Stitch Documentation with the Scannr App – Documentation 2019

Scannr App Download:

IOS Store - https://apps.apple.com/app/apple-store/id880966829

Google Play - https://play.google.com/store/apps/details?id=co.infinum.scannrapp&referrer=utm_source%3Dscannrapp.com%26utm_medium%3Dwebsite%26utm_campaign%3Dscannrapp.com

Step 1: Log in to MongoDB using this link:

https://cloud.mongodb.com/user?signedOut=true#/atlas/login

- Register here to create an account:

https://cloud.mongodb.com/user?signedOut=true#/atlas/register/accountProfile

- Proceed to Step 2 once logged in.

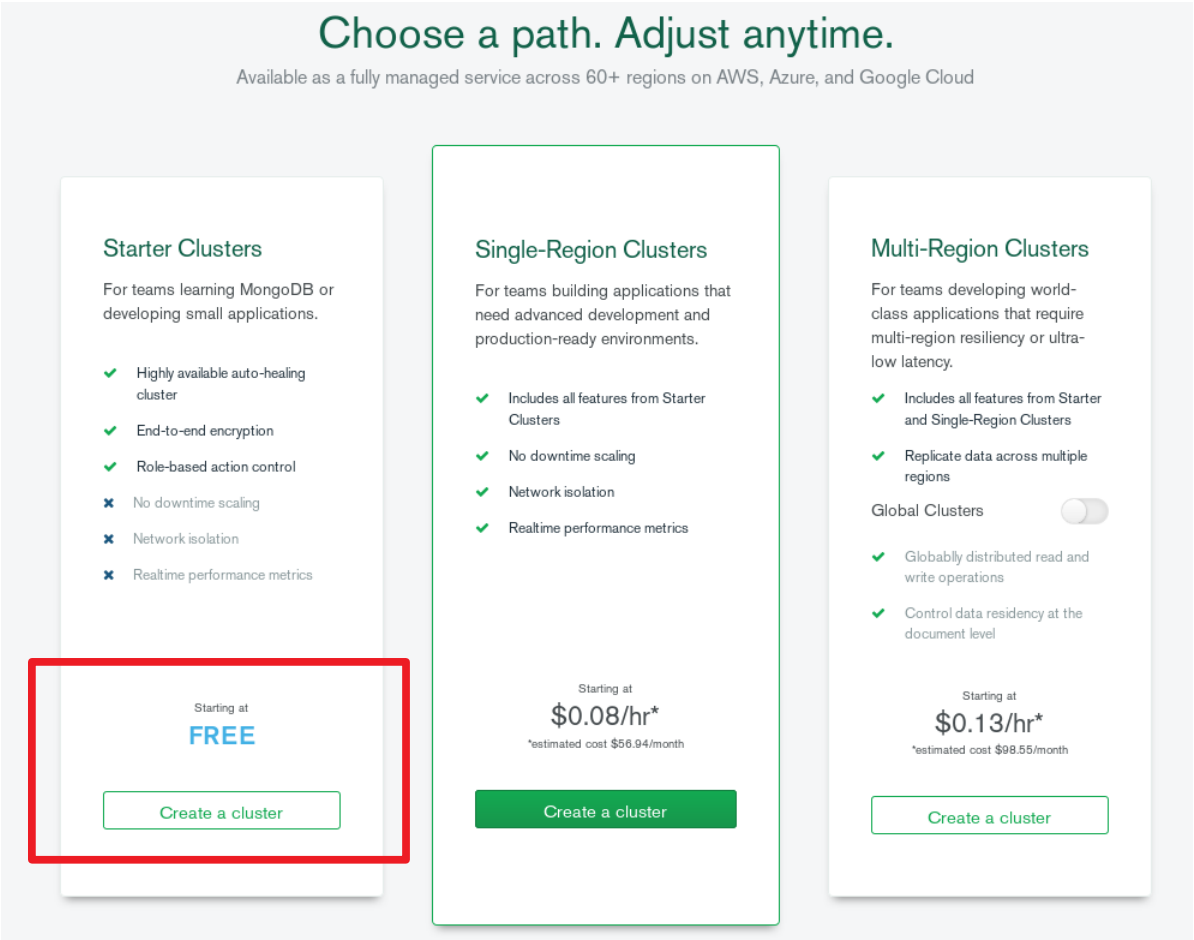
Step 2: Creating a new Project

- After logging in, the next window should display existing projects.
- Typically new or existing accounts already have a default project created named "Project 0." (Skip to Step 3 if continuing with Project 0).

- If users desire to make a new project anyway, click the green "New Project" button at the top right corner of the screen
- Name the project and then click "Next" to continue.
- The following window will allow users to add members and set permissions for those members to the new project. By default users will automatically be added as the project owner. Once the user settings are satisfactory, click "Create Project."
- Proceed to Step 3 once the new Project is created.

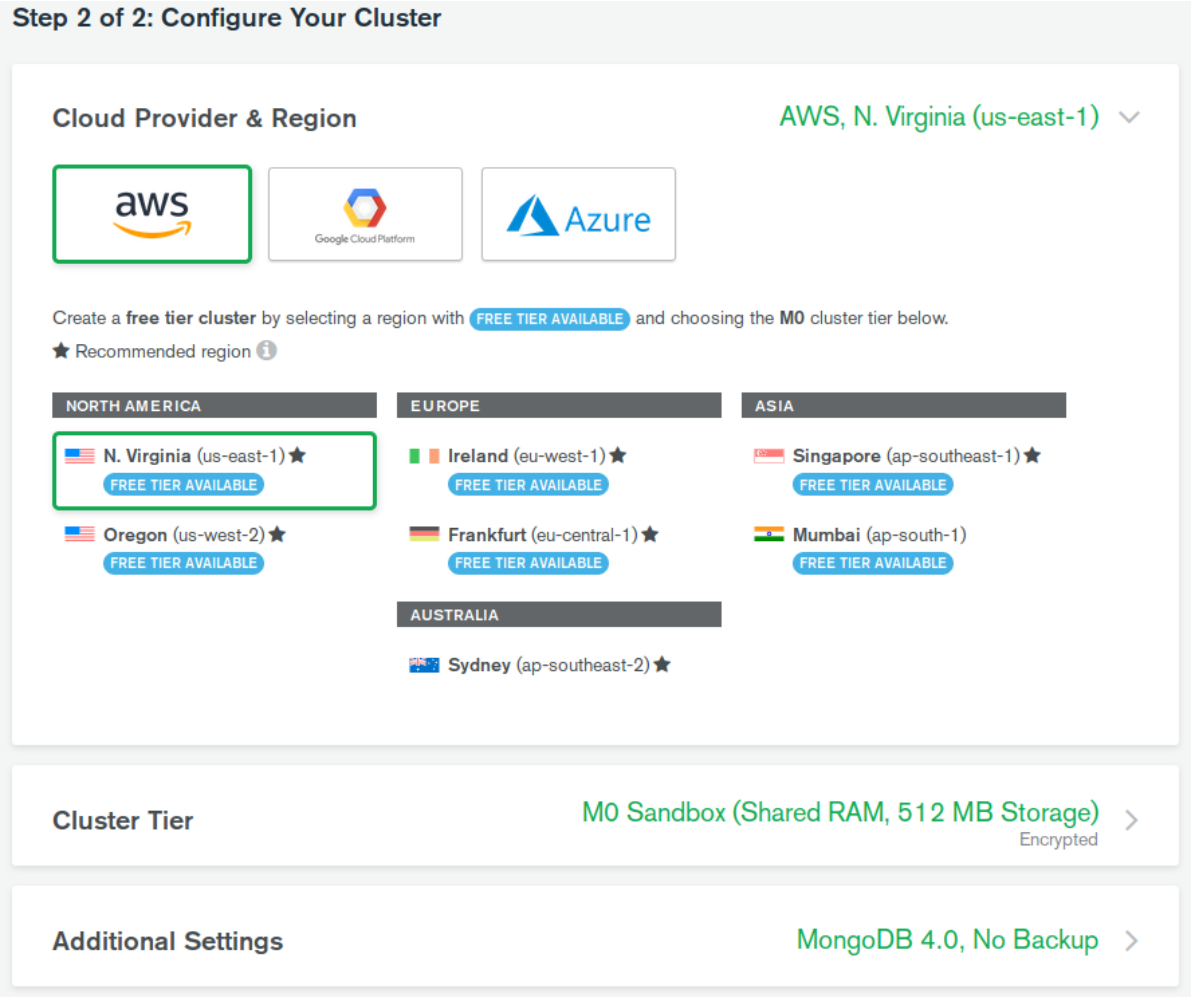
Step 3: Creating and setting up a new Cluster

- A new project will not have an existing cluster which will be needed to store information. Click "Build a Cluster" to create a new cluster.
- A new window like the one shown below will appear. Choose the "Free Starter Clusters" option.



- Enter the Cluster's new name and click "Continue."

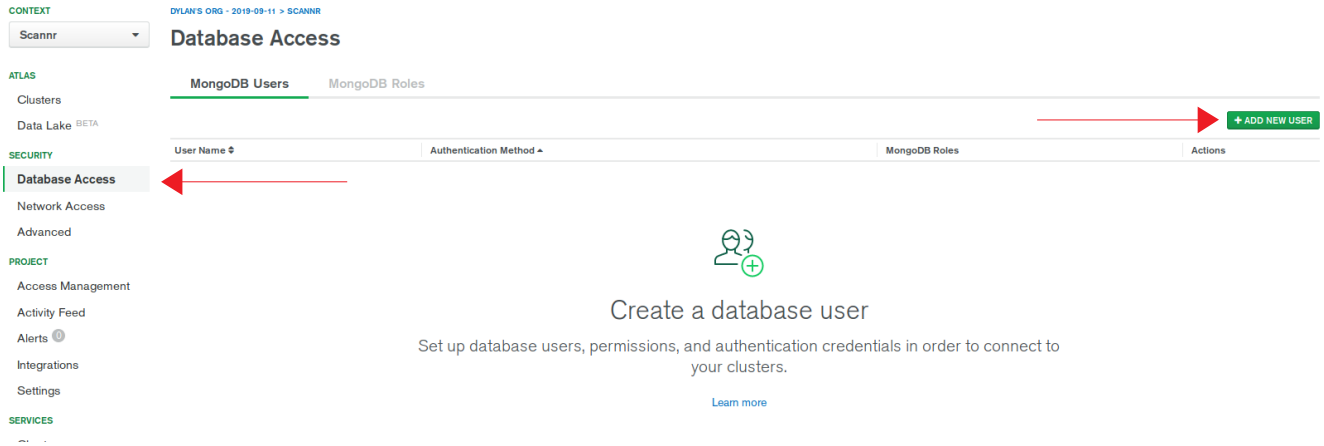
- Everything in the next window shown below may be left as they are by default. However, choose the preferred settings should there need to be any changes. Select "Create Cluster" to continue at the bottom of the screen.



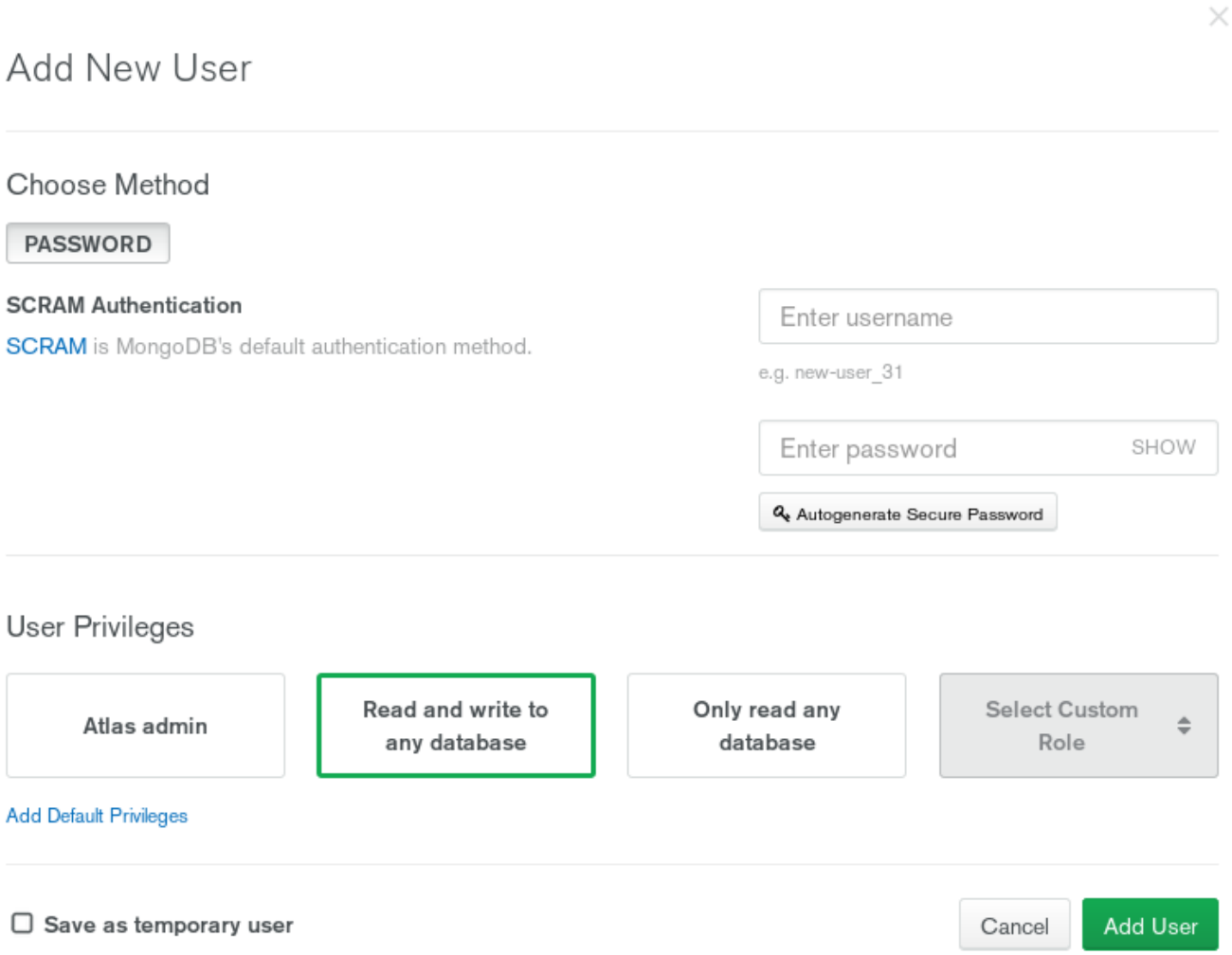
- The new cluster will take approximately 7-10 minutes to be established. Once it is finished, users will need to create a Database User and Whitelist their IP address.

3.1 Adding a Database User

- To add a new Database User, click "Database Access" under "Security" on the left hand of the screen. Then, select the green "Add New User" button on the right hand side of the screen.



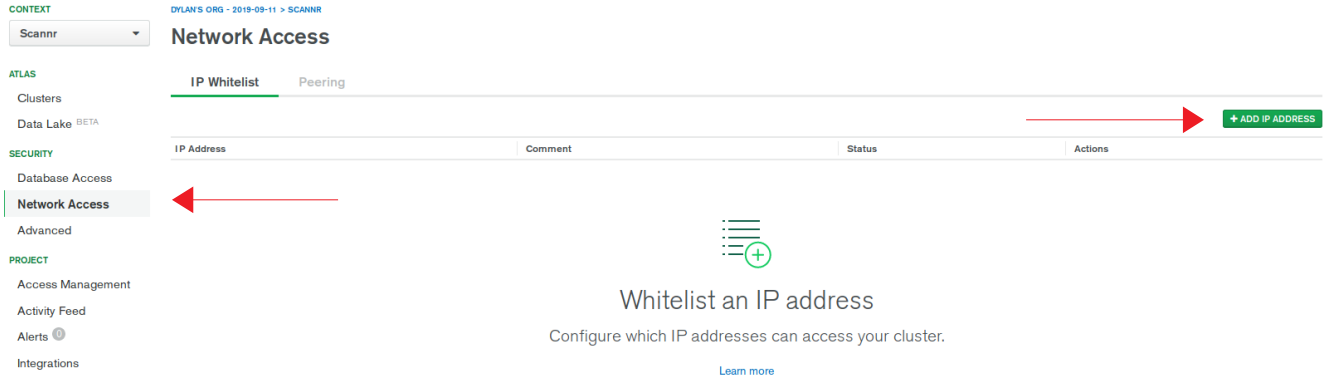
- A new window will appear where users can add their own username and password for the database user like the one below. Enter a new username and password and select "Add User" once finished. The "User Privileges" section may be left at its default setting.



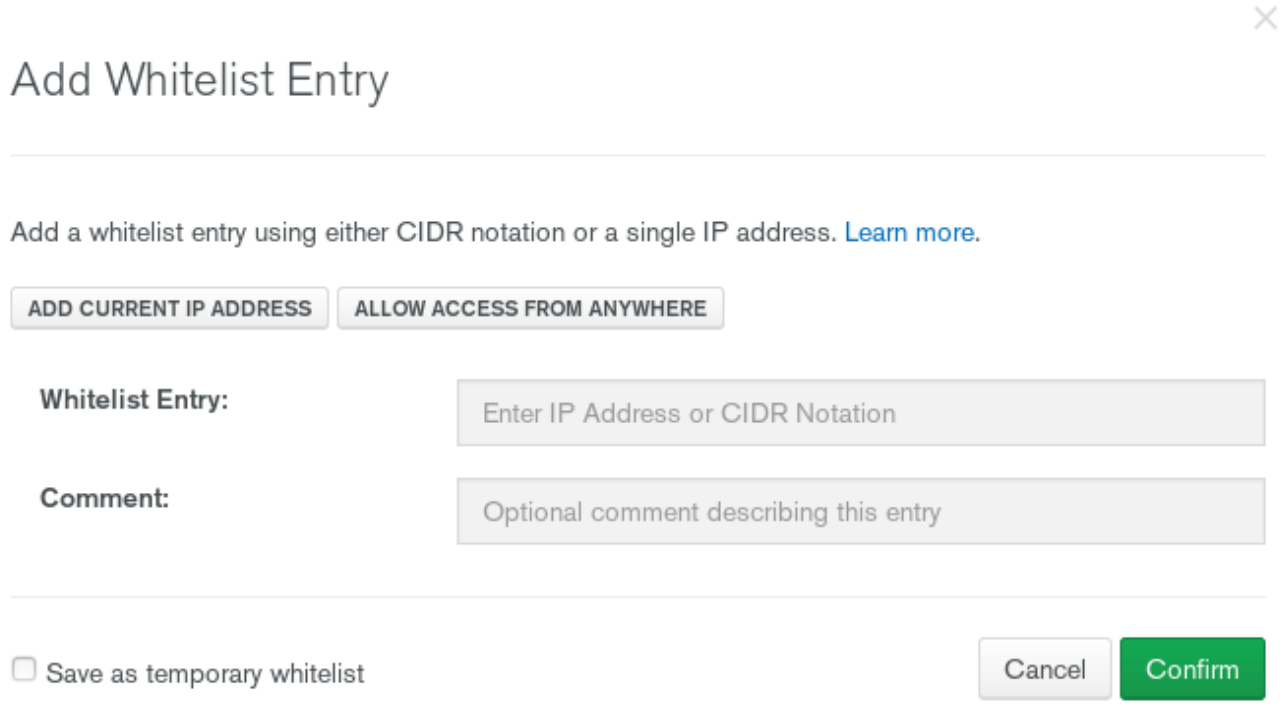
- Adding a new Database User make take a few moments to implement in the new cluster. Users can return to the cluster by clicking on “Clusters” under the “Atlas” section on the left hand side of the screen.Once completed please proceed to Step 3.2.

3.2 Whitelist IP Address

- Going back to the left hand side of the screen, choose “Network Access” under “Security.” From there, click the green “Add IP Address” button on the top right hand corner of the screen.



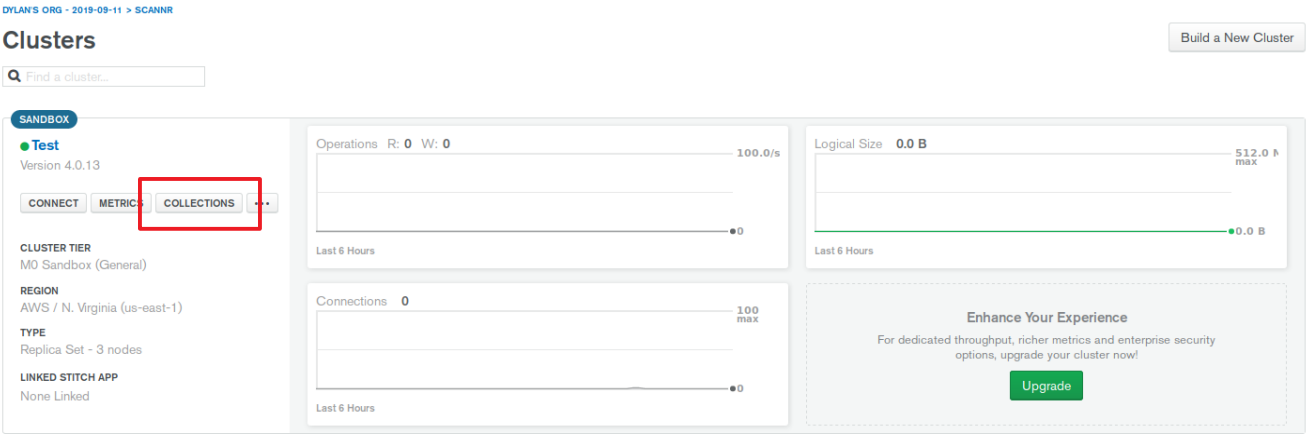
- On the following window, simply click the “Add Current IP Address” to whitelist an IP address. After that, select “Confirm.”



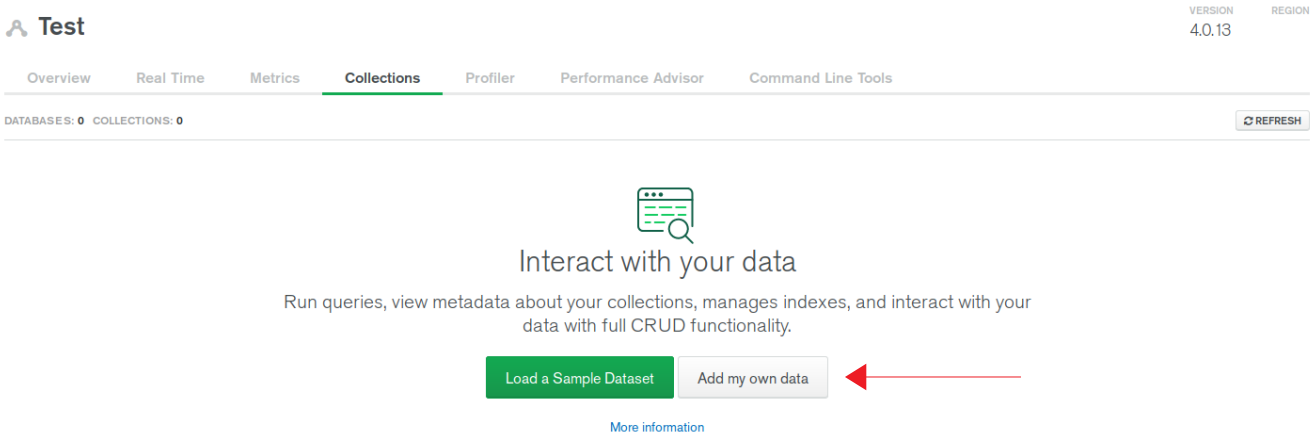
- Adding a whitelisted IP address will take a few moments to configure with the Cluster. Users can return to the cluster by clicking on "Clusters" under the "Atlas" section on the left hand side of the screen. Once that is completed, please proceed to Step 3.3.

3.3 Adding the Database and Collection

- Once a Database User and IP Address has been added the final addition needed for the cluster is to manually add a database and a collection. Choose "Clusters" under the "Atlas" section to return to the created cluster.
- To create a database and a collection for the cluster, select "Collections."

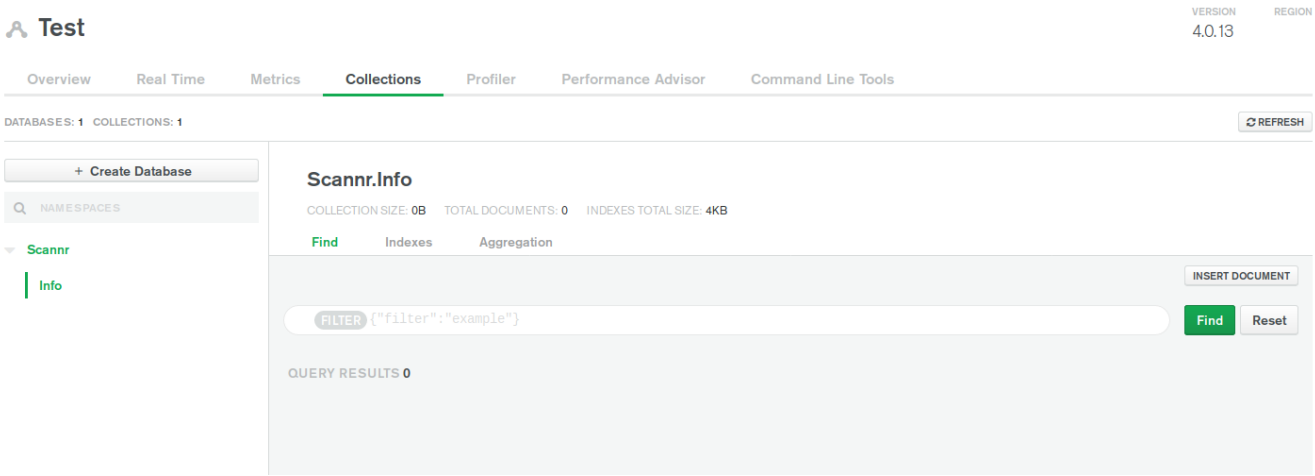


- Users will be brought to a new screen as shown below. Click on "Add my own data" to continue.



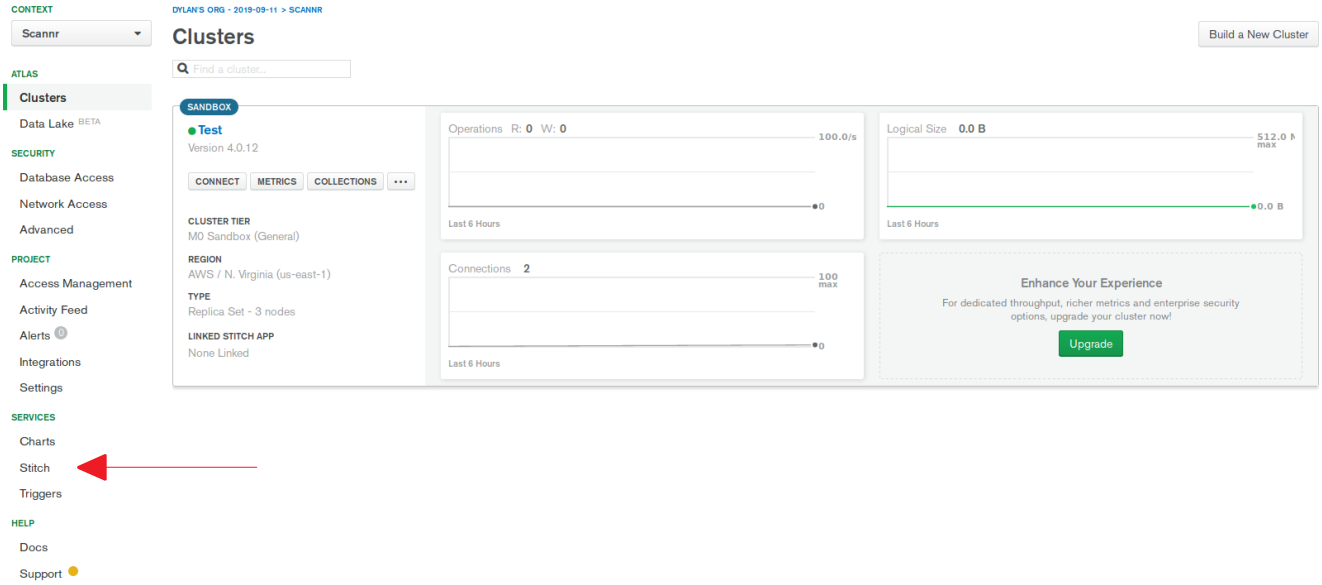
- A new window will appear to allow users to enter a database name and a collection name. Enter a name for the database and collection and click "Create."

- If the Database and Collection had been created successfully, then users will be directed to a new window like the one shown below:



- This is where information will be stored using MongoDB Stitch. **Remember the database and collection name as they will be used for coding later on.**
- When ready, proceed to Step 4.

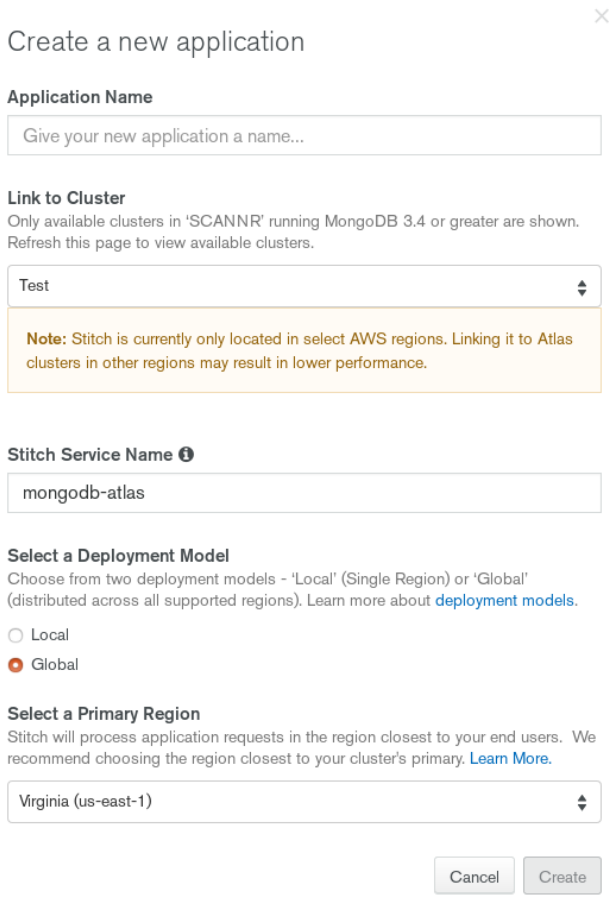
Step 4: Link a Stitch Service to the new Cluster



- Return to the cluster by selecting “Clusters” under the “Atlas” section. At this point, this image above should be the same to the information on screen. This is where Stitch will be linked to the cluster. To link Stitch to the new cluster, click “Stitch” at the bottom left under the “Services” tab indicated by the red arrow.

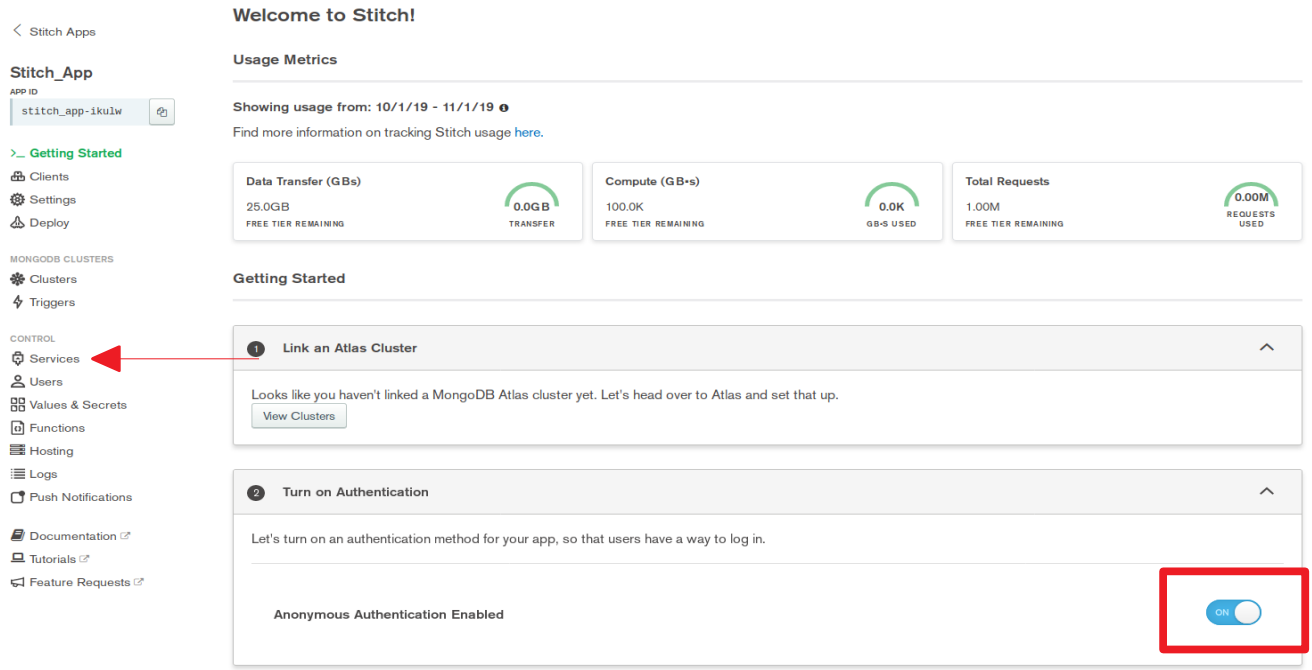
- Select “Create New Application” in the next window.

- The window indicated to the right allows users to detail certain aspects of the Stitch application. For now, the default selections will be suitable. Simply give a name to the application and select “Create.”



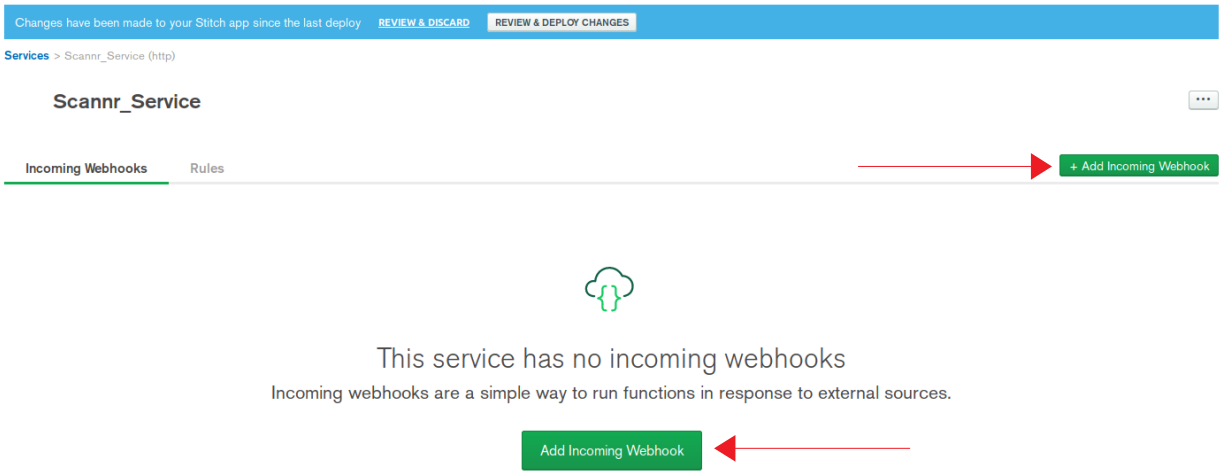
- The Stitch Application takes 1-3 minutes to create. Once it is finished, please proceed to Step 5.

Step 5: Stitch Setup – Creating a Service

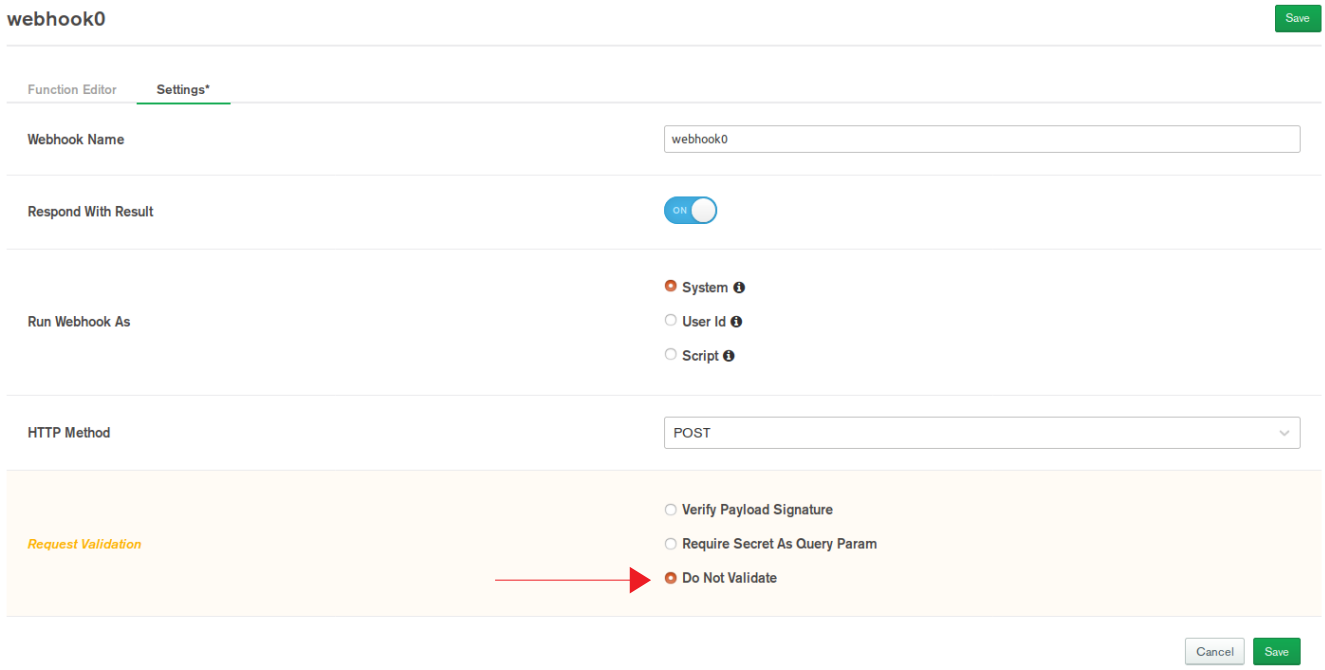


- There are many choices here in the next window; however, the "Services" tab under the "Control" section on the left hand side of the screen and the "Anonymous Authentication Enabled" under "Turn on Authentication" will be the only necessities to interact with. Enable "Anonymous Authentication" and then select the "Services" tab when ready.
- Click "Add a Service."
- Select "HTTP" as the service and then name your new service. Click "Add Service" when finished.
- Continue to Step 6 when ready.

Step 6: Webhook Setup – Creating a Webhook



- The next step is to create a Webhook for the newly created Service. To do so, select either button that says, “Add Incoming Webhook.”



- The next window allows you to modify the settings of the Webhook. Here the Webhook’s name can be changed. The only setting that needs to be changed is “Request Validation.” Select “Do Not Validate” and then click “Save” at the top right or bottom right portion of the screen.
- Continue to Step 7 once ready.

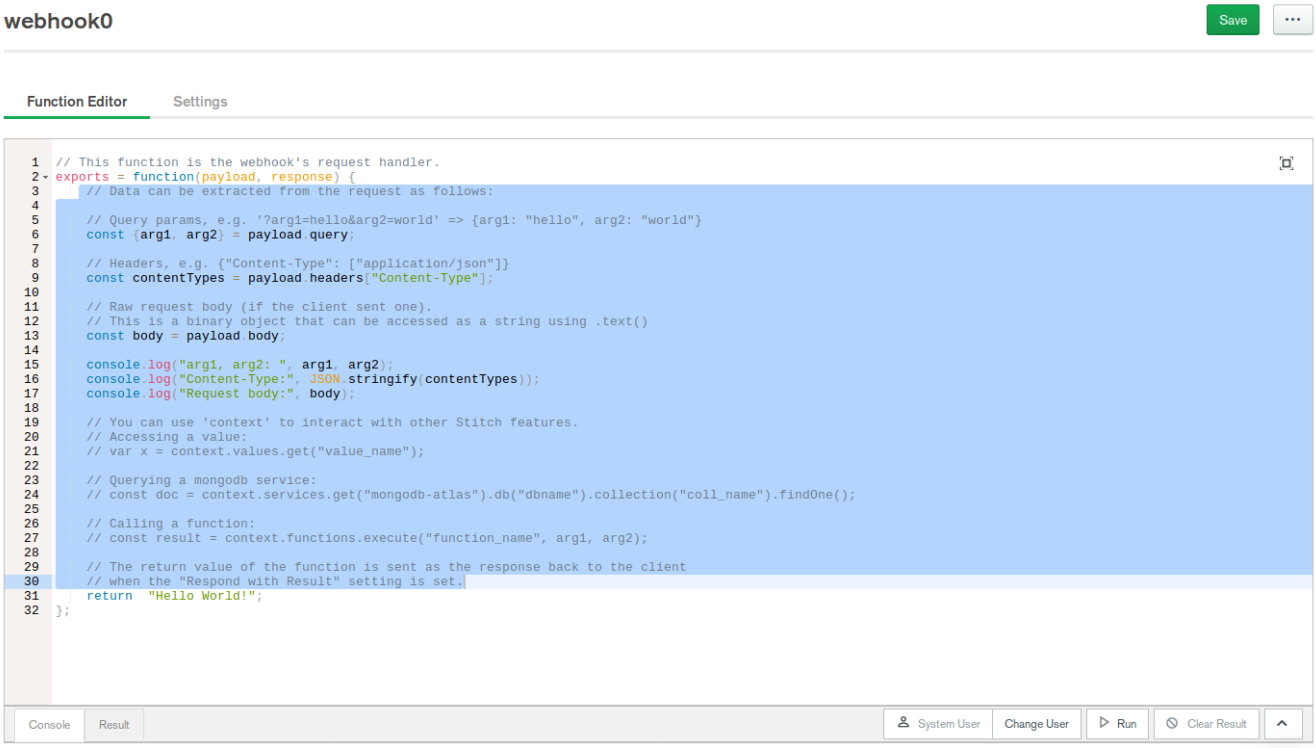
Step 7: Writing The Code

- Users will be brought to the “Function Editor” once the Webhook is created. This is where code will be written to establish connection to the Scannr app to the database. Additionally, users will be able to test their code by using the Function Editor. The following information will provide small steps to ensure proper connectivity from Scannr to MongoDB.
- Proceed to Step 7.1 when ready.

7.1 Remove Initial Code:

- Stitch uses JavaScript for back-end connectivity. This the type of code users will use to create a Stitch application.

- The first step is to remove the body of the code. The argument parameters for the function at the top and the return statement at the bottom will stay. Refer to the picture below. The highlighted section will be removed.



- Continue to Step 7.2 when ready.

7.2 Create a variable to hold the Service

- This new variable will store the service created earlier. It will be used for later on. The name 'mongodb-atlas' comes from the "Linked Cluster Name" in the "Getting Started" section of Stitch. This should be the same. However, if users wish to check what it is, follow these instructions:

1) Save the progress on the Webhook

2) Click on "Getting Started" on the left hand side of the screen

3) Scroll down to "Initialize a MongoDB Collection"

The Linked Cluster name should be listed under this section.

- To get back to your service:

Click on Services under the Control section → Select your service under Services → Select your Webhook.

Function Editor* Settings

```
// This function is the webhook's request handler.
exports = function(payload, response) {

  // 1) Add a variable that holds your Service
  const DB_SERVICE = context.services.get('mongodb-atlas');

  return  "Hello World!";
};
```

- Proceed to Step 7.3 when ready.

7.3 Add a variable to store the database/collection

- The next addition to the code is to initialize a variable to hold the database and collection. Previously the database and collection were created in Step 3. Here, users will utilize those names for the newly created line.

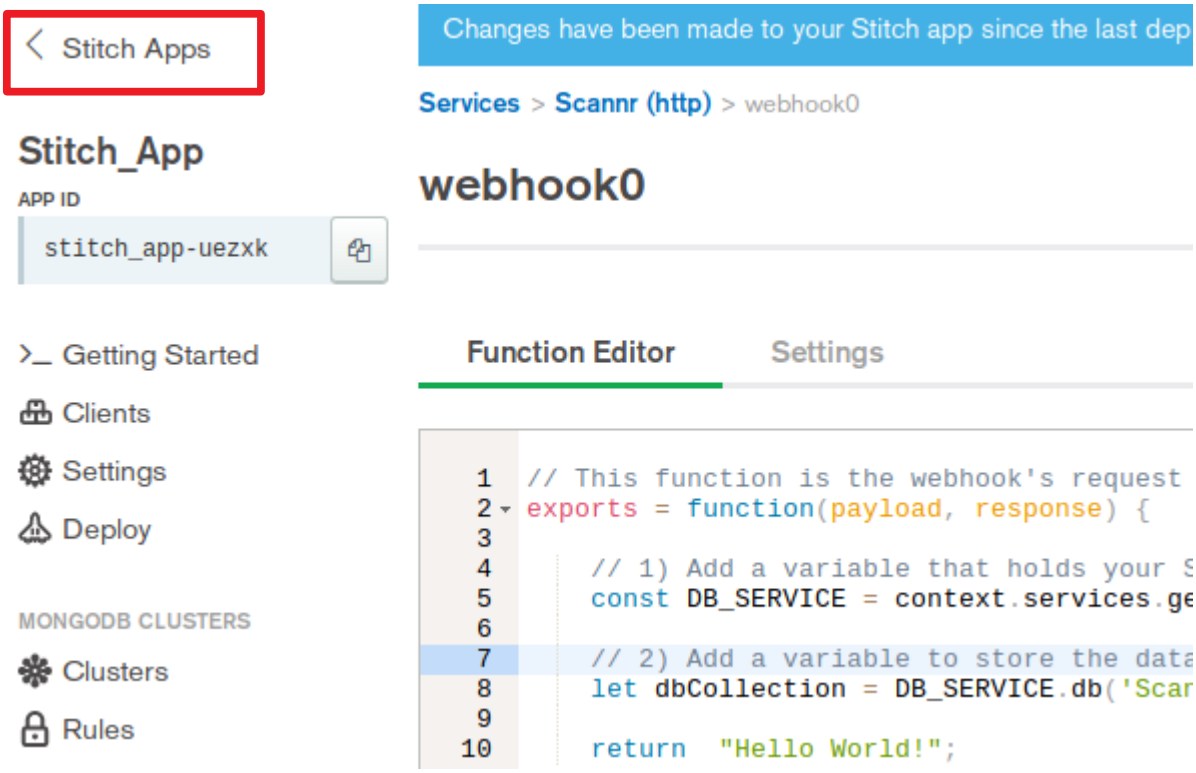
Function Editor Settings

```
// This function is the webhook's request handler.
exports = function(payload, response) {

    // 1) Add a variable that holds your Service
    const DB_SERVICE = context.services.get('mongodb-atlas');

    // 2) Add a variable to store the database/collection
    let dbCollection = DB_SERVICE.db('Scannr').collection('Info');

    return  "Hello World!";
};
```

- To refer back to the database and collection name:

1) Save the progress on the Webhook

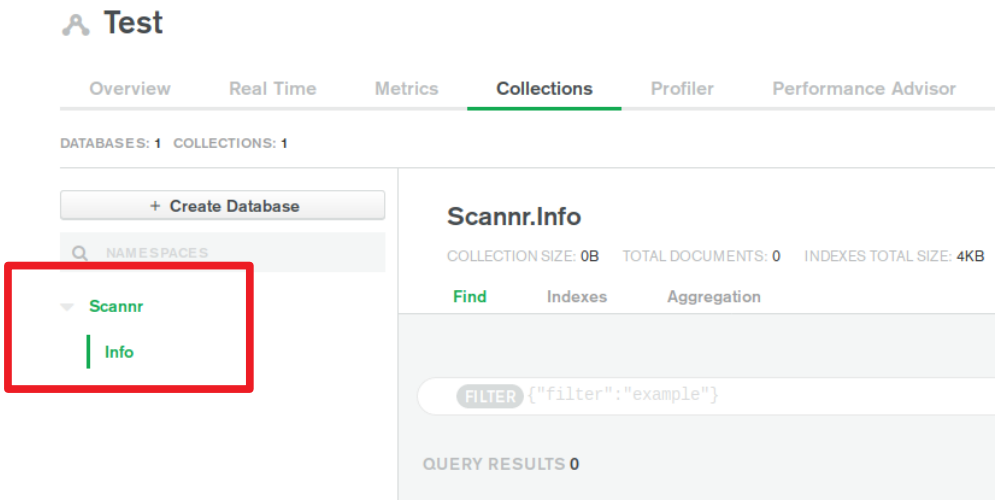
2) Click “Stitch Apps” at the top left corner of the screen



3) Navigate to “Clusters” under the “Atlas” section on the left hand side of the screen

4) Select Collections

The name of the database and cluster will be located on the left hand side of the screen. Here, “Scannr” is the database name and “Info” is the collection name.



- Next is Step 7.4.

7.4 Add a variable to store the object document Scannr passes.

- The Scannr app can pass an object document into the Mongo database. This object lists all the properties that Scannr uses to identify an individual when their driver's license is scanned. An example of this object has been provided on the right.
- This object is important to have to see what the properties are as they are needed for Stitch programming.
- For now preparing a variable to hold this information will complete this step. Refer to the picture below on how to do so.


```
_id: ObjectId("5da871150bacfb0490aa24a1")
scanning_result: Object
    kPPCustomerFullName: "DOE,JOHN,A"
    kPPAddressJurisdictionCode: "AA"
    kPPIssuerIdentificationNumber: "123456"
    kPPCustomerFirstName: "JOHN"
    kPPCustomerFamilyName: "DOE"
    kPPAddressStreet: "1234 Address"
    kPPCustomerMiddleName: "A"
    kPPCustomerIdNumber: "D01234567"
    kPPAddressCity: "CITY NAME"
    kPPOrganDonor: "Y"
    kPPHeightCm: "180"
    kPPJurisdictionVersionNumber: "0"
    kPPJurisdictionRestrictionCodes: "B"
    kPPHeight: "68 IN"
    kPPAamvaVersionNumber: "1"
    kPPSex: "1"
    kPPDocumentExpirationDate: "01012020"
    kPPDocumentIssueDate: "12345678"
    kPPWeightPounds: "154"
    kPPIssuingJurisdiction: "AA"
    kPPJurisdictionVehicleClass: "A"
    kPPWeightKilograms: "70"
    kPPAddressPostalCode: "12345-1234"
    kPPEyeColor: "GRN"
    kPPNonResident: "N"
    kPPHairColor: "BR"
    kPPHeightIn: "68"
```


Function Editor Settings

```
// This function is the webhook's request handler.
exports = function(payload, response) {

    // 1) Add a variable that holds your Service
    const DB_SERVICE = context.services.get('mongodb-atlas');

    // 2) Add a variable to store the database/collection
    let dbCollection = DB_SERVICE.db('Scannr').collection('Info');

    // 3) Add a variable to store the object document Scannr passes
    const body = payload.body;
    // This variable allows us to read the object passed in by the Scannr app
    scannrObject = JSON.parse(body.text());

    return  "Hello World!";
};
```

- Proceed to Step 7.5 to continue.

7.5 Write Code to Insert a New Object

- To insert the Scannr object document mentioned in the previous step, the insertOne function will come into play. This function will be used with the variables created in steps 7.3 and 7.4.

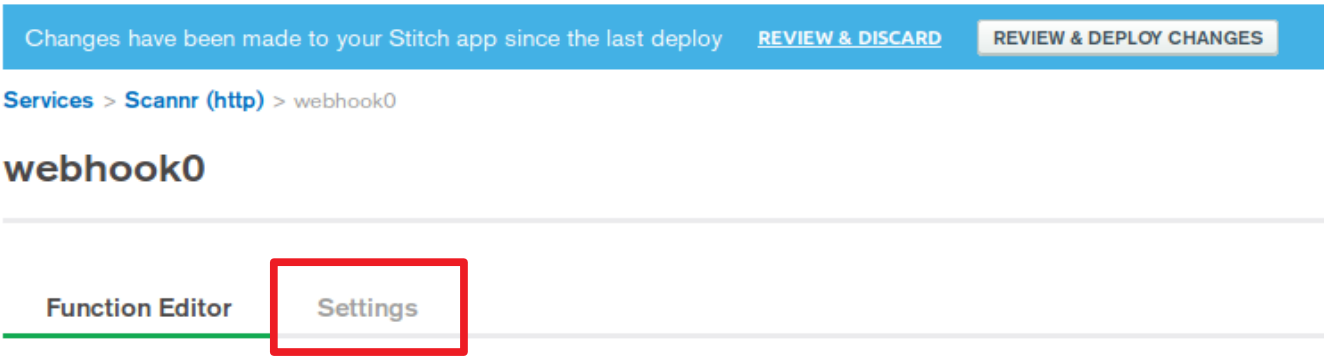
Function Editor Settings

```
// This function is the webhook's request handler.
exports = function(payload, response) {

    // 1) Add a variable that holds your Service
    const DB_SERVICE = context.services.get('mongodb-atlas');

    // 2) Add a variable to store the database/collection
    let dbCollection = DB_SERVICE.db('Scannr').collection('Info');

    // 3) Add a variable to store the object document Scannr passes
    const body = payload.body;
    // This variable allows us to read the object passed in by the Scannr app
    scannrObject = JSON.parse(body.text());

    // 4) Inserting a new object
    dbCollection.insertOne(scannrObject)
    .then(result => console.log(`Successfully inserted a new person with id: ${result.insertedId}`))
    .catch(err => console.error(`Failed to insert new person: ${err}`));

    return  "Inserted 1 new record!";
};
```

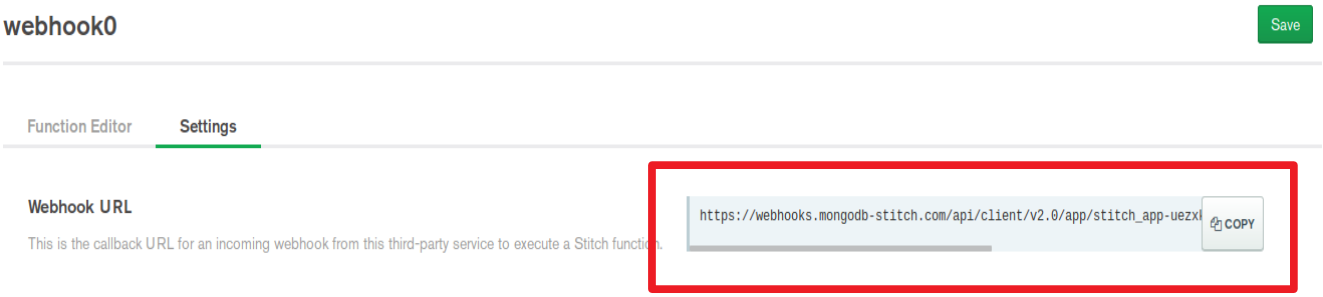
- Please continue to Step 7.6 when ready.

7.6 Deploying Changes and Using Webhook

- At this point, the Webhook should be ready to pass the Scannr object document into the database.
- Save the progress on the Webhook and towards the top of the screen there is a blue bar with the button that reads "Review & Deploy Changes." Click it, review the changes made, select "Deploy", and then select the "Settings" tab below.

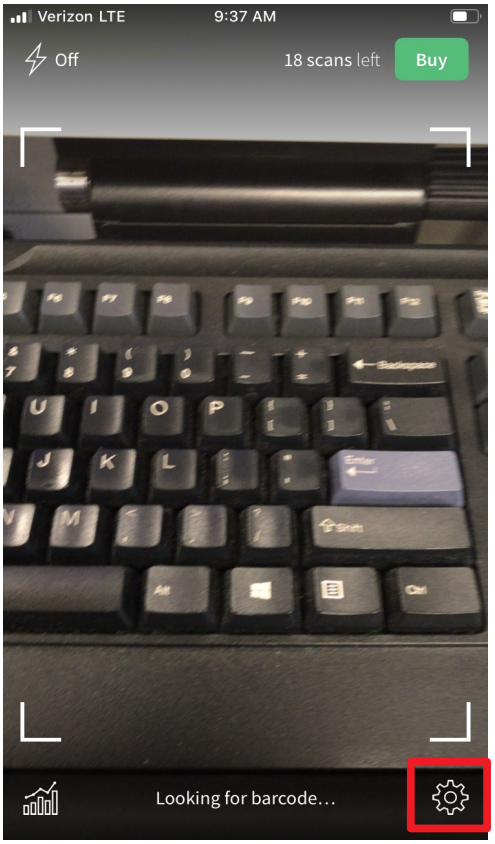


- From here, the Scannr App will be used. To download it, go to the IOS Store or Google Play and download the app to a phone. The link below is the Webhook URL and will be used with Scannr. To get this link to the Scannr app, users can copy the link below and email it through G-mail or by other conventional means.



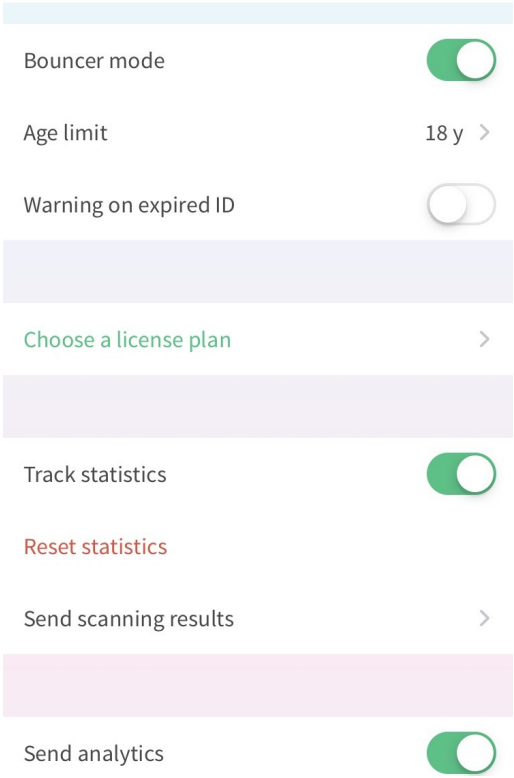
- Once users have transferred the Webhook URL from their computer to their phone, they will need to open the Scannr app and follow the pictures below. Note that these pictures were taken from an iPhone. Scannr may look different on an Android.

1



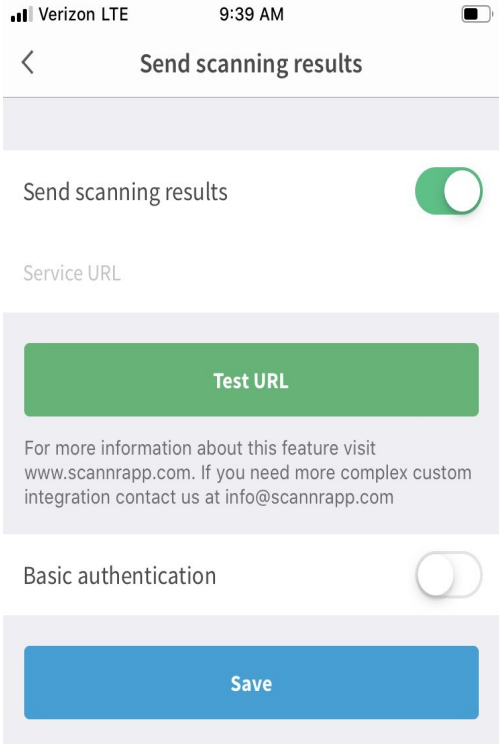
Select the Settings button

2



Select "Send scanning results"

3



Paste the URL in the "Service URL" section. Then click "Test URL"

- After this, users should now have a new object (like the one shown in step 7.4) listed in their database.
- To review the database, select "Stitch Apps" at the top left → Click "Clusters" under the "Atlas" section → Click "Collections" in the Cluster panel. Refer to the pictures below.
- Click ">" next to "scanning_result" to expand the object.

Scannr

ATLAS

Clusters

Data Lake BETA

SECURITY

Database Access

Network Access

Advanced

PROJECT

Access Management

Activity Feed

Alerts 0

Integrations

Settings

Clusters

Find a cluster...

SANDBOX

Test

Version 4.0.13

CONNECT | METRICS | COLLECTIONS | ...

CLUSTER TIER

M0 Sandbox (General)

REGION

AWS / N. Virginia (us-east-1)

TYPE

Replica Set - 3 nodes

LINKED STITCH APP

Stitch_App

Overview | Real Time | Metrics | Collections | Profiler | Performance Advisor

DATABASES: 1 COLLECTIONS: 1

+ Create Database

NAMESPACES

Scannr

Info

Scannr.Info

COLLECTION SIZE: 950B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 16KB

Find | Indexes | Aggregation

FILTER {"filter":"example"}

QUERY RESULTS 1-1 OF 1

```
_id: ObjectId("5daf16989f04c83a4d28f66b")
> scanning_result: Object
scanning_timestamp: 1421336230
```

- If there are issues adding the document object in the database, make sure to save the current version of the Webhook, select "Review & Deploy Changes", and click "Deploy." Deploying the Webhook will allow it to use the existing code in the Function Editor terminal. Sometimes users will edit the code, save it, and then forget to deploy the Webhook. This will cause the Webhook to utilize code previously entered before its deployment which can understandably cause confusion.
- To return to the Function Editor for the Webhook, select Stitch under Services on the bottom left hand side of the screen → choose the created Stitch app → select Services under the Control section → choose the created Service → select the created Webhook.
- If operations are performing properly, please proceed to Step 8.

Step 8: Modifying Object Information

- In this step, the object that Scannr passed to the database will be used to pass only certain properties to the database.
- To do so, users will need to navigate back to the Function Editor in the Webhook.
- All the information listed on the right may not be necessary to have within a new object. To individually grab the properties, users will need to write a new object that pulls specific properties from the object on the right.
- For example, if only the first name and last name is needed, then users will need to use the properties **kPPCustomerFirstName** and **kPPCustomerFamilyName**.
- In the Function Editor, the new object will be implemented. Additionally, users can create another insert method like the one shown below:


```
_id: ObjectId("5da871150bacfb0490aa24a1")
scanning_result: Object
  kPPCustomerFullName: "DOE,JOHN,A"
  kPPAddressJurisdictionCode: "AA"
  kPPIssuerIdentificationNumber: "123456"
  kPPCustomerFirstName: "JOHN"
  kPPCustomerFamilyName: "DOE"
  kPPAddressStreet: "1234 Address"
  kPPCustomerMiddleName: "A"
  kPPCustomerIdNumber: "D01234567"
  kPPAddressCity: "CITY NAME"
  kPPOrganDonor: "Y"
  kPPHeightCm: "180"
  kPPJurisdictionVersionNumber: "0"
  kPPJurisdictionRestrictionCodes: "B"
  kPPHeight: "68 IN"
  kPPAamvaVersionNumber: "1"
  kPPSex: "1"
  kPPDocumentExpirationDate: "01012020"
  kPPDocumentIssueDate: "12345678"
  kPPWeightPounds: "154"
  kPPIssuingJurisdiction: "AA"
  kPPJurisdictionVehicleClass: "A"
  kPPWeightKilograms: "70"
  kPPAddressPostalCode: "12345-1234"
  kPPEyeColor: "GRN"
  kPPNonResident: "N"
  kPPHairColor: "BR"
  kPPHeightIn: "68"
  kPPIssueTimestamp: "2001-12-01 00.00.00.000000"
  kPPDateOfBirth: "01012000"
scanning_timestamp: 1421336230
```


```
    // 4) Inserting a new object
    dbCollection.insertOne(scannrObject)
    .then(result => console.log(`Successfully inserted a new person with id: ${result.insertedId}`))
    .catch(err => console.error(`Failed to insert new person: ${err}`));

    // 5) Creating a new object
    newPerson = {
      "firstName": scannrObject.scanning_result.kPPCustomerFirstName,
      "lastName": scannrObject.scanning_result.kPPCustomerFamilyName
    };

    // 5.1) Inserting the object above to the database
    dbCollection.insertOne(newPerson)
    .then(result => console.log(`Successfully inserted a new person with id: ${result.insertedId}`))
    .catch(err => console.error(`Failed to insert new person: ${err}`));

    return  "Inserted 1 new record!";
};
```

- Once that is completed, save the Webhook and choose “Review & Deploy Changes” then click “Deploy.”

- Go back to Scannr and click “Test URL” once more. Go back to the cluster and the results should look like the picture below:

- The second document object is another “scannrObject” that came from running the program again. The third object is the result of the “newPerson” object being inserted into the database using only certain properties from the original object document. It only holds the first and last name just like it was coded to.