Getting Started with MongoDB Stitch

Overview

This tutorial demonstrates how to use MongoDB Stitch to create a realtime dashboard using the following Stitch features:

- JavaScript SDK
- Integrated HTTP Service
- Data access rules (read/write)
- Email/password, API Key, and Anonymous authentication

Here we will be dealing with simple Point-of-Sale data. However, this data is similar to Point-of-Sale, Telemetry, IoT, or Log data that you might regularly encounter.

We'll start off with all users being able to read and write data, but quickly move to a model where:

- A public API may be used to access a specific, aggregated subset of data
- Data can only be loaded using a special API key
- Authenticated users can view a realtime dashboard, but only have access to non-PII fields of the data

Prerequisites

For this app, you will need the following:

- A MongoDB Atlas cluster using MongoDB version 3.4+. The tutorial uses an Atlas Free Tier cluster.
- Node v6 or above to run the application and npm (install here)
  - The mongodb-stitch, http-server (installed with -g option) and chance packages

Tutorial

A. Create a MongoDB Atlas Instance *(~5 minutes)*

First, Log into Atlas.

To use MongoDB Stitch, you must be logged into MongoDB Atlas. If you do not have an Atlas account, follow the instructions in the Atlas documentation to create an account.

Next, create an Atlas cluster (3.4 or greater).

If you do not already have an Atlas cluster for use with MongoDB Stitch, create a cluster.

Important: You must deploy the Atlas cluster with MongoDB version 3.4 or greater.

Atlas provides a Free Tier M0 replica set as well as paid M10+ clusters. Free Tier deployments have restrictions as compared to paid M10+ deployments but will work for the purposes of this tutorial. For complete documentation on these restrictions, see Atlas M0 (Free Tier) Limitations.

Wait until your cluster finishes provisioning, then add a MongoDB Stitch application.

1. In Atlas, click Stitch Apps in the left-hand navigation.
2. Click the Create New Application. The name can only contain ASCII letters, numbers, underscores, and hyphens.
3. Select your Atlas cluster for your MongoDB service.
4. Click Create.

Upon creation of your app, you will be redirected to the MongoDB Stitch console. Your app is created with a MongoDB service named mongodb-atlas.

B. Get Started in Stitch (*~5 minutes)*

On the homepage of MongoDB Stitch, use the ‘Get started here’ steps to perform the following:

1. Using Step 1, turn on Anonymous authentication to your application
2. Using Step 2, initialize a MongoDB Collection. We’ll be using a collection with the following information:
    a. Database Name: SalesReporting
    b. Collection Name: Receipts

After we’ve completed these steps, we’ll open up access to this collection that we just initialized.

To view the collections in the attached MongoDB instance that you’ve made available to Stitch, click ‘mongodb-atlas’ and then click on the ‘Rules’ tab.

Important: If you want Stitch to be able to access data within a certain collection, you must add that collection to this list of Namespaces available and set the rules appropriately.

Important: MongoDB Stitch rules do not override the read and write access (i.e. authorization) that may have been set up separately in MongoDB. That is, MongoDB Stitch rules determine whether the fields are readable or writable; not whether the client has authorization to read or write to a particular database or collection.

Similarly, MongoDB Stitch validation rules do not override document validation rules set up separately in MongoDB.

1. Review the read rule for the top-level document of SalesReporting.Receipts.
2. Modify the read rule for the top-level document to the following and click Save:
3. { }
4. The new rule specifies that all the fields in the documents are always readable. For more information on MongoDB read rules, see MongoDB Service Read Rule.
5. Review the write rule for the top-level document of SalesReporting.Receipts.
6. Modify the write rule for the top-level document to the following and click Save:
7. { }
8. The new rule specifies that all the fields in the documents are always writable. For more information on MongoDB write rules, see MongoDB Service Write Rule.
9. For this collection, the owner_id field does not exist. Delete the owner_id field by hovering over it and clicking the x on the right-hand side.
10. By default, Allow All Other Fields flag is enabled so that any unlisted fields are readable/writeable as long as the document meets the read/write rules. Alternatively, you can explicitly define all the fields for this collection and disable Allow All Other Fields to control the shape of the documents.
11. Delete the rule ‘Filter’ for SalesReporting.Receipts.
    a. Click on the Filters.
    b. Delete the default filter.
    c. For more information on filters, see MongoDB Service Filters.
12. Save your changes.

See MongoDB Service Rules for more information about rules.

C. Download the Dashboard Application Source (*~3 minutes)*

If you haven’t downloaded the code for Stitch examples, you will need it to continue the tutorial. Download the examples from the MongoDB Stitch GitHub example repository and unzip. The source code for the Dashboard application can be found in the Dashboard directory.

D. Start loading data through Stitch (*~7 minutes)*

Next we’re going to use the data_generator.js script to load data through Stitch into the collection that you’ve initialized and added rules for.

Since the data_generator.js script uses Stitch to do this, you will have to add your App ID to the script. You can find your App ID in the ‘Clients’ section of Stitch’s left hand menu.

Now, edit the data_generator.js file to add your App ID.

1. From the downloaded examples directory, navigate to the Dashboard folder.
2. Edit the data_generator.js file (line 3):
    a. For “APP ID”, enter "<YOUR APP ID>" where <YOUR APP ID> is the App ID for your app.
3. Now run the data_generator.js file by using the terminal to navigate to the folder that contains the dashboard tutorial and using the command ‘node data_generator.js’

As long as the script continues to run, data will continually be loaded. In order for the dashboard to run correctly you will need to keep this script running, but we recommend stopping it after you have finished the tutorial.

D. Creating Simple API Access with Stitch (*~10 minutes)*

Now let’s look at how simple it is to set-up scoped down access to your data through Stitch. To do this, we’ll be creating a public HTTP endpoint to allow anyone who holds a special key to get the toppings that were ordered on the last 100 pizzas.

To start, we’re going to create a pipeline that aggregates our data:

1. In Stitch, click Pipelines and then click ‘New Pipeline’.
2. Enter the following properties for the pipeline:

Name	AggregateToppings
Private	Leave unselected. If selected, the pipeline is inaccessible by a client and is only accessible in other MongoDB Stitch definitions, such as webhooks and other pipelines.
Skip Rules	Leave unselected. If selected, rules do not apply to the service actions in the pipeline stages.
Can Evaluate	Leave as {}. The Can Evaluate condition determines whether the pipeline can be run. An empty document {} always evaluates to true, indicating that the pipeline can always be run.
Parameters	This Pipeline won’t require any parameters

For the output type, select Array
For the displayed stage, edit the following:

	Service – Select mongodb-atlas

	Action – Select aggregate

Pipeline	In the text box that appears, enter the following aggregation:
	
	`{`
	`  "database": "SalesReporting",`
	`  "collection": "Receipts",`
	`  "pipeline": [`
	`    {"$sort": {"timestamp": -1}},`
	`    { "$limit": 100},`
	`    { "$sortByCount": "$toppings"}`
	`  ]`
	`}`

Bind data to %%vars	We are not binding any data to variables in this pipeline.

3. Click Done
4. Click Save.

See Named Pipelines for more information.

Now we're going to use the HTTP service to create an endpoint that a user can interact with to get this data.

1. In Stitch, click 'Add Service' and then click 'HTTP Service' and name it 'ToppingsAPI'.
2. After creating the service, we will create an incoming Webhook that we will use to serve the necessary data. Click the Webhook tab and then click 'New Webhook' then enter the following properties:

Name	ToppingsAPI
Respond With Result	Select this so that API users will receive documents in their response
Request Validation	Select 'Require Secret As Query Param'
Secret	Set to a short string, you will use this when making your API call.

For the output type, select Single Document
For the displayed stage, edit the following:

	Service – Select built-in Action – Select namedPipeline

Pipeline	In the text box that appears, enter the following aggregation:
	
	`{`
	`  "name": "AggregateToppings",`
	`  "args": {}`
	`}`
	
	We're using this to call the AggregateTopping pipeline that we just created whenever this HTTP service receives a request.

Bind data to %%vars	Unselect as We will not bind any data to Vars.

3. Click Done
4. Click Create

See the HTTP Service documentation for more information.

Finally, let's try accessing this data from the terminal.

1. Open up a new terminal window and execute the following command:

   ```
   curl -X POST -H "Content-Type: application/json" [WEBHOOK
   URL]?secret=[SECRET] -d '{}' | json_pp
   ```

   Where the WEBHOOK URL is the URL provided in your webhook configuration and SECRET is the secret that you added to your webhook.

As long as you have loaded more than 100 orders into your database, you should see the distribution of toppings for the last 100 orders.

Now, let's move onto something a bit more complex, in the next example we'll look at serving data through a web application.

E. Setting up your Dashboard (*~10 minutes*)

Now we're going to use the rest of the code provided to take the data that we're loading into Stitch and create a realtime dashboard.

In order to do this, we're going to need another pipeline, which is going to take the a start time and end time and pass us data about the sales which were recorded within that time interval

1. In Stitch, click Pipelines and then click 'New Pipeline'.
2. Enter the following properties for the pipeline:

Name	SalesTimeline
Private	Leave unselected. If selected, the pipeline is inaccessible by a client and is only accessible in other MongoDB Stitch definitions, such as webhooks and other pipelines.
Skip Rules	Leave unselected. If selected, rules do not apply to the service actions in the pipeline stages.
Can Evaluate	Leave as {}. The Can Evaluate condition determines whether the pipeline can be run. An empty document {} always evaluates to true, indicating that the pipeline can always be run.
Parameters	This pipeline has the following parameters: start (Select as Required) end (Select as Required)

For the output type, select Array
For the displayed stage, edit the following:

	Service – Select mongodb-atlas Action – Select find

Pipeline (Stage 1)

```
{
  "database": "SalesReporting",
  "collection": "Receipts",
  "query": {
    "timestamp": {
      "$gt": "%%vars.start",
      "$lt": "%%vars.end"
    }
  },
  "project": {
    "_id": 0,
    "timestamp": 1,
    "total": 1
  },
  "sort": {
```

	`"timestamp": 1`
	`}`
	`}`

Bind data to %%vars	Enable and paste the following to bind the start/end parameters to variables:
	
	`{`
	`"start": "%%args.start",`
	`"end": "%%args.end"`
	`}`

3. Click Done

4. Click Save.

Now let’s kick off the code that will run our dashboard. Once again, since we’re using the Stitch SDK we will need our App ID. As a reminder, you can find your App ID in the ‘Clients’ section of Stitch’s left hand menu.

Now, let’s edit the index.html file to add your App ID.

1. From the downloaded examples directory, navigate to the Dashboard folder.
2. Edit the index.html file (line 9):
    a. For “APP ID”, enter "<YOUR APP ID>" where <YOUR APP ID> is the App ID for your app.
3. To start, first open a new terminal window and then navigate to the Dashboard folder. Next, to start your dashboard run ‘http-server &’
4. To view your dashboard, go to the local server address presented in the terminal.

Now you should see a dashboard graphing all of the sales that have happened in the last 5 minutes that continually updates.

If you can’t see any data or your graph is not updating, check to ensure that your data generator script is still running.

F. Adding Authentication *(~10 minutes)*

Now, before we release this application, we’re going to want to make sure that the correct people have access to read/write the correct data – nothing more and nothing less. Luckily, Stitch has rules that ensure that setting this up is easy.

To start, we’re going to tighten up the authentication a bit. Specifically, We’ll be using the integrated API Key and e-mail/password authentication that Stitch offers.

1. To start, go to the Stitch UI and find ‘Authentication’ in the left hand menu.
2. Find “Email/Password” under ‘Provider’ and click ‘edit’
    a. Click the slider to enable e-mail/password authentication
    b. Set Email Confirmation URL to ‘http://localhost:3000/#/confirm’
    c. Set Password Reset URL to ‘http://localhost:3000/#/reset’
    d. Click ‘Save’
3. Now let’s add our first e-mail/password user
    a. Click the ‘Users’ tab
    b. Click ‘+ Add User’
    c. Enter an email address and password of your choosing and click ‘Create’
    d. **Note**: Remember your login information, you’ll need it later.
4. Next go back to ‘Providers’, find ‘API Key’ authentication and click ‘edit’
    a. Click the slider to enable API Key authentication
    b. Click ‘Create API Key,’ assign your API Key a name, and click ‘Save’
    c. Click ‘Show Key...’ and then copy your key

Now that we’ve got our authentication methods set-up, we’re going to add the authentication code to the data_generator.js and index.html files.

Let’s start with data_generator.js. It will authenticate with the API token. This will represent a point-of-sale device that has an API key it uses to authenticate back to Stitch. While we’ll use one ‘device’ and API key, it would be easy to have multiple keys, each with the ability to write data for a single store location.

1. Take the key that you’ve copied from your API Authentication settings
2. Open the data_generator.js file in an editor
3. Go to the bottom of the file, comment out line 39 (adding ‘//’) and uncomment line 42 (removing ‘//’)
4. Replace ‘<YOUR API KEY>’ in with the API key that you have copied and save the file.
5. Now, if data_generator.js is still running, restart it (and if it is stopped, start it up again).

Next, let's add rules to tie everything together. To do this, we're going to add another pipeline that checks the type of authentication that a user is logged in with. This will allow us to scope writes to only users with API key validation and reads to only users logged in with an e-mail/password.

1. In Stitch, click Pipelines and then click 'New Pipeline'.
2. Enter the following properties for the pipeline:

Name	CheckAuth
Private	Leave unselected.
Skip Rules	Leave unselected. If selected, rules do not apply to the service actions in the pipeline stages.
Can Evaluate	Leave as {}. The Can Evaluate condition determines whether the pipeline can be run. An empty document {} always evaluates to true, indicating that the pipeline can always be run.
Parameters	Enable and paste the following to bind the start/end parameters to variables: UserAuth (Select as Required) AuthType (Select as Required) Note: This will be a two stage pipeline, so after you enter the first stage of the pipeline click "+ Add Stage"

For the output type, select Boolean

For the displayed stage, edit the following:

	Service – Select built-in Action – Select literal
Pipeline (Stage 1)	In the text box that appears, enter the following aggregation: `{ "items": [{ "UserAuth": "%%vars.UserAuth" } ]}`

Bind data to %%vars	Use the following to bind the UserAuth parameter to the doc above `{"UserAuth": "%%args.UserAuth"}`
Pipeline (Stage 2)	Service – Select built-in Action – Select match
Pipeline (Stage 2)	In the text box that appears, enter the following aggregation: `{ "expression": { "UserAuth": "%%vars.AuthType" }}`
Bind data to %%vars	Use the following to bind the UserAuth parameter to the doc above `{"AuthType": "%%args.AuthType"}`

3. Click Done
4. Click Save.

Now that we've added the necessary pipeline, let's tie everything together with a set of comprehensive rules.

1. To get started go to your linked MongoDB instance in the Stitch UI by clicking 'mongodb-atlas' on the left hand menu.
2. Then click the 'Rules' tab, and the 'SalesReporting.Receipts' namespace to edit the rules for the specific database and collection our dashboard uses.
3. We will start by removing the Read rule that we placed on 'Top-Level Document' (replace with "").
4. Next, we will define a write rule on 'Top-level Document' that will only allow API Key authenticated user to write to this namespace.

Rule

In the text box that appears, enter the following rule:

```
{ "%%true": {
    "%pipeline": { "name": "CheckAuth",
      "args": { "AuthType": "server",
        "UserAuth": "%%user.type"  }  }  }
}
```

5. Finally, we'll make only the timestamp and total fields readable by SDKs (any only by users that have authenticated with a username/password). To start, use the "+ Add" button to to add the fields "timestamp" and "total"'
6. In **both** of these fields, add the following read rule to check that a user is authenticated with username/password.

Rule

In the text box that appears, enter the following aggregation:

```
{  "%%true": {
    "%pipeline": { "name": "CheckAuth",
      "args": { "AuthType": "local/userpass",
        "UserAuth": "%%user.identities.provider"} } }
}
```

Finally, let's add authentication to index.html (covering our dashboard):

1. Open the index.html file in an editor
2. Near the top of the file, comment out line 16 and uncomment line 17
3. Refresh your web browser and now you will be asked to authenticate before you can view the dashboard you've created.
    a. To authenticate, you the e-mail and password that for the Stitch user that you created.

Note: index.html uses very simple way to collect authentication data as it shows your password in clear text, if we had more time we would use a more robust method of collecting user name and password, but we've simplified it for this example.
Now, with these steps, we've created a system where:

- Data can only be loaded using a special API key
- Authenticated users can view a realtime dashboad, but only have access to specific, non-PII data
- A public API may be used to access a specific, aggregated subset of data

H. What next?

This tutorial has lots of potential for expansions beyond the scope of what we've provided above. Some ideas are:

- Building out a more robust public API to access your data by adding more pipelines and webhooks

- Adding pipelines and using the data to create more graphs in your realtime dashboard
- Using one of our integrations to build extra features like e-mail receipts for customers or a slackbot for placing an order.
- Taking the ideas here and building an API or dashboard with your own data