Lab: Building a near real-time risk assessment service

# Overview

You should have already seen a partial overview of the IoT for Insurance service offering on Bluemix and its supporting Watson services. This lab also assumes basic knowledge of deploying and managing applications and services on Bluemix.

Measuring risk is one way insurance companies determine premium rates for their customers. This measurement is based on various factors about the customer and their assets, and ultimately is analogous to the probability the customer will file a claim. Some of these factors like age and driving records change rarely, but customers can lower their rates by completing safety courses, maintaining good driving records, installing security systems, etc.

In this lab you will become familiar with the IoT for Insurance service which is comprised of various service offerings from Bluemix. We will write new shields and action handlers to extend the functionality of the base offering, and utilize the service to simulate a different insurance use case. The lab mission (i.e. risk assessment service) is a hypothetical use case meant to show what is possible.

# Mission

Instead of paying a pre-determined amount based on static or rarely changing data (age, gender, residency, etc), we (the insurer) want to provide a service that calculates customer insurance rates as a function of a stream of near real-time data regarding the customer’s assets and behavior.

## Prerequisites

You need the following software:

* Internet Explorer, Safari, Firefox, or Chrome web browser

You need the following accounts:

* Bluemix

# Section 1. Introduction to IoT4I

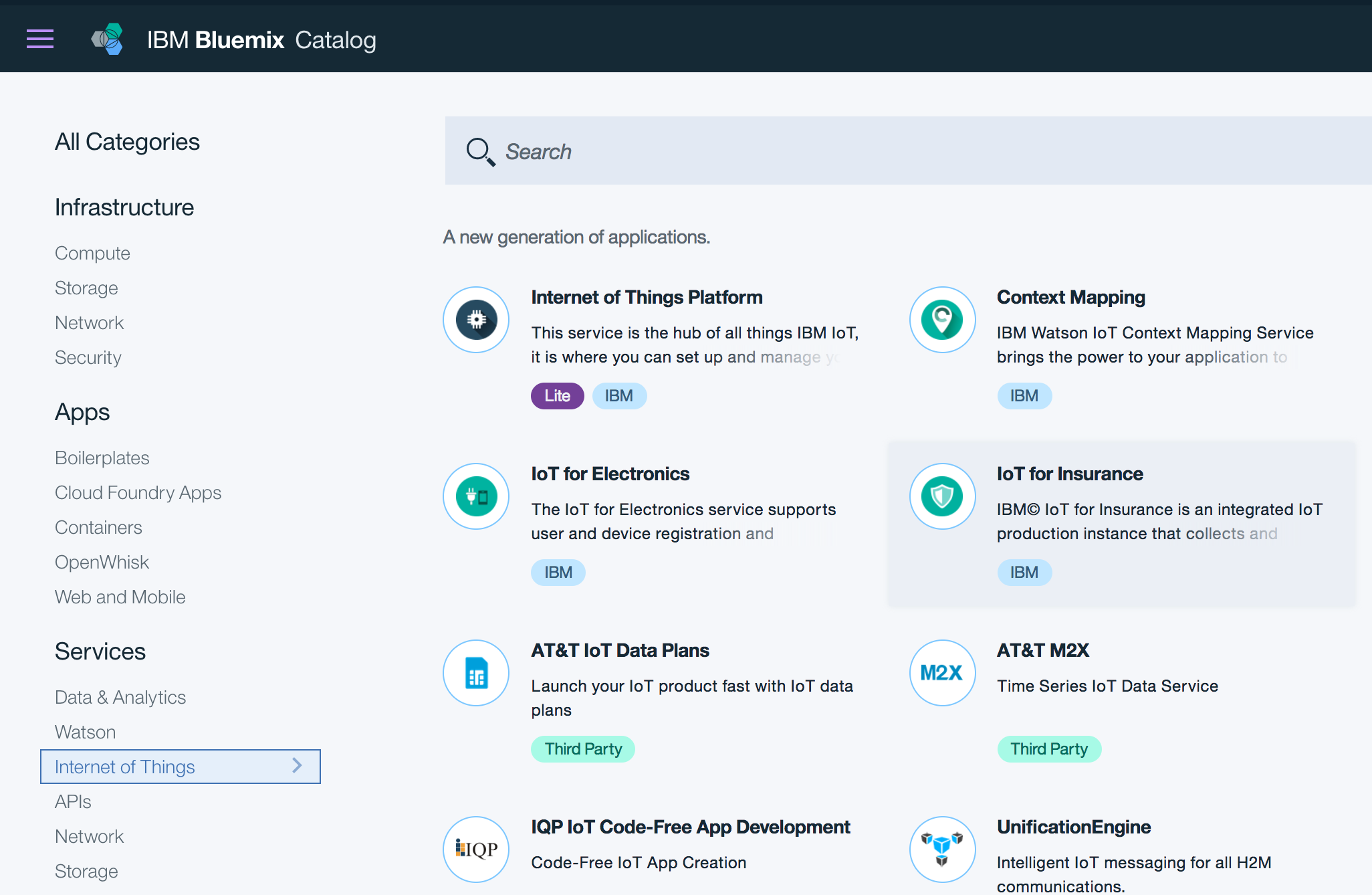
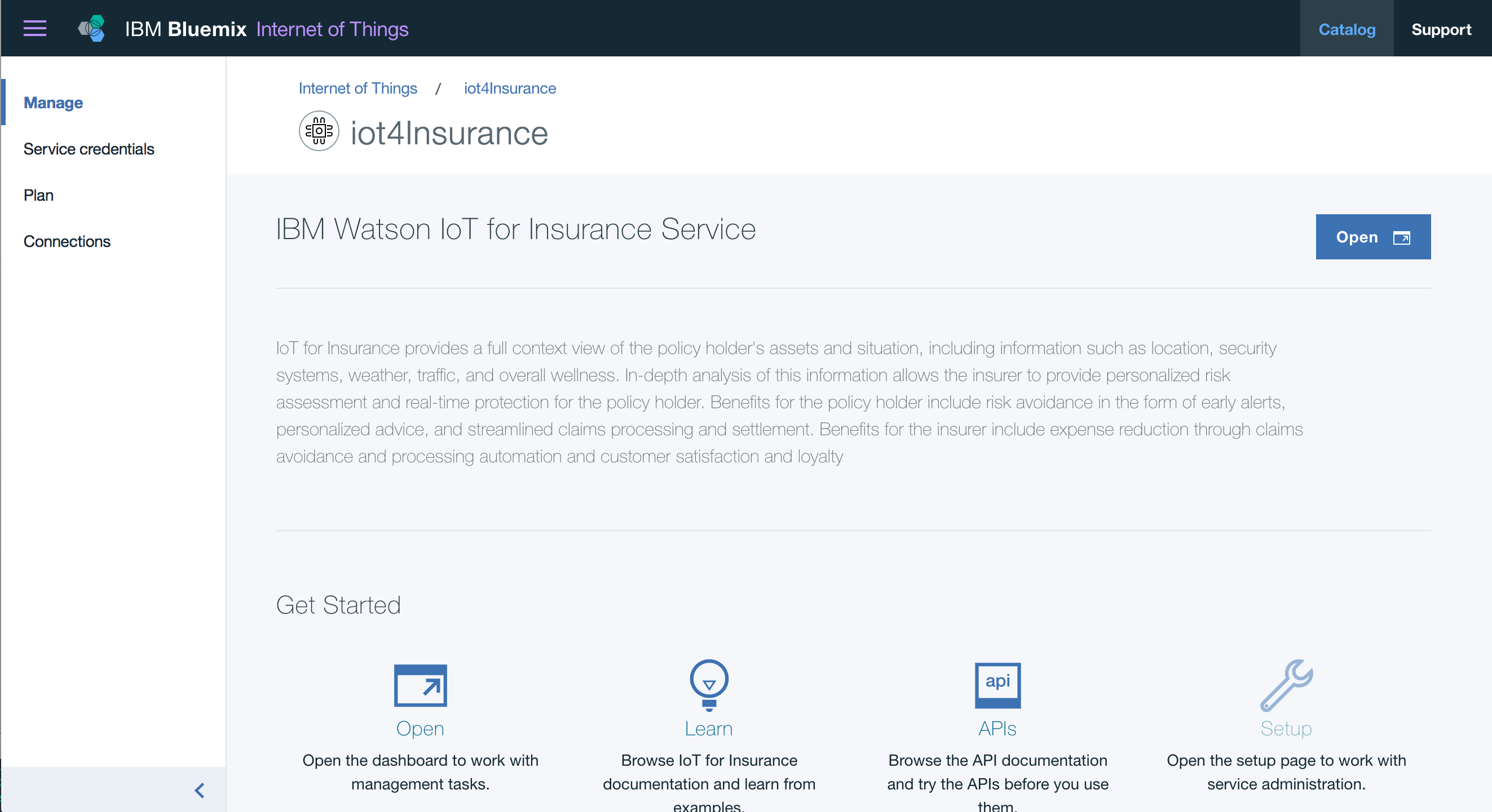
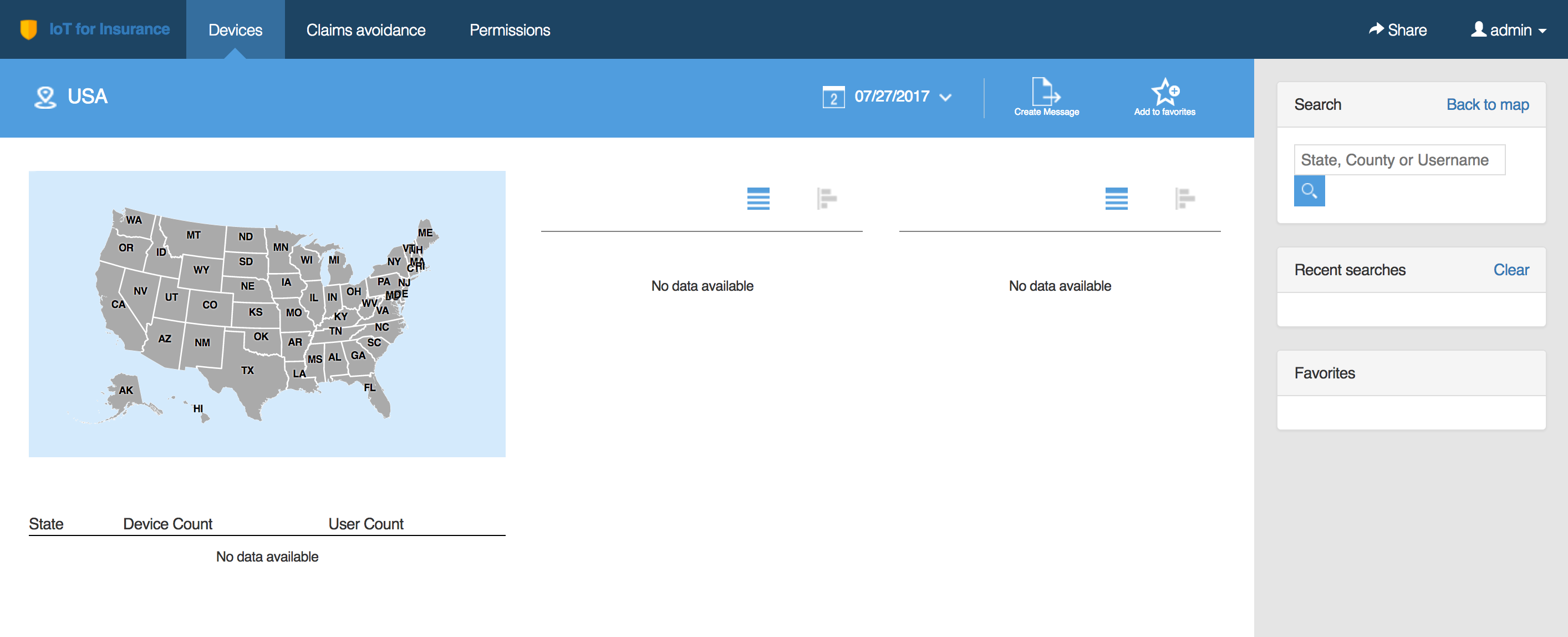
In this lab section, we’ll gain a better understanding of the IoT for Insurance service and how it works. You’ll provision the IoT for Insurance service offering, populate the service with some example data and events, practice querying the IoT4I APIs, and explore the IoT4I dashboard from an insurance provider perspective.

1. Log in to Bluemix to procced with the rest of this lab so that your browser session is authenticated. In a browser, navigate to one of the following regions:

* <https://bluemix.net/>: This link should take you to your default region.
  + - <https://console.ng.bluemix.net/>(Region: US South)
    - <https://console.eu-gb.bluemix.net/>(Region: United Kingdom)
    - <https://console.eu-de.bluemix.net/>(Region: Frankfurt)
    - <https://console.au-syd.bluemix.net/>(Region: Sydney)

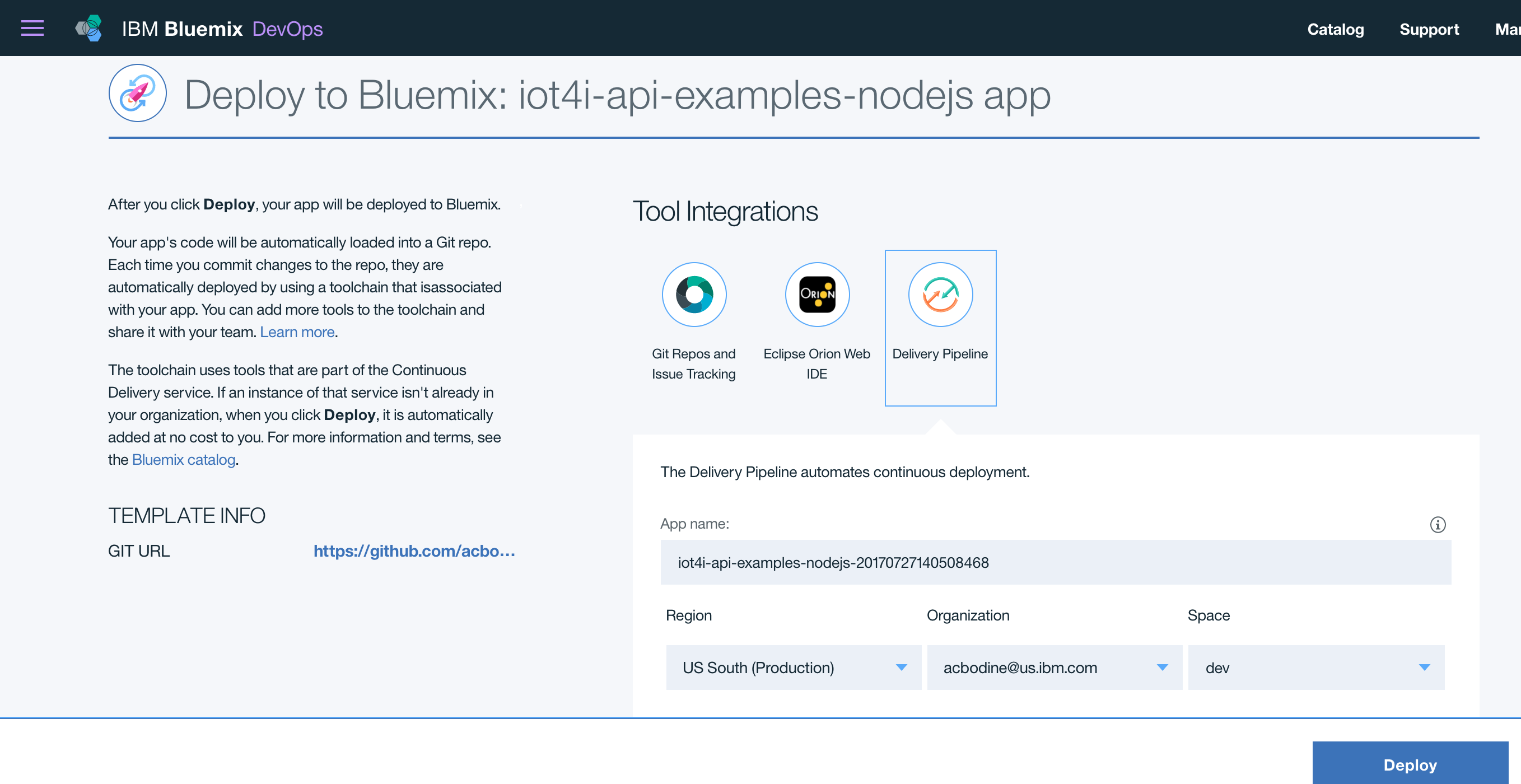
1. Click **Log In** and then enter your login information on the IBM id page and click **Sign in**. You will see your dashboard view.



1. Search the catalog for the **IoT for Insurance** service and provision it to an available organization associated to your lab user account. Name the IoT4I service **iot4Insurance**, leave it unbound, and choose the Standard pricing option.  
     
   
2. Once provisioned you will be prompted on the service landing page to finish deploying the Cloud Foundry applications that make up the rest of the IoT4I service.  
   
3. Review applications and services that have been deployed in Bluemix dashboard and compare them to architecture diagram.  
     
   TODO: Insert architecture diagram (simplified one)
4. Go to the IoT4I dashboard which you can access from the service’s landing page in Bluemix and login with the pre-filled values.   
     
   Note: If the pre-filled values “go away”, use the Service Credentials that were provisioned with the IoT4I service to login to the dashboard.  
     
     
   
5. You’ll notice that the dashboard is pretty empty at the moment. Let’s fix that by using the provided IoT4I examples repository to simulate some devices and events for a policy holder. Navigate to this Github repository in a browser: <https://github.com/acbodine/iot4i-api-examples-nodejs>  
     
   Note: The maintained repository is <https://github.com/IBM-Bluemix/iot4i-api-examples-nodejs>, we are using my fork to leverage fixes that haven’t been merged upstream yet.
6. Scroll down on the page and click on the **Deploy to Bluemix** button.

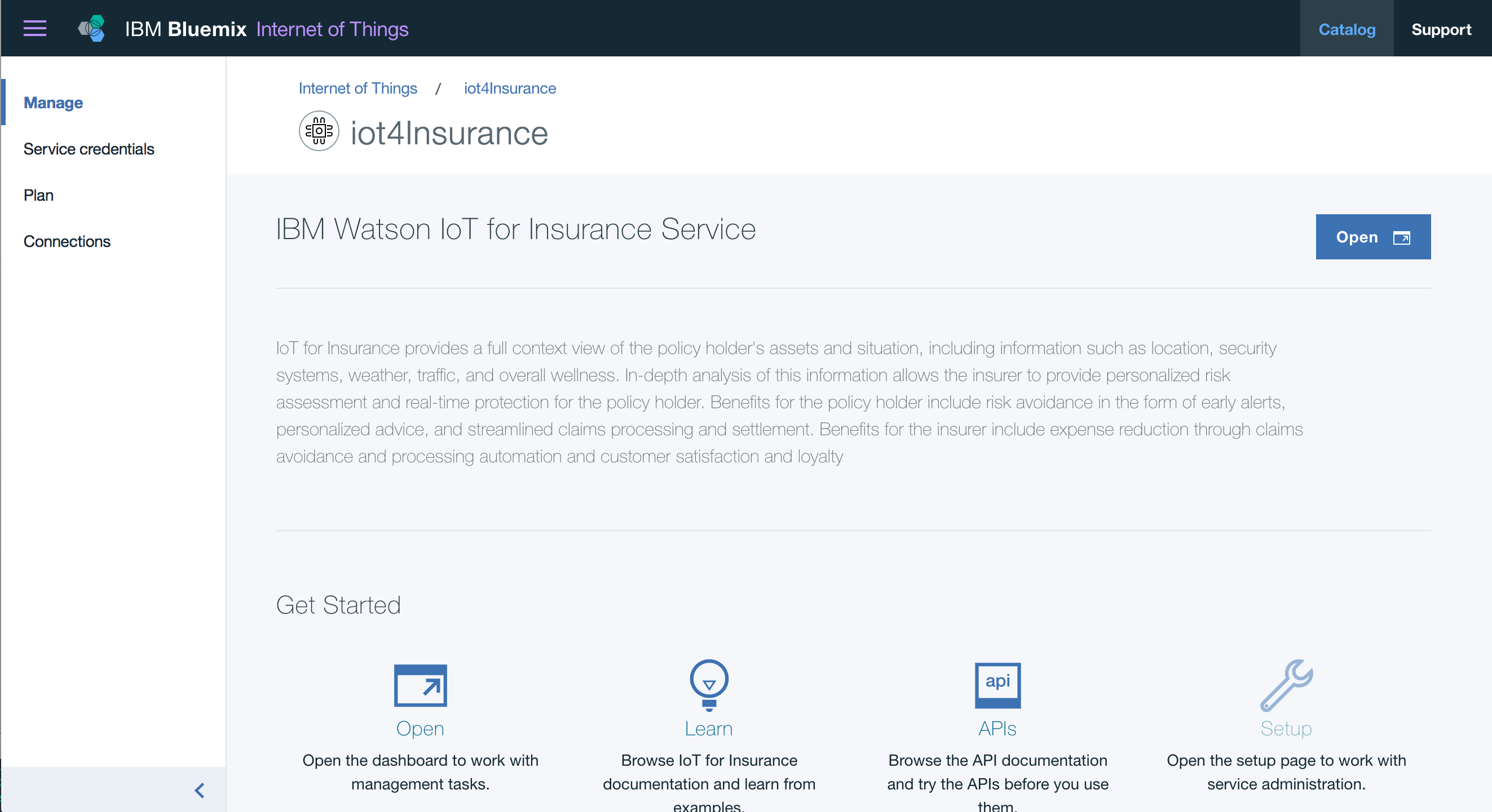
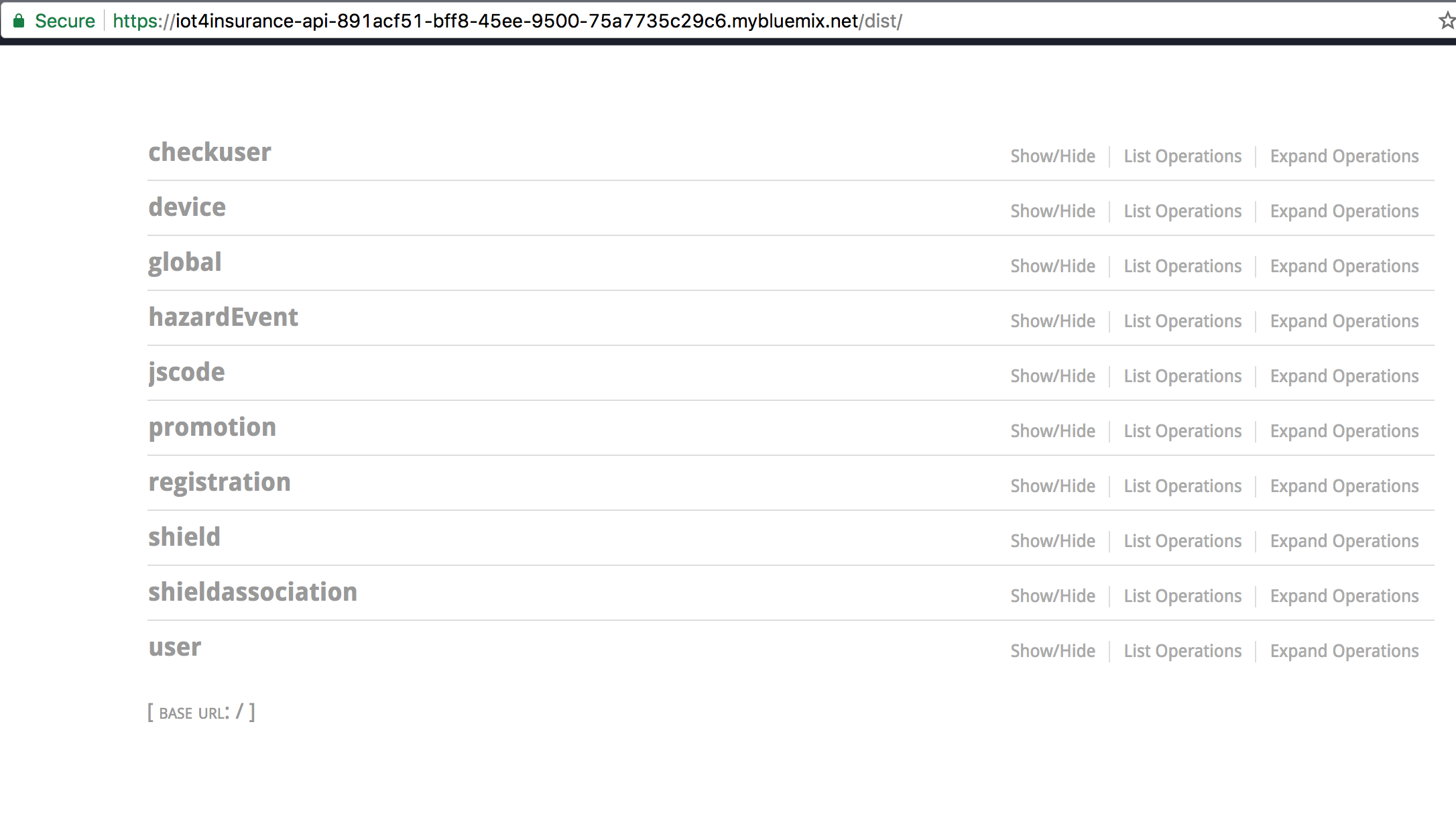
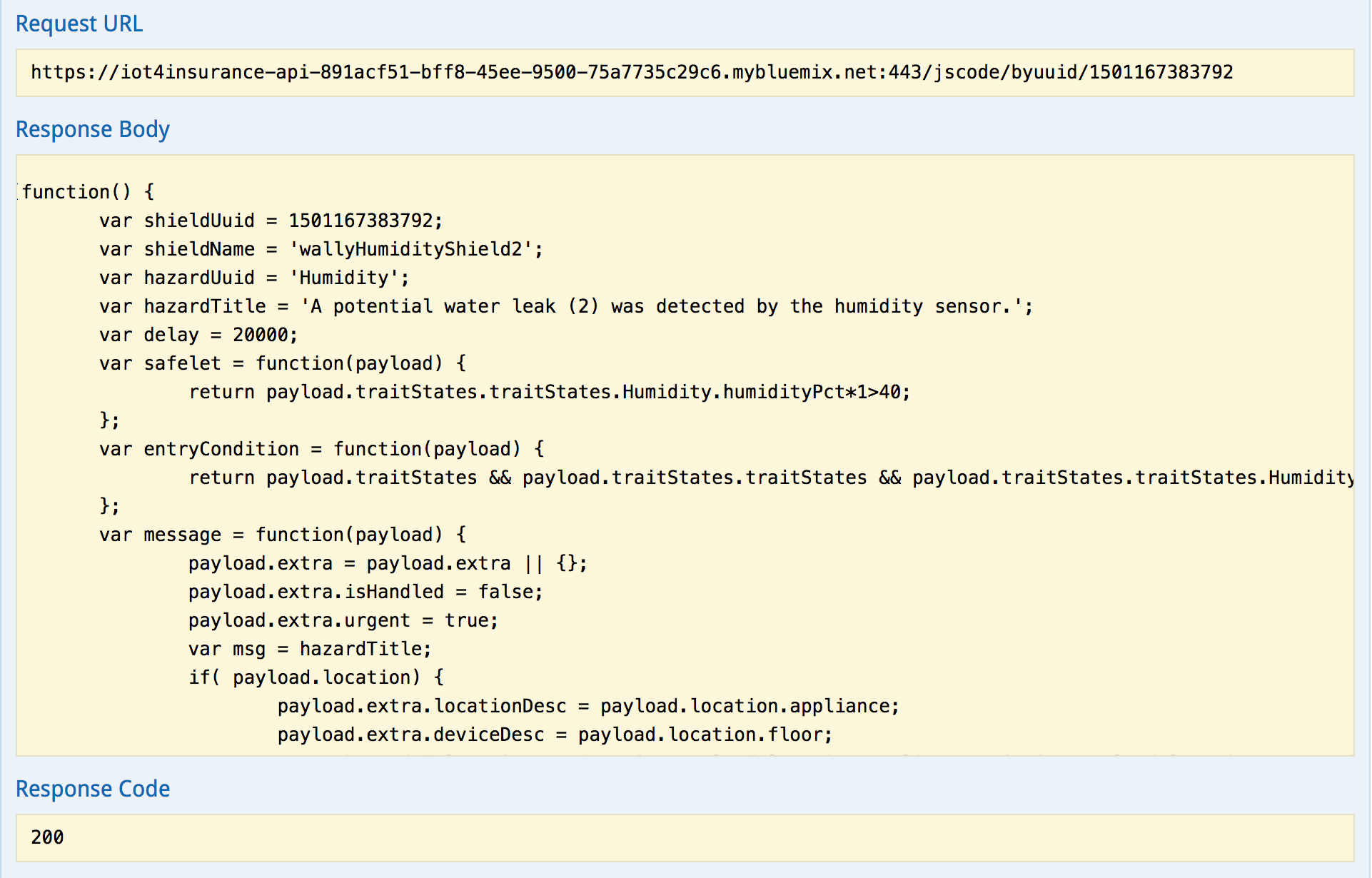
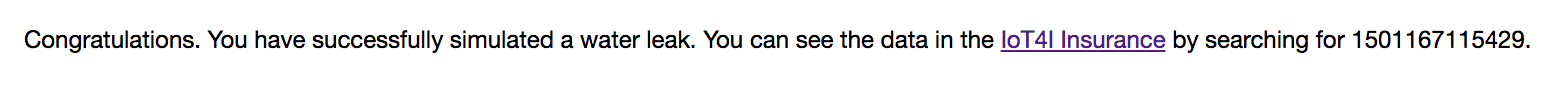
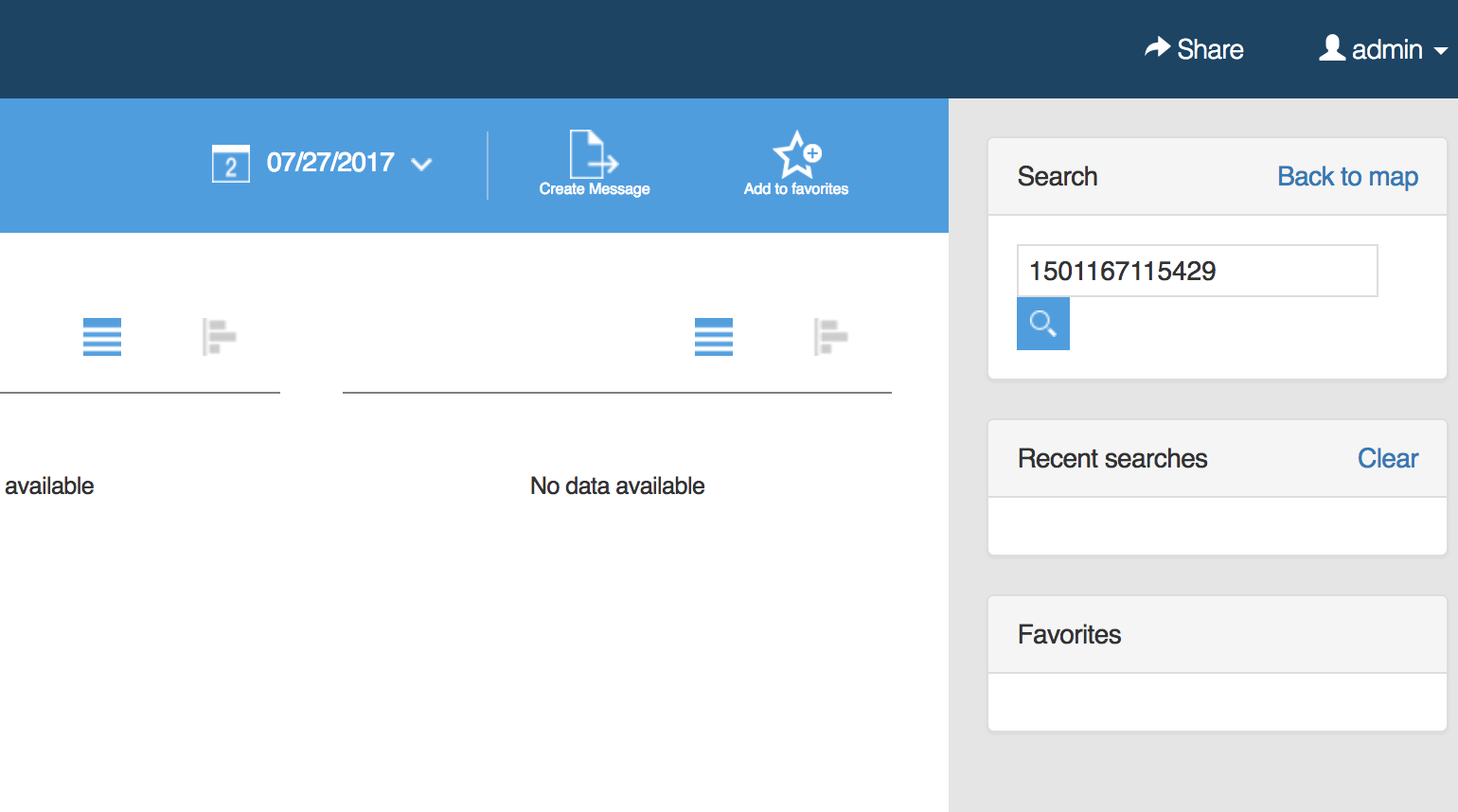
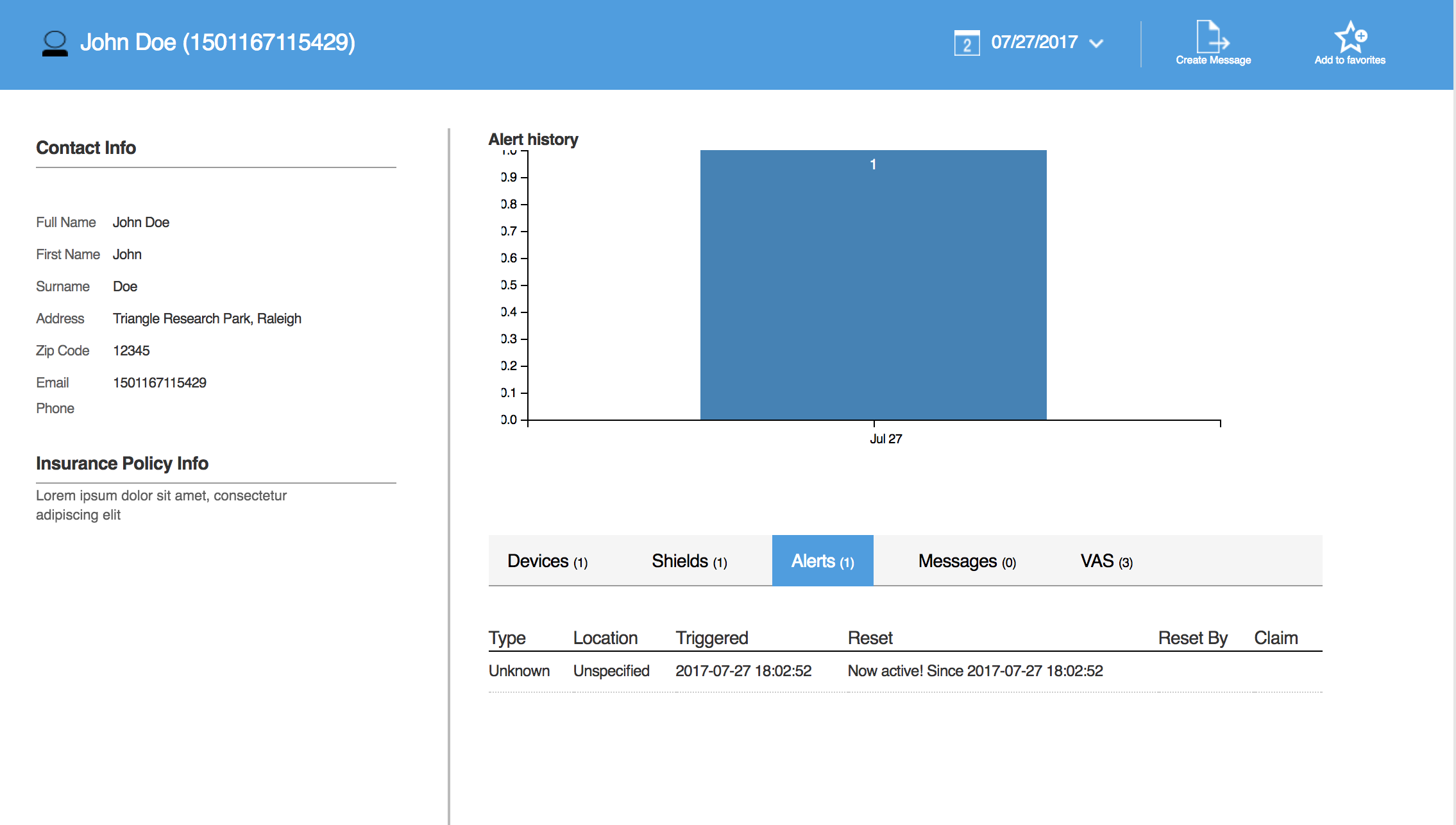


1. The toolchain creation page for your new application will be shown with the Delivery Pipeline options. These options include region, organization, space, and an app name based on the repository name.



Go ahead and click **Deploy** and when the pipeline has finished you can click **View App** to navigate to the examples application we just deployed. You should see something like this:



1. We are going to step through each of these to understand what is happening in the backend. We’ll utilize the browser based IoT4I API documentation app that you can access by click on the **APIs** button on the service landing page in Bluemix:  
     
     
   Try querying **GET /user/all** using the API application. You should see only the Admin user currently in the backend.
2. The first step in the examples application will create a new user in the IoT4I system. To be clear, a user here represents a Policy Holder. Go ahead and click the button, then again query **GET /user/all** with the API application to verify the user has been created.  
     
   
3. The second step will create a shield that detects a water leak. Before clicking the button check what shields exist in the IoT4I service already by querying **GET /shield/all** with the API application. Click the second button and verify the shield has been created in the backend. You should see a shield with the name **Wally Humidity Shield 2**.  
     
   Note: Remember shields trigger actions, and thereby action handlers to do something when a risk has been detected on a policy holder’s asset.  
   
4. The third step will create shield code (Javscript) and associate it to the shield that was created in the previous step. After clicking the button, query the **GET /jscode/byuuid/{id}** with the API application substituting in the shield id that was generated from the previous step. This will give you an idea of what shield code looks like.  
   
5. Now that we have a user and a shield that has shield code, we need to associate the shield to the user. Click the button for step 4, and use the **GET /shieldassociation/all** query with the API application to inspect the associations afterwards.  
     
   Notice there was already an association between the extra shield that is in the IoT4I service by default and the Admin user.  
   
6. The last step will simulate a hazardous event based on the shield and user we just setup. Go ahead and click the last button, and examine the hazard event with the API application **GET /hazardEvent/all**  
     
   
7. When you are finished with the examples application steps, it will prompt you with a user id at the bottom of the page:  
     
     
   Use the userid to search in your IoT4I dashboard for the example data you just injected into the system.  
     
     
     
   

## Summary

You have setup the baseline applications and services that we will make use of for the rest of the lab. We aren’t experts yet on the IoT4I service, but we are starting to get a feel for what the IoT4I service does and how it works. You should feel comfortable with inspecting data in the IoT4I system now, and have a basis for the next lab sections.

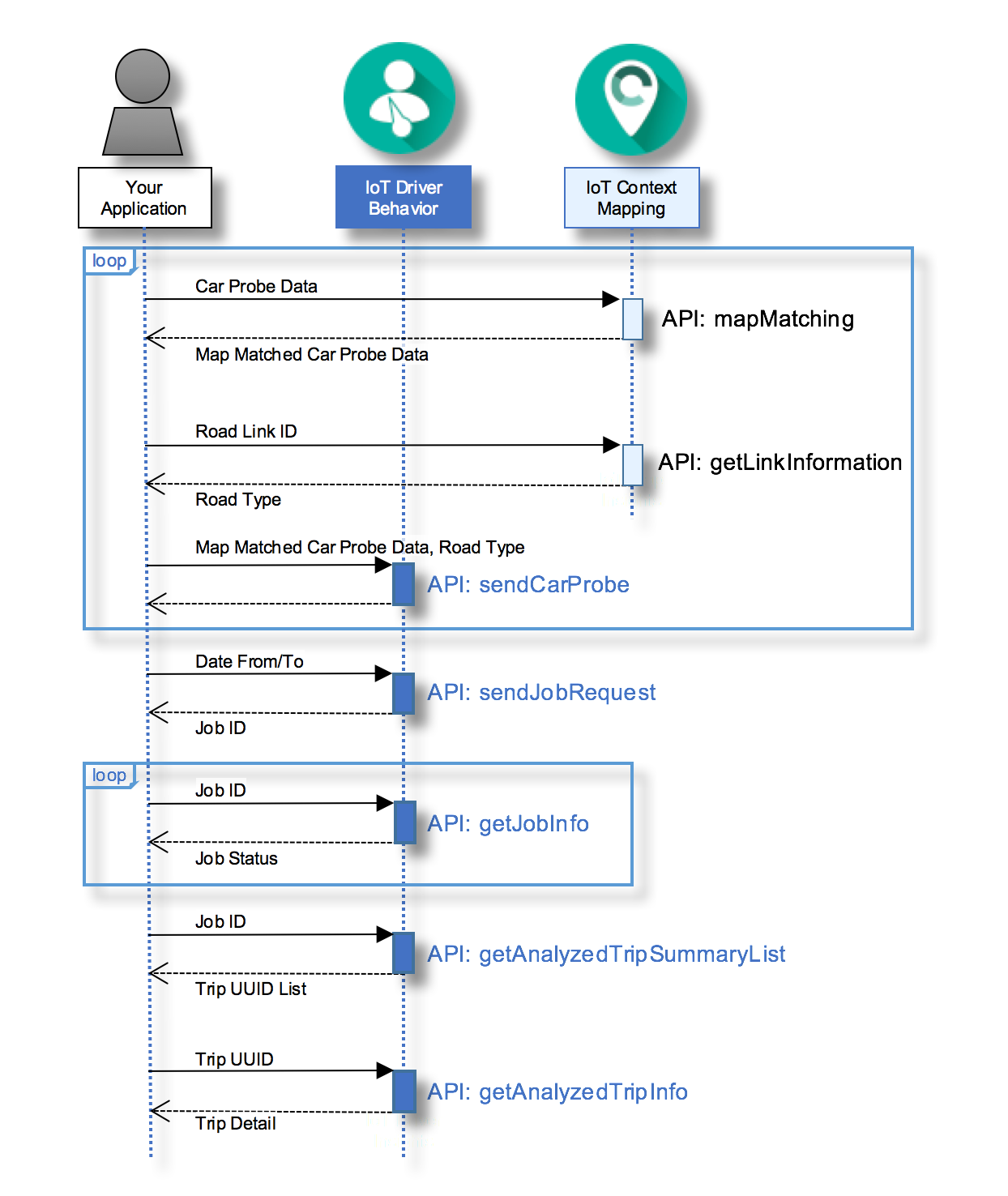
Remember: the Watson IoT Platform is the backbone for how IoT4I ships data from edge devices (insurance assets) to Bluemix. The service wraps and enterprise grade MQTT broker some helpful models to facilitate data propagation to our backend.

# Section 2. Applying Business Logic to IoT4I

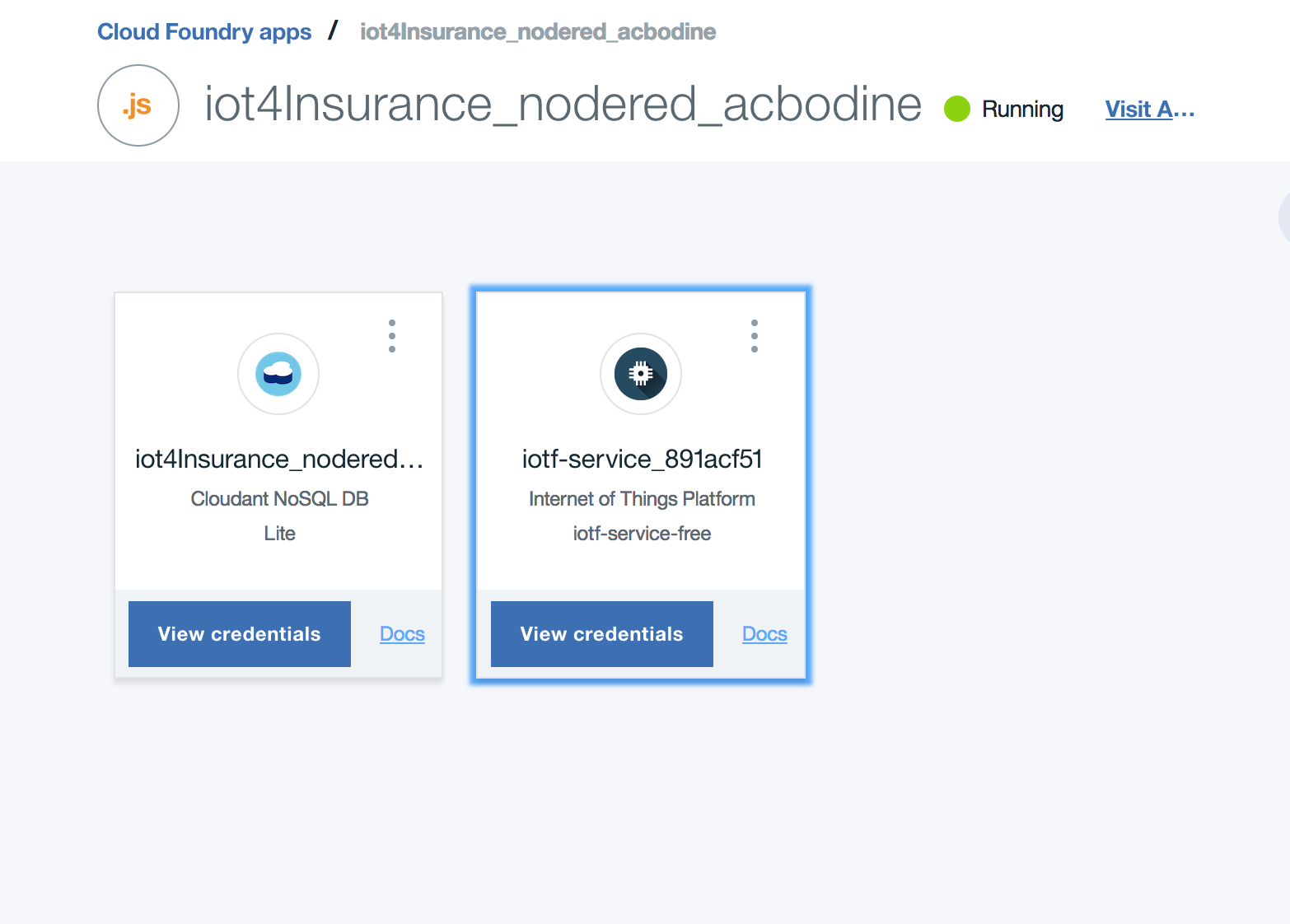
In this section, we will dive deep into the inner-workings of the IoT4I service and implement some new functionalities. This section aims to give insurance industry developers a good sense of how to leverage the IoT4I service and it’s components for their specific use cases.

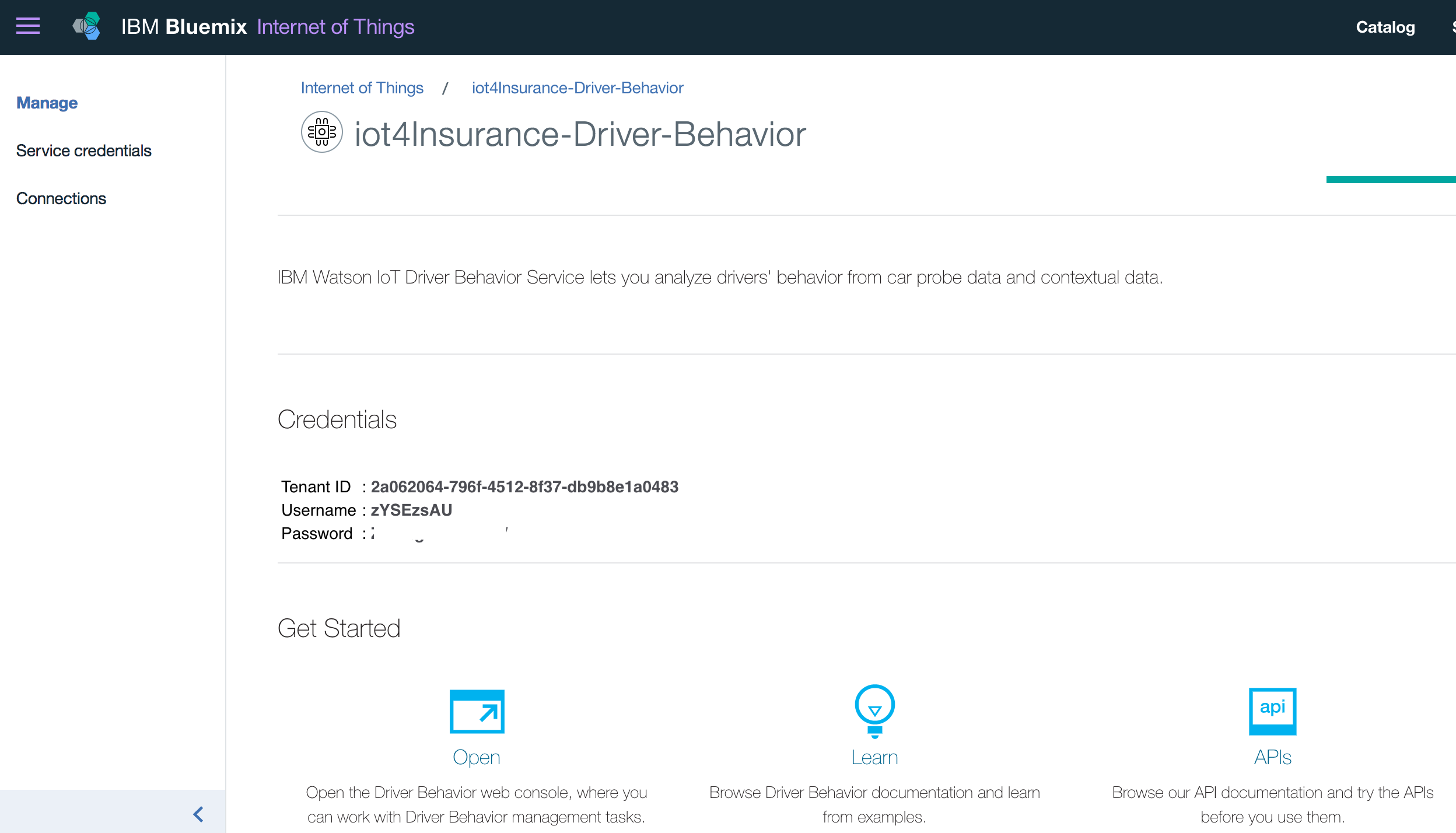
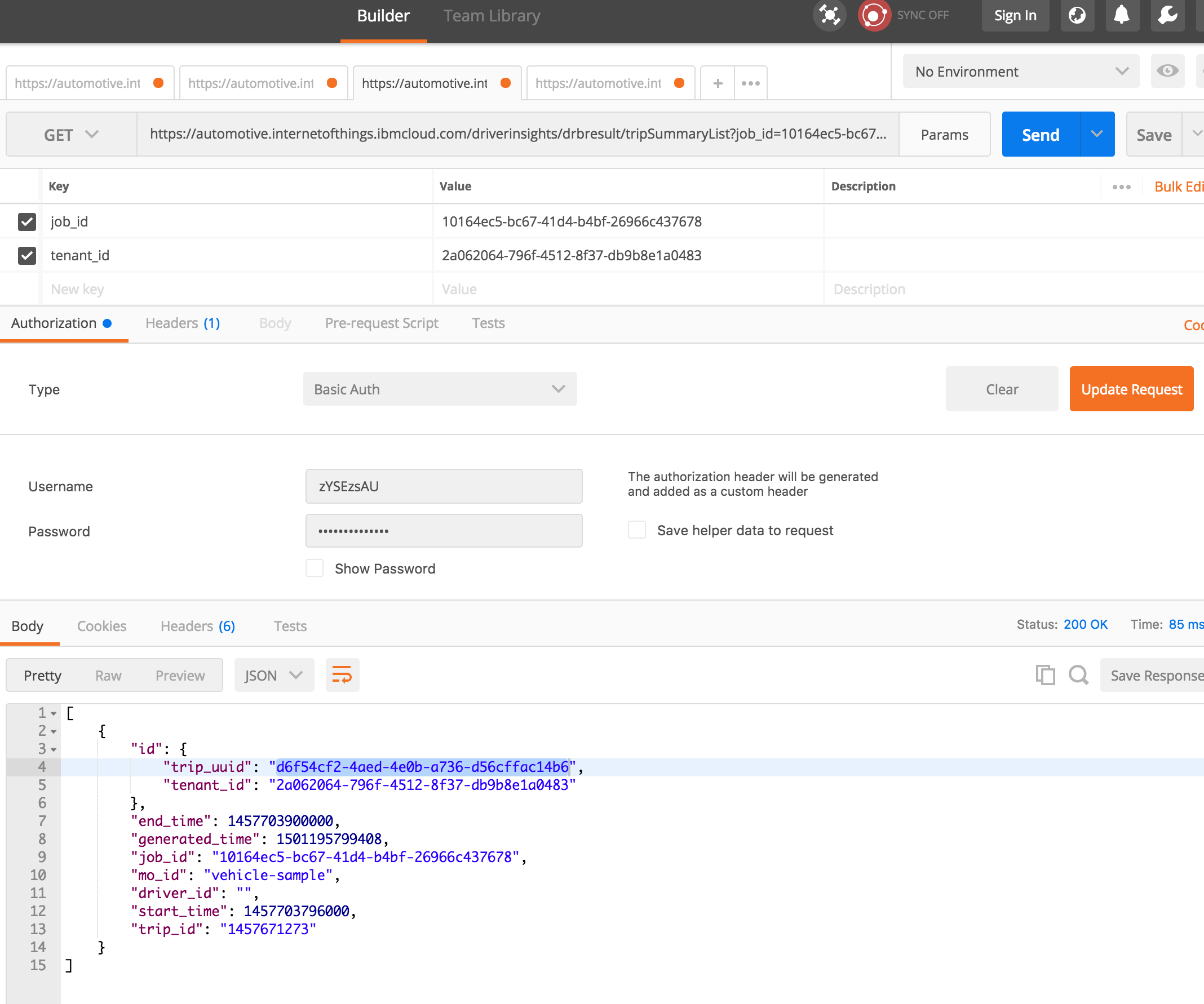
To develop a bit of context, imagine we (the insurers) want to notify policy holders when operators of their vehicles are driving abnormally or dangerous even. We hope this can cut down on the number of claims associated with accidents involving vehicles. Policy holders who opt in will be provided with a “back seat driver” (built in later lab sections) bot that will provide driving tips to encourage safer driving.

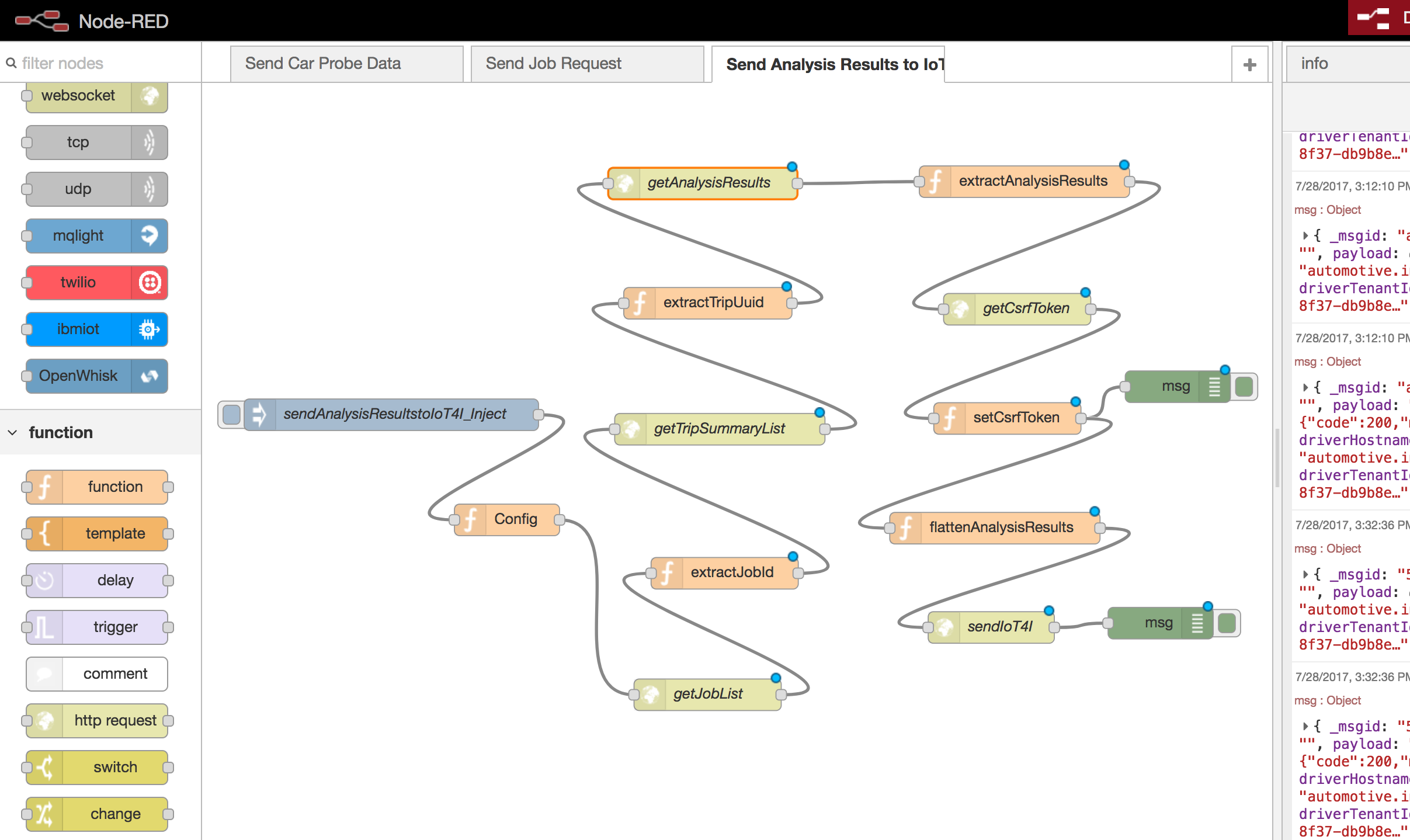
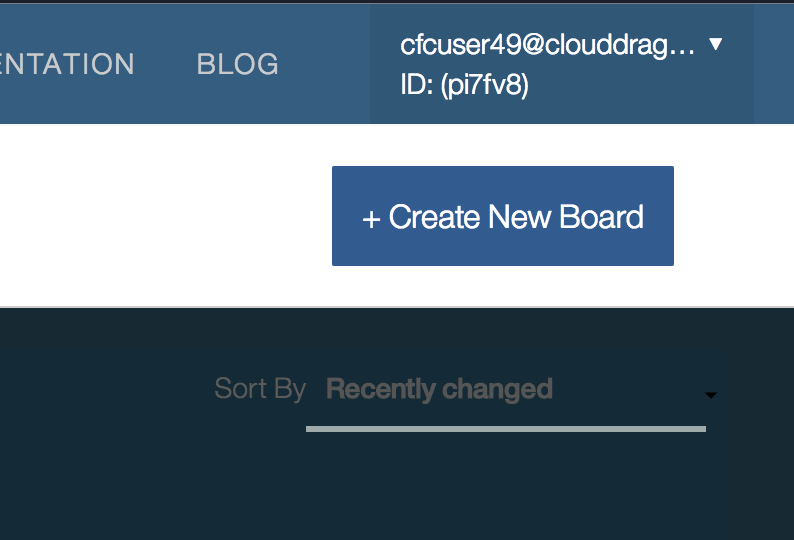
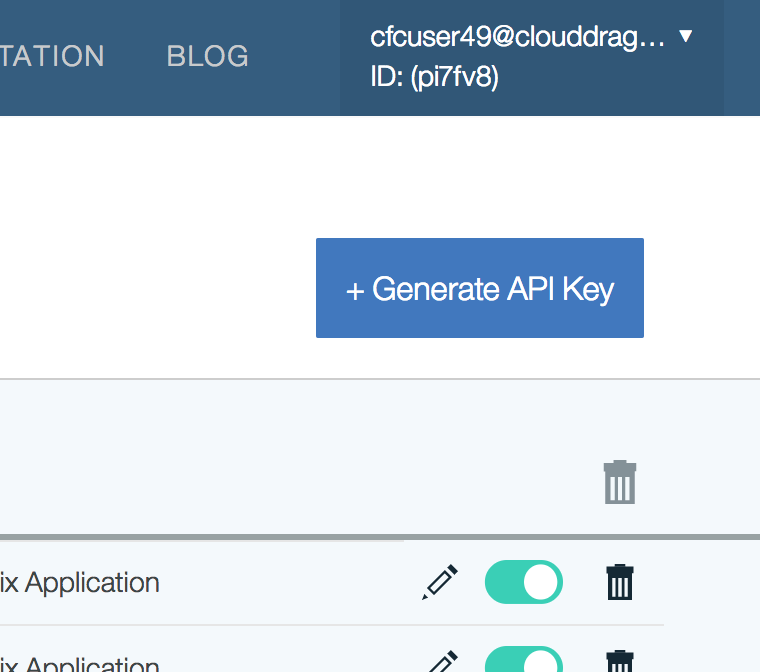
1. We can re-use a repository that has most of the components we need to add to our setup from Section 1 to implement our new dangerous driving features. The following steps are links to said repository. Please follow them one by one:
   1. [Step 1](https://github.com/acbodine/car-data-management#step-1-setting-up-context-mapping): Setting up Context Mapping  
        
      Note: Although the name doesn’t matter, you can name it **iot4Insurance-Context-Mapping** for consistency.
   2. [Step 2](https://github.com/acbodine/car-data-management#step-2-setting-up-driver-behavior): Setting up Driver Behavior  
        
      Note: Again, the name doesn’t matter here but you can call it **iot4Insurance-Driver-Behavior**
   3. [Step 3](https://github.com/acbodine/car-data-management#step-3-creating-the-node-red-application): Creating the Node-RED application  
        
      Note: Name it **iot4Insurance\_nodered\_<dojo-user-id>** replacing the dojo-user-id with the one you were provided. This is the application that will pick up car data from the MQTT broker in Watson IoT Platform, and enhance the data with results from Context Mapping before sending it over to the Driver Behavior service for analysis.

Here is a diagram of the application’s workflow:  
  


Note: Before proceeding, add the existing Watson IoT Platform service (bundled as part of the IoT4I service deployment) as a connection to the Node-Red application we just deployed. Otherwise you won’t receive events in Node-Red when you get to Step 8.



* 1. [Step 4](https://github.com/acbodine/car-data-management#step-4-set-up-the-watson-iot-platform-instance): Set up the Watson IoT Platform instance  
       
     Note: We already have a Watson IoT Platform instance with the IoT4I instance we provisioned in Section 1. You can skip the first 4 parts of this step, so you should start with the “Go to your **Internet of Things Platform** service and click **Launch dashboard**”. The remainder of this step sets up a **Device** inside Watson IoT Platform to represent our car simulator.
  2. [Step 5](https://github.com/acbodine/car-data-management#step-5-setting-up-the-node-red-workflow-for-sending-car-probe-data): Setting up the Node-RED workflow for sending car probe data  
       
     Note: This step is configuring the Node-RED application we deployed previously to point at the Context Mapping and the Driver Behavior services. Be careful with what values you put where.
  3. [Step 6](https://github.com/acbodine/car-data-management#step-6-setting-up-the-node-red-workflow-for-sending-analysis-job-requests): Setting up the Node-RED workflow for sending analysis job requests  
       
     Note: This step is configuring the Node-RED application we deployed previously to send analysis job requests to the Driver Behavior service.
  4. [Step 7](https://github.com/acbodine/car-data-management#step-7-setting-up-the-java-client-of-the-watson-iot-platform): Setting up the Java client of the Watson IoT Platform  
       
     Note: This step builds the local Java application that is our car simulator.
  5. [Step 8](https://github.com/acbodine/car-data-management#step-8-running-the-send-car-probe-data-sample): Running the Send Car Probe Data sample
  6. [Step 9](https://github.com/acbodine/car-data-management#step-9-running-the-send-analysis-job-request-sample): Running the Send Analysis Job Request sample  
       
     Note: You will get back a job\_id after running the analysis request. There is documentation for how to interact with the Driver Behavior APIs via the service’s landing page.  
       
       
       
     Note: Fire up your favorite REST client, here we show Postman. Using the Credentials values we configure our Postman workspace with the following parameters, where the target url without query string is: https://automotive.internetofthings.ibmcloud.com/driverinsights/drbresult/tripSummaryList  
       
       
     Now you have the trip\_uuid to query the action analysis result with. Prepare another Postman tab pointing at the url: https://automotive.internetofthings.ibmcloud.com/driverinsights/drbresult/trip and subsitute in the appropriate parameters like so:  
       
       
     Here you can see that some of the results reported things like “Harsh Breaking” and “Frequent acceleration”.

1. Import the Node-RED workflow [nodered-workflow-submit-results-to-iot4insurance.json](https://raw.githubusercontent.com/acbodine/car-data-management/master/nodered/nodered-workflow-submit-results-to-iot4insurance.json) into a new flow tab in Node-RED like we did before. This flow will automate the task of fetching analysis results from Driver Behavior based on job\_id, and sending those results over to IoT4I so our shield can analyze them.  
     
     
   You need to configure the “Config” node, and enter the User/Password for IoT4I service into each of the yellow nodes (http request nodes). When you have everything configured correctly you should see messages in the debug panel indicating “{“code”: 200, “message”: “ok”}”  
     
   Optional: Now is a good time to fire up the provided tools/hazard-listener.js. You will need to modify tools/local-vcap.json with the correct values from your **Watson IoT Platform** instance associated with IoT4I. Navigate to the Watson IoT Platform dashboard in your browser. The **iotCredentialsIdentifier** is the name of the Service Credentials that were provisioned with the IoT4I service. The **orgid** is in the top right corner or in the url:  
     
     
     
   You can generate a new **apiKey** and **apiToken** in the **Apps** section of the Watson IoT dashboard by clicking the button **Generate API Key**:  
     
     
     
   This tool subscribes to the MQTT broker, and will echo messages as you simulate events and your shields fire actions. Useful for debugging.  
     
   Note: You will want to pay attention to the format of the payloads you are sending from Node-RED to IoT4I.
2. You will modify the iot4i-api-examples repository you cloned in Section 1 for this step, there have been some helpful modifications made to make things simpler.  
     
   Write a driver behavior IoT4I shield, called **resources/driverBehaviorShieldCode.js** to utilize the analysis job results performed in the Node-RED workflow against the Driver Behavior service to fire an action when the risk threshold is reached.  
     
   There is a reference implementation under **resources/refDrBehaviorShieldCode.js** but try to implement the shield on your own.  
     
   For reference, there is a collection of various shields here that the Watson IoT team manages: <https://github.com/ibm-watson-iot/ioti-shields>  
     
   When you are finished, modify your **config.js** to include **mode: “auto”**, and run `$ node app.js` to start the example app like you did in Section 1. The mode: “auto” makes the examples application use your new shield. Step through all steps except the last one, because we are going to use the Node-RED flow we just imported to simulate a driver.
3. Once your new user and shield are in place, configure the Node-RED flow to user the userid and run the flow to simulate a driving behavior analysis result. Verify action handlers fire when your driver-behavior shield signals them.
4. Visualize events in IoT4I dashboard.
5. Optional: There is a separate Github repository (<https://github.com/IBM-Bluemix/map-driver-insights>) that utilizes the Driver Behavior analysis results and constructs a map with overlay representing the drivers behavior relative to location.  
     
   TODO: Deploy this application as a service in Bluemix.  
     
   TODO: Modify shield code to inject url to redirect from the message payload to a rendering of the driver behavior results on a map.

## Summary

In this section, we will dive deep into the inner-workings of the IoT4I service and implement some new functionalities. This section aims to give insurance industry developers a good sense of how to leverage the IoT4I service for their specific use cases.

# Section 3. Create “back seat driver” Chatbot for customers

We have a system that will analyze car probe data to report risky driving behavior. Each time our driving shield fires a hazardous event is created in the storage associated to the IoT4I service. In this section you will utilize the IoT4I apis and prior Watson Conversation experience to create a Chatbot that reports detected hazards to the policy holder. Your chatbot should be able to answer questions the policy holder has in detail, with regards to hazardous events.

# Section 4. Bonus: Enable Chatbot to handle claims with Blockchain

TODO: