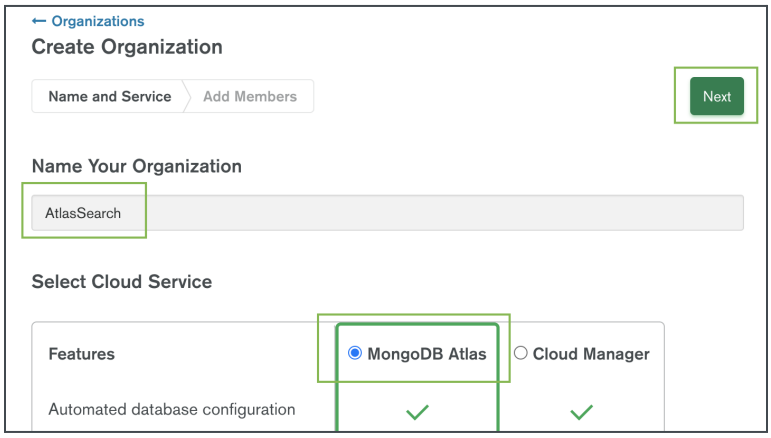
Once you're logged in, you should see the following options. Click on 'Visit MongoDB Atlas' under 'Develop and Deploy'.



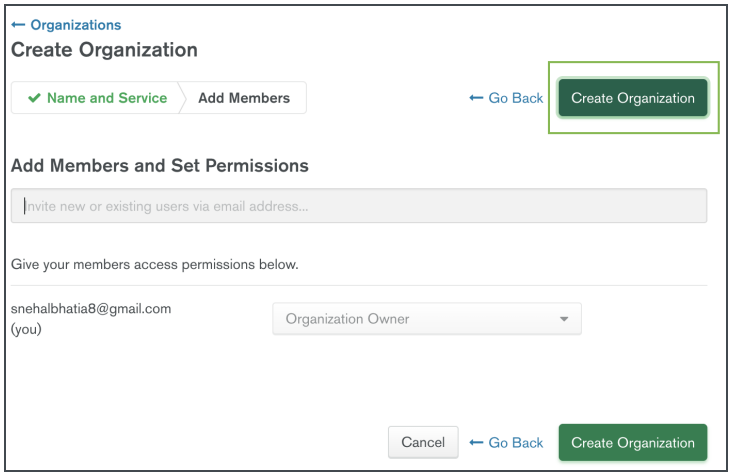
Creating an Organization and Project

Give your organization a name under the 'Name your Organization' field. Note that it can also be changed later.

Select 'MongoDB Atlas' from the two options under 'Select Cloud Service' and click on 'Next'.

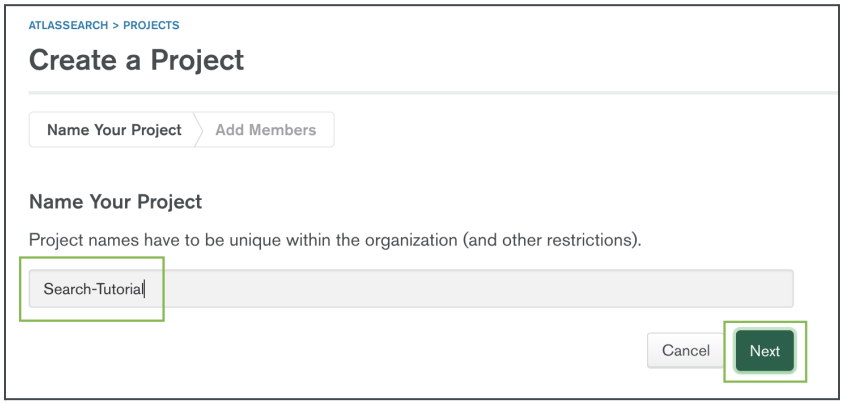


You will be then asked to add members to your organization and set their permissions. You will be granted access with the role of 'Organization Owner' by default. Leave the default settings selected and click on 'Create Organization'.

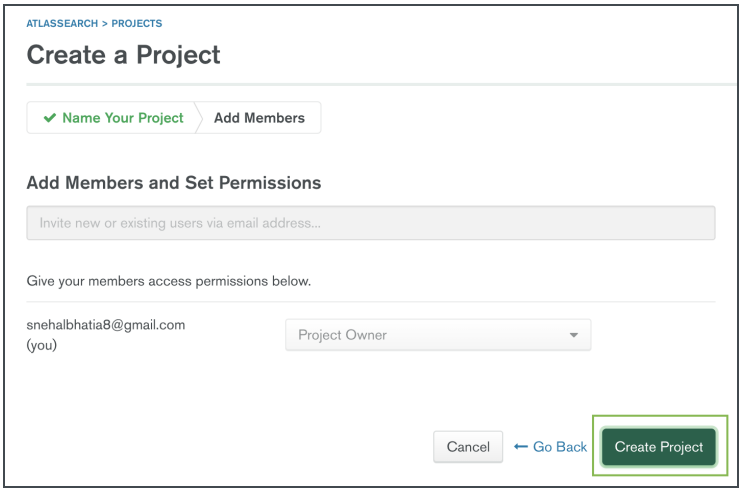


You will then be navigated to the 'Projects' page for the Organization you just created. To create a project, click on 'New Project', on the top right corner of the screen.

Give your project a name by typing it under the 'Name Your Project' field (however, remember that this cannot be changed later!) and click on 'Next'.

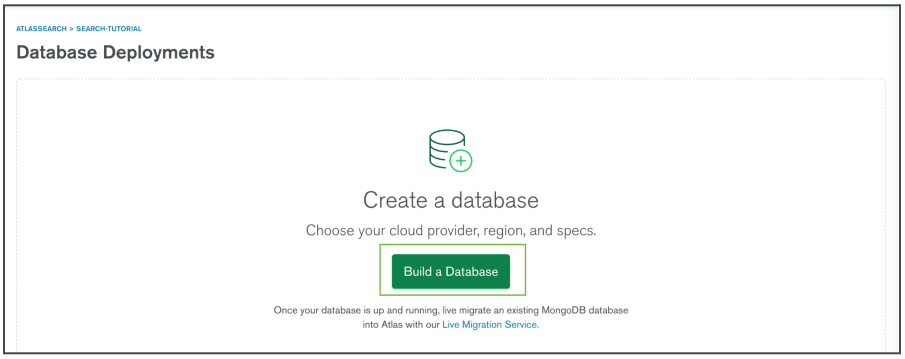


You will then be navigated to the 'Add Members' tab for the project you just created. You will automatically be added as a member to this project with the permission of 'Project Owner'. Leave the default settings selected and click on 'Create Project'.

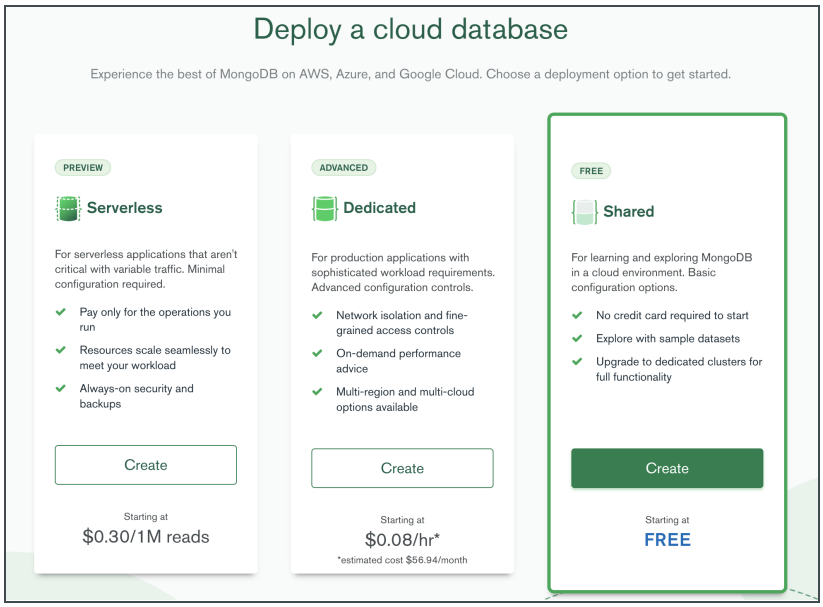


Deploying a Database Cluster

Once you are inside the project you just created, click on ‘Build a Database’.

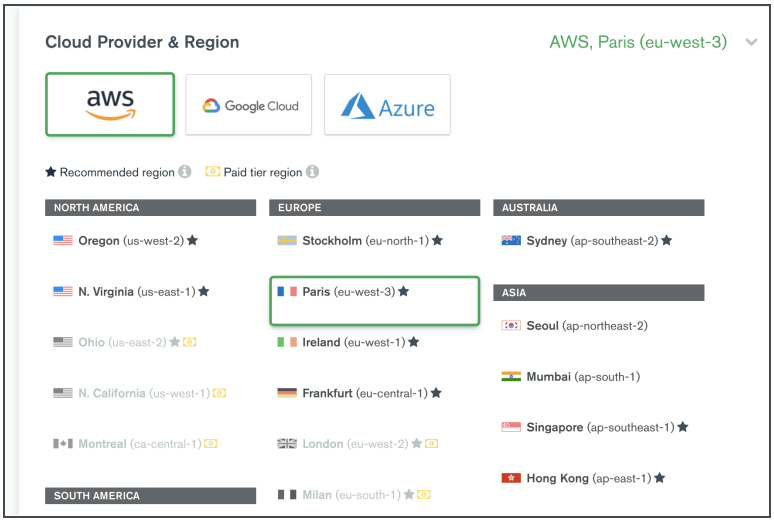


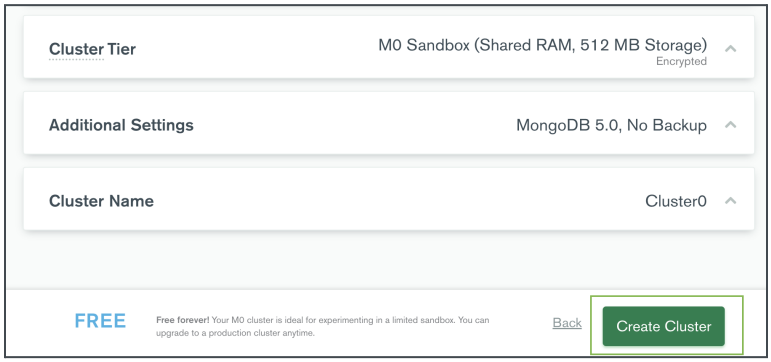
It should take you to the setup wizard. For this tutorial, we will choose the ‘Shared’ cluster type, which is the free tier of MongoDB Atlas. Once selected, click on ‘Create’.



This should then bring you to the setup wizard. Select the cloud provider and region of your choice, and leave the default settings for the rest. If you prefer, you can scroll all the way down and change the cluster name to your liking (remember, this cannot be changed later).

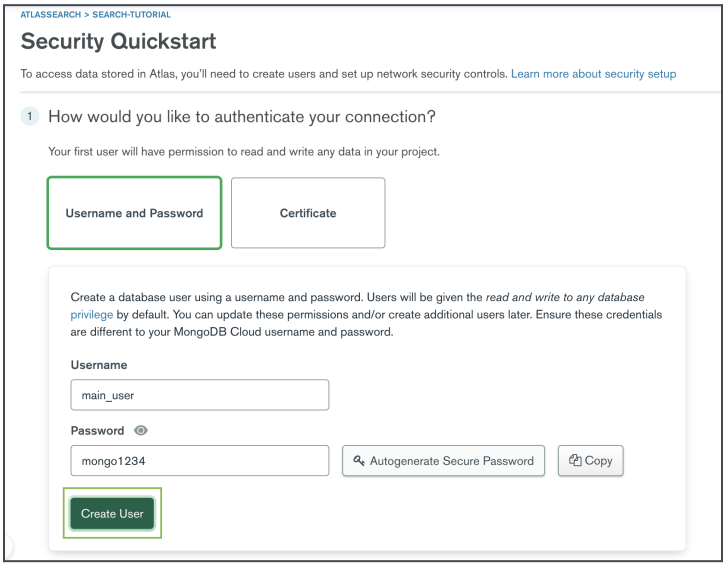
Then, click on ‘Create Cluster’.





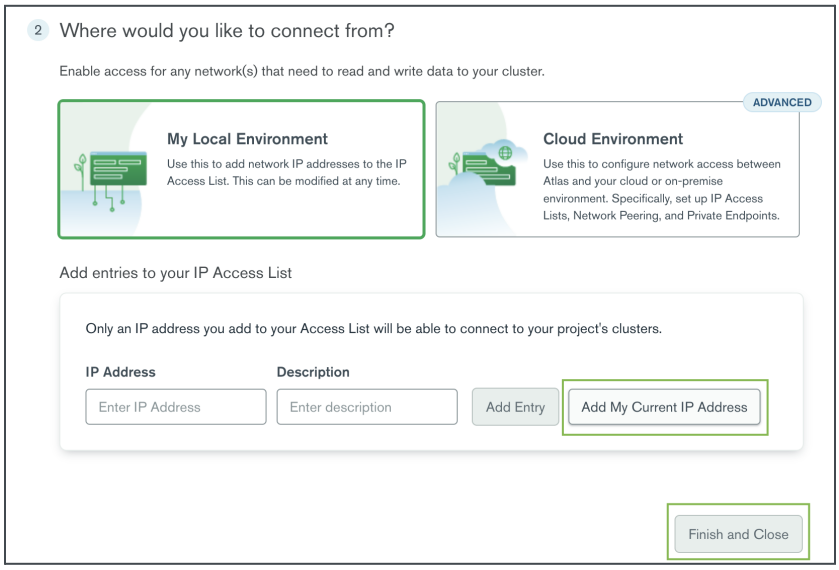
You will then be navigated to the Security Quickstart wizard.

First, under "How would you like to authenticate your connection?", select the 'Username and Password' option. Enter the Username and Password of your choice (remember to take a note of this!), and click on 'Create User'.

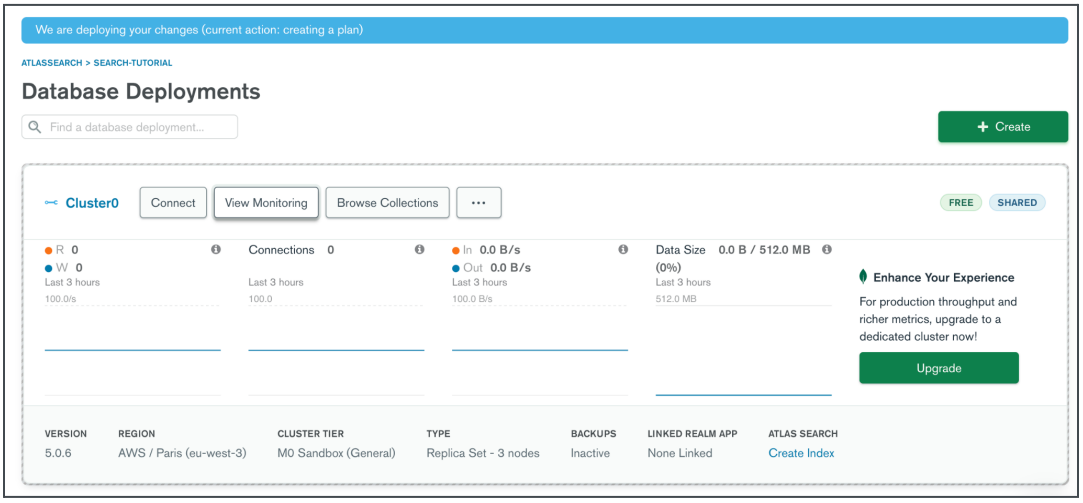


Once this is done, you can then proceed to setting up network access for your database cluster under “Where would you like to connect from?”. Select the option ‘My Local Environment’ and click on ‘Add my Current IP Address’.
For the purposes of this tutorial only, you may also add ‘0.0.0.0/0’ under ‘IP Address’ to allow access to your database from anywhere (remember that this is just for simplicity and is not recommended for the database environments of your actual projects).

Once done, click on ‘Finish and Close’.



You should then see your Database Cluster being deployed. This should take around 3-5 minutes.



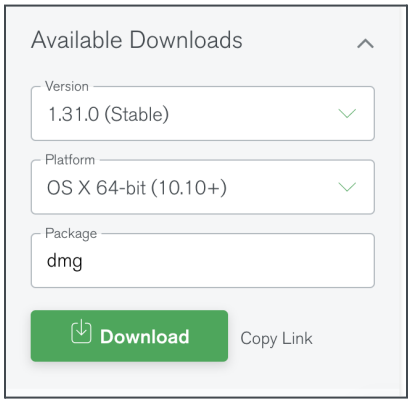
Downloading MongoDB Compass: The GUI for MongoDB [Optional]

MongoDB Compass is a GUI tool which allows you to easily explore and manipulate your database. It is intuitive, flexible and provides features such as detailed schema visualizations, real-time performance metrics, sophisticated querying abilities, and much more.

The use of Compass is optional for this workshop, but it is recommended.

To get started, go to: https://www.mongodb.com/try/download/compass?tck=docs_compass

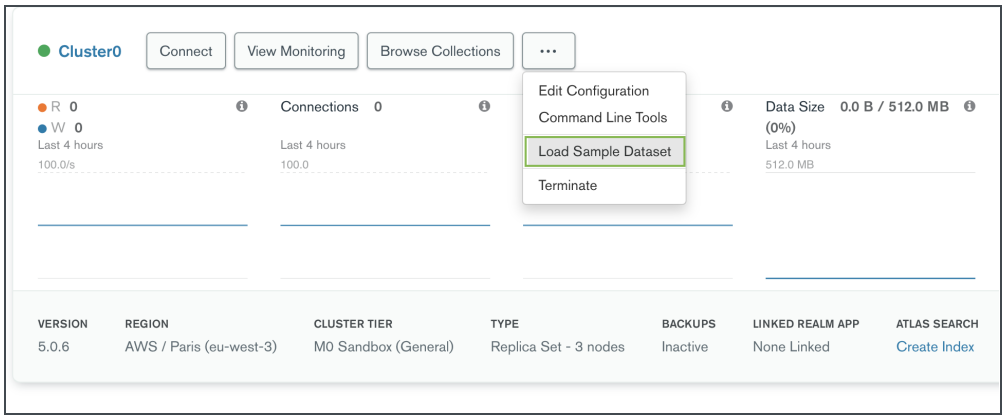
Select the version and platform (OS) that corresponds to your system, and click on 'Download' (note that the website identifies your system specifications automatically, so you can leave the default options selected).



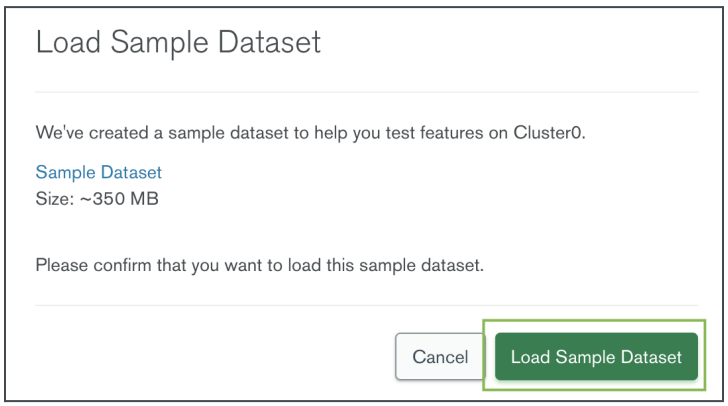
Click on the downloaded file, and follow the setup wizard. For more details on downloading and installing Compass, visit our documentation: https://www.mongodb.com/docs/compass/current/install/ .

Step 2: Load Sample Data

Atlas has sample data that we're going to leverage for this workshop. Click the  button and select 'Load Sample Dataset' from the drop down:

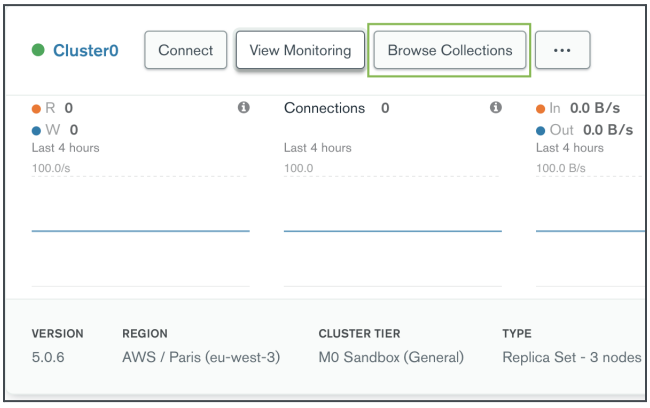


Click on the 'Load Sample Dataset' button when the pop-up window appears. It should take less than 5 minutes to load the dataset into your cluster.



Step 3: Review Movies Collection

Once the data is downloaded, let's have a look at the collections, which you can do by clicking on the 'Browse Collections' button:

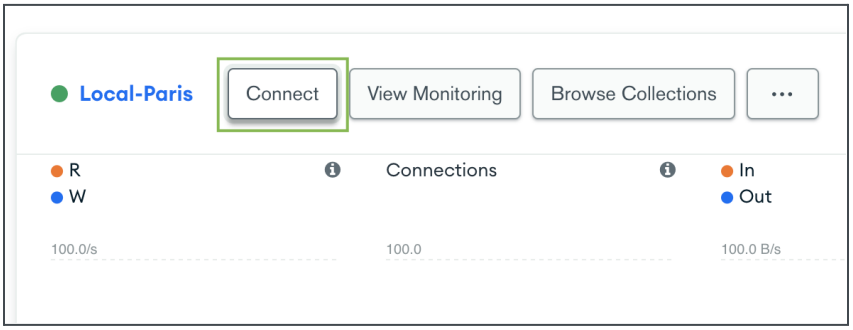


For this workshop, we will be using the 'movies' collection in the 'sample_mflix' database. This collection has over 23,000 movies with a variety of text, date, and numeric fields that we can query against.

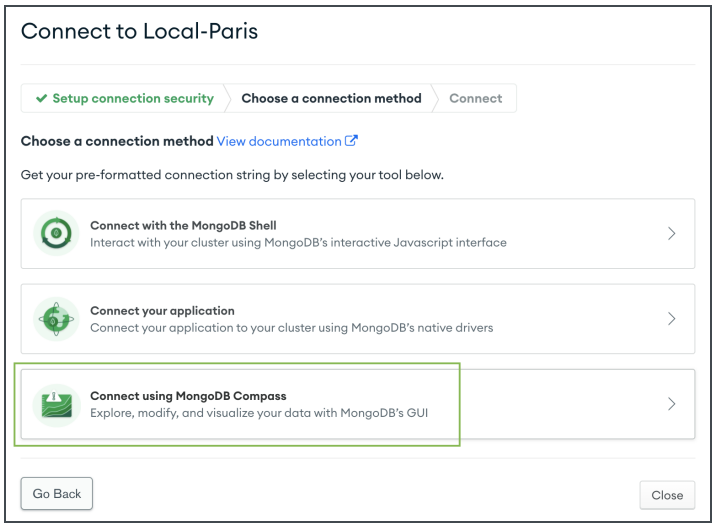
Connecting to MongoDB Compass [Optional]

If you have MongoDB Compass downloaded, you can use it to explore and manipulate your data instead.

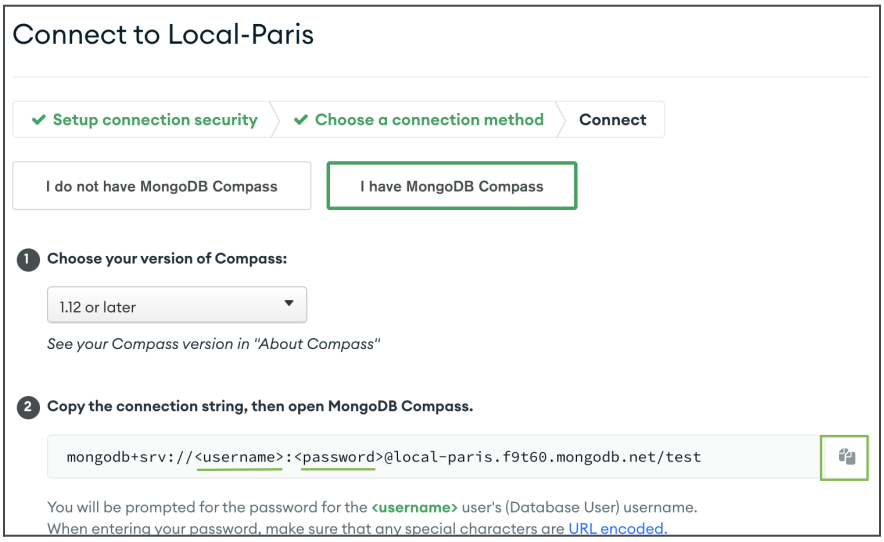
Before we can do that, we need to connect to Compass. Head over to your cluster and click on 'Connect'.



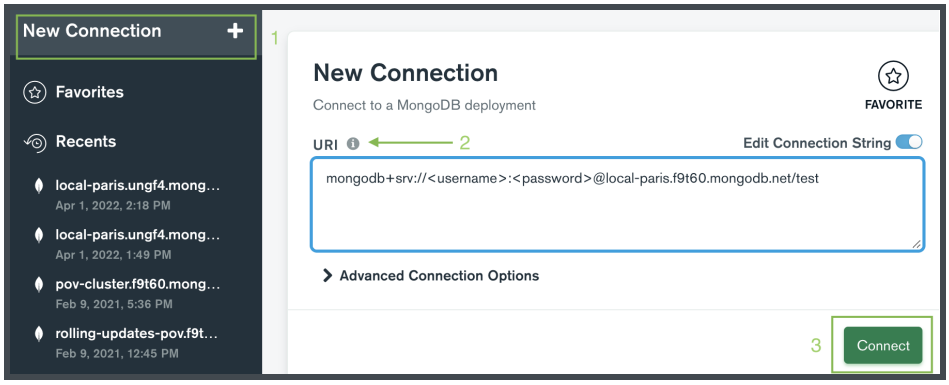
Select 'Connect using MongoDB Compass' from the list.



Select ‘I have MongoDB Compass’, and click on the  icon in step 2. Make sure to replace <username> and <password> in the connection string with the username and password that you had set up while configuring User Security.



Then, open MongoDB Compass, and click on ‘New Connection’. Paste your connection string in the ‘URI’ field and click on ‘Connect’.



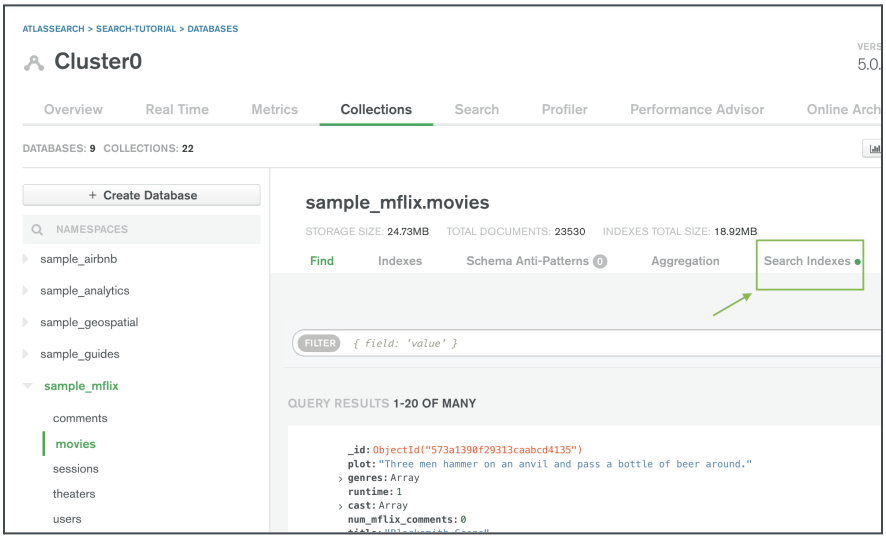
Once connected, you should be able to see your databases and collections, and start exploring them.

For additional information on MongoDB Compass and how to best leverage it, refer to the MongoDB Documentation: https://www.mongodb.com/docs/compass/current/

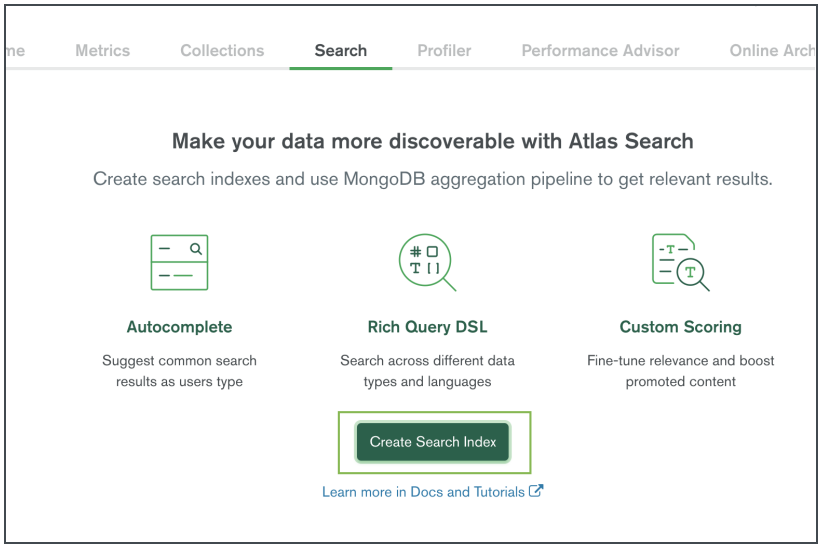
Step 4: Build a Search Index

Before we can build full-text search capabilities, a Search Index must be created.

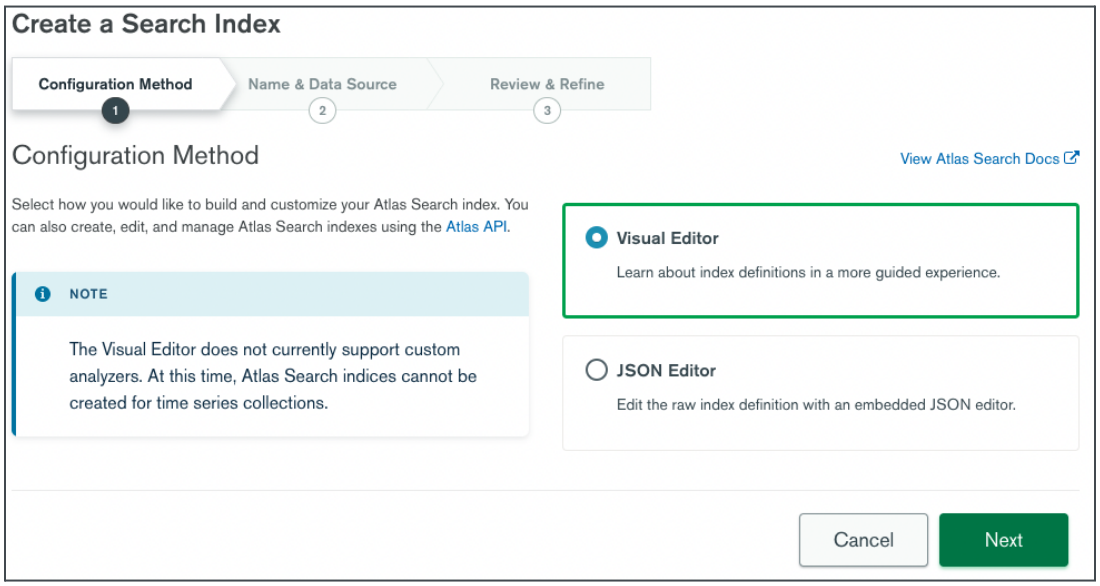
While viewing the movies collection, click on the 'Search Indexes' tab:



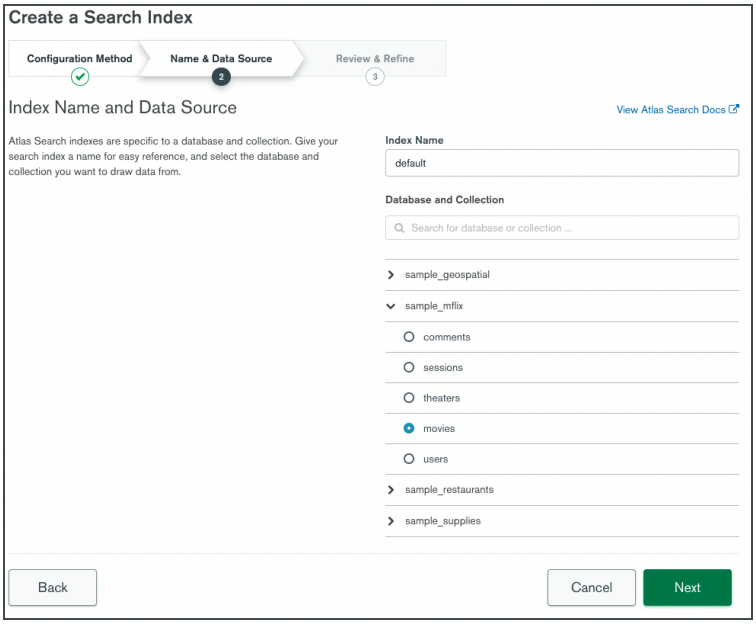
Then, click on the 'Create Search Index' button:



Atlas Search provides a powerful Visual Editor to guide you through the experience, and we're going to use that today. Select the 'Visual Editor' option as your Configuration Method, and click on the 'Next' button:



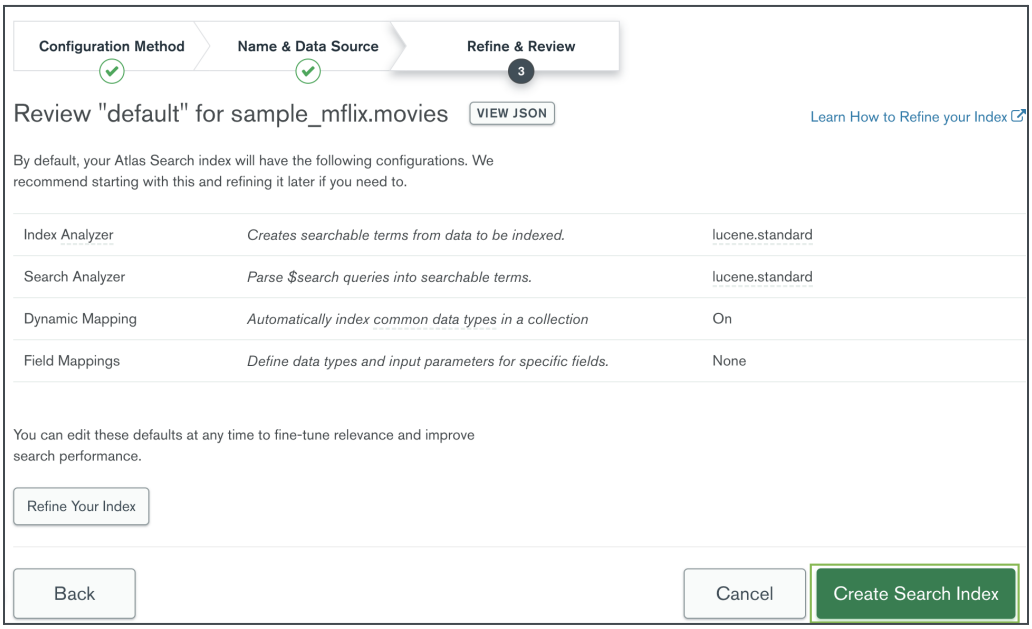
You'll be now taken to Step 2 of creating a Search Index, which is under the 'Name & Data Source' tab. Give your index a name and confirm the correct collection (movies) to index is selected. You can leave the default settings here and click on 'Next'.
By leaving the Index Name as "default", we will not have to specify the index name in our search queries, which is recommended for this tutorial.



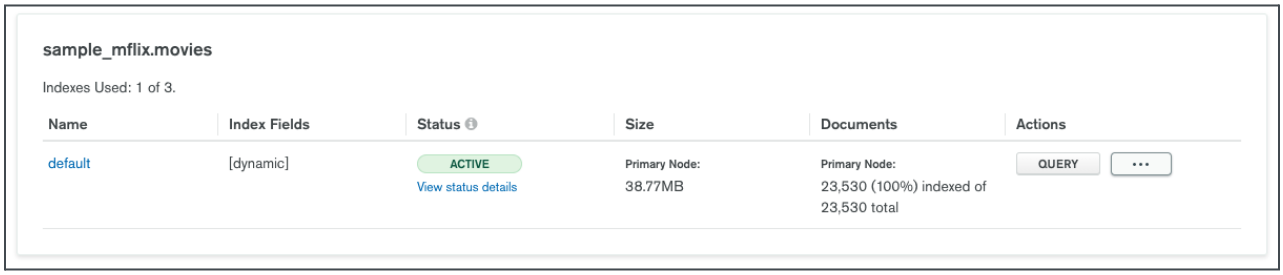
You will then be brought to the third step of creating a Search Index, which is 'Refine and Review'.

Atlas Search is built on the popular Apache Lucene search library. As such, Atlas Search is highly configurable: allowing you to employ different analyzers and map field types. For this first exercise, we are going to stick with all the defaults.

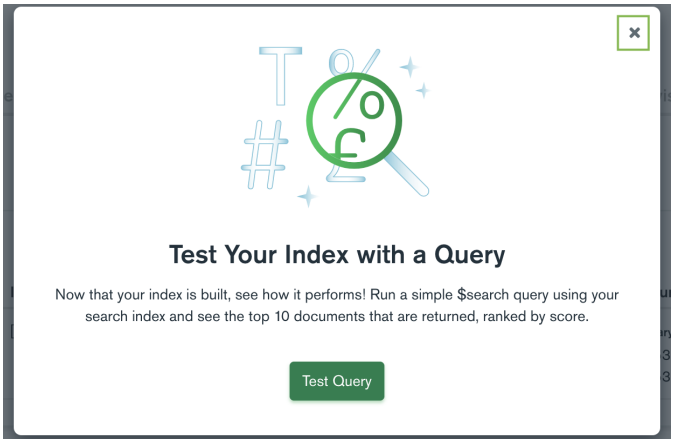
Leave the default settings and click on 'Create Search Index'.



The index build is asynchronous and will start soon. Once complete, you will see that the index status turns to ACTIVE, as shown below:



Once the Search Index is created, you have the option to test your search index with a query. We will skip this for now and instead do this during the following exercises. Click on ✖ to close the pop-up.



NOW WE HAVE AN ATLAS CLUSTER, SAMPLE DATA AND SEARCH INDEX!! *And that's all you need to do to start taking advantage of Apache Lucene on top of your MongoDB Atlas data!* 👊

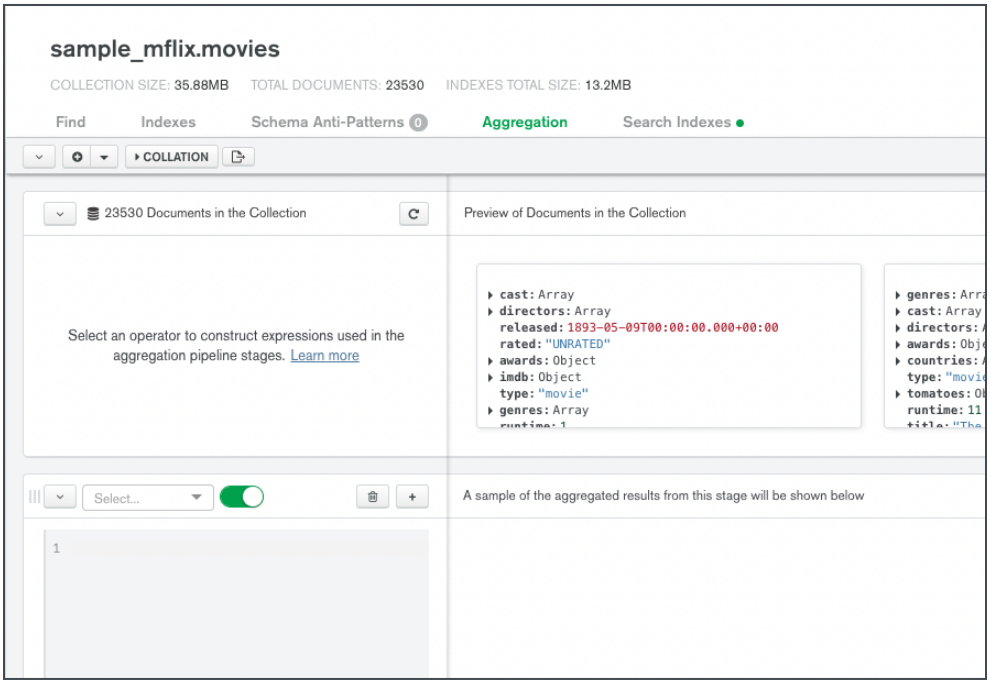
Step 5. Build Aggregation Pipeline Using $search

Atlas Search queries take the form of an aggregation pipeline stage using the $search pipeline stage.

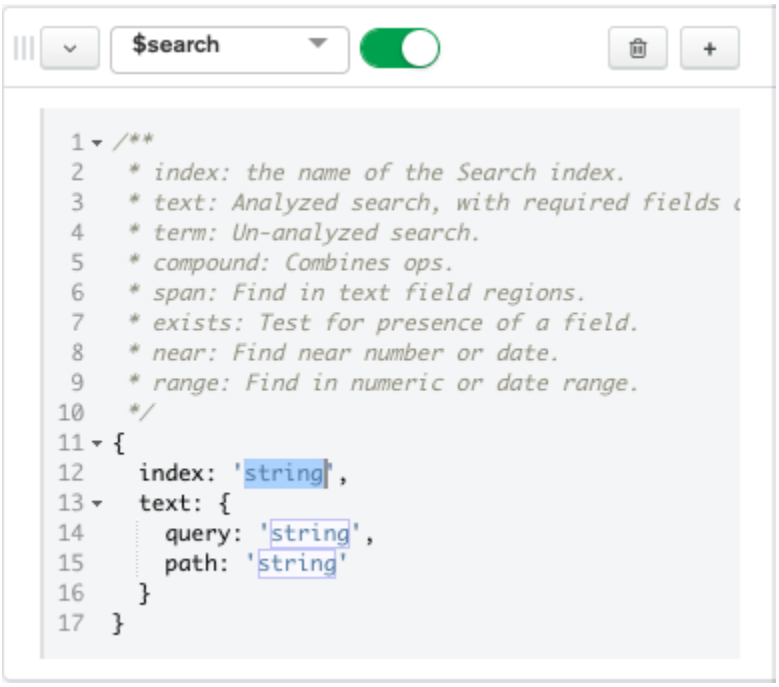
To build our pipeline, let's use the Aggregation Pipeline builder here in the Collections tab.

If you haven't used the aggregation pipeline builder yet, we highly recommend it. This simple UI makes building and troubleshooting your aggregation queries a snap.

Start by making sure you are in the **sample_mflix.movies** collection. Then find your way to the **Aggregation** tab:



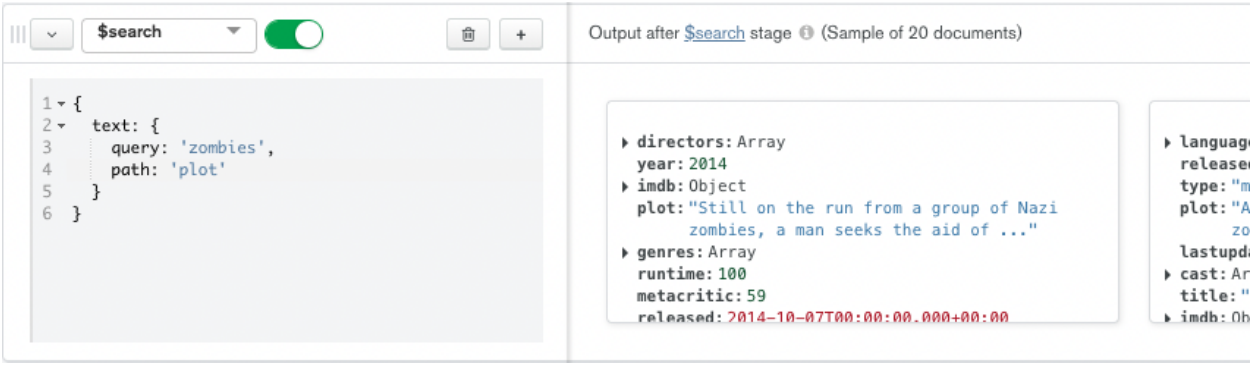
The $search stage is always the first stage in an Atlas Search aggregation, so select **$search** in the dropdown:



Let's use our default index to query for **'zombies'** in the '**plot**' field. Replace the contents of the dialog with the following expression:

```
{
  text: {
    query: 'zombies',
    path: 'plot'
  }
}
```

You can see the resulting movie documents from the query in the right preview panel. Previewing the results is very helpful to make sure we are building out the aggregation correctly.



Now let's add another stage, $project, to get back only the fields we will use in our movie search application. We also use the $meta operator to surface each document's searchScore. This score signifies how well this movie matches our query term "zombies".

Click ADD STAGE and select **$project** in the dropdown. Then replace the contents of the dialog with the following expression:

```
{
  title:1,
  year:1,
  poster:1,
  plot:1,
  score: {$meta: 'searchScore'}
}
```

And as you can see, the movie documents are returned to us with the score in descending order -which means we get the best matches first. Dead Snow 2 is our top result.

Also, we want to point out that, right now, we are searching just in the plot field, but if we wanted, we can search across many different fields simply by combining those field names in an array... so let's extend the path in the $search stage to include "**title**" and "**fullplot**":

```
{
  text: {
    query: 'zombies',
    path: ['plot', 'fullplot', 'title']
  }
}
```

And now that change has affected the scoring of our movie documents. Cockneys vs Zombies comes out on top with a score of 4.3.

Let's say we're on the search for Indiana Jones, I'll get Indiana Jones and the Last Crusade. BUT I am not a good typist, so if I put Indiana Ones - and I fail to get the results I want... Atlas Search offers "fuzzy matching" to forgive us for our typos and poor spelling. You only need to add a fuzzy field to the text operator in my $search stage.

```
{
  text: {
    query: 'Indiana Ones',
    path: ['plot', 'fullplot', 'title'],
    fuzzy:{}
  }
}
```

And now we are back to Harrison Ford movies.

As easy as it is to see the scoring metadata, it is just as simple to surface highlights. Adding the highlight option to the Atlas $search stage will show your queried search term in its proper context alongside the adjacent text. To enable highlighting for the 'fullplot' field, add

```
highlight:{path:'fullplot'}
```

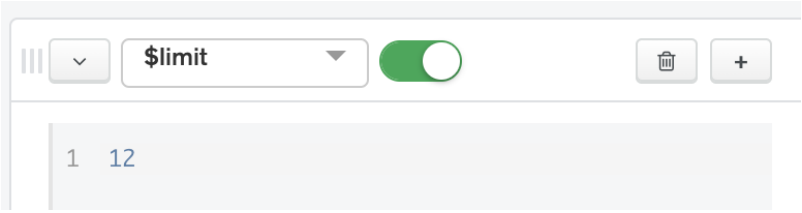
to the $search stage, as well as

```
'highlights': {
       '$meta': 'searchHighlights'
     }
```
to $project below the score line.

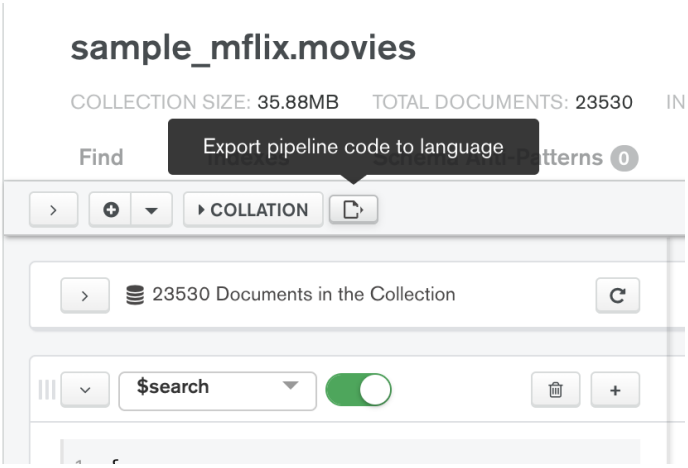
$limit

Let's finish off our $search aggregation pipeline with $limit. $limit is extremely important in Search because speed is important. $limit: 12 will bring the 12 most relevant movie documents to your search query.

Pro tip!! In fact, $limit is so important in Search that I can move the $limit stage before the $project stage. Since I know that my movie documents are already sorted by score, this will help performance by not executing $project on all of the 25,000 movie documents.



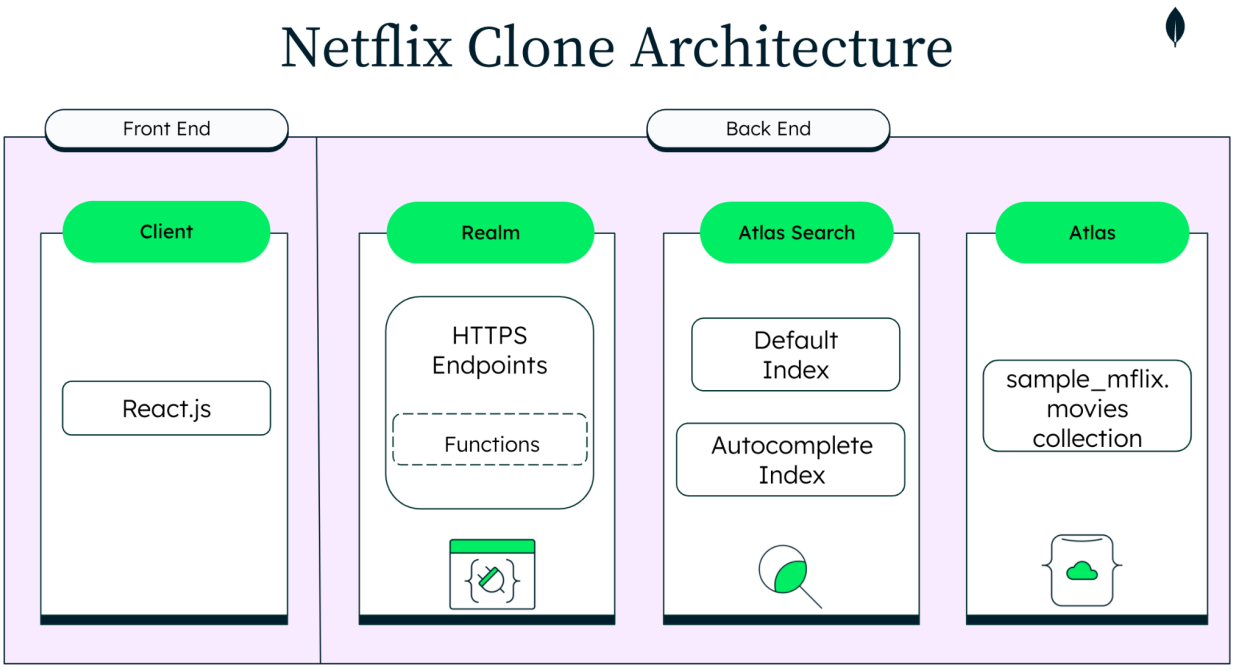
Finally, let's return to the top of the Pipeline Builder and click Export Pipeline Code to Language:



We will pick NODE as our language. Then copy and paste it in a text for safe keeping. Here it is as well:

```
[
  {
    '$search': {
      'text': {
        'query': 'Harrry Pottter',
        'path': [
          'title', 'plot', 'fullplot'
        ],
        'fuzzy': {
          'maxEdits': 1
        }
      },
      highlight:{path:'fullplot'}
    }
  }, {
    '$limit': 12
  }, {
    '$project': {
      'title': 1,
      'year': 1,
      'plot': 1,
      'fullplot':1,
      'released':1,
      'poster': 1,
      'imdb':1,
      'score': {
        '$meta': 'searchScore'
      },
      'highlights': {
        '$meta': 'searchHighlights'
      }
    }
  }
]
```

This small snippet of code - A mere 3-stage aggregation- powers our movie search application! Now that we have the heart of our movie search engine, I will simply create a RESTful API to expose this data - and for that, we will use MongoDB App Services here in Atlas.

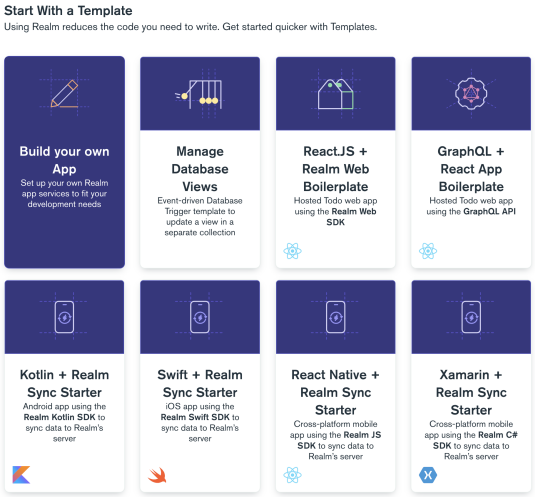


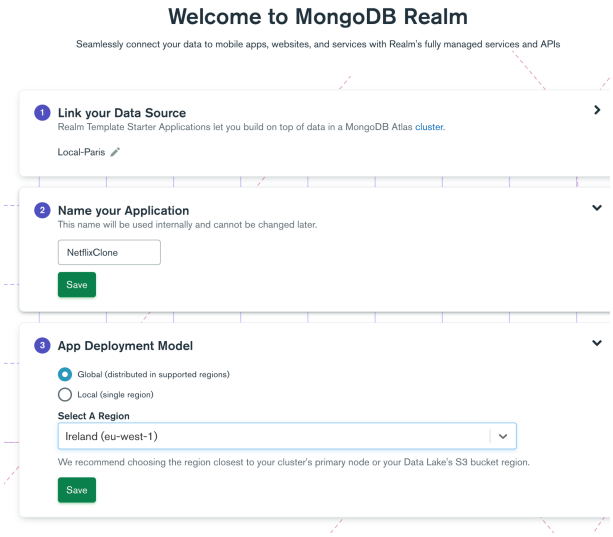
Step 6. Create an Application in Atlas App Services and Microservice

App Services is MongoDB's serverless backend. It offers many things, including serverless functions, triggers, GraphQL, and HTTP Services. To take advantage of any of them, we first need to create a App Services application.

To make it easy to bounce between Atlas and App Services, right click the App Services tab and select **Open Link in New Tab**:

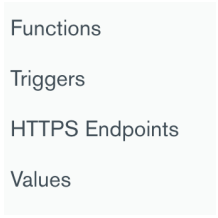
Use the basic **Build your own App** template, click Next. Name your application **NetflixClone**. After ensuring it is linked to the correct cluster, click the **Create an App Services** button.





You'll see that App Services provides a handful of Application Guides that you're encouraged to visit at a later time. For now, click **Close Guides**.

On the left menu, select **HTTPS Endpoints**:

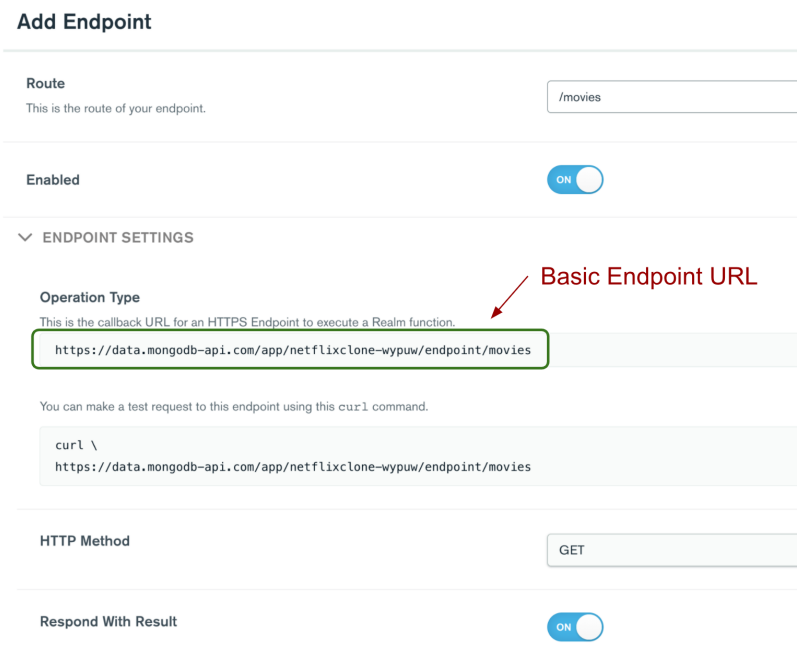


And click the **Add an Endpoint** button.

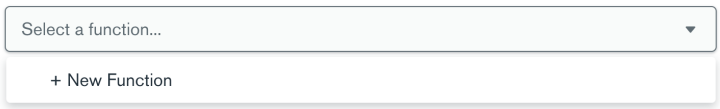
Select the HTTP service, and name it movies.

1. For **Route,** type **/movies.**
2. Set your **Operation Type** to **GET**
3. Enable **Respond with Result.**

Make note of your **BASIC ENDPOINT URL.** You will be using this in your application.



This endpoint will be calling a **New Function:**



Name the function **getMovies** and replace the code in the **Function Editor** with following code from **snippets 3_basicMoviesEndpoint.txt**:

```
exports = async function({ query, headers, body}, response) {

  // GET A HANDLE TO THE MOVIES COLLECTION
  const moviesCollection =
context.services.get("mongodb-atlas").db("sample_mflix").collection("movies");

  // GET SEARCHTERM FROM QUERY PARAMETER. IF NONE, RETURN EMPTY ARRAY
  let searchTerm = query.searchTerm;
  if (!query.searchTerm || searchTerm ===""){
    return [];
  }

  const searchAggregation =[];

  const results = await moviesCollection.aggregate(searchAggregation).toArray();
return results,


};
```

In the above code, we use the global context variable to get a handle to the movies collection in the sample_mflix database:

```
3   // GET A HANDLE TO THE MOVIES COLLECTION
4   const moviesCollection = context.services.get("mongodb-atlas").db("sample_mflix").collection("movies");
5
```

Then we get an argument from my function payload query and set it to the **searchTerm** variable.

This will be passed in from our application:

```
6   // GET SEARCHTERM FROM QUERY PARAMETER. IF NONE, RETURN EMPTY ARRAY
7   let searchTerm = query.arg;
```

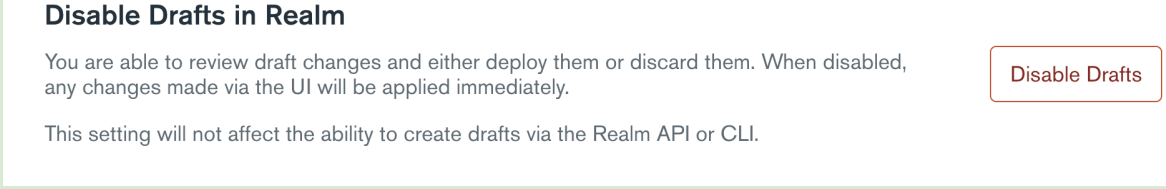
Currently our **searchAggregation** is an empty array.

Then finally we execute the aggregation against the movies collection and set the body of our response to that result.

```
14    const results = await moviesCollection.aggregate(searchAggregation).toArray();
15
16  // pro tip!!
17    response.setHeader("content-type", "application/json");
18    response.setBody(JSON.stringify(results));
19
```

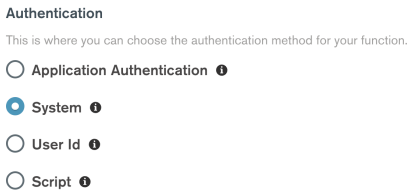
Save your Draft. Then **Review Draft & Deploy.**

Quick Dev Tip!! To save time in the workshop, in **Deployment** settings on the left menu, in the Configuration tab, click **Disable Drafts.**



Now in the **Functions** section, you will now see your **getMovies** function!

We will do one more thing to set up by adjusting the Settings of your new function. App Services supports multiple authentication methods. **SET** the default Authentication method to **System** to keep things simple and be sure to **Save.**



USE SEARCH AGGREGATION:

In the **Function Editor,** paste your newly

At the top of the page, switch to the Function Editor tab, and set your **searchAggregation** variable

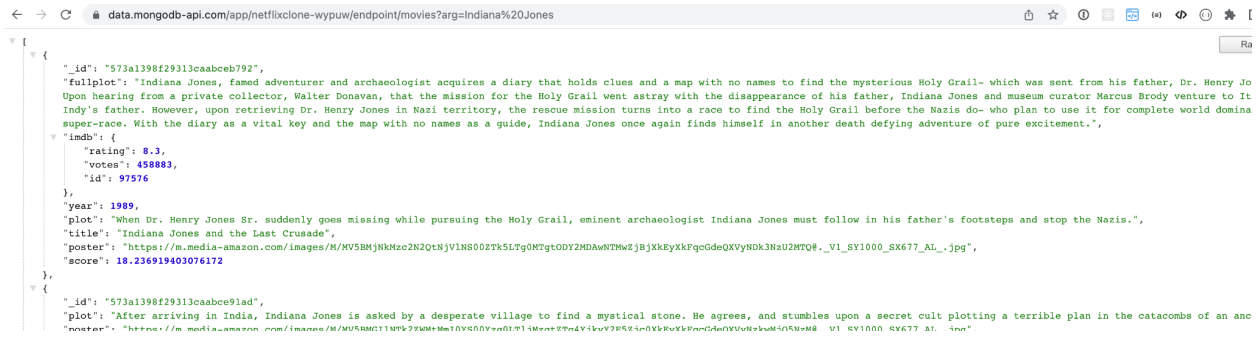
```
12    const searchAggregation =[];
```

to what you built in Compass in **Step 5.** You will find it above or in **4_completeBasicAggregation.** Be sure to replace your query of 'Harrry Pottter' to the variable **searchTerm,** else your function will be quite limited.

Now we can test our new App Services endpoint in the browser. Get the HTTP endpoint you created earlier and append ?arg=Indiana Jones to the end of the url.

https://data.mongodb-api.com/app/netflixclone-hazaz/endpoint/movies?arg=Indiana%20Jones

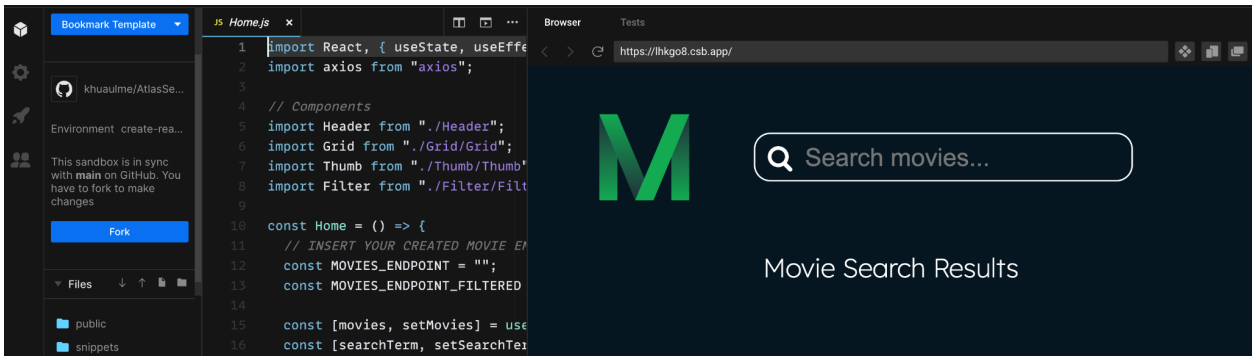
Now paste into your browser to see the results:



If you are coding along and receive an output like this, congratulations! You have successfully created a movie search API that you can call from anywhere!

Step 7. Call the API from the Front End.

Returning to AtlasSearchWorkshop.com and opening up the embedded Code Sandbox in the browser will give you your own forked sandbox of our Netflix Clone application. You can edit away as you please!



Currently, typing anything in the search box will bring you no results. We will be building out this functionality now.

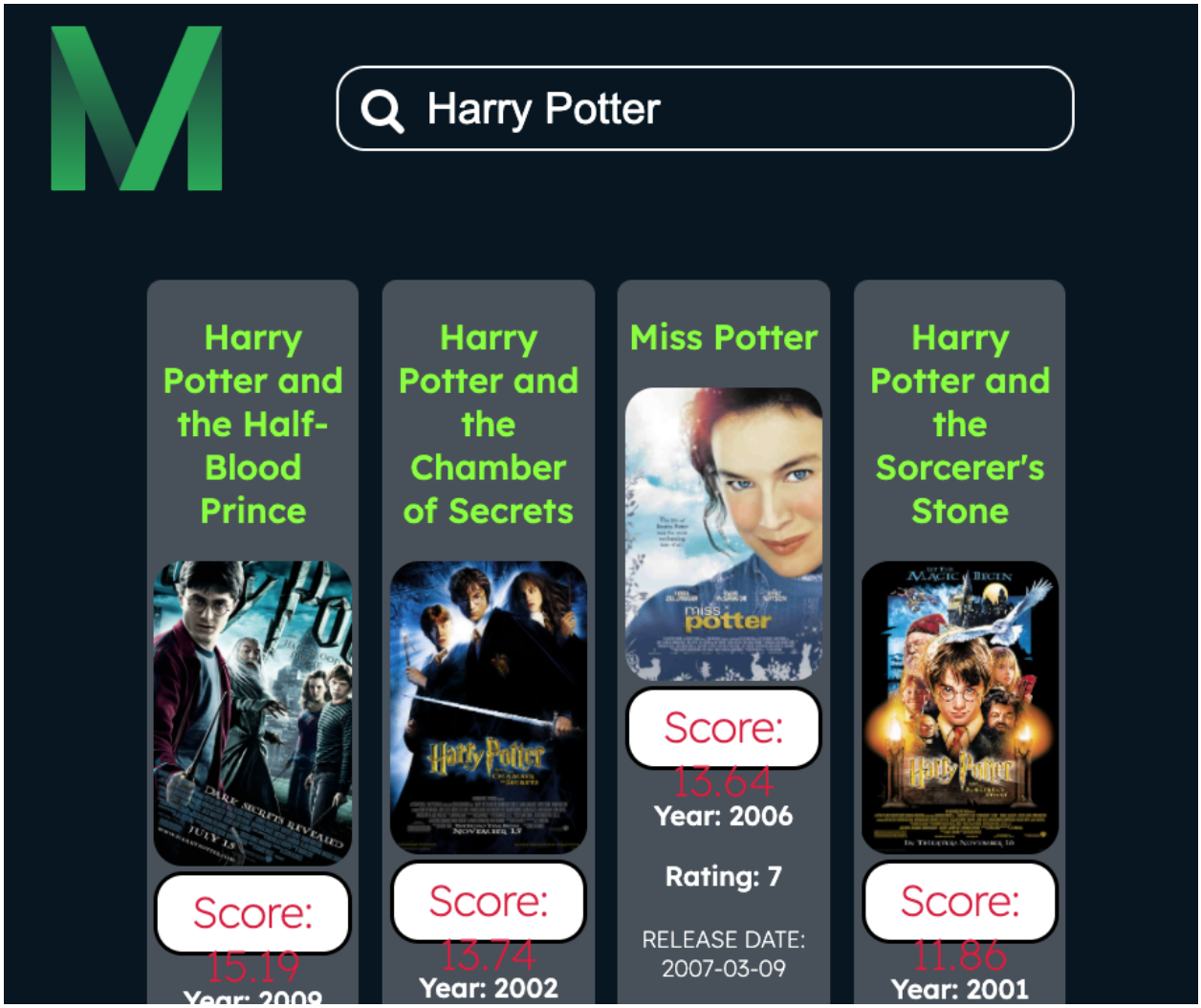
Our main point of integration is in the **Home.js** file found in the **src/components** directory.

```
import React, { useState, useEffect } from "react";

// Components
import Header from "./Header";
import Grid from "./Grid/Grid";
import Thumb from "./Thumb/Thumb";
import Filter from "./Filter/Filter";

const Home = () => {
  const [movies, setMovies] = useState([]);
  const [searchTerm, setSearchTerm] = useState("");
  const [showFilter, setShowFilter] = useState(false);
  const [dateStart, setDateStart] = useState(new Date(1970, 12, 1));
```

Paste your new App Services HTTP endpoint as the value for **MOVIES_ENDPOINT** variable on line 22:

```
21  // INSERT YOUR CREATED MOVIE ENDPOINTS
22  const MOVIES_ENDPOINT = "";
23
```

Now perform the same search for **“Harrry Pottter”**. Et voilà! You can now search movies about ‘zombies’ or ‘sports’ or your favorite Avenger characters! If it’s in the **sample_mflix.movies** collection, Atlas Search will bring it to you!



A few features you see fresh off the bat:

- Relevance-based scores
- Fuzzy Matching
- Highlighting

Step 8. Live Coding: Play with Search Operators

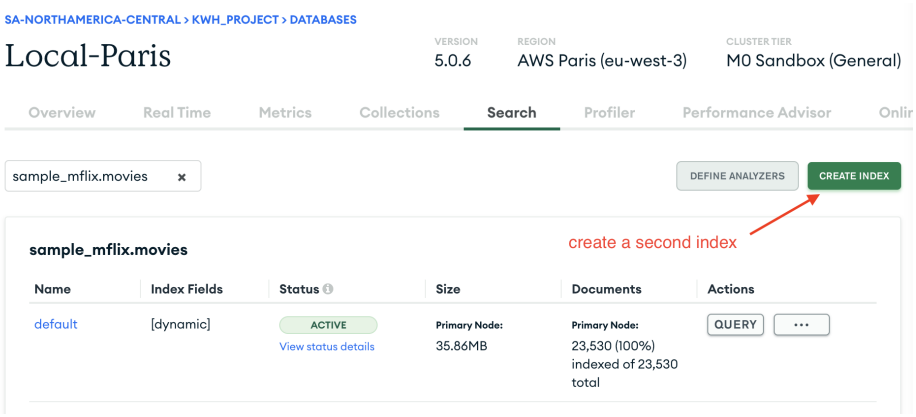
- Text
- Fuzzy
- Phrase
- Compound
- Boost score modifier

Step 9. How to Implement Autocomplete

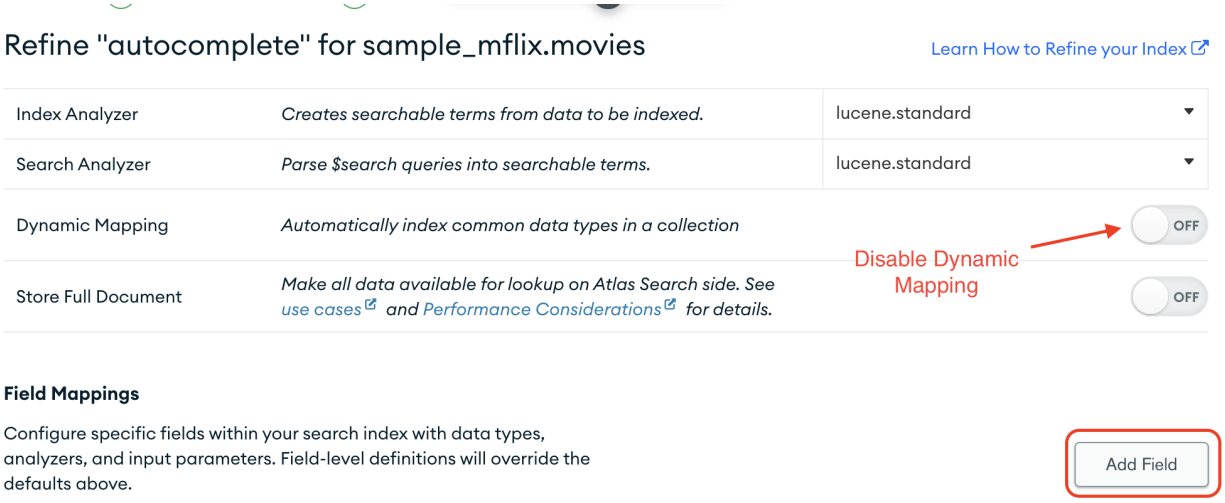
Define an Index with Autocomplete

For our Netflix Clone, we want to make it easy for our users to find the titles of the movies they are looking for. This is easy with the **autocomplete** operator or type-ahead on the **title** field of our documents. But first we need to create an index that will allow for **autocomplete** as its own data type.

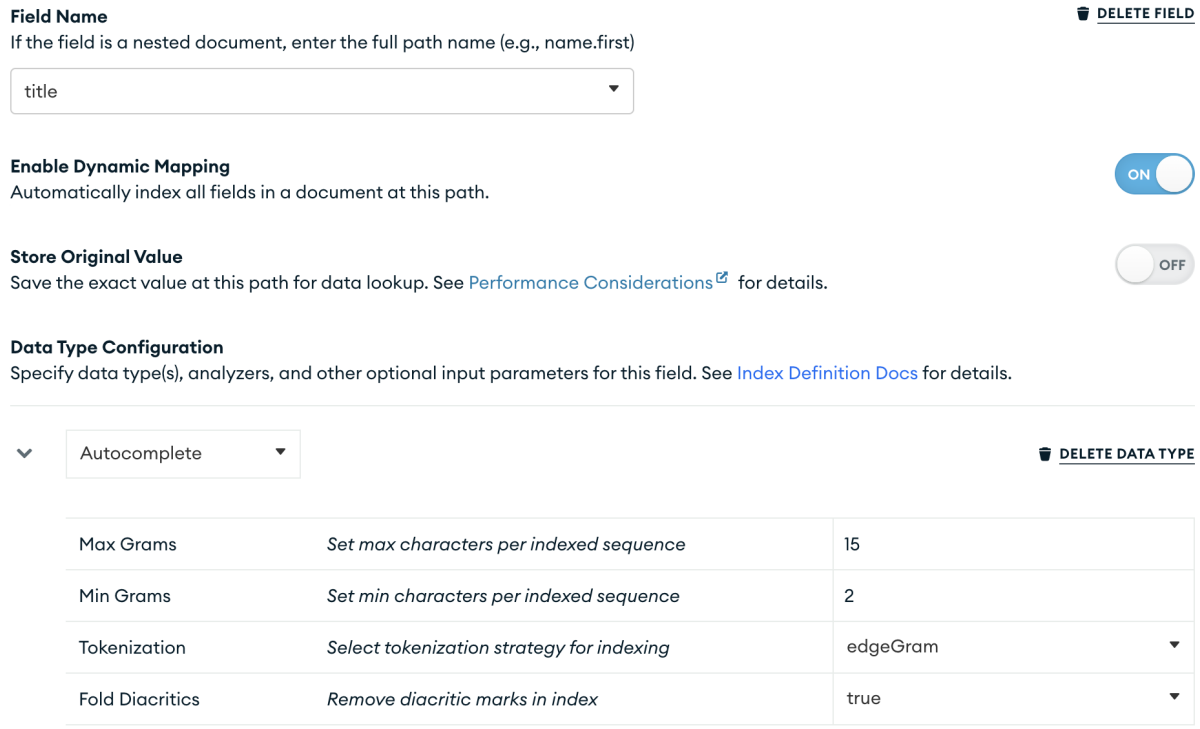
Return to the **sample_mflix.movies** collection in the Atlas UI Data Explorer to create another index. Name this index **"autocomplete"** if you wish.



Now we will need to **Refine Index.** Since we are only looking for titles, disable dynamic mapping on the document and click **Add Field.**



Use the **title** field name and under **Data Type Configuration,** use **Autocomplete** before saving changes.



Write an Autocomplete Search Query

Now we are ready to use our new **autocomplete** index created on our movie documents' **title** field to write our new type-as-you-go functionality in our Netflix Clone application.

Return to Compass and in our search stage, replace the code with:

```
{
  index:'autocomplete',
  autocomplete:{
    query:"Indiana J",
    path:"title"
  }
}
```

Notice that since we are no longer using the **'default'** index, we need to specify the **index: 'autocomplete'** as the first line in our **$search** stage.

Now we want to be careful about over-fetching data unnecessarily and burdening our front end application. Since we only need the **title** field, we can then simplify our **$project** stage to only have the **title** field.

```
{
      'title': 1
}
```

And also, we can **$limit** to only 10 returned documents.

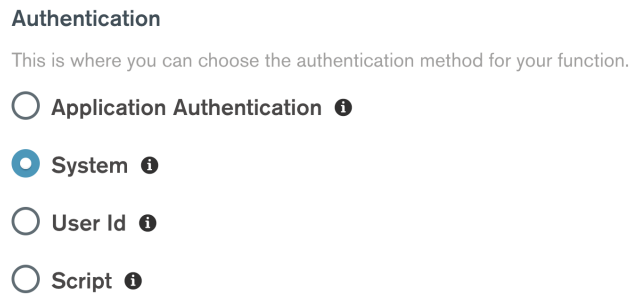
Our final aggregation pipeline will look like this:

```
[
  {
    '$search': {
      'index': 'autocomplete',
      'autocomplete': {
        'query': 'Inndiana Jon',
        'path': 'title'
      }
    }
  }, {
    '$limit': 20
  }, {
    '$project': {
      'title': 1
    }
  }
]
```

Very simple! Now we can simply create another **HTTPS Endpoint** in our same Netflix Clone App Services application as we did above in **Step 6** using the function code below. I used the endpoint route: **/titles ,** and named my new function **getTitles.**

```
exports = async function({ query, headers, body}, response) {

   // GET A HANDLE TO THE MOVIES COLLECTION
  const moviesCollection =
context.services.get("mongodb-atlas").db("sample_mflix").collection("movies");

  // GET SEARCHTERM FROM QUERY PARAMETER. IF NONE, RETURN EMPTY ARRAY
  let searchTerm = query.searchTerm;

  if (!query.searchTerm || searchTerm ===""){
    return [];
  }

  const searchAggregation =[
  {
    '$search': {
      'index': 'autocomplete',
      'autocomplete': {
        'query': searchTerm,
        'path': 'title',
        'fuzzy': {
          'maxEdits': 1
        }
      }
    }
  }, {
    '$limit': 12
  }, {
    '$project': {
      'title': 1
    }
  }
];

  const results = await moviesCollection.aggregate(searchAggregation).toArray();

};
```

Don’t forget to set the authentication method in the settings for your function to System:



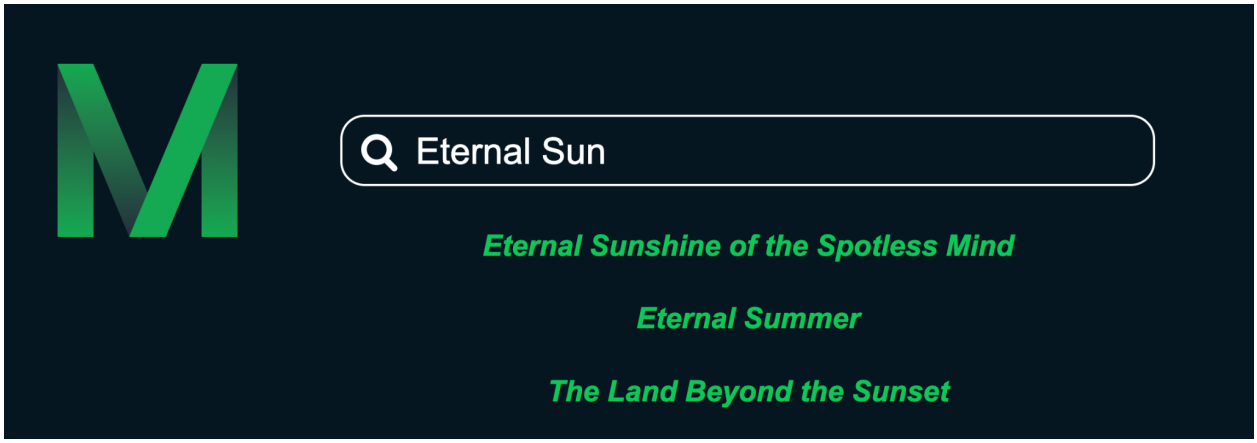
Now finally grab the **HTTPS Endpoint** you just created and insert into **line 15** in the **Searchbar.js** component file in the **src/components/SearchBar** directory:

```
7   const SearchBar = ({
8     searchTerm,
9     setSearchTerm,
10    setMovies,
11    setSubmitted,
12    showSuggestions,
13    setShowSuggestions,
14  }) => {
15    const TITLES_ENDPOINT = "";
16
```

Much like our original basic search query, on line 27 of this file, the value in the search bar will be appended to the end of the endpoint to begin displaying the **autocomplete** results of the movie titles after the user has typed in at least 3 characters (line 53).

```
25      let endpoint = TITLES_ENDPOINT;
26      if (searchTerm) {
27        endpoint = TITLES_ENDPOINT + `?searchTerm=${searchTerm}`;
28      }
29      try {
30        let names = await (await fetch(endpoint)).json();
```

Play around now with your Netflix to look for those movies whose names have been just sitting on the tip of your tongue!



Congratulations!
You've completed this lab on text searching using Atlas Search. Refer to Atlas Search documentation for examples and details about the various analyzers and query options available.