



Hands-On Workshop

Part 2 - Stitch

Overview

In Part 1 of this workshop you've set the foundation by creating a MongoDB cluster and loading some data. Now it's time to put that data to action. In part 2 of this workshop we'll create microservices to expose the data via REST APIs and create a basic front-end application that leverages those APIs.

Specifically, we'll create APIs to query movies. Then we'll host all of this on MongoDB Stitch!

Hands-on Lab

Lab 7 - Create a Microservice

Next we'll create a microservice that we'll expose to our application teams as a REST API. We'll accomplish this via a [MongoDB Stitch Function](#) and [HTTP Service](#). Our microservice will allow us to query for movies by name.

7.A. Create the Stitch Application

Stitch is a serverless platform, where functions written in JavaScript automatically scale to meet current demand. Return to the Atlas UI and click **Stitch Apps** on the menu on the left and then click the **Create New Application** button.

Name the application **Workshop**. The other defaults are fine:



Build your application with Stitch, our backend as a service.

Stitch manages data manipulation, integrations, and infrastructure so you can focus on building your app.

Create New Application

You will need an active cluster running MongoDB 3.4 or greater.
[Learn more about Stitch](#)

×

Create a new application

Application Name

Workshop

Link to Cluster

Only available clusters in 'HUAULME' running MongoDB 3.4 or greater are shown. Refresh this page to view available clusters.

Workshop

Note: Stitch is currently only located in select AWS regions. Linking it to Atlas clusters in other regions may result in lower performance.

Stitch Service Name ⓘ

mongodb-atlas

Select a Deployment Model

Choose from two deployment models - 'Local' (Single Region) or 'Global' (distributed across all supported regions). Learn more about [deployment models](#).

☐ Local

☒ Global

Select a Primary Region

Stitch will process application requests in the region closest to your end users. We recommend choosing the region closest to your cluster's primary. [Learn More](#).

Virginia (us-east-1)

Cancel

Create


Click **Create**, which will take you to the **Welcome to Stitch!** page.

7.B. Create the Function getMovieByTitle

Now we'll create the function that queries movies by name. Click **Functions** on the left and then **Create New Function**. Name the function **getMovieByTitle** and choose **Run as System**:

Functions

Create New Function



This application has no functions

Server-side JavaScript functions let you build complex logic and orchestrate data between clients, services, and MongoDB.

Create New Function

function0

Save

Function Editor

Settings*

Function Name

This is the name of your function. You use this name to call your function from a client. You can come back here to change it at any time.

getMovieByTitle

Private

If private, this function may be called from incoming webhooks, rules, and other functions defined in the Stitch console. Private functions may not be called from Stitch client application.

☐

Run As System

If enabled, this function runs with unrestricted access to services and may bypass rules.

☒

Can Evaluate

This is a JSON expression that must evaluate to TRUE before the function may run. If this field is blank, it will evaluate to TRUE. This expression is evaluated before other service-specific rules.

1

Click **Save**, which will open the Function Editor.

Replace the example code in the editor with the following:

```
exports = async function(arg){

  var collection = context.services
    .get("mongodb-atlas").db("sample_mflix").collection("movies");

  console.log ("IN GETMOVIEBYNAME FUNCTION");

  //Return a single document to matching the arg/movie name.
  var doc = await collection.findOne({name: arg});
  if (typeof doc === "undefined") {
    return `No movies named ${arg} were found.`;
  }
}
```

```

    }

    console.log(`FOUND A MATCHING MOVIE: ${arg}.`);

    return doc;
}

```

You can ignore the “Missing semicolon.” warnings shown in the editor.

Let’s review the code together. MongoDB has idiomatic [drivers](#) for most languages you would want to use. In this example we’re using the [findOne](#) method to return a single document.

Click the **Console** tab below the editor to expand it. In the Console, change the argument from ‘Hello world’ to **‘Speed’**:

The screenshot shows a console interface with two tabs: 'Console' and 'Result'. The 'Console' tab is active, displaying a block of text that reads: 'To Run the function: - Select a user - Type 'exports()' to run the function with no arguments - Type 'exports(arg1, arg2, args...)' to run the function with arguments - To run a saved function type 'context.functions.execute(<function-name-string>, args...)' - Click 'Run function as''. Below this text, the command `exports('Speed')` is entered.

Then click **Run** to test the function. You should get something similar to below with the result being the full document of the **Speed** movie.

The screenshot shows the 'Result' tab of the console interface. It displays the output of the function run, which is a JSON document for the movie 'Speed'. The output includes fields such as '_id', '_id_', 'fullplot', 'rating', 'numberDouble', and 'votes'. The 'fullplot' field contains a detailed description of the movie's plot.

Click **Save** and **Review & Deploy Changes** to save the function.

Changes have been made to your Stitch app since the last deploy [REVIEW & DISCARD](#) [REVIEW & DEPLOY CHANGES](#)

Optional: You can easily modify the function to look for movies by actor by making the changes below :

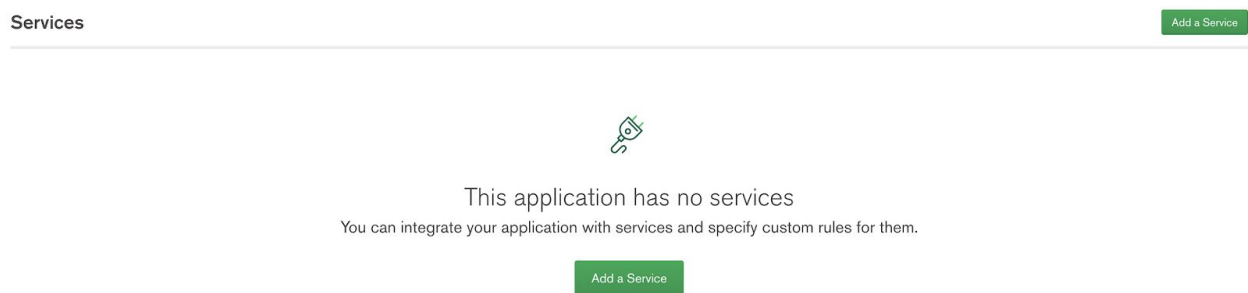
- Change field name to **“cast”**

- Change **findOne()** to **find()**
- Return results as an array: **.toArray()**

```
exports = async function(actor){  
  
  var collection = context.services  
    .get("mongodb-atlas").db("sample_mflix").collection("movies");  
  
  console.log ("IN GETMOVIEBYACTOR FUNCTION");  
  
  //Return a single document to matching the arg/cast name.  
  var docs = await collection.find({cast: actor}).toArray();  
  if (docs.length === 0) {  
    return `No movies with ${actor} were found.`;  
  }  
  
  console.log(`FOUND A MATCHING MOVIE WITH: ${actor}.`);  
  
  return docs;  
}
```

7.C. Expose the Function as a REST service


Click the **Services** menu on the left and then **Add a Service**.





You'll notice Stitch supports service integrations with [Twilio](#), [AWS](#) and [GitHub](#), making it very easy for you to leverage these providers' unique capabilities. More generically, Stitch also provides an [HTTP Service](#), which we will use to expose our function as a REST API.


Select the HTTP service and name it **movies**:

Add a Service

**Twilio**
Send and receive text messages

**HTTP**
Send requests over HTTP

**AWS**
Access AWS services

**GitHub**
Respond to GitHub events

Service Name
Enter a unique name for this service. You can add multiple instances of any type of service.

movies

Cancel

Add Service


Click **Add Service**. You'll then be directed to add an incoming webhook.

</> movies

Incoming Webhooks

Rules

+ Add Incoming Webhook



This service has no incoming webhooks
Incoming webhooks are a simple way to run functions in response to external sources.

Add Incoming Webhook

Click **Add Incoming Webhook** and configure the settings as shown below (the Webhook Name is **getMovieByTitle** and be sure to enable **Respond with Result**, set the HTTP Method to **GET** and Request Validation **Do Not Validate**):

webhook0 Save

Function Editor

Settings*

Webhook Name

getMovieByTitle

Respond With Result

☒

Run Webhook As

☒ System ⓘ

☐ User Id ⓘ

☐ Script ⓘ

HTTP Method

GET

POST

PUT

DELETE

PATCH

Request Validation

☐ Verify Payload Signature

☐ Require Secret As Query Param

☒ Do Not Validate

Cancel

Save

To keep things simple for this introduction, we're running the webhook as the System user and we're skipping validation. Click **Save**, which will take us to the function editor for the service.

In the service function we will capture the query argument and forward that along to our newly created function. Note, I could have skipped creating the function and just coded the service functionality here, but the function allows for better re-use, such as calling it [directly from a client application](#) via the SDK. Replace the code with the following:

```
exports = function(payload) {  
  
    var queryArg = payload.query.arg || '';  
    return context.functions.execute("getMovieByTitle", queryArg);  
  
};
```

Then set the **arg** in the Console to **'Speed'**:

**** note you must change from arg1 to arg ****

```
1 exports = function(payload) {  
2  
3   var queryArg = payload.query.arg || '';  
4   return context.functions.execute("getMovieByTitle", queryArg);  
5  
6 };  
7 |
```

Console

Result

```
/*  
To Run the function:  
- Select a user  
- Type 'exports()' to run the function with no arguments  
- Type 'exports(arg1, arg2, args...)' to run the function with arguments  
- To run a saved function type 'context.functions.execute(<function-name-string>, args...)'  
- Click 'Run function as'  
*/
```

```
exports({query: {arg: 'Speed'}, body: BSON.Binary.fromText('{ "msg": "world" }')})
```

and click **Run** to verify the result:

Console

Result

```
> ran on Fri Jul 19 2019 20:07:26 GMT-0500 (Central Daylight Time)  
> took 790.844838ms  
> logs:  
IN GETMOVIEBYNAME FUNCTION  
FOUND A MATCHING MOVIE: Speed.  
> result:  
{  
  "_id": {  
    "$oid": "573a1399f29313ca9bceeb67"  
  },  
  "fullplot": "Bomber terrorist's elevator plan backfires, so he rigs a bomb to a LA city bus. The stipulation  
  "imdb": {  
    "rating": {  
      "$numberDouble": "7.2"  
    },  
    "votes": {  
      "$numberInt": "241786"  
    },  
    "id": {  
      "$numberInt": "111257"  
    }  
  }  
}
```

Click **Save** to the service. The **Review & Deploy Changes**.

7.D. Use the API

The beauty of a REST API is that it can be called from just about anywhere. For the purposes of this workshop, we're simply going to execute it in our browser. However, if you have tools like [Postman](#) installed, feel free to try that as well.

Switch back to the **Settings** tab of the getMovieByTitle service and you'll notice a Webhook URL has been generated.

getMovieByTitle

Save

Function Editor

Settings

Webhook URL

This is the callback URL for an incoming webhook from this third-party service to execute a Stitch function.

https://webhooks.mongodb-stitch.com/api/client/v2.0/app/workshop-vxkrn

COPY

You can make a test request to this webhook using this curl command.

curl \

https://webhooks.mongodb-stitch.com/api/client/v2.0/app/workshop-vxkrn/service/movies/incoming_webhook/getMovieByTitle

CC

Webhook Name

getMovieByTitle

Respond With Result

☒

Click the **COPY** button and paste the URL into your browser. Append the following to the end of your URL:

?arg=Speed

REMEMBER THIS URL TO USE LATER IN YOUR CODE!

and submit (your output from your browser may look different):

← → ↺ https://webhooks.mongodb-stitch.com/api/client/v2.0/app/workshop-vxkrn/service/movies/incoming_webhook/getMovieByTitle?arg=Speed#

```
{
  "_id": {
    "$oid": "573a1399f29313caabceeb67"
  },
  "fullplot": "Bomber terrorist's elevator plan backfires, so he rigs a bomb to a LA city bus. The stipulation is: once armed,",
  "imdb": {
    "rating": {
      "$numberDouble": "7.2"
    },
    "votes": {
      "$numberInt": "241786"
    },
    "id": {
      "$numberInt": "111257"
    }
  },
  "year": {
    "$numberInt": "1994"
  },
  "plot": "A young cop must prevent a bomb exploding aboard a city bus by keeping its speed above 50 mph.",
  "genres": [
    "Action",
    "Crime",
    "Thriller"
  ],
  "rated": "R",
  "metacritic": {
    "$numberInt": "78"
  },
  "title": "Speed",
  "lastupdated": "2015-08-25 00:03:05.463000000",
  "languages": [
```

Feel free to play around with the API argument to find other movies such as:

?arg=Constantine

or

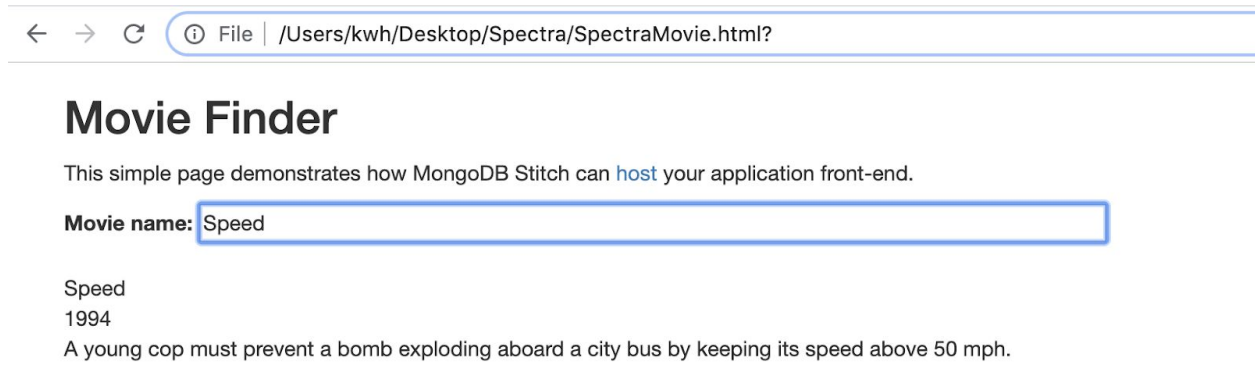
?arg=The%20Matrix

Lab 8 - Host your Application

Yes, Stitch can also [host](#) your application, therefore supporting the entire application stack. Let's see this in action using a very simple front-end that will use the REST API we just created and allow us to search for movies by title.

Download and Test the UI

Download this [index.html](#) file and open it in your browser. It should work as is because it's currently pointing to a pre-existing REST API:



← → ↻ ⓘ File | /Users/kwh/Desktop/Spectra/SpectraMovie.html?

Movie Finder

This simple page demonstrates how MongoDB Stitch can [host](#) your application front-end.

Movie name:

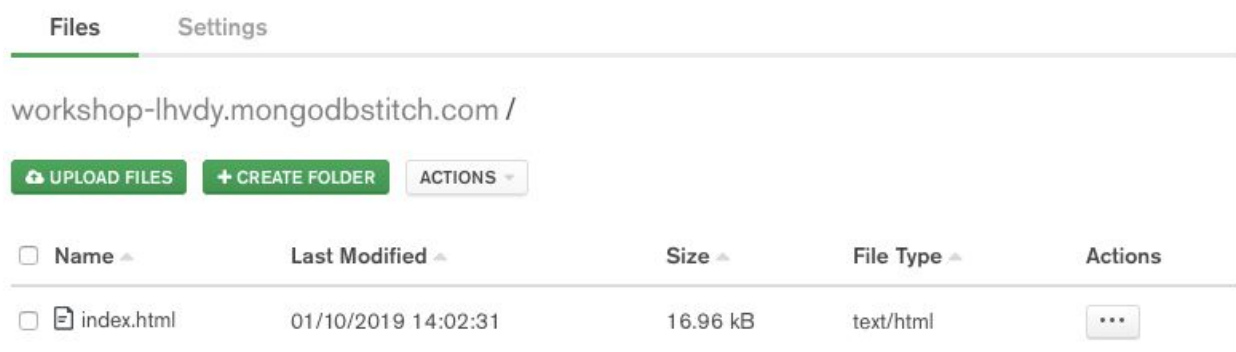
Speed
1994
A young cop must prevent a bomb exploding aboard a city bus by keeping its speed above 50 mph.

Open the index.html file in a code editor and familiarize yourself with the contents. Then replace the value of the **webhook_ur1** variable around **line 40** with the Webhook URL from the Stitch Service you created earlier. Save and test the UI.

Host the UI on Stitch


In the Stitch UI, click the **Hosting** in the left navigation bar and then click **Enable Hosting**:

Hosting (Beta)




Files Settings

workshop-lhvdy.mongodbstitch.com /

| <input type="checkbox"/> | Name ▲ | Last Modified ▲ | Size ▲ | File Type ▲ | Actions |
|--------------------------|--|---------------------|----------|-------------|----------------------------------|
| <input type="checkbox"/> |  index.html | 01/10/2019 14:02:31 | 16.96 kB | text/html | <input type="button" value="⋮"/> |

Upload your index.html file using the **UPLOAD FILES** button. When prompted if you want to overwrite the existing index.html file, click **Upload**. Then select the action to open your file in a browser:

| <input type="checkbox"/> Name ▲ | Last Modified ▲ | Size ▲ | File Type ▲ | Actions |
|---|---------------------|---------|-------------|---------|
| <input type="checkbox"/>  index.html | 01/10/2019 14:05:28 | 1.24 kB | text/html | ... |

Open in Browser
Copy Link
Edit Attributes...
Rename...
Move...
Copy...
Delete...

← → ↻ <https://stitch-statichosting-prod.s3.amazonaws.com/5d3260e9bb56a128fdf1930e/index.html>

Movie Finder

This simple page demonstrates how MongoDB Stitch can [host](#) your application front-end.

Movie name:

Speed

1994

A young cop must prevent a bomb exploding aboard a city bus by keeping its speed above 50 mph.

Notice the URL in your browser. Your movie application is now live on the Internet! Test it and confirm that the app is still successfully using your movie microservice.

And that's a wrap!

Lab 9 Optional - View the Stitch Logs

Logs

| Type (All) ▼ | Status (All) ▼ | From: dd/mm/yyyy | To: dd/mm/yyyy | Filter by UID... 🔍 | Filter by Request ID... 🔍 | Apply |
|--------------|---------------------------|------------------|----------------------|--------------------|--------------------------------------|---------|
| Status | Time | Time Taken | Id | User | Name | Type |
| ▶ OK | 2018-11-15T10:19:06-05:00 | 98ms | 5bed8e6a3e0e0b017... | -- | getRestaurantsByName | Webhook |
| ▶ OK | 2018-11-15T10:19:00-05:00 | 24ms | 5bed8e64e707c0eae... | -- | getRestaurantsByName | Webhook |